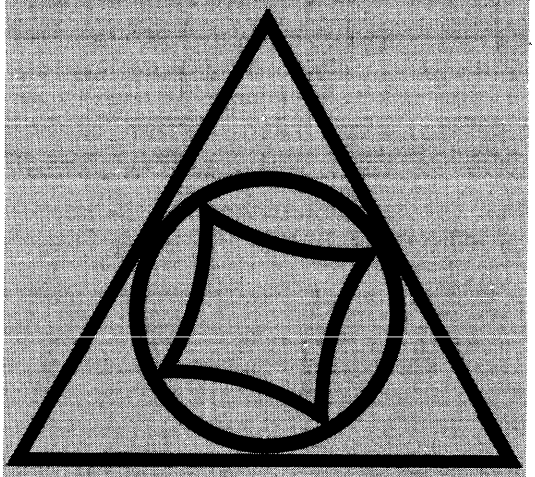


AFIPS

CONFERENCE PROCEEDINGS VOLUME 32

1968

SPRING JOINT COMPUTER CONFERENCE

APRIL 30 - MAY 2
ATLANTIC CITY, NEW JERSEY

THOMPSON BOOK COMPANY
National Press Building,
Washington, D. C. 20004

The ideas and opinions expressed herein are soley those of the authors and are not necessarily representative of or endorsed by the 1968 Spring Joint Computer Conference Committee or the American Federation of Information Processing Societies.

Library of Congress Catalog Card Number 55-44701
THOMPSON BOOK COMPANY
National Press Building
Washington, D.C. 20004

CONTENTS

DIGITAL SIMULATION TECHNIQUES

FAULT DIAGNOSIS

Time-sharing versus batch processing: the experimental evidence

by H. SACKMAN
System Development Corporation
Santa Monica, California

INTRODUCTION

Time-sharing of computer facilities has been widely acclaimed as the most significant evolutionary step that has been taken in recent years toward the development of generalized information utilities. The basic techniques of interactive man-computer time-sharing were developed in the 1950's in connection with realtime command and control computing systems, initially in SAGE air defense. Time-sharing was practiced in these pioneering systems in the sense that many military operators at separate consoles—consoles equipped with push-buttons, CRT displays and light guns—were able to request and receive information from the central computing system at essentially the same time. These historical roots reveal that time-sharing is an outgrowth of realtime system development.

The emergence of time-sharing systems as general-purpose online computing facilities is primarily a development of the 1960's. The users of such systems are a more or less random and changing collection of people at any point in time, typically but not necessarily working on unrelated tasks with different computing programs, entering and leaving the system independently of one another, and using it for varying and largely unpredictable periods of time; such use approaches that of a public utility, roughly analogous to the quasi-random pattern of telephone traffic.

Experimental time-sharing systems were designed and operated in the first half of this decade. The Massachusetts Institute of Technology developed the Compatible Time-Sharing System (CTSS) used for Project MAC (Corbato, Merwin-Daggett, and Daley, 1962);[1] the System Development Corporation developed TSS, the Time-Sharing System for the Advanced Research Projects Agency of the Department of Defense (Schwartz, Coffman and Weissman, 1964),[2] and RAND developed JOSS, the Johnniac Open-Shop System (Shaw, 1964).[3] Commercial applications have sprouted and are rapidly spreading with practically all computer manufacturers marketing or developing some version of time-sharing hardware, software, and support facilities.

In batch or offline processing—the operational workhorse of most contemporary data processing and the evolutionary predecessor of time-sharing—the user typically has indirect contact with the computer. Batch processing has been the rule for economical operation, with stacked jobs done one at a time on a waiting-line basis. Job scheduling is often mediated by programed operating systems based on job priority and estimated computer running time. Turnaround time may take minutes, hours, days or even more than a week before completed outputs are returned in response to job requests. Proponents of stacked-job systems argue that throughput time, useful computations per unit time, is at a maximum with minimum waste of computer resources.

In contrast, time-sharing permits fast and direct access to the computer when the user wants it (provided that guaranteed access is available). For many types of data-processing tasks, the user can get what he wants in minutes rather than hours or days. He may exert continual control over his program and he is free to change his mind and do things differently, at least within system capability, as he interacts with the computer. Time-sharing typically means expense-sharing among a large number of subscribers, with reduced computing costs for many kinds of applications. And perhaps most significant of all, the online nature of time-sharing permits direct man-computer communication in languages that are beginning to approach natural language, at a pace approaching normal human conversation, and in some applications, at graded difficulty levels appropriate to the skill and experience of the user. Time-sharing systems, becuase of requirements for expanded hardware and more extensive software, are generally more expensive to build and operate than closed-shop systems using the same central computer.

Time-sharing advocates feel that such systems more then pay for themselves in convenience to the user, in more rapid program development, and in manpower savings.

Time-sharing, however, has always had its critics. Their arguments are often directed at the efficiency of time-sharing, that is, at how much of the computational power of the machine is actually used for productive data processing as opposed to how much is devoted to relatively non-productive functions (program swapping, idle time, etc.). These critics claim that the cost-effectiveness of time-sharing systems is questionable when compared to modern closed-shop methods, particularly the most advanced versions of fast-turnaround batch systems. Since online systems are presumably more expensive than offline systems, there is little justification for their use except in those situations where online access is mandatory for system operations (for example, in realtime command and control systems).

Time-sharing advocates respond to these charges by saying that, even if time-sharing is more costly with regard to hardware and operating efficiency, savings in programmer man-hours and in the time required to produce working programs more than offset such increased costs. The critics, however, do not concede this point either. Many believe that programmers grow lazy and adopt careless and inefficient work habits under time-sharing. Easy access to the computer, they claim, tends to make users more prone to casual and costly trial and error computer runs with poorly prepared problems, in an effort to trade off computer time against human time, as compared to the batch environment in which computer time is at a premium and programers do more extensive desk checking. In fact, they claim that instead of improving, user performance is likely to deteriorate.

While the controversy continues to rage, many computer installations, pursuing their own unique evolutionary paths, are quietly assimilating the best of both worlds. Time-shared systems are tending to find it convenient to run short jobs to completion and to interleave stacked production jobs into long pauses in online operations as "background" tasks. Conventional operating systems are becoming less conventional by incorporating, in novel forms, many features associated with time-sharing (e.g., direct coupled and remote batch systems). Of special interest are the high capacity, fast turnaround batch systems such as those reported by Lynch (1967).[4] With the continued growth of computer installations, the evolutionary varieties of online and offline facilities are diversifying into new forms and are also converging into hybrid forms. It may well be that many large computer complexes of the future will offer a variety of services in a spectrum of optional online, offline and mixed operational modes.

The above arguments are characteristic of the speculative controversy that has attended the recent rapid growth of time-sharing. For various and complex reasons—which range beyond the purpose and scope of this paper but which are treated elsewhere in detail by the author (1967)[5]—the growth of an applied experimental tradition in man-computer communication has not been vigorously pursued in the computer sciences. Over the last two years, however, this subjective and predisciplinary tenor has finally, and somewhat belatedly, taken a more objective and scientific turn with the advent of experimental comparisons of time-sharing and batch processing in the literature. Five such studies are available and together they comprise an instructive and valuable body of knowledge on methodology and findings (Erikson, 1966,[6] Gold, 1967;[7] Grant and Sackman, 1967[8] Schatzoff, Tsao and Wiig, 1967[9] and Smith, 1967[10]). The objectives of this paper are to critically review and evaluate these studies, summarize areas of agreement and disagreement, point up key gaps in these initial experiments, and sketch the more promising avenues for future research.

Comparative methodology of the experimental studies

Table I outlines and summarizes the main characteristics of the five experimental studies. Unfortunately, an outline of this kind can not do justice to the extensive details of each study, and the interested reader is referred to the original articles. The aim of this section is to review comparative methodology to help determine the technical scope and limitations of these studies. Table I breaks the description of each study down into five categories—subjects, problems, computer system facilities, experimental procedure, and performance measures. Each of these is discussed in turn.

There are a total of 212 subjects in all five studies. It probably comes as no surprise to anyone that college students form the bulk of this population, with only one sample showing a highly experienced group of programmers (Grant and Sackman). It will be noted later that the three studies with small samples were organized around relatively efficient experimental designs to optimize the information yield from the results.

A critical experimental control factor, not shown in Table I, enters into the selection of subjects. This factor is the nature of the computer-related experience of the subjects and their bias, as a result of their

Experimental Characteristics	Erikson (1966)	Gold (1967)	Grant and Sackman (1967)	Schatzoff, Tsao and Wiig (1967)	Smith (1967)
SUBJECTS					
Sample Size	9	60	12	4	127
Type of Subjects	Programmer trainees	Undergraduate and graduate students	Experienced programers from an R&D setting	Undergraduate students with high programing aptitude	Undergraduate and graduate students in an introductory programing course
Experience Level	Less than one year	78% of subjects had taken at least one programing	Average of 7 years experience	"Some" programing experience	Most subjects had less than a year experience
PROBLEMS					
Number and Types of Problems	Two problems, a sorting routine and a cube puzzle	One problem, simulation model of construction industry	Two problems, algebra and maze	Four problems: Monte Carlo integration, algebraic sorting, Pig Latin translator, text format conversion	Two easy "warmup" problems and four experimental problems; cosine infinite series, matrix sorting, language translation, heuristic program
Difficulty Level	Conceptually simple	Moderately difficult, open-ended problem	Moderately difficult for highly experienced programers	Moderately difficult for skilled student subjects	Moderately difficult for beginners
Average Completion Time	A few hours	15 to 20 hours	Approximately 60 hours to complete both problems	Approximately 40 hours to complete all problems	Approximately 60 hours to complete all problems
ONLINE/OFFLINE FACILITIES					
Online Facility	SDC Q-32 Time-Sharing	MIT Time-Sharing IBM 7094	SDC Q-32 Time-Sharing	MIT Time-Sharing, IBM 7094	Burroughs B-5500 batch system at Stanford with "instant" turnaround
Batch Facility	Same facility–simulated offline conditions	MIT Batch Facility, IBM 7094	Same facility–simulation of offline conditions	IBM 7094 scientific batch facility	Same facility, with normal turnaround
Language Used	TINT–interpretative higher-order language for time-sharing	DYNAMO–simulation language used in time-sharing and batch modes	JTS (higher-order language) and SCAMP (machine language)	Not mentioned	Burroughs Extended ALGOL
Batch Turnaround Time	Usually several minutes, variable	6 hours (daytime), 10 hours (overnight), variable	Constant at two hours	Not mentioned	Variable, usually hours
EXPERIMENTAL PROCEDURE					
Experimental Design	2X2 Latin Square: two problems vs. on/off comparison	Two matched groups of subjects	2X2 Latin Square: two problems vs. on/off comparison	Graeco-Latin Square: 4 problems, 4 subjects, vs. on/off compaison	Matched groups of subjects, each subject taking two problems on "batch" and two on "instant"
Statistical Tests	Analysis of variance, some nonparametric tests	A variety of nonparametric statistics comparing the two groups	Analysis of variance, factor analysis	Analysis of variance, correlational analysis	Descriptive statistical comparisons; no tests of statistical significance
Experimental Controls	Counterbalanced order of problems and experimental variables	Questionnaire items, deadline for completed problem	Biographical items, counterbalanced experimental order	Counterbalanced order of experimental design	Counterbalanced order of "batch" and "instant" modes
Motivational Controls	Trainee class grades	Class grades	Job assignment	Not mentioned	Class grades
Recording Procedures	Computer records and personal logs	Computer records, student logs and questionnaires	Computer records, experimenter logs, paper and pencil test	Computer recording, work logs, paper and pencil test	Computer recording, student logs and questionnaire
KEY PERFORMANCE MEASURES	Debug man-hours Computer time Program size Individual differences Basic Programing Knowledge Test Scores	Problem-solving man-hours Computer time Task performance Ratings of written reports Cost comparisons Questionnaire items	Debug man-hours Coding man-hours Computer time Program size Program running time Individual differences Basic Programing Knowledge Test scores	Elapsed time Analysis Programer's time Computer time Number of compilations Total Cost	Initial program preparation Keypunch time Time to prepare new run Number of runs Computer runs per trip Elapsed time Computer time Submission intervals Questionnaire items

TABLE I – Comparative characteristics of five experimental studies comparing time-sharing with batch processing

experience, toward time-shared or batch systems. For example, Erikson's subjects were trained primarily in online programing, whereas Schatzoff, Tsao and Wiig indicated that their subjects had most of their previous experience in batch systems and used batch-oriented procedures in the experimental time-sharing mode. The other three studies had subjects with various degrees of mixed online and offline experience. Obtaining equal familiarity and equal skill in online and offline activities is a difficult kind of experimental control. An antidote to this problem, only partially encountered in these studies, is to deliberately select subjects on the basis of equal experience and to offer them extensive and equal practice sessions in both modes up to some standard level of proficiency.

The problems cover a fairly wide area of programming and problem solving. They include mathematical problems, various puzzles, sorting procedures, and a simulation model. While many of these are typical of program tasks, they can hardly be put forth as representative. For example, there are no large data

base or statistical analysis problems—the kinds requiring large data storage and much computation, which often lend themselves more efficiently to batch processing. On the other hand, neither were there any particularly long, exploratory programs, such as those encountered in graphics and display-centered systems, that lend themselves more efficiently to time-sharing. All problems were individual rather than team-oriented tasks. Perhaps most basic of all, there are no empirical norms available to determine the representativeness of the various data processing tasks.

The difficulty level of most studies varies from "conceptually simple" to "moderately difficult." There were no reported cases of subjects who were unable to complete the experimental tasks even though some studies indicated missing data. The average time for subjects to complete their experimental tasks varied from a few hours up to 60 hours. The longer problems give some idea of the manpower costs of conducting this kind of research and underscore the general tendency to use students or trainees.

The problem posed by Gold for his student subjects differed from the other four studies in that it was not a programming task. The experimental vehicle was a computerized simulation model of the construction industry and its market; the student's task was to formulate and construct a set of decision rules to maximize his profits as an independent, small-scale builder in this simulated, cyclical market. The computerized simulation model provided criterion performance scores which constituted feedback for the students by indicating their profit level in response to decision rule inputs for this open-ended problem.

The online/offline facilities reveal key dilemmas faced by the experimenters in attempting to construct unbiased and equal conditions for an objective comparison between time-sharing and batch processing. In the two SDC studies, time-sharing was real and batch processing had to be simulated on the Q-32 Time-Sharing System. In Smith's study, the basic system was batch and time-sharing was simulated by providing "instant" turnaround time (several minutes); there were no conversational or interactive features in this simulated online condition. While Smith's study is primarily a comparison between conventional batch and fast-turnaround batch, it is included here because of the useful information it contributes to timing and feedback aspects of the time-sharing/batch controversy. The two MIT studies were the only ones offering ostensibly comparable online and offline modes without resorting to some form of simulation.

The computer language employed is another difficult control variable. Gold and Smith were able to have their subjects use the same language which, they claimed, was equally applicable and useful for both modes. Erikson used TINT, an interactive language, for the noninteractive mode. In the Grant-Sackman study, most subjects used JTS, originally a batch processing language, later adapted to time-sharing. Schatzoff, Tsao and Wiig do not mention any languages at all; since they indicate that their subjects used batch procedures in the time-sharing mode, and that their subjects only had a brief indoctrination in time-sharing, one cannot help but wonder whether their comparison provided reasonably comparable starting conditions under both experimental modes. This same criticism applies, at least in part, to the two SDC studies.

Experimental control problems are compounded further with respect to turnaround time under the batch mode. These turnaround times vary from minutes, to hours, to next-day turnaround. Only Grant and Sackman controlled this variable at a constant value of two hours. While this procedure provided rigorous experimental control over turnaround time, it was obviously unrealistic in not providing variability in turnaround service. The other investigators apparently left their subjects to the vagaries of their particular operational batch system without obtaining exact measures of turnaround time for each run. In addition, for all studies, it is not clear whether subject waiting time during batch turnaround was spent working on the problem, or not working on the problem, and for some of the studies, whether it was included or excluded in subject logs of man-hours spent on the experimental task. Future studies in this area should incorporate systematic variation and control of machine turnaround time, and careful recording of what the subject does during this time. Lack of experimental controls in this area unquestionably increases error variance in performance measurement and decreases the reliability of the final results.

The next category in Table I, experimental procedure, reveals a remarkable spectrum of experimental designs for the five studies. The Graeco-Latin Square configuration of the Schatzoff, Tsao and Wiig study is the most sophisticated experimental design, whereas the Smith study merely compared mean scores of matched groups without any reported measures of dispersion or any tests of statistical significance. With a sample of four subjects, the Schatzoff, Tsao and Wiig study had to have optimal statistical efficiency to demonstrate reliable results, whereas with Smith's sample of 127 subjects, ob-

served mean differences are correspondingly more reliable. Nevertheless, the absence of statistical tests and neglect in reporting measures of dispersion in the data from which statistical tests may be constructed, are to be deplored since these practices reduce the cost-effective yield of an experiment, leave quantitative results ambiguous, and deprive the larger community of useful information on individual differences.

The three experiments using Latin-Square designs employed analysis of variance and correlational techniques to the findings, which not only provided statistical tests for online/offline comparisons, but also yielded valuable information on problem and individual performance differences. The Grant-Sackman study was the only one which included an exploratory factor analysis of subject performance. Gold's tests were exclusively non-parametric, and as in Smith's study, no quantitative findings on individual differences were reported.

The experimental controls included matching of groups in the studies with the largest samples (Gold and Smith) primarily on the basis of questionnaire items. The remaining three studies, using Latin-Square designs, involved stratified samples of subjects (e.g., experienced programmers, high-performance students, trainees) with random assignments of subjects to the various test conditions in accordance with the experimental design. Motivational controls essentially consisted of class grades for students and fulfillment of job assignments for the experienced SDC programmers. Individual competition probably spurred most subjects to work hard at their assigned tasks and to keep most of their problem strategy and tactics to themselves, at least in the three small sample experiments. These motivational constraints were probably less effective in the two experiments with the larger subject samples.

The recording procedures characteristically included computer recording for machine usage, subject logs for man-hours spent on experimental tasks, questionnaires for selecting and matching subjects and for collecting observations and ratings on self-performance. Gold collected the most comprehensive questionnaire data on his subjects before, during, and after the experiment. Items included biographical data, problem-solving behavior, and comparative attitudes toward time-sharing and batch processing. Paper and pencil tests of programmer ability were used in three studies. Schatzoff, Tsao and Wiig selected students who received a grade of A on the IBM Data Processing Aptitude Test; in the two SDC studies, the Basic Programming Knowledge Test (developed at the University of Southern California) was administered to the subjects. Of the various recording procedures, the computer records were probably the most objective and the subject logs were the ones most open to intentional and unintentional errors. In the three studies with small samples, it was easier to keep the subjects under surveillance, to monitor their manual reporting procedures, and to tactfully resolve discrepancies as they arose. In the two larger sample studies, experimenter monitoring of individuals had to be more indirect. Neither Smith nor Gold discuss possible errors or bias in student reporting procedures in any detail.

The last category in Table 1 covers the experimental payoff, performance measures. The two key performance measures running through all five studies are man-hours and computer time required to complete experimental tasks. The computer time measure is the most straightforward. Man-hour measures appear in various forms and are partitioned in different ways. For example, the two SDC studies distinguish coding time from debugging time; Gold uses a single measure of problem-solving time; the other two studies incorporate an overall measure of elapsed time with different ways of slicing man-hours spent on experimental tasks. Cross-comparisons are somewhat difficult because measures are defined differently for different contexts.

The three studies utilizing Latin-Square designs devote some attention to the analysis of individual performance differences. Although individual differences were not originally a key objective of these studies, there was an unavoidable serendipitous fallout of human differences from the analysis of variance in each investigation. The study of individual differences was carried furthest in the Grant-Sackman experiment through an exploratory factor analysis of performance measures.

Questionnaries bearing on subject preference between online and offline operations were used in the Gold and Smith studies. This performance measure, while subject to the problems that plague questionnaire reliability and validity, is of special interest in the time-sharing/batch controversy in providing an index of user attitudes and in testing for a bandwagon effect.

The SDC studies used final program size and running time as measures of performance. It is surprising that these objective, easily obtainable, and obvious measures of programing efficiency were not reported in the other two studies requiring completed programs. It would be of value to test whether programs are written more efficiently, as measured by these two indices, in the online or the offline mode.

The performance measures in two cases (the Gold study and the Schatzoff, Tsao and Wiig study) in-

clude estimates comparing online/offline costs which incorporate man and machine factors. In both cases these costs were derived from experimental measures of human and machine time which were used as empirical parameters in simple cost models.

The Gold study had some unique measures of performance. The most notable is task effectiveness—how well the subject performed his task, as measured by his profit in the simulated construction industry model. Whereas the other studies measured effectiveness in terms of how long it took the subject to complete a standard task, Gold was also able to obtain a quantitative measure of how *well* the subject performed (profit). Gold's study was also unique in obtaining written reports from each subject to assess their mastery and grasp of the experimental task from an independent source of (verbal) data. He was also the only experimenter who required his subjects to give a standardized account of their computer runs on a run-by-run basis. These various measures enabled Gold to obtain more diversified data than any of the other studies on problem-solving and decision-making activities in the online and offline setting.

In an attempt to explore the relation between paper and pencil tests and performance on experimental tasks, the two SDC studies incorporated scores on the Basic Programming Knowledge Test in their analyses of individual differences. Since sample sizes were small, and since validity correlations of successful paper-and-pencil tests of job performance are traditionally moderate to low, these tests, at best, were tentative probes.

Summing up, what are the chief methodological characteristics, strengths, and weaknesses of these five studies in regard to subjects, problems, computer facilities, experimental procedure, and performance measures? The subjects were primarily students or trainees—experienced data processing personnel were used in only one study. While the experimental problems ranged over a broad area, involving many types of data processing tasks and procedures and requiring many hours for successful solution, certain types of tasks prominently occurring in batch processing and in time-sharing are not encountered, and it is difficult to assess how representative these problems are for data-processing in general and how well they are balanced for an objective online/offline comparison. Some of the toughest problems were met in providing comparable time-sharing/batch facilities; matched computers and equivalent languages posed many problems, and the crucial variable of batch turnaround time was generally not systematically controlled. The experimental procedures show diverse levels of experimental sophistication, with the most critical problems occurring in the observation and measurement of human performance. Even at this early stage, the range of performance measures is impressive, covering a variety of man-machine indices; on the other hand, the paucity of automatically collected measures of human and program performance, particularly in the online setting, is somewhat disappointing. More powerful online techniques, such as regenerative recording of user performance—a technique for capturing the complete real-time interaction between the user and the computer so that it can be played back in its entirety for later analysis (Sackman, 1967)[11]—should be developed and applied to the experimental investigation of a broad spectrum of user tasks.

Results of experimental studies

In this section the key results of each of the five experiments are successively summarized in tabular form and briefly evaluated; a composite box-score of the results of all five experiments is also presented. The next section, Interpretation, provides a cross-comparison and an overall evaluation of method and findings. The tabular format for the results of each experiment essentially consists of a list of key performance variables, with observed scores in the online and offline mode, and obtained statistical significance for the observed difference (providing such tests were conducted); additional notable findings follow this list, and each table concludes with a box-score listing what the author believes to be the most significant results of the given study.

Performance Measure	Time-Shared Mode	Batch Mode	Statistical Significance
Debug Man-Hours	5.0	9.6	.06*
Computer Time (sec.)	146	492	.04*
Number of TINT Statements	51	53	-
Range of Individual Differences in Debug Man-Hours	8:1 and 3:1	7:1 and 6:1	-
Range of Individual Differences in Computer Time	5:1 and 4:1	4:1 and 3:1	-

*Nonparametric tests of mean differences in adjusted scores.

BOX SCORE

1. Time-sharing requires fewer man-hours and much less computer time for debugging with programer trainees than a simulated noninteractive mode when an interpretive language developed for time-sharing is used in both modes.
2. Individual differences in performance are larger than online/offline system differences.

TABLE II—Main results of the Erikson study

Performance Measure	Time-Shared Mode	Batch Mode	Statistical Significance
Problem-Solving Man-Hours	15.5	19.3	.05
Task Performance (profit)	$1404	$1215	.002
Computer Time (min.)	*7.13	1.25	.001
Understanding of Problem (rating of written report)	Higher	Lower	.04
Overall Cost	No appreciable difference		-
Subject Preference	More desirable	Less desirable	.001
Range of Individual Differences in Problem Solving Man-Hours	7:1	4:1	-

*There is an additional editing load under time-sharing with DYNAMO that is not present under the batch mode. Comparable adjusted figures are not available.

BOX SCORE

1. Time-sharing requires fewer man-hours than batch processing in a problem-solving task with a sample of 60 students.
2. Time-sharing is accompanied by a higher level of effectiveness than batch processing.
3. Batch processing requires much less computer time than time-sharing for the given problem.
4. Time-sharing is strongly preferred by student subjects over batch processing, and this preference grows with increasing exposure to both modes.

TABLE III – Main results of the Gold study

Performance Measure	Time-Shared Mode	Batch Mode	Statistical Significance
Debug Man-Hours	19.3	31.2	.05*
Computer Time (sec.)	747	548	-
Program Size (machine words)	2534	2339	-
Program Run Time (sec.)	3.7	3.7	-
Range of Individual Differences in Debug Man-Hours	14:1 and 13:1	6:1 and 9:1	-
Range of Individual DIfferences in Computer Time	3:1 and 11:1	7:1 and 8:1	-
Factor Analysis of Performance Measures	Two factors: programing speed and program economy		

*Analysis of variance on transformed scores.

BOX SCORE

1. Time-sharing requires fewer man-hours to debug programs for highly experienced programers than a simulated batch system with a two-hour turnaround time.
2. Computer time, program size, and program running time are not significantly influenced by batch versus time-sharing modes under the conditions of this experiment.
3. Individual performance differences in a highly experienced group of programers are considerably larger than observed system differences between time-sharing and batch processing.
4. An exploratory factor analysis of the experimental data revealed two basic programing skills--programing speed and program economy.

TABLE IV – Main results of the Grant-Sackman study

Performance Measure	Time-Shared Mode	Batch Mode	Statistical Significance
Elapsed Time (days)	29.5	46	.08
Analysis Time (min.)	3059	2295	-
Programer Time (min.)	5672	2737	.02
Computer Time (min.)	92	101	-
Compilations	118	49	.05
Total Cost (dollars)	1579	1075	.08
Range of Individual Differences for above variables			3:1 to 4:1

BOX SCORE

1. Students experienced in batch techniques and who are inexperienced in time-sharing techniques, and who essentially use batch procedures under both modes, use less of their own time and incur lower man-machine costs to prepare, code and debug programs under the batch mode than in time-sharing.
2. Time-sharing, even with subjects unfamiliar with its use, requires less total elapsed time than batch processing to prepare, code and debug programs.
3. In a select group of students, individual differences in performance are much larger than system differences between time-sharing and batch processing.
4. While time-sharing required more compilations than batch processing under the conditions of this experiment, there was no significant difference in the expenditure of computer time under both modes.
5. The above conclusions are contingent upon the type of programing languages used in both modes and the extent and variability of batch turnaround times--both of which were not reported in the original article.

TABLE V – Main results of the Schatzoff, Tsao and Wiig study

Performance Measure		"Instant"* Mode	Batch* Mode
Initial Program Preparation (min.)		440	405
Time to Keypunch Original Program (min.)		109	108
Time to Prepare New Run		311	293
Number of Runs per Student		7.1	6.6
Computer Runs per Trip		2.5	1.9
Elapsed Time (days)		3.0	3.7
Computer Time (average min. per run)		.277	.186
Intervals Between Successive Computer Runs (min.)	First Quartile	40	210
	Median	205	450 (est.)
	Third Quartile	not reported	not reported
Student Preference		70%	24%

*No tests of statistical significance were reported.

BOX SCORE

1. Instant turnaround batch results in less elapsed time than conventional batch to prepare, code and debug programs for a relatively large sample of student users.
2. Instant turnaround results in heavier computer time expenditure than conventional batch processing.
3. Instant turnaround is preferred by substantially more students than conventional batch processing.
4. Instant turnaround is associated with changes in programing working patterns that are characterized by shorter intervals between successive job runs and earlier completion of the experimental task.
5. The above conclusions are contingent upon the variability of the data and derived tests of statistical significance which were not reported in the published study.

TABLE VI – Main Results of the Smith Study

Interpretation

What are the consistent patterns, the ambiguities, and the gaps in the findings of the five studies? Six types of performance measures in the data are reviewed: subject time, computer time, system costs, user preference, individual differences and special measures. Composite results for the first four measures are shown in Table VII.

	Man-Hours	Computer Time	Costs	User Preference
Erikson	Time-Sharing* 1.9:1	Time-Sharing 3.4:1	Time-Sharing	Time-Sharing
Gold	Time-Sharing 1.2:1	Batch 5.7:1	Approx. Same	Time-Sharing
Grant and Sackman	Time-Sharing 1.6:1	Batch 1.4:1	Approx. Same	Time-Sharing
Schatzoff, Tsao and Wiig	Batch 2.1:1	Time-Sharing 1.1:1	Batch 1.5:1	Not Reported
Smith	Instant** 1.2:1	Batch 1.5:1	Approx. Same	Instant
Median for All Studies	Time-Sharing 1.2:1	Batch 1.4:1	Approx. Same	Time-Sharing Preferred

*The mode showing a reported advantage appears in each box together with its favorable ratio; e.g., this entry shows less man-hours for time-sharing at a 1.9:1 ratio.

**"Instant" batch is treated in this table as a simulated version of time-sharing.

TABLE VII—Composite Experimental Box-Score: Time-Sharing Versus Batch Processing

Four out of five studies show time-sharing (or its simulated equivalent) to result in less human time in producing programs or solving problems than batch processing (or its simulated equivalent). Only the Schatzoff, Tsao and Wiig study shows a reverse trend, and these authors admit to their subjects' use of batch techniques under time-sharing. Further, these authors do, in fact, show less *elapsed* time for completion of experimental tasks under time-sharing. On the other hand, the Erikson study, which shows the greatest relative performance advantage for time-sharing (almost 2:1 in trainee man-hours), was based on the use of an interactive interpretive language in both modes, which created a favorable bias for time-sharing. With these qualifiers at both extremes in mind, it appears that time-sharing does tend to require less elapsed time and fewer man-hours to produce programs and solve problems. The magnitude of this performance advantage is not very large—the median improvement for all five studies is roughly 25 percent less human time under time-sharing than in batch processing. No claims are made for the meaning or the stability of this or the other medians, but they do give a crude rule of thumb for the pooled results of these five studies.

The comparative results on computer time show no clear-cut trend. They range from a 6:1 ratio in favor of batch processing in Gold's study (an admittedly inflated ratio since computer times in the two modes are not strictly comparable), to middle-of-the-road ratios varying from 1.5:1, to 1.4:1 in favor of batch in the next two studies (Smith, and Grant and Sackman) to 1.1:1 in favor of time-sharing in the Schatzoff, Tsao and Wiig study to a 3:1 ratio in the same direction in Erikson's study. The conservative conclusion is that computer time is highly sensitive to the unique conditions of each experiment and that no consistent advantage seems to accrue to either mode as far as the pooled data of these studies are concerned. On the other hand, the median ratio is 1.4:1 in favor of batch computer time, and perhaps this might serve as a "best" estimate for the pooled data.

The combined results for human time and computer time, assuming that the reported trends are reliable, reinforce the hypothesis that in time-sharing the user trades off computer time for his own time. That is, to state the extreme case, rather than check out his program as thoroughly as he can at his desk, the time-sharing user is more likely to take a less-polished version or only a partially checked program to the computer for a trial run than his batch counterpart. Time-sharing critics will assail this practice by claiming that the user develops careless and lazy work habits through excessive reliance on extra computer runs; time-sharing advocates will assert that such behavior allows more intelligent exploration and testing of alternative solutions at a natural pace for the user when and as problems arise. While there is probably some truth to both positions (which are not mutually exclusive), it is hoped that future experimental analyses of problem-solving stages in both modes will lead to improved hypotheses in the dynamics of man-computer communication that will supersede these rather crude stereotypes of user behavior under time-sharing and batch processing.

The data on system costs also shows no definite trend. While only two studies reported cost estimates, the overall results indicate that one study shows definitely less expense for time-sharing (Erikson—less computer time and fewer man-hours), three studies show roughly equal costs for both modes (computer time and man-hour results in opposite cost directions), and one (Schatzoff, Tsao and Wiig) shows

a 50 percent cost advantage for batch processing. Here again the results are contingent upon unique experimental conditions.

The comparative results on user attitudes show a decided preference for time-sharing in Gold's study and a strong preference for "instant" over conventional batch in Smith's study. In the two SDC studies, although a formal poll was not taken, most subjects apparently preferred time-sharing over the simulated offline conditions. Schatzoff, Tsao and Wiig do not report any opinion data. The available evidence, such as it is, indicates that time-sharing and "instant" batch (minutes of turnaround time) are preferred over conventional batch (hours of turnaround time). There are no data to indicate how time-sharing would fare against fast-turnaround batch. While it is not at all surprising that the subjects liked easy access to computers and fast computer response, it is nevertheless desirable to demonstrate this experimentally. User preference for the interactive conversational features of time-sharing over and above the fast response of instant batch is still a moot point.

Individual differences were investigated in those three studies using analysis of variance techniques. In each case, performance differences between subjects were larger and overshadowed system differences between time-sharing and batch processing. The observed ranges were sometimes at an order of magnitude between best and poorest performers—even with relatively stratified subject samples. Although no measures of the dispersion of subject performance were reported in the Gold and Smith studies, it is hoped that such analyses will be forthcoming since these two studies have the largest user samples. Except for the Grant-Sackman exploratory factor analysis of individual performance differences, no systematic analysis of human differences was attempted.This factor analysis resulted in two well-defined and essentially independent factors—one concerned with programming speed (low coding and debugging time, and low computer time) and the other with program economy (smaller and faster running programs). While the entire area of individual differences in man-computer communication, from economic, system performance and humanistic points of view, is probably more important than operating system differences, nevertheless, little has been done and virtually nothing is known about such individual differences.

Gold's study is the only one that attempted to assess how well the experimental task was done and how well it was understood. He found that the time-sharing group made a significantly larger profit in the simulated construction industry market and that they also understood the problem better than the batch processing group, at least as determined by independent ratings of written reports from both groups. These findings, but just for this one study, support the contention that time-sharing leads to a higher-quality end product than conventional batch.

The distribution of successive computer runs in Smith's study shows interesting differences between the instant and the conventional batch modes. Median turnaround time for subjects to prepare their next run is more than twice as short in the instant mode. Problem-solving speed is apparently slower under conventional batch. As time-sharing adherents have often pointed out, ready accessibility of computer services lends itself to natural pacing in problem-solving tasks, whereas the forced delays inherent in conventional batch turnaround time tend to disrupt normal problem-solving patterns and inhibit spontaneous closure. Unfortunately, the intervals between successive computer runs under batch, and between successive console sessions under time-sharing were not reported in the other studies, thus, the above hypothesis is still conjectural. Nor does this hypothesis bear upon the differences between instant batch versus interactive time-sharing.

The two SDC studies, at least as far as program size and running time are concerned, are neutral with respect to Gold's results in that no significant program differences were found between time-sharing and simulated batch modes. It would be of interest if online/offline experiments were conducted in which subjects were instructed to write short and fast-running programs in addition to solving the experimental tasks. Without such instructions subjects are likely to concentrate primarily on working solutions rather than on operating costs of the finished product. Program size and running time can be used to measure comparative "quality" of final programs—a useful measure in realtime computing systems, for example, where space and running time are often at a premium.

What is the composite picture of experimental comparisons of time-sharing and batch systems, at least as depicted by the available studies, and what are the main gaps in this portrait? The rather blurred portrait that emerges seems to show that time-sharing is more likely to get the job done faster, perhaps at higher quality, at a working pace preferred by users. Batch processing may, more often than not, require less computer time, and perhaps at somewhat less cost than time-sharing. Prior familiarity with batch or time-sharing, and built-in individual or institutional bias toward one or the other, especially if coupled to computer system tools or languages built for one mode rather than the other, could easily shift the balance in the familiar direction. Overshadowing these system differences are wide-ranging individual differences which seem to account for most of the observed variance in performance.

Except for Gold's exploratory work on the quality of the user's final product, virtually nothing has been done on human creativity in the online/offline setting. No studies have been performed on the distinctive characteristics of conversational interaction in time-sharing and whether these characteristics offer any advantage over fast batch systems. No work has been done on a comparative error analysis of user performance between time-sharing and batch processing except for some preliminary tabulations listed by Smith (1967).[12] There are no detailed case histories on the real time pattern of problem-solving—a kind of time-and-motion study of human decision making—that occurs under online and offline conditions, away from the computer as well as at the computer. Until we understand the behavioral dynamics of man-computer communication we can hardly expect to understand the relative tradeoff between alternative modes of data processing, including the comparison between time-sharing and batch processing. It is not within the scope of this paper to develop a systematic framework for comparative analyses of user performance; this has been done elsewhere by the author.[11] Suffice it to say that it is an encouraging sign of the times that significant experimental attempts have been made to obtain open scientific data on camparative man-computer systems, and that the application of computers to human affairs is becoming more a shared, applied science and less a secretive, crude, trial-and error technology.

REFERENCES

1 F J CORBATO M MERWIN-DAGGETT R C DALEY
An experimental time-sharing system
Proceedings of the Spring Joint Computer Conference 1962 pp 335-355

2 J I SCHWARTZ E G COFFMAN C WEISSMAN
A general purpose time-sharing system
Proceedings of the Spring Joint Computer Conference 1964 vol 25 pp 397-311

3 C J SHAW
The JOSS system
Datamation vol 10 no 11 November 1964 pp 32-36

4 W C LYNCH
Description of a high capacity, fast turnaround university computer center
Proceedings of 22nd National Conference Association for Computing Machinery Thompson Book Co 1967 Washington D C pp 273-288

5 H SACKMAN
Experimental investigation of user performance in time-shared computing systems: retrospect prospect and the public interest
SP-2846 System Development Corporation Santa Monica California 5 May 1967

6 W J ERIKSON
A pilot study of interactive versus noninteractive debugging
TM-3296 System Development Corporation Santa Monica California 13 December 1966

7 M GOLD
Methodology for evaluating time-shared computer usage
Doctoral Dissertation Massachusetts Institute of Technology Alfred P Sloan School of Management 1967

8 E E GRANT H SACKMAN
An exploratory investigation of programmer performance under on-line and off-line conditions
SP-2581 System Development Corporation Santa Monica California 2 September 1966

9 M SCHATZOFF R TSAO R WIIG
An experimental comparison of time sharing and batch processing
Communications of the ACM vol 10 no 5 May 1967 pp 261-265

10 LYLE B SMITH
A comparison of batch processing and instant turnaround
Communications of the ACM vol 10 no 8 August 1967 pp 495-500

11 H SACKMAN
Computers system science and evolving society
John Wiley and Sons Inc New York 1967

12 LYLE B SMITH
Part one: a comparison of batch processing and instant turnaround
Part two: a survey of most frequent syntax and execution-time errors
Stanford Computation Center February 1967

Computer scheduling methods and their countermeasures

by EDWARD G. COFFMAN, JR*
Princeton University
Princeton, New Jersey
and
LEONARD KLEINROCK**
University of California
Los Angeles, California

INTRODUCTION

The simultaneous demand for computer service by members from a population of users generally results in the formation of queues. These queues are controlled by some computer scheduling method which chooses the order in which various users receive attention. The goal of this priority scheduling algorithm is to provide the population of users with a high grade of service (rapid response, resource availability, etc.(, at the same time maintaining an acceptable throughput rate. The object of the present paper is to discuss most of the priority scheduling procedures that have been considered in the past few years, to discuss in a coherent way their effectiveness and weaknesses in terms of the performance measures mentioned above, to describe what the analysis of related queueing models has been able to provide in the way of design aids, and in this last respect, to point out certain unsolved problems. In addition we discuss the countermeasures which a customer might use in an attempt to defeat the scheduling algorithm by arranging his requests in such a way that he appears as a high priority user. To the extent that we can carry out such an undertaking, the single most important value of this consolidation of the results of analysis, experimentation, and experience will be in the potential reduction of the uncertainty connected with the design of a workable service discipline.

By a grade or class of service we mean the availability of certain resources (both software and hardware), a distribution of resource usage costs, and a well-defined distribution of waiting or turn-around times which applies to the customer's use of these resources. In multi-access, multiprogramming systems throughput may conveniently be measured in terms of computer operating efficiency defined roughly as the percentage of time the computer spends in performing user or customer-directed tasks as compared with the time spent in performing executive (overhead type) tasks. We shall avoid trying to measure the programmers' or users' productivity in a multi-access environment as compared with productivity in the usually less flexible but more efficient batch-processing environment. For discussions on this subject see References 1 and 2 and the bibliography of Reference 3.

With a somewhat different orientation some of these topics have been covered elsewhere. In particular, Coffman,[4] Greenberger[5] and in more detail Estrin and Kleinrock[6] have reviewed the many applications of queueing theory to the analysis of multiprogramming systems. In addition, Estrin and Kleinrock have surveyed simulation and empirical studies of such systems. However, in contrast to the purposes of the present paper, the work cited above concentrates on mathematical models and on service disciplines to which mathematical modeling and analysis have been to some extent successfully applied. We shall extend this investigation to several priority disciplines not yet analyzed and to others which more properly apply to batch-processing environments. Furthermore, as indicated earlier, the present treatment investigates on a qualitative basis the detailed interaction of the customer and the overall system with the service discipline.

Classification of priority disciplines

Before classifying priority disciplines consider the following very general notion of a queueing system. In Figure 1 we have shown a feedback queueing system consisting of a computer (service) facility, a queue or system of queues of unprocessed or incompletely

*This research was supported in part by the Bell Telephone Laboratories, Murray Hill, New Jersey.
**This research was supported in part, under Contract DAABO7-67-0540, United States Army Electronics Command, and also in part by the Advanced Research Projects Agency (SD-184).

processed jobs (or more generally, requests for service), a source of arrivals requesting service and a feedback path from the computer to the system of queues for partially processed jobs. The system will be defined in any given instance by a description of the arrival mechanism, the service required from the computer, the nature of the computer facility, the service discipline according to which the selection of service requests from the system of users is determined, and the conditions under which jobs are "fed back" to the system of queues. In all of the service disciplines discussed in the next section we make the following assumptions: 1) the arrival mechanism is such that if the arrival source is not empty it generates new requests according to some probability distribution, 2) the service disciplines are such that the computer facility will never be idle if there exists a job in the system ready to be executed.

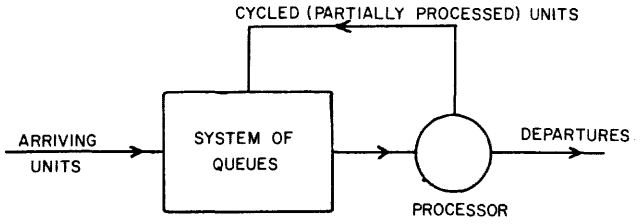


Figure 1 – Feedback queueing system

There are a variety of ways to classify priority service disciplines. Indeed, one point of view is expressed by saying that priorities may be *bought* (e.g., in disciplines where bribing is allowed,[7] *earned* e.g., by a program demonstrating favorable characteristics in a time-sharing system), *deserved* (e.g., by a program exhibiting beforehand favorable characteristics), or a combination of the above. For our purposes we shall classify a given priority method according to the properties listed below.

A. Preemptive vs. non-preemptive disciplines

This characteristic generally determines how new arrivals are processed according to the given discipline. If a low priority unit is being serviced when a higher priority unit arrives (or comes into existence by virtue of a priority change of some unit already in the system) then a preemptive service discipline *immediately* interrupts the server, returns the lower priority unit to the queue (or simply ejects it from the system), and commences service on the higher priority unit. Note that non-preemptive disciplines involve preplanning in some sense. However, the extent of pre-planned "schedules" may vary widely.

B. Resume vs. restart

This characteristic determines how service is to proceed on a previously interrupted (preempted) job when it comes up for service again. With a *resume* priority rule no service is lost due to interruption and with a *restart* rule all service is lost. Assuming that the costs of lost service are intolerable in the applications of concern to us we shall treat only resume rules and systems in which such rules are feasible.

C. Source of priority information

Service disciplines may be classified according to the information on which they base priority decisions. Such a list would be open-ended; however, the sources of the information may be considered to fall in one of three not necessarily disjoint environments: 1) the job environment whereby the information consists of the intrinsic properties of the jobs (e.g., running time, input/output requirements, storage requirements, etc..), 2) the (virtual) computer system environment (e.g., dynamic priorities may be based on the state of the system as embodied in the number of jobs or requests waiting, storage space available, location of jobs waiting, etc.) and 3) the programmers' or users' environment in which management may assign priorities according to urgency, importance of individual programmers, etc.

D. Time at which information becomes known

Classical service disciplines assume that the information on which priority decisions are based is known beforehand. On the other hand, time-sharing disciplines are a prime example of service disciplines in which decisions are based on information (e.g., running time and paging behavior, which is obtained only during the processing of service requests. Such information, of course, is used to establish priorities based on the predicted service requirements of requests which at some time were interrupted and returned to the queue.

All of the priority scheduling methods to be discussed are applicable to the infinite input population case (in which the number of possible customers is unbounded) as well as to the finite population case (in which a finite number of customers use the system) – see Reference 6.

Priorities based only on running times

The intent of systems using a so-called running-time priority discipline is that the shorter jobs should enjoy better service in terms of waiting times. The FCFS (first-come-first-served) system is commonly used as a standard of reference to evaluate the suc-

cess of this intent. The first two algorithms below assume job running times are *known at the time they arrive* at the service point, while most of the remainder, which have arisen primarily in connection with multiprogramming systems, assume that no indications of running times are known until after jobs have been at least partially run.

A. The shortest-job-first (SJF) discipline[8]

This is a non-preemptive priority rule whereby the queue is inspected only after jobs are completely processed (served) at which times that job in the queue requiring the least running time is the next to receive service to completion (and thus there are no cycled arrivals). The SJF descipline has the obvious advantage of simplicity and the somewhat less obvious advantage that the mean customer waiting time in the system is less than in any system (including the FCFS system) not taking advantage of known running times. However, it is clear that significantly long running jobs suffer more in an SJF system than in an FCFS system. Thus, the reduction in the first moment of the waiting time comes at the cost of an increase in the second moment (or variance). We discuss the SJF discipline further in connection with the following discipline.

B. The preemptive shortest-job first (PSJF) discipline

With this discipline the SJF priority rule is applied whenever a job is completed as well as whenever there is a new arrival. If a new arrival has, at its time of arrival, a service requirement less than the *remaining* running time required by the job, if any, in service, then the latter job is cycled back to the single queue and the computer given over to the new arrival. The job returned to the queue is subsequently treated as if its running time were that which remained when it was interrupted; i.e., we have a preemptive, resume discipline.

The PSJF discipline has the advantages over the SJF and FCFS disciplines of further accentuating the favoritism enjoyed by the short running jobs and further reducing the average waiting time in the system. (Again, the variance will be increased.) Indeed, it has been shown that the PSJF discipline is the optimum running-time priority discipline in these last two respects. This relationship between the SJF and PSJF disciplines is seen in part by observing that time-of-arrival receives some consideration in the SJF discipline but, because of the preemption property, none at all in the PSJF discipline.

The principal disadvantage of the PSJF discipline in the computer application is the cost associated with interrupting a job in progress, placing it into auxiliary (queue) storage, and loading the higher priority job for execution. Although this swapping process with auxiliary storage devices may not always be necessary, depending on the size of main storage, it may outweigh the advantage of PSJF scheduling over SJF scheduling. Since it is usually difficult to expect advance knowledge of exact running times, it is encouraging to note that in a thorough study by Miller and Schrage[9] it is shown that even with partial indications of running times significant improvements in mean flow time are possible at the expense of increases in the second moment.

It is obvious that the major effect of these disciplines on the programmer-users of such systems is a salubrious one in that it causes them to produce faster, more efficient jobs. However, the reaction of a user with a job ready to be submitted to an SJF or PSJF system depends to some extent on what information regarding the state of the system is available to the user. If the user can see only the queue length (and this is usually available) then whether or not he balks (refuses to join the queue) depends on how long his job is, his knowledge of the distribution of the running times of jobs submitted to the system, and his assessment of his chances were he to decide to come back later. If indications of the running times of jobs in the queue are known at arrival time then good estimates of waiting time are possible; thus, longer jobs wanting fast service are more likely to balk. With knowledge only of queue length, however, it would appear that less balking would occur with the SJF and PSJF systems than with the FCFS system. On the other hand, reneging (leaving the queue after joining it and before being completely serviced) would be more likely since long jobs are likely to progress rather slowly toward the service point.

The countermeasures available to the users of the SJF system in attempts to defeat (or take advantage of) the computer scheduling algorithm are rather obvious. Firstly, it is clearly advantageous to submit as *short* a job as possible. The natural consequence of this action suggests that a user partition his request into a sequence of short independent requests. Secondly, unless special precautions and penalties are provided, the users may purposefully lie about (i.e., underestimate) their required running time. Such countermeasures lead to a situation in which the attempted discrimination among jobs becomes ineffective and preferred treatment is given to those users displaying the use of clever and/or unethical tactics. We will continue to observe this unfortunate result in the other scheduling algorithms.

So far we have been discussing computer operating disciplines as if there existed but one queue of jobs

waiting in the central processor. This is not generally true if we consider also the queue or queues of jobs waiting for the use of input/output (I/O) devices. The executive or supervisor system itself may be one of the "jobs" in the I/O queues. In a general batch-processing or time-sharing system it is more realistic to assume that jobs consist of phases or tasks whose requirements alternate between the use of the central processor and the use of some I/O device. Then the SJF policies may be defined just for the central processing time (as implied above) or by the sum of central processing and I/O time.

It is not our intention to discuss I/O scheduling disciplines in any detail but it should be kept in mind that a job completion in the central processor system may simply mean that the given job has reached a point where it requires an I/O process before continuing. Similarly, an arrival may mean a job returning from the I/O system for more computing time. This is not to say, however, that I/O and central processing scheduling are independent processes; indeed, it may be necessary that one of the criteria for assigning priority (external or internal to the comput system) be which I/O devices are required and for how much time. This is taken up again below.

C. The round-robin (RR) discipline

This well-known scheduling procedure was first introduced in time-sharing systems as a means for ensuring fast turn-around for short service requests when it is assumed that running times are not known in advance. As seen below the RR discipline falls within a class of so-called quantum-controlled service disciplines in which the size (q) of the quantum or basic time interval is to be considered as a design parameter. In an RR system the service facility (computer) processes each job or service request for a maximum of q seconds; if the job's service is completed during this quantum then it simply "leaves" the system (i.e., the waiting line and the central processor), otherwise the job is cycled back to the end of the single queue to await another quantum of service. New arrivals simply join the end of the queue.

As can be seen, the use of running time as a means of assigning priorities is *implicit* in the RR discipline, whereas it is *explicit* in the previous two disciplines. Running time priorities are assigned *after* a job has been allocated a quantum of service—if the job requires additional service it suffers an immediate drop to the lowest (relative) priority and sent to the end of the queue. Furthermore, it is clear that the RR policy uses both running time *and* time-of-arrival to make (implicit) priority decisions. This latter dependence is seen by noting that all jobs arriving at any time earlier than a given job will have been allocated at least one more quantum of service when the given job reaches the service point.

The extent to which the RR discipline maintains the shortest-job-first policy (in a posterior fashion) depends on the quantum size. Clearly, if q is allowed to be infinite we have a FCFS system. On the other hand as q approaches zero we have in the limit a so-called processor-sharing system[10] in which the part of the processor not devoted to executive or overhead functions is "divided up" equally among the jobs currently in the system. In short, we have a system with no waiting line wherein it is possible to execute all jobs simultaneously but at a rate reduction for an individual job which is proportional to the number in the system. (The computer systems with multiple program counters approximate to some extent the RR behavior with q very close to zero). Despite the better service for short jobs as q decreases, questions of efficiency generally dictate against quantum sizes too small in conventional computer systems. We elaborate on this shortly.

The extent of discrimination by the RR discipline in favor of short jobs also depends on the distribution of job running times. In particular, the RR discipline clearly does not take advantage of any knowledge gained by the quantum-execution of a job beyond the fact that the job simply requires more. For example, the distribution of job running times may be such that any job requiring more than two quanta will with probability .95 require in excess of 10 quanta. In this event the desire to favor shorter jobs would indicate that all jobs having received two quanta should not come to the service point for the third time until all jobs in the system have received at least two quanta. The distribution of job running times that *is* applicable to RR scheduling in this respect is the exponential distribution, which also corresponds to the assumption that has been found analytically tractable in most queueing theoretic studies of the RR discipline. This arises from the so-called memoryless property of the exponential distribution which means in our application that after executing a job for q seconds (with q arbitrary) the distribution of the remaining time to completion is always the same (and equal to the original distribution). Thus, it is the *continued identical uncertainty* in job running times that constitutes the primary job characteristic making the simple RR discipline desirable.

It is immediately evident from the definition of the RR discipline that the basic disadvantage consists of the *swapping* (removing one job from and placing another job into service) necessary for jobs requiring in excess of q seconds of service. Many approaches

to the solution of this problem have been taken. For example, increasing the size of main memory so that many jobs may coexist there eliminates much of the need for swapping. Of course, this is a limited and possibly expensive solution. Also, overlapping the swapping of one job with the execution of another in systems with appropriate memory control and storage capacity makes the swapping process a latent one so that the suspension of the central processor for input/output processes is reduced. Nevertheless, with modern, large-scale multiprogramming systems, the swapping process remains a principal bottleneck to efficient operation with many users.

Several analytical studies of RR disciplines have been carried out[11,12,26] with the goal of determining, for a system defined by a given arrival process and job running time distribution, the interaction between system performance (efficiency, throughput, or waiting times) and the swap-time and quantum size parameters. As verified by experience, analysis has shown how performance deteriorates sharply when the quantum size for a given swap time and system loading is made lower than a certain minimum range of values (or alternatively when loading becomes too heavy for a given quantum size). The priority disciplines described in the next section illustrate techniques whereby this excessive deterioration of service is avoided to some extent.

As regards countermeasures, the mere reduction of one's job length gains little. However, if a user were to partition his job into many smaller jobs, then he would achieve superior performance than a user with an identical job which was left intact. Again, the clever user wins. An interesting property of the $q = 0$ case is worthy of note, namely, that all customers have *identical* ratios of service time to mean time spent in system!

D. The multiple-level feedback (FB) discipline

The FB discipline differs from the RR discipline in a way which is analogous to the way in which the PSJF rule differs from the SJF rule. In an FB system a new arrival preempts (following the quantum, if any, in progress) all jobs in the system and is allowed to operate until it has received at least as many quanta as that job(s) which has received the least number of quanta up to the time of the new arrival. Alternatively, the FB system may be viewed as consisting of multiple queue-levels number 1, 2, 3, ... with new arrivals put in queue-level 1, jobs having received 1 quantum and requiring more in queue-level 2, etc. After each quantum-service the next job to be operated will be the one at the service point of the lowest numbered, non-empty queue-level.

Once again, shorter jobs receive better service at the expense of the longer jobs, and large jobs are not allowed to interfere or delay excessively the execution of small jobs. However, the mean flow time is the same in the RR and FB systems. (In this regard, we note the existence of a conservation law[13] which gives the contraint* under which one may trade the speed of response among a population of users.) The choice between the RR and FB priority disciplines is determined basically by how much one wants to favor short jobs, for the basic algorithms involve the same amount of swapping. It is true, however that the FB discipline is somewhat more costly to implement in the sense that indicators must be used to keep track of the amount of service received by each job.

In the FB system, the users' countermeasure is again to partition his work into many smaller jobs each requiring a small number of quanta (one quantum each, optimally).

Observe that the RR, FB, and FCFS disciplines may be combined in a variety of useful ways. Two combinations that have been used are described below.

E. The two-level FB or limited RR discipline

With this discipline jobs are permitted to "round-robin" only until they have received a fixed number of quanta. At this point they are put into a "background" which is only serviced when there are no other jobs in the system. The background queue may be executed in a FCFS fashion or in a RR fashion with perhaps a larger quantum size. Here, the user countermeasures by forming many jobs from one, each such requiring no more than the fixed number of quanta which prevent his falling into the background queue.

F. The FB discipline with a finite number of levels

In this system a job after receiving a fixed maximum of quanta according to the FB rule (case D) is made to join a background which is to be serviced in one of the ways mentioned above. A further degree of freedom may be added to this or the simple FB rule by removing the constraint that the quanta allocated at different queue-levels be the same.

A question that immediately arises is how one goes about establishing the values of the quanta, the number of levels, or any of the other parameters of these running time priority disciplines. With a specified arrival process and job running time distribution, analysis has been only partly successful in the attempt

*In particular, the sum of the products for each class of users of the utilization factor (mean arrival rate times mean service time) and the mean waiting time remains constant.

to relate the performance measures of interest to the structural parameters of the system.[12] For further results simulation or empirical study[25] will, in many cases, prove the more rewarding approach.

Observe that if q is made zero in the basic FB system we have a processor-sharing type of system in which the jobs sharing the processor at any point in time are those which have received the least amount of service. Thus, a new arrival immediately preempts those jobs sharing the processor and is allowed to operate until its attained service is equal to that of the former or until it completes. It can be seen that the RR and FB processor-sharing disciplines differ in structure in precisely the same way as the SJF and PSJF disciplines. Furthermore, from our earlier remarks it is evident that the former priority rules represent the best that one can do with, respectively, non-preemptive and preemptive priority disciplines designed to favor shorter jobs when running time is not known in advance.

G. Declaration of mode—interactive or batch

The time-sharing system at the University of California at Santa Barbara[14] uses an interesting variation of the above methods. A user is required to state whether his job is interactive (short, frequent requests) or batch (longer, usually single requests). Time is divided into fixed length segments such that during the first half of each segment, the interactive jobs are served in a round robin fashion until their half-segment is exhausted (or until their collective requests are satisfied). The remainder of the segment is then used to service (to completion, if possible) as many of the batch jobs as possible. During the first half-segment, some interactive jobs will drop from the queue after one quantum of service (e.g., those requiring the acceptance of a single button-push), etc; thus in this system, there is a benefit in declaring the nature of your job in an honest way, since in the case where there is only one batch job and many interactive jobs, the batch job receives better service if that user declares himself as being in the batch mode.

The discussion of balking and reneging when running time is assumed unknown applies without change to the RR and FB disciplines analogous to the SJF and PSJF disciplines, respectively. Another effect that may need to be taken into account in the design of a time-sharing discipline is that of "jockeying" among queues of users awaiting service at a console. (This complication of balking and reneging has received little attention in the literature.) As before, users are encouraged to produce fast running jobs for the RR and FB disciplines. In time-sharing applications this might be better stated by saying that users are encouraged to produce "frequently interacting" jobs. In a given multiple-level RR system, for example, long jobs may avoid being put into a background by communicating (artificially, if necessary) with the on-line user at a frequency such that its running time never exceeds the foreground quantum during any operation interval. In effect, the job is alternating between the foreground and input/output queues in a way that provides better service than if it were put into the background. Obviously, this is an example where the more clever users of a system tend to defeat the purpose of the service discipline.

As a final remark it should be noted that the existence of saturation in a system with any of the disciplines we have discussed, except for the RR discipline, depends on the class of jobs being considered. For example, in an SJF, PSJF, or FB system it is clearly possible that loading be such that jobs with less than say five minutes running time will have a finite expected wait while those with greater than five minutes running time will have an infinite expected wait (in the infinite population case). Of course, the system *as a whole* is saturated if a finite threshold of this nature exists, since the processor is not able to complete all jobs submitted to it. For the RR discipline (as with the FCFS system) there exists a single saturation point which, when reached, causes all jobs to have an infinite expected wait. (This stems from the continued interference of long jobs with the execution of short jobs.)

State-dependent running time priority disciplines

The principal motivation behind state-dependent disciplines is the desire to reduce the overhead and swapping costs in the execution of quantum-controlled service. It is of particular interest to prevent or minimize the collapse of RR system performance under heavy loading; i.e., to provide a more graceful deterioration of service with increases in loading.

A. Cycle-oriented RR disciplines

A basic design parameter of such systems is the so-called cycle or response time which is set and used to control as desired the maximum amount of time required to execute one round-robin through all active jobs. In one variation,[12] after completing a given cycle or round-robin, the subsequent round-robin quantum is determined by dividing the fixed cycle time parameter by the number of jobs requiring service; the time represented by the result of this division is then allocated to each of the jobs requiring service at the beginning of the cycle. Subsequent cycles are then determined in the same fashion. Usu-

ally, some minimum allocation time (quantum) is always given because of the otherwise seriously degrading effects of swapping during heavy loading. Although the cycle time therefore increases at excessive loading, it is clear that system performance degrades more gracefully than otherwise in that the loss of swapping is reduced as the load increases. The present discipline is state-dependent in the sense that the amount of time received by a given job in a given cycle depends on the number of active jobs in the system at the beginning of the cycle.

In another (two-level) RR variation[26] a maximum cycle time is similarly imposed on the amount of time taken to process a "foreground" queue (i.e., a queue of interactive, on-line user jobs) and a background queue consisting of conventional production type jobs. In this time interval each foreground job is given a fixed quantum; if there is any time remaining in the cycle it is devoted to the background job processing, otherwise another cycle is initiated. To limit swapping overhead (and thereby provide graceful degradation during periods of heavy loading) the cycle is extended in the event there are too many foreground jobs to process for one quantum in one cycle.

It is clear that the advantage of a reduced variance in RR response times is compensated in cycle-oriented disciplines by the slower reaction to changes in loading or input activity (we have implied that the queue is examined only at the end of the round-robin cycles). Statistically, however, this disadvantage would seem to be a minor one. Of course, it is also possible to make the value of the cycle time parameter dependent on system loading (number in the system). Just how this is to be done in a useful way, however, presents a difficult problem. In these disciplines, the countermeasure of partitioning a long job into many small jobs is extremely effective.

B. Input-dependent disciplines

In one such (RR) discipline each time a new arrival occurs the job, if any, in service is allocated an additional quantum of execution time.[15] In this way the RR discipline reacts to heavy input activity by increasing time allocation and thereby reducing the amount of overhead and swapping. During light to moderate input activity the straight RR discipline is little effected by this change.

Another such discipline orders the queue of interactive user's jobs by interarrival time. Thus, those users communicating with the system at the faster rates will receive the shorter response time. The obvious countermeasure here is to initiate false I/O commands. This technique, of course, may be combined in various ways with both RR and FB disciplines.

C. Priorities based on storage allocation

Apart from the number in the system the most important other source of internal priority information is the current allocation of storage and the availability of I/O devices to perform storage allocation functions. This, of course is tied in with the swapping problem discussed earlier. In batch processing systems requiring maximum efficiency this information may serve as the only criterion for assigning priorities; at a decision point with this type of discipline a schedule of job operations is computed as far as necessary in advance such that storage is well utilized in some reasonable sense.

This sort of scheduling is also applicable to the cycle-oriented RR disciplines described earlier. Thus a job is executed once per cycle, but where it executes in the cycle is made dependent on what turns out to be the best way (or at least a good way) to sequence the use of main storage so that I/O is minimized and overlapped as much as possible with computation. Clearly, the cost that must be compensated in this operation is the (potentially substantial) overhead time required to produce "good" schedules.

In the latest generation of multiprogramming systems, storage structure and the processes of storage allocation have been made more elegant by the concepts of virtual memory and paging.[16] In paging systems jobs (programs and data) are paginated into sets of fixed length pages or blocks of computer words so that the logical unit of information transfer within the supervisory system becomes a page. This also means that jobs may be operated (at least in part) when only a proper subset of the job's pages are in main storage. The synthesis and analysis of efficient storage dependent, running time priority disciplines that take advantage of this added flexibility is a difficult, important, and as yet unsolved problem.

Inclusion of externally generated priorities

The classical priority disciplines in which priorities are determined external to the computer system are described as follows. At any point in time with preemptive rules or just after service completions with non-preeemptive rules the job next to be serviced is the one with the highest priority (i.e., the job having been assigned the lowest priority number). That discipline receiving the most attention in the literature is one for which the number of priorities is finite or countably infinite; i.e., in one-to-one correspondence with the integers.[17] This gives rise to levels of queues containing jobs of the same priority; these queues are generally ordered by time of arrival to the system. The advantages and disadvantages of preemptive vs. non-preemptive priority rules are the

same as those discussed for the PSJF and SJF rules in an earlier section.

There has been a variety of ways by which externally generated priorities have been included in the running time priority disciplines described in previous sections. These are classified as follows.

A. RR disciplines with external priorities

A technique used with RR disciplines consists of making the quantum size to be allocated dependent on the priorities of the active jobs.[10] Thus, a higher priority job would be allocated a larger quantum than a lower priority job. In the limit where the quantum size is zero we have a processor-sharing system in which the fraction of the processing rate received by a job is determined by an externally assigned priority.

Another technique which smacks of the multiple level schemes given below involves the specification of (relative) time delays as priorities.[28] Consider the following implementation, for example: Each arriving job is assigned a (priority) number which is based on the given job's externally assigned priority *and* on the number currently possessed by the other active jobs in the system. After a given quantum-service (which may consist of multiple quanta) the next job to receive service is the one having the lowest priority number. The jobs having the same priority number are ordered by time of arrival and serviced in that sequence. Each time a job is serviced for one quantum its priority number is increased by one. Thus, with a non-preemptive system we see that the number assigned to an arriving job indicates how much time it is to be allocated for operation before it joins the round-robin of jobs already in the system; this in turn is determined by an external priority assignment.

B. Multiple level disciplines with externally assigned priorities

The simplest and most natural method of including external priorities applies to the multiple level disciplines in which each level is used to correspond to an external priority number as well as to a given level of attained service.[12] In this way conventional priority scheduling is combined directly with the various FB disciplines described earlier. However, variations in these types of disciplines may be obtained by the selection made for the means of ordering the queue-levels. Specifically, the queues may be ordered by time of arrival to the system, by time of arrival to the queue, or by a combination of one of these with ordering by priority; i.e., by the queue level of original entry to the system. (Note that ordering by time of arrival to the queue and by time of arrival to the system amounts to the same thing in the basic FB discipline without external priorities.)

One time-sharing scheduling discipline of the above type that has received considerable attention is one in which priorities are assigned at arrival time according to job size (storage requirements), queues are ordered by time of arrival to the queue, and quantum sizes are exponentially increasing with the queue level (i.e., level one provides on quantum, level two provides two quanta, level three provides four quanta, and so on.[2] Priorities based on storage requirements are assigned so that the larger jobs receive the lower priorities (enter at the higher queue levels). In this fashion efficiency is kept high by reduction in swapping time, since large jobs are given more time to operate between swaps. Furthermore, the large jobs are not allowed to interfere with the small, presumably faster, and more efficiently scheduled jobs. Clearly, users of such systems are further encouraged to write small jobs, again possibly by breaking larger jobs up into autonomous and small sub-jobs.

For a given application the design of the general disciplines just discussed requires a means for evaluation of the best values for the quantum distribution, the specification of the storage-dependent priority assignment rule, and the selection of the best method for ordering queue levels. Here again is a synthesis problem similar to the one mentioned earlier. At present no generally applicable, well-defined procedure exists; experiment by simulation and empirical study has been used thus far. Some encouraging work towards the optimal synthesis of such systems has been reported by Fife.[18]

In all of the externally assigned priority methods, the user who can influence the assignment of external priorities has a great advantage over the others. This is taken up next.

C. User controlled priority assignment

According to this type of discipline the user is allowed in some way to bid for, or simply buy the priority he desires (or can afford) for his job. One such discipline is the so-called bribing model[7] in which system users offer a bribe (based on an "impatience" factor of their own) to obtain a preferred position in the queue of waiting jobs. All those bribing strategies are then considered which minimize an appropriately defined cost function over the set of users.

Another (quantum-controlled) system[19] of this sort has been used which provides a quantum size that is proportional to the priority a user decides to assign his job, and which increases with the size of his job (in order to maintain a reasonable operating

efficiency). What constrains the priority a user assigns to his job is the fact that he is charged a fee for use of the system which is proportional to the product of the priority he has selected times the sum of the computing and I/O time required by the job. In this particular case, as well as in the bribing case, if a user is capable of learning what all the other users have assigned as their priorities (or bribes) then he need merely "go one better" and choose a slightly higher priority (or bribe) to achieve superior service. Note when the system is heavily loaded that all users see a "slow" system and so they tend to increase their self-assigned priorities (or bribes) in an iterative fashion; the result is an ever increasing cost to the user for a constantly decreasing grade of service! Clearly, the user population as a whole should in such case, act in collusion to prevent such runaway conditions.

Dynamic or time-dependent priorities

There have been a number of priority disciplines proposed in which jobs receive an external priority that changes in a dynamic way once the job is in the system. These disciplines were motivated by other applications but it will be clear that they may be considered candidates for scheduling computer operations.

So far we have treated disciplines in which the waiting time experienced by a job is not used to directly influence the priority decision. The disciplines below are structured so that this information, weighted by an external priority number is used in the process of selecting from the queue which job is to be serviced next.

A. Delay-dependent disciplines

In the first variation to be discussed[20] a job's priority is increased, from zero at the time of arrival, linearly with time in proportion to a rate (externally) assigned to the job's priority class. Each time a new job is to be selected for service according to the non-preemptive or preemptive variations of this scheme the "attained" priorities of jobs in the system are compared, and that job with the highest attained priority is selected next for service. If the priority classes are assigned according to a shortest-job-first policy it can be seen that this rule moderates the SJF and PSJF rules by reducing the probability of excessively long waits.

In another variation[21] jobs are similarly assigned external priorities related to the urgency of service. However, with this discipline a given job takes precedence over another job in the queue if, and only if, the difference between the former's (external) priority and that of the latter is not less than the time the latter has spent waiting. This scheme may be implemented as a preemptive as well as a non-preeemptive discipline. Again, this is an example in which management becomes concerned about a job that has waited for a long time.

B. Priorities based on general cost accrual[5]

In the most general such system arriving jobs have associated with them a cost accrual rate which is some arbitrary function of time. In a preemptive or non-preemptive mode the discipline is executed by servicing (at each decision point) that job which minimizes over all jobs the cost accrued by the subsequent waiting time. The cost accrual attributable to a given job over a given time interval is calculated simply by integrating the cost rate curve over the given time interval. It is of interest also that one may introduce deadlines by making the slope of the cost rate function infinite after a suitable interval of time.

The simplest case of the general discipline exists when the cost functions are constant and identical for each user. It can be seen that this corresponds to a system in which cost accrues linearly with time and where the priority minimizes the average wait (i.e., we have the PSJF or SJF rules depending on whether or not we have preemption). If we assume constant valued functions that may differ for each job we have the so-called c/t rule. This rule amounts to selecting for service that job whose ratio of constant cost rate (c) to known or average service time (t) is the smallest.

In these systems, it is not usually to one's advantage to partition jobs into smaller ones in an attempt to defeat the system. In fact, it may pay to group many jobs together so that they all enjoy an early arrival time to the system.

Priority disciplines in multiprocessor systems

In providing the additional degree of freedom of the number (and perhaps types) of processors many different possibilities come into existence, most of which have not been fully tested or analyzed as yet. In this section we shall examine briefly the application of previously discussed disciplines to multiprocessor systems and certain disciplines for which analyses exist in the literature but which arose out of other applications.

A. Processors-in-series systems

Multiple channel (server) queueing systems are broadly classified in the literature according to whether the servers are being used in parallel or in

phase type service. By phase type service we mean that the processors are being used in sequence:[22] one phase of a given job are being serviced on a given processor. In general, each processor will have a queue of jobs (phases) which is fed by the output of some other processor or by an external source. Such a system applies, for example, to the alternating I/O and computing job structure described in another section. Here, we assume that at least one central processor and at least one I/O processor is being used in a cyclic way by a given job. Other applications of this discipline)which may be combined with other disciplines at each of the separate queues) are to be found in computer systems dedicated to the processing of certain, large phase-structure jobs.

B. Processors used in parallel

The simplest extension of the previous disciplines to multiple processors is simply to treat the processors in a first-available-first-used fashion. [8] In large, general-purpose systems in which the processors are identical this single queue approach offers the advantages of simplicity, flexibility, and efficiency. The disciplines may be made more effective in those systems where jobs are designed to operate on one or more processors depending dynamically on availability.[23]

However, several other variations of "parallel" priority disciplines exist when the set of processors or the input jobs are not homogeneous. The simplest such case arises when jobs fall into one of two categories; a foreground of fast service jobs and a background of perhaps less important production type jobs. These conditions are appropriate to the so-called variable-channel discipline in which the number of processors made available to the foreground jobs is made dependent on (increases with) the number of foreground jobs in the system. The number of foreground processors increases only after a maximum queue length has been reached; i.e., with each newly arriving job after the fixed maximum has been reached a new processor is made available (if possible) to serve the job at the head of the queue.

Two other possibilities are 1) the existence of special purpose processors, and 2) processors of different computing speeds that may be allocated by external priority or by job requirements. In this regard, we may ask that a user estimate his job length or type and then assign him to a processor which has been optimized for such jobs. If, after some processing, it is found that a user has made a poor estimate, he is penalized in some way (e.g., by being forced to move to the end of a queue on another processor). His penalty for overestimating his work is to be placed on a processor which is not "tuned" to jobs of this type.[6]

The coming importance of networks of computers[29] creates another source of applications for the above types of multiple-queue disciplines. Computer network disciplines will also have to be dependent on transmission delays of service requests and jobs or parts of jobs from one computer to another as well as on the possible incompatibilities of various types between different computers. The synthesis and analysis of multiprocessor and multiple processor network priority disciplines remains a fertile area of research whose development awaits broader multiprocessor application and an enlightening experience with the characteristics of these applications.

CONCLUSION

We have listed above a variety of possible computer scheduling methods suitable for many situations. This list is by no means complete. In fact, this wealth of possible algorithms produces an "embarrassment of riches" in that we do not really know how to select the most useful scheduling methods. As we have indicated, the possibilities are considerable.

We have also attempted to discuss briefly the possible counter-measures available to a user of the computer which would allow him to defeat or take advantage of the system for the various algorithms described. Indeed we have shown that in most cases, there is such a countermeasure! One hopes that there exists an efficient scheduling method which is immune to such manipulations.

REFERENCES

1 J I SCHWARTZ E G COFFMAN C WEISSMAN
A general purpose time-sharing system
Proc SJCC 1964

2 F T CORBATO M MERWYN-DAGGETT R C DALEY
An experimental time-sharing system
Proc SJCC 1962

3 G BELL M W PIRTLE
Time-sharing bibliography
Proc IEEE December 1966

4 E G COFFMAN
Studying multiprogramming systems through the use of queueing theory
Datamation July 1967

5 M GREENBERGER
The priority problem
MIT Project Rep MAC-TR-22 November 1965

6 G ESTRIN L KLEINROCK
Measures models and measurements for time-shared computer utilities
Proc ACM Natl Conf August 1967

7 L KLEINROCK
Optimum bribing for queue position
Journal of operations research (to appear)
8 T E PHIPPS JR
Machine repair as a priority waiting-line problem
Operations Research vol 4 1956
9 L W MILLER L E SCHRAGE
The queue M/G/1 with the shortest remaining processing time discipline
Rand Corp Report P 3263 November 1965
See also
L E SCHRAGE
Some queueing models for a time-shared facility
PhD Dissertation Dept of Indust Engineering Cornell Univ February 1966
10 L KLEINROCK
Time-shared systems: A theoretical treatment
Journal of the ACM April 1967
11 L KLEINROCK
Analysis of a time-shared processor
Naval Res and Log Quart March 1964
12 E G COFFMAN
Stochastic models for multiple and time-shared computer systems
PhD Dissertation Dept of Engineering UCLA June 1966
13 L KLEINROCK
A conservation law for a wide class of queueing disciplines
Naval Res and Log Quart June 1965
14 G CULLER
Univ of Calif at Santa Barbara (Private Communication)
15 E G COFFMAN
Analysis of two time-sharing algorithms designed for limited swapping
Journal of the ACM (to appear)
16 J B DENNIS
Segmentation and the design of multiprogrammed computer systems
Journal of the ACM October 1965
17 A COBHAM
Priority assignment in waiting line problems
Operations Research Feb 1954
18 D W FIFE
An optimization model for time-sharing
Proc SJCC 1966
19 G SUTHERLAND
Paper presented at the symposium: *Computers and communication: Their system interaction*
Sponsored by the IEEE groups on Communication Technology and Electronic Computers Santa Monica Calif January 1967
20 L KLEINROCK A FINKELSTEIN
Time-dependent priority queues
Journal of ORSA (to appear)
21 J R JACKSON
Waiting time distribution for queues with dynamic priorities
Naval Res and Log Quart March 1962
22 L KLEINROCK
Sequential processing machines (SPM) analyzed with a queueing theory model
Journal of the ACM April 1966
23 D F MARTIN
The automatic assignment and sequencing of computations on parallel processor systems
PhD Dissertation Dept of Engineering UCLA January 1966
24 A L SCHERR
An analysis of time-shared computer systems
PhD Dissertation Dept of Electrical Engineering MIT June 1965
(See also the bibliography in reference 6)
25 B KRISHNAMOORTHI R C WOOD
Time-shared computer operations with both interarrival and service times exponential
Journal of the ACM July 1966
26 *Time-sharing system/360 development workbook*
IBM Internal Document
27 E T IRONS
A rapid turn-around multi-programming system
Comm of the ACM March 1965
28 E G COFFMAN
Bounds on the parallel processing of bulk queues
Naval Res and Log Quart September 1967
29 T MARILL L G ROBERTS
Toward a cooperative network of time-shared computers
Proc FJCC 1966

Some ways of providing communication facilities for time-shared computing

by HOWARD L. STEADMAN and GEORGE R. SUGAR
ESSA Research Laboratories
Boulder, Colorado

INTRODUCTION

Since July 1965 we have been using time-shared computing services as a computing aid for technical programs at the Boulder laboratories of the National Bureau of Standards and the Environmental Science Services Administration (ESSA). We proceeded slowly through the steps of first studying time-shared computing, visiting various installations, installing a terminal and using an outside service, experimenting with a variety of services, acquiring several more terminals and providing regular service, and finally acquiring our own time-shared computer. At first service was obtained from the General Electric Company in Phoenix, then in addition from System Development Corporation in Santa Monica, from IBM in Los Angeles, from Tymshare in Palo Alto, and from Com-Share in Ann Arbor. Service is now being provided by an SDS-940 time-sharing system operated by ESSA in Boulder.

Throughout the build-up period there has been a steady increase in our needs for Teletype terminals, communications lines, switching facilities, data sets, etc. We have obtained most of these from the local Telephone Company (Mountain States Telephone and Telegraph Company). However, because we have often been the first customer in our area to ask for a particular service or feature, we have in effect educated the telephone company in the communications requirements for time-shared computing. This was a slow process and in some cases the services obtained were not as good as we had expected.

The purpose of this paper is to relate some of the problems we encountered in providing communication for time-shared computing, the solutions for some of these problems, and some possible approaches to solving the others. While many organizations using a time-sharing computer have had some experience dealing with computer manufacturers, their experience with communications problems had often been limited to obtaining conventional telephone service for their staff. We hope that by relating some of our problems and solutions we will help other time-sharing users solve their own communications problems. The body of this paper is organized into three major sections. The first discusses the Teletype terminals themselves and some of the options that we found useful. The second part discusses communicating with a computer in a distant city and describes our initial facilities for the SDS-940. The third part discusses a number of alternate ways of providing the required communication facilities.

Teletype terminals

While most telephone companies are well equipped to handle the installation of telephones, even colored Princess or Trimline instruments, such items as Teletypes and data sets are sometimes alien to their experience. The special terminal options which many time-sharing systems require create additional problems. These are not limited to technical details such as full-duplex and half-duplex terminals, upright and inverted operation, TWX versus TWX' service, etc., but extend to simple mechanical problems such as making a terminal "portable." For example, after much discussion the Telephone Company decided that it was indeed possible to furnish wheels and a telephone plug on a Model 35 Teletype for $7.50 per month extra. However this was subject to the restriction that the terminal not be moved between floors in our building since the wheels might catch in the crack between the elevator and its shaft and tip the terminal over. We have long tried to persuade the Company to furnish a portable Model 33 Teletype and have recently succeeded. They had considered this smaller machine to be less stable than the larger Model 35 and therefore more dangerous to put on wheels. (We had heard that some telephone companies would furnish a portable Model 33 terminal and we

have made our own Model 33 portable at a cost of $20 including parts and labor.) In fairness to the Telephone Company we must say that they have tried to meet all of our needs. The principal problem seems to be that anything that has not been done before locally may require a very long time to accomplish.

Our first terminal was a Model 33ASR Teletype connected for half-duplex Data-Phone service ("upright"). As a next step we wanted a terminal to communicate with the SDC time-sharing system. At that time it too was equipped for Data-Phone service but for "inverted" mode in contrast to the upright mode that our terminal used. The difference between these two modes lies in which of two carrier frequencies is used by the originating terminal for sending and which is used for receiving. Since normal Teletypes are equipped both to originate and to answer calls it is necessary that the Teletype be able to use either carrier for either sending or receiving. However the standard Teletype does not have the flexibility required to operate in both upright and inverted modes. The solution proposed by the Telephone Company was to install another terminal for inverted service and this was done. It was not until some time later that we persuaded the Company to equip all of our terminals with a simple switch that allowed us to operate them in upright or inverted mode. (It now appears that upright mode has been adopted as standard for time-shared computer service and the special switch is no longer needed.)

The next special feature we required was full-duplex operation to permit a lecture and demonstration of the Berkeley time-sharing system. It was not until late on the day before the lecture that the terminal was installed and it was not clear whether it worked or not. The installer had not worked with a full-duplex terminal before and his uncertainty about the operating condition of the terminal was not the least reassuring to us. After the speaker arrived, a quick test online to the computer showed that indeed the terminal was the proper one. (We later learned that the only internal difference between a full-duplex Model 35 Teletype and a half-duplex one was the location of one wire on the terminal board of the data set. When later it became clear that we would require full-duplex terminals to use the Tymshare system in Palo Alto we ordered full/half-duplex switches for all of our terminals. This option, much to our surprise, became available at no extra cost.)

Other features that we have obtained include forms control and sprocket-feed platens on Model 35 Teletypes, slashed zeros (Ø), escape keys, and remote reader-start and TD call-in on Model 33 Teletypes. (This last option allows the computer to start and stop the paper tape reader on the Teletype. It is standard on Model 35 but a special option on Model 33 terminals.) We have also obtained multi-contact distributors. These are devices that provide an interface to other apparatus such that signals originating in the other apparatus are transmitted as if they were sent from the Teletype keyboard itself.

As can be seen from the above, we have obtained a large variety of special features on our terminals. Table I lists the options which we have been able to obtain along with the charges made by the Telephone Company. Others in our area now have no difficulty in getting these same features. However, new features may still be hard to get. We can offer two hints to those who want special features that are new to their telephone company. First, ask for only one new feature at a time. Second, find another telephone company that has installed the feature you want and inform your local company. If you can do at least one of these two things and then wait a while, you can probably get any feature that is technically feasible.

TABLE I

Availability and Monthly Rental Costs for Various Teletypewriter Options

	Model Number			
	33ASR	33KSR	35ASR	35KSR
Base cost	$42 + $50	$33 + $50	$110 + $50	$70 + $50
Data set (always required)	$25 + $25	$25 + $25	$25 + $25	$25 + $25
Paper tape reader and punch	S	X	S	X
Full/half duplex switch	$.50	$.50	$.50	$.50
Upright/inverted switch	$2	$2	$2	$2
Escape key	NC	NC	NC	NC
Slashed zero (Ø)	NC	NC	NC	NC
Even parity keyboard	NC	NC	NC	NC
TD call-in and remote reader start	$4	X	$4	X
Portable machine (wheels and telephone plug)	$7	$7	$7	X
Parallel-input interface from external apparatus (Multi-contact distributor)	$14 + $35	$14 + $35	$14 + $35	$14 + $35
Parallel-output interface to external apparatus	$5 + $15	$5 + $15	$5 + $15	$5 + $15
Sprocket feed platen and forms control	$5 + $25	$5 + $25	$5 + $25	$5 + $25

Notes: X means not available
NC means no charge
S means standard feature

When two costs are given, the first is the monthly rent and the second is the installation charge.

Communications

Our first terminal was connected as an extension on our internal telephone system. We reached the computer in Phoenix by first dialing an outside line and then dialing the computer over the regular toll telephone network. The communication cost for these calls to Phoenix was about $25 per hour. We could have used TWX service instead of Data-Phone service but the cost of calls to Phoenix would have been slightly higher. However, for other distances TWX may be less expensive. As usage built up we found it advantageous to get a WATS line that provided unlimited service to Phoenix for $1,000 per month.

As use of time-sharing increased, we added other terminals and at first connected them to the one

WATS line. This arrangement turned out to be singularly unsatisfactory since, for at least one combination of terminals, a new user could come on the line and disconnect another user. It was never clear whether this was done accidentally. While we were puzzling over how to provide better communications at lower cost, we learned of so-called FX or foreign exchange lines and it was by using these that we were able to effect a major reduction in communication costs. Foreign exchange lines provide direct connections to a telephone exchange in a distant city. In our case, for example, an FX line to Phoenix provided us with the same telephone connection we would have had if we had been in Phoenix. This meant that a call to Phoenix was a "local" call for us insofar as the mechanics of placing the call was concerned. This gave us, in effect, a sort of private line to the Phoenix computer without requiring any special arrangements at the computer. Now this would be of little advantage relative to WATS service except for one thing. Since the Federal Government is a large user of TELPAK service we were able to get our FX lines at essentially TELPAK D rates which are about 45¢ per mile per month for each voice-grade circuit. A small private user would have to pay the private line rate of about $2.00 per mile per month for this same service. The low rate for FX lines put the break-even point for FX *versus* regular toll rates at about 20 hours use per month. At one point our total monthly use reached about 700 terminal hours and we had 2 FX lines to Phoenix and 8 FX lines to Palo Alto (for the Tymshare computer) to handle this load.

During the period when we relied completely on outside time-sharing service, all of our terminals (about 30) were connected to our internal telephone exchange and the FX lines were terminated at the switchboard. Thus any terminal could, by dialing our own telephone operators, connect with any of the services which we used. This arrangement also provided control over the use of the FX lines for unauthorized calls. Further, if the FX lines were all busy, our operator could place regular toll calls for the user. Obtaining time-sharing service became a matter of dialing "operator" and stating which computer service was desired.

In installing our own system, an SDS-940, we sought to provide an interim communication system which would allow access to either our system or the commercial services which we were using. Our purpose was to avoid any sharp cut-over from the outside services to our own system just in case the new machine didn't work right immediately. The interim system described below was chosen primarily because it was the cheapest one which provided the needed facilities.

Figure 1 shows the current communication arrangement for our time-sharing system. Lines are provided for internal and outside users. All equipment is provided by the Telephone Company. The total cost for the arrangement in Figure 1 is $6,361 per month.

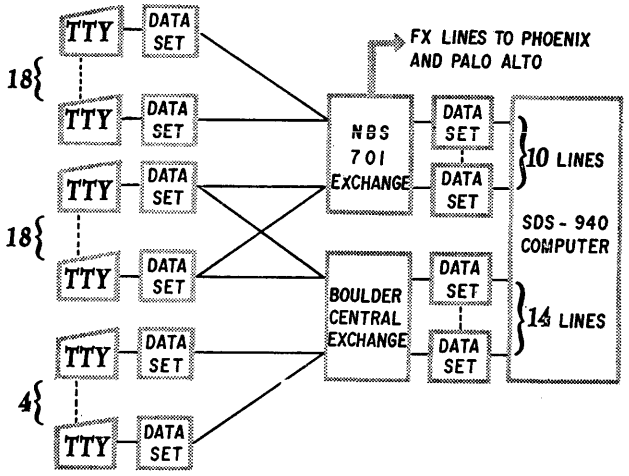


Figure 1 – Present system

This is broken down in Table II. Although this system works moderately well, it has several significant drawbacks. The first is that it is somewhat expensive and the telephone company expects to increase the rates on some of the items. Second, it places a considerable load on the in-house exchange which at the same time must absorb increases in normal telephone use. Third, it lacks flexibility with regard to connection to other communication networks such as ARS or Western Union Telex service.

TABLE II

Monthly Cost For Present System

(See Figure 1)

Equipment	No.	Unit Cost	Total Equipment Cost
Teletypes	40	$100[1]	$4,000
Data sets	64	25	1,600
Central exchange lines	36	17.50	630
701 exchange lines	46	2.85	131
		Total	$6,361

[1] This is an average price for a mixture of Model 33's and Model 35's with a variety of options.

Some other alternatives for obtaining facilities

For communication with our DSD-940 we require: (1) terminals, (2) transmission facilities, and (3) a "line-concentrator." The need for the first two is clear. The need for the third is also clear when we consider that there are more Teletype terminals than entry ports to the computer. For each of the three facilities there are several choices of equipment and often several choices of suppliers or methods of supply for each piece of equipment. We first consider facilities supplied completely by the Telephone Company and then consider composite facilities supplied from any combination of sources. This separation is a logical one since the Company insists on supplying either complete systems or just leased private lines.

Complete systems available from the telephone company

The Telephone Company has proposed the system shown in Figure 2. The costs for this system are shown in Table III. Clearly this system offers little savings over our current system and in fact, on the basis of the cost and trouble for converting, is not a desirable change. The actual cost will probably be greater than shown in the table. This is a result of the Telephone Company policy of requiring 5-year leases on some switching equipment. The system proposed has a $250-per-month termination penalty for the unexpired part of a 5-year lease. In view of the rapid developments in time-shared computing it is unlikely that any system designed now will meet our needs for the next 5 years. We would therefore incur a considerable termination penalty and the actual cost of the system in Figure 2 will probably be higher than that for the current system.

TABLE III

Monthly Costs For Telephone Company Isolated Switching System

(See Figure 2)

Equipment	No.	Unit Cost	Total Equipment Cost
Teletypes	40	$100[1]	$4,000
Data sets	64	25	1,600
Lines:			
Radio Bldg.	28	1.50	42
Main Campus	10	4	40
30th St. Bldgs.	2	6	12
Computer Ports[2]	24	15	360
		Total	$6,054

[1] This is an average price for a mixture of Model 33's and Model 35's with a variety of options.

[2] The cost of the concentrator is included in this item. There is a termination charge of $250/mo. for the unexpired part of a five year lease.

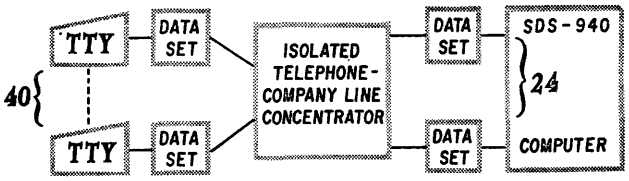


Figure 2 – Telephone Company isolated switching system

Another proposed arrangement would use only our internal telephone system (701). In studying Table II one quickly notices that lines connected through the central exchange are far more expensive than lines connected through the 701. We have the present split arrangement, because the 701 does not have enough capacity to handle the whole time-sharing load. Naturally one question is whether the capability of the 701 can be increased and the cost of doing it. This can be done and Figure 3 shows this arrangement. The costs are shown in Table IV. Although this system does not offer significant savings it is clearly preferable to the system of Figure 2. There is still a termination penalty for the extra switching equipment required. However, it is probable that this equipment would simply be kept and used to meet increased demands for normal telephone service and no penalty would be incurred.

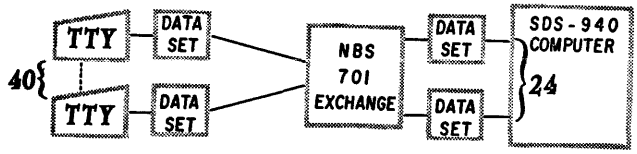


Figure 3 – Expanded 701 system

Composite systems

Terminals and data sets – Terminals could in theory be any one of several devices. However from the viewpoint of cost, availability and compatibility with the current system, Model 33 or 35 Teltypes are the best choice. These can be either leased or purchased. In principle terminals can be leased from the Tele-

phone Company or Western Union. However the Telephone Company will lease Teletype units only in conjunction with other equipment and Western Union will lease only Model 35 Teletypes. If terminals are purchased, we would favor the Model 33 over the Model 35 because of a 1:4 cost ratio. If terminals are purchased, we must also arrange for their maintenance. At present we know of four possibilities; maintenance by ESSA, GSA, Western Union, and RCA Service Company. Initial estimates are that maintenance would cost about $25 per month per terminal.

TABLE IV

Monthly Cost For Expanded 701 System

(See Figure 3)

Equipment	No.	Unit Cost	Total Equipment Cost
Teletypes	40	$100[1]	$4,000
Data sets	64	25	1,600
701 exchange lines	64	2.85	182
Increase 701 Capacity[2]	35	3.95	138
		Total	$5,920

[1]This is an average price for a mixture of Model 33's and Model 35's with a variety of options.

[2]There is a termination charge of 1/2 the cost for the remaining part of a 5 year lease.

We do not yet have final estimates of maintenance costs or of life expectancy for the terminals. Based on information obtained so far and assuming a useful life of 2 years for Model 33 equipment, a purchased Model 33 KSR Teletype would cost approximately $20 per month plus maintenance or $45 per month. The Telephone Company lease cost is $42 per month. Western Union has not yet filed a tariff on Model 33 Teletypes. They have offered to lease Model 35 Teletypes to us at $70 to $95 per month including maintenance.

Any of the terminals discussed above will require some form of data set or modem. In examining the various arrangements available from the Telephone Company we find that they all include a substantial number of data sets which must be leased at $25 per month each. In many cases there is no technical need for an elaborate data set and this is therefore an area where substantial saving can be effected. For example, when private lines are used, suitable modems can be purchased for less than $100 and Western Union leases modems for $13.75 per month. The Telephone Company charges $25 per month for a modem to connect Model 33 and 35 Teletypes to private lines. This makes the rental of Teletypes from them more costly than any of the other alternatives.

Another type of modem that is readily available is the acoustic or magnetic telephone coupler. These are available from a number of sources at purchase prices as low as $250. We are already using a number of them and find that they are adequate for most of our needs. The use of these telephone couplers permits most telephones to be used as entry points to the computer.

Transmission facilities—Off our main campus and perhaps outside of the building where the SDS-940 is installed there is no choice of lines other than from the Telephone Company. There are two types of lines available—type 1000 for DC and low frequency signalling and type 3000 for voice frequency signalling. Either type has an approximate monthly lease cost of $1.50 per half-duplex circuit within the same building and $1.00 per ¼ mile per half-duplex circuit outside the building, with a $4.00 minimum per line. Such leased lines are the only answer for off-campus buildings. For terminals on the main campus we have considered the installation of our own wiring, however, we have not yet obtained any cost estimates for doing this. An additional possibility for remote locations where we expect to have a significant number of terminals is to place a line concentrator there to reduce the number of lines to the computer. Lines within the main building could then be provided either by us or the Telephone Company.

A problem in using type 3000 lines is that they transmit a.c. only (300-3000 cps); therefore tone signalling must be used. Fortunately, as mentioned earlier, suitable modems are readily available at low cost. A problem in using type 1000 lines is that either the SDS-940 system must be modified to accommodate them or a special interface must be provided external to the computer. Therefore there is no opportunity to effect a major saving by using type 1000 lines in place of type 3000 lines.

Switching facilities—The switching facility need not be a general exchange. In particular only one-way switching is required. Also no choice needs to be exercised by the originating terminals. Therefore a simple line concentrator is all that is required. (The Rand Corporation has taken this approach for their JOSS time-sharing system.) A suitable system can be obtained for approximately $6000. This would accomodate up to 200 terminals and 40 computer "ports."

Optimum systems

A comparison of the alternatives shows that a com-

posite system will be less expensive than one obtained completely from the Telephone Company. The two best alternatives are a system using telephone couplers on the regular internal telephone system or a system using private lines and a purchased line concentrator. Figure 4 shows a system using telephone couplers. It is similar to the system of Figure 3 but eliminates the need for obtaining Teletypes and data sets from the telephone company. Table V gives the projected costs for this arrangement. We would save $1640 per month by converting to it.

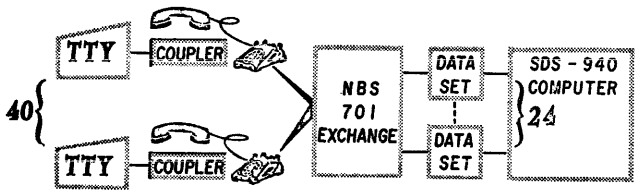


Figure 4—Telephone coupler - 701 system

TABLE V

Equivalent Monthly Cost For Telephone Coupler - 701 System

(See Figure 4)

Equipment	No.	Unit Cost	Total Equipment Cost
Teletypes	40	$85[1]	$3,400
Data sets	24	25	600
Couplers	40	10	400
701 exchange lines	64	2.85	182
Increase 701 Capacity[2]	35	3.95	138
		Total	$4,720

[1]This is an average price for a mixture of Model 33's and Model 35's with a variety of options. We estimate that leasing from Western Union or purchase of Teletypes will average at least $15 per month less than Telephone Company rates.

[2]There is a termination charge of 1/2 the cost for the remaining part of a 5 year lease.

Figure 5 shows the system using private lines and a purchased line concentrator. The costs for this system should not exceed those shown in Table VI and will probably be somewhat lower. With it we would expect to save $2330 per month over our present communication facilities. This arrangement has the additional advantage that it is the one best able to provide for connections to the time-sharing system from ARS, Telex, the regular telephone system, etc. Any system we get must be able to accommodate a small number of such special inputs. One alternative is simply to dedicate some of the computer ports to these types of services. This is however an undesirable method since the number of ports is limited. With our own switcher (and probably with one leased from Western Union) we can connect these terminals to the exchange input lines as shown in Figure 4, thus providing access to outside and special transmission facilities without having to dedicate ports to particular kinds of inputs.

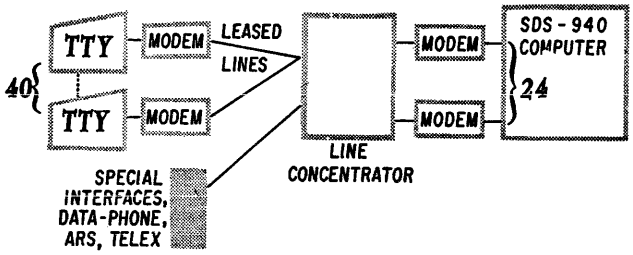


Figure 5—Purchased switching system

TABLE VI

Equivalent Monthly Cost For Purchased Switching System

(See Figure 5)

Equipment	No.	Unit Cost	Total Equipment Cost
Teletypes	40	$ 85[1]	$3,400
Modems	64	3[2]	192
Exchange	1	250[2]	250
Lines:			
Radio Bldg.	28	3	84
Main Campus	10	8	80
30th St. Bldgs.	2	12	24
		Total	$4,030

[1]We estimate that leasing from Western Union or purchase of Teletypes will average at least $15.00 per month less than Telephone Company rates.

[2]Based on a two year service life after installation.

Tariffs

Tariffs are often mentioned by the common carriers as a reason why this or that cannot be done. To a considerable extent, tariffs are the carriers' catalogs and price lists and simply reflect the business policies of the carriers. They are not, although you may get the impression they are, immutable laws writ in stone. They are sometimes easily changed upon application to the appropriate regulatory agency. Unfortunately, the carriers' business policies are sometimes rather

inflexible and may be difficult to get modified to accommodate new services properly. This has been especially true in regard to connecting customer-owned equipment to the telephone system. However the trend within the FCC is toward increased flexibility in regard to the use of these "foreign attachments," and this should make it much easier to use composite systems than it has been in the past. Our experience has been that it is useful to vigorously pursue our requests for new or unusual services even though the initial reaction from a carrier is that it cannot provide these services. In several such cases service arrangements that we proposed were initially rejected by the carrier as illegal or unacceptable for other reasons and were later found to be legal and acceptable.

CONCLUSIONS

There are a number of alternatives for meeting the communications needs of a time-shared computing system. We have found that an amazing variety of services can be obtained from the telephone company and that there are often good alternatives to use of telephone company equipment. In our case the investigation of these alternatives is leading toward substantial cost saving and improved service to our users.

The Baylor medical school teleprocessing system—operational time-sharing on a system / 360 computer*

by WILLIAM F. HOBBS and ALLAN H. LEVY
Baylor University College of Medicine
Houston, Texas
and
JANE McBRIDE
IBM Corporation
Houston, Texas

INTRODUCTION

The Baylor Teleprocessing System (BTS) is designed to operate as a time-sharing system. It accomplishes the following functions:

1. It allows several jobs initiated from various terminals to run concurrently with one batch job stream.
2. It permits the use of high-level languages for the construction of all programs, including those designed for remote terminals.
3. It insulates the user program from changes in the operating system by providing a set of macro-instructions and interface routines for input and output over telecommunication lines.
4. It provides certain utility functions for the terminal user, including the ability to build, alter, and retrieve data sets, and to communicate with the machine operator and other terminal users.
5. It provides a means by which programs originally written to run as batch jobs may be used from a remote terminal.
6. It insulates user programs from hardware errors originating during data transmission.

The system has been operational since July, 1967 on an IBM System/360 Model 50 with 256,000 positions of core storage. The terminals which the system supports include a cathode ray tube-keyboard terminal (IBM 2260 Display Station), and 2 types of typewriter terminals (IBM 2740 and IBM 1050). Both local CRT terminals (cable-connected to a contol unit which is directly attached to the channel) and remote CRT terminals (connected by telephone lines) are supported by the software. See Figure 1 for the configuration of the Baylor machine.

The primary programming support under which BTS was developed is Operating System/360—MFT (multiprogramming with a fixed number of tasks). OS/360—MFT, when it is loaded, divides core storage into a number of sections or partitions. These partitions are fixed in size until the Operating System is reloaded. OS/360—MFT must be utilized with at least two partitions if teleprocessing *and* batch jobs are to operate concurrently. Minor modifications are required to OS/360. These changes are incorporated into the standard systems generation procedures. Other programming support under OS/360 which is used by the teleprocessing system includes BTAM (Basic Telecommunications Access Method) and Basic Graphics Support.

If there is no batch job work to be done, one partition, for teleprocessing only, may be used. The teleprocessing monitor dynamically subdivides its partition as terminal jobs are requested until all available core is used. As soon as a terminal job ends, its core is freed and made available for another terminal user. Each job is storage protected (write protect only) so that it cannot alter or destroy any other job in the system. The system time-shares between the teleprocessing programs and the batch job stream if there is one. Time-slicing and control transfer during wait status are both utilized to accomplish time-sharing. Additional details are set forth in a later section.

The system provides extensive language interfacing

*Supported in part by grant FR-259 from the National Institutes of Health, grant HM-509 from the Division of Hospital and Medical Facilities, United State Public Health Service, and grant RT-4 from the Bureau of Social and Rehabilitation Services, HEW.

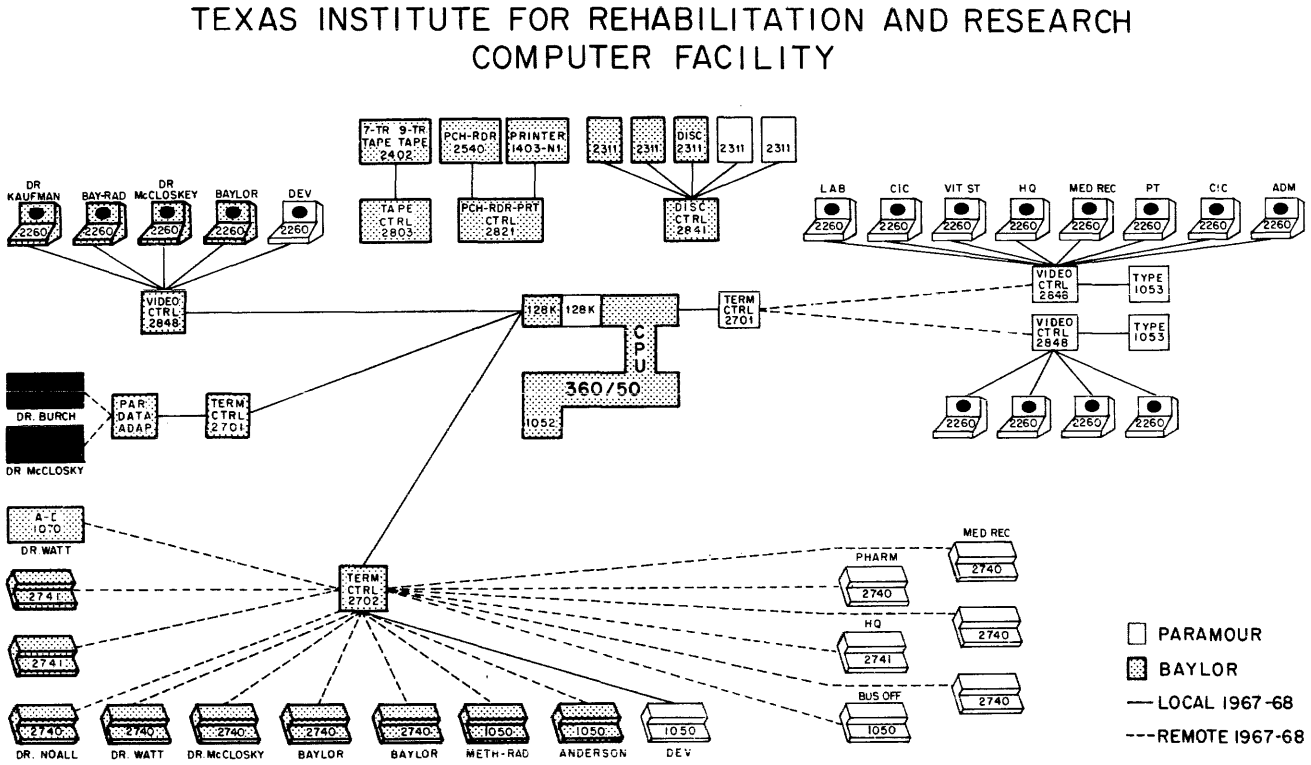


Figure 1—Operational configuration. The system serves both scientific users at Baylor University College of Medicine and the hospital data management system at the Texas Institute for Rehabilitation and Research (TIRR)

so that teleprocessing programs may be written in Assembler Language, PL/1, FORTRAN or COBOL. Programs written in Assembler Language input and output data to and from a terminal through macro-instructions. All high level language programs access terminals through CALL statements.

Each terminal is assigned a unique 8-character symbolic name which identifies its location and terminal type. This terminal name plus the time and date comprise the message prefix which is added by the system to every input message. A program can thus explicitly identify the source of every input message by inspection of the terminal name in the message prefix.

Space for all data sets for all teleprocessing jobs must be allocated at the time the teleprocessing system is loaded and initialized each day. Terminal users cannot create additional data sets. They can, however, read or alter any existing data sets that are defined for their use at system initialization time. Definition of files used in the system is handled by the Job Control Language of OS/360.

It is advantageous, although not essential, to have only checked-out programs running in the teleprocessing partition. BTS thus includes a teleprocessing simulator which allows any programmer to debug his teleprocessing program as a batch job. The simulator uses the card reader and the printer to simulate terminal input and output devices.

Terminal operation

The command language or set of control statements is dependent of the type of terminal in use. A control statement is recognized by its first two characters—"$$." The teleprocessing monitor responds to the successful entry of any control statement with an "OK."

The command language consists of seven control statements:

$$ACCOUNT
$$EXECUTE (or $$EXEC)
$$END
$$DDNAMES
$$EDIT
$$EOT
$$CONSOLE

A terminal user ordinarily would first wish to enter a $$ACCOUNT statement as follows:

$$ACCOUNT (B123, B1234, JONES)

This statement records on disk the user's pertinent accounting information: department number, project number, name and time of sign on. This accounting information must be recorded before a user is allowed to execute a program. Accounting information, once recorded, is not cleared until a user sends a $$EOT control statement. At this time another accounting record is written to clock off the user.

Once a user has established his accounting record, he can then execute a program. Programs to be executed from terminals must be stored in one of several teleprocessing libraries on disk. When a user wishes to execute a program, he must tell the teleprocessing monitor the program name, the amount of core it requires, and the program type. The program type may be:

1) SINGLE—only one copy of the program may be in core at any time. (Example - a file update program)
2) COPY—the program handles only one terminal but multiple copies may be in core and executing concurrently.
3) MULTIPLE—the program handles multiple terminals so that only one copy ever need be in core.

If no program type is specified, the monitor assumes type SINGLE. A user might call into execution the message switching program with the following control statement:

$$EXECUTE (SWITCH, 2000, MULTIPLE)

He is asking for the program named SWITCH which uses 2,000 bytes of core storage and handles multiple terminals. If 2,000 bytes of core are available, the program will be executed immediately and the user will receive a message from the program SWITCH with instructions on how to message-switch.

As another example, a user might enter the following $$EXECUTE statement:

$$EXEC (ELCOMP, 40000)

He is requesting the program named ELCOMP which uses 40,000 bytes of core and is type SINGLE. If 40,000 bytes of core are available, the program will be executed immediately; if not, the user will be informed that all core is in use, and he should try again later.

When a user wishes to end a program he is executing, he issues the following control statement:

$$END

This statement frees the core of the program he is executing, but does not clear the accounting information of the user. Thus he is now free to execute any program he wishes.

A terminal user may request the exclusive use of a data set by means of the $$DDNAMES control statement as follows:

$$DDNAMES (DATA1, DATA2)

The teleprocessing monitor checks its list of data set names previously entered with a $$DDNAMES statement and notifies the user if DATA1 or DATA2 is already in use. If neither is already in use, both are added to the list and the user receives his "OK" response.

A terminal user can request special editing functions by means of the $$EDIT control statement. Two levels of editing are available. Level 1 editing merely translates all character codes to internal EBCDIC codes. Level 2 editing removes all typewriter control and function key characters from the text of input message. The system default is level 2 editing. A user may also request that blanks be suppressed on incoming data and that small letters be translated to corresponding capital letters.

To end a session at the terminal, a user might enter the following control statement:

$$EOT

If he is executing a program, the program will be terminated and its core will be freed. In any case, his accounting information will be "logged off" on disk. The monitor will respond, as usual, with an "OK" when the user is cleared.

A terminal user may communicate with the machine room operator by means of a $$CONSOLE statement. He might send the following statement:

$$CONSOLE (MY JOB WILL RUN 5 MIN PAST SHUTDOWN TIME. MAY WE DELAY SHUTDOWN?)

The entire message within the parentheses will be printed on the console typewriter in the machine room, together with a prefix which identifies the send-

ing terminal. Thus the operator may send a reply back to the terminal if necessary.

There are seven different control statements in the command language of BTS. Frequently a user wishes to specify special data set or editing information and he must enter several statements. Therefore, the use of catalogued control statements is the substantial assest for the terminal user that OS/360 catalogued procedures are for the Job Control language user. A disk data set (hereafter called the Control Statement Library) contains sets of control statements which may be called by the terminal user with just one entry. For example, a set of control statements called SWITCH exists on the Control Statement Library. The set contains the following control statements:

```
$$ACCOUNT (B999, X9999, ANYNAME)
$$EDIT (2,F)
$$EXEC (SWITCH, 2000, MULTIPLE)
```

A terminal user can invoke these control statements (and, hence, execute the program named SWITCH) by entering the following command:

```
$$SWITCH
```

The terminal user can also add, change, display, or delete sets of control statements in the Control Statement Library. However, password protection is available (if wanted) so that a set of control statements may be displayed and used but not altered.

Time-sharing the teleprocessing partition

One feature of BTS is its ability to handle multiple teleprocessing jobs within one OS/360 partition. To implement this capability requires two changes to OS/360:

1) The task control block table in the nucleus must be expanded to handle additional teleprocessing task control blocks.
2) Two system termination (ABEND) modules must be changed. Both of these changes can be incorporated into the standard systems generation, using the source modules for the termination routines.

When OS/360 is loaded, the system is initialized with at least one partition. Under OS/360–MFT one task control block (TCB) is created for each partition of core storage. The TCB controls the execution of each successive job within the partition. See Figure 2 for a sample layout of core storage after OS/360–MFT has been initialized with two partitions.

BTS is the first job scheduled by the Operating System. The initialization Routine builds a number, N, of teleprocessing task control blocks where N will be the maximum number of teleprocessing jobs that

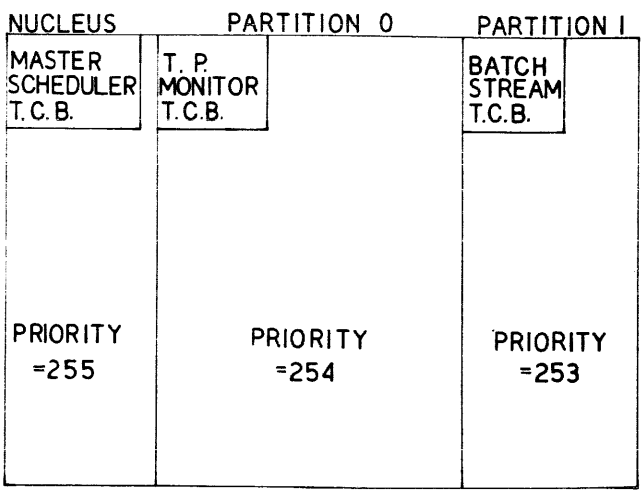


Figure 2–Layout of core storage after initialization of OS/360-MFT for two partitions

can execute concurrently. N is defined in the Teleprocessing System Tables and can be changed by reassembling that module. The teleprocessing task control blocks (TCB's) are added to the OS/360 TCB chain and given priorities below the teleprocessing monitor, while the priorities of the other lower Operating System partitions are shifted down. See Figure 3 for a sample layout of core storage after OS/360--MFT has been initialized with two partitions *and* BTS has been initialized for three concurrent teleprocessing jobs.

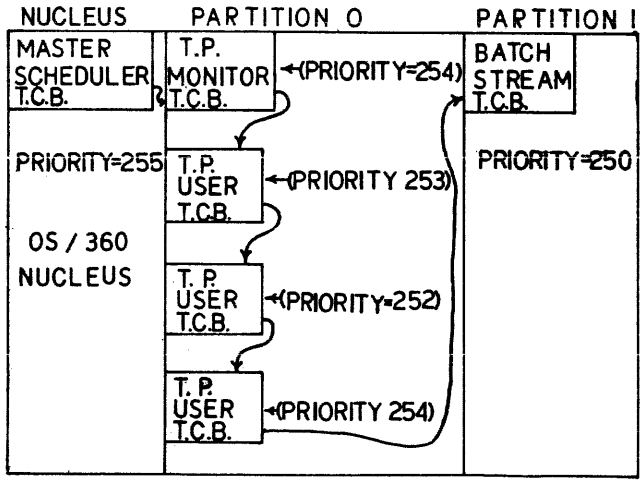


Figure 3–Layout of core storage after initialization of OX/360-MFT for two partitions and BTS for three concurrent teleprocessing jobs

The teleprocessing task control blocks wait in a non-dispatchable mode until a terminal user requests a program. At that time, the monitor gets core for the

requested program, completes other Operating System control blocks and puts the program into execution under the control of one of the teleprocessing task control blocks. From this time until the teleprocessing program terminates, it is handled by OS/360 like any other program executing under a standard task control block.

When the teleprocessing program terminates, BTS frees all of the core used by the terminating program, and the task control block is made non-dispatchable, waiting to be re-used by another teleprocessing job.

The system time-shares using two methods:

1) Time-slicing with circular rotation of task control block priorities.
2) CPU control transfer when any task goes into wait status.

Time-slicing is accomplished using an equal priority algorithm. At the end of each specified time interval (for example, 100 milliseconds) BTS takes the user task with the highest priority and gives it the lowest priority. The priorities of all other tasks on the chain are incremented by one, thereby giving a new job the highest user priority. When each time interval expires, the circular chain of task priorities is re-adjusted. If a batch job stream is in use, its priority is also rotated on the circular chain.

CPU control transfer when a task goes into wait status is handled by the OS/360–MFT dispatcher. Dispatching is attempted on a task priority basis, searching down the chain until a task is found that can utilize the CPU. If no task is found, the system waits until an event (such as I/O) completes so that some task can use the CPU.

The combination of the two methods described above provides the advantages of concurrent execution of multiple jobs and attempted full CPU utilization.

Language interfacing to terminals

The large majority of teleprocessing programs need to communicate directly with one or more terminals. In order to facilitate this communication, the capability is provided to get a message from a terminal (GETMSG), to put a message out to a terminal, (PUTMSG), and to break conversational mode with a terminal (BREAK). Macro-instructions provide these capabilities to the Assembler Language programmer. Interface routines allow the high level language programmer to make use of these functions via a CALL statement.

For example, an Assembler Language program may issue the following macro:

```
GETMSG DATAAREA, 96
```

The next message received from a terminal in conversational mode with this program will be placed in DATAAREA. The maximum length of the message will be 96 bytes which includes the message prefix of 15 bytes, 80 data bytes, and an end of block character (EOB). The program can then look at the 8-character symbolic terminal name in the message prefix to find the source of the data.

The same Assembler Language program may issue the following macro:

```
PUTMSG ADDRESS=DATAAREA+15,
LENGTH=80 DEST=BCOMPS60,
PRIOR=0, LINE=12
START=1, ERASE=NO
```

The monitor will then send (with a priority of zero) to the terminal named BCOMPS60 (*B*aylor *Comp*uting *S*cience 2*260*) 80 characters of data beginning at DATAAREA + 15. The 80 characters of data will be written on Line 12 of the 2260 and a START MI symbol will then be put in the first position of Line 1. The 2260 screen will *not* be erased prior to the write. If the terminal indicated by the parameter 'DEST' were not a 2260, the last three parameters would be ignored.

An Assembler Language program may also issue the following macro:

```
BREAK BCOMPS60
```

The monitor will then break conversational mode between the specified terminal and the program issuing the BREAK. If the specified terminal is the last or only terminal in conversational mode with the program, the program will be terminated and its core will be freed.

The same three teleprocessing capabilities (GETMSG, PUTMSG, and BREAK) are available to all high level language programs written in PL/1, COBOL, or FORTRAN.

For example, a PL/1 program may request a message as follows:

```
CALL GETMSG (STRING, RETURN);
```

The next message received from a terminal in conversational mode with this program will be placed in STRING (which is a character string variable of varying length). The return code will be placed in RETURN.

Likewise, a FORTRAN program may wish to send a message:

```
CALL PUTMSG (ARRAY, LENGTH, DEST,
PRIOR, RETURN)
```

The data in ARRAY will be sent to the terminal specified by. DEST with the priority specified in

PRIOR. The return code from the write will be placed in RETURN.

In the same manner, a COBOL program may break conversational mode as follows:

```
CALL 'BREAK' USING TERM, RETURN.
```

Conversational mode will be broken between the program and the terminal specified by TERM. If the specified terminal is the last or only terminal in conversational mode with the program, the program will be terminated and its core will be freed.

With the language interfacing facilities of the Baylor Teleprocessing System, no special training is required for application programmers to make active use of the system. They are free to use the language with which they are familiar, and by doing so, they can access any terminal on the system.

Terminal input/output

Two basic types of terminals are supported by BTS. Local CRT terminals are connected to the central processor by cables. All other terminals are connected by means of telephone lines. Different programming methods are used for these terminal types.

Because of the relatively high speed of local CRT terminals, no queues of messages are maintained for them. Read and write operations involving these terminals are always carried out immediately when they are requested by a problem program.

For all other types of terminals, two output queues at different levels of priority are maintained for each telecommunications line. Each PUTMSG to one of these terminals causes a message to be placed in one or the other queue according to its priority. These queues are maintained both in core and on direct access storage. A sufficient amount of text is kept in core to insure that the communications lines are kept active. Additional text is spilled to direct access devices.

When a line becomes idle, the following method is used to determine what operation to carry out next:

If any high priority messages are enqueued for transmission on the line, the next message (on a first in-first out basis) is dequeued and sent. If no high priority messages are awaiting transmission and some terminal on the line is using a problem program the line is left idle until the program issues a GETMSG. At this time a read operation is begun on the line.

If no terminal on the line is using a problem program, low priority messages (if any) are transmitted. When no low priority messages remain to be sent, a general polling operation is begun on the line. That is, each terminal in turn is interrogated to see if it wishes to send a message.

A dynamic buffering technique is used. This method allows buffers to be assigned to a line from a common pool of available buffers even while a channel input operation is in progress. Thus, individual input messages may vary greatly in length without tying up large amounts of storage to handle worst-case situations.

The Baylor Teleprocessing System has proven useful for the research environment in which it was developed. It is presently supporting several groups of scientific users, including those involved in mass spectroscopy, radiotherapy treatment planning, and gynecological cancer control. It is also the support system for a comprehensive computer-oriented hospital data management project at the Texas Institute for Rehabilitation and Research. With this operational load, it has been found that the size of core storage is the single most important limitation. We plan system support for low speed large core storage and for the IBM 2314 direct access device. It is felt that such additional machine capacity will remove present limitations and enhance the operational capabilities of the system.

Some techniques for shading machine renderings of solids

by ARTHUR APPEL
IBM Research Center
Yorktown Heights, N. Y.

INTRODUCTION

Some applications of computer graphics require a vivid illusion of reality. These include the spatial organization of machine parts, conceptual architectural design, simulation of mechanisms, and industrial design. There has been moderate success in the automatic generation of wire frame,[1] cardboard model,[2] polyhedra,[3,4] and quadric surface[5] line drawings. The capability of the machine to generate vivid sterographic pictures has been demonstrated.[6] There are, however considerable reasons for developing techniques by which line drawings of solids can be shaded, especially the enhancement of the sense of solidity and depth. Figures 1 and 2 illustrate the value of shading and shadow casting in spatial description. In the line drawing there is no clue as to the relative position of the flat plane and the sheet metal console. When shadows are rendered, it is clear that the plane is below and to the rear of the console, and the hollow nature of the sheet metal assembly is emphasized. Shading can specify the tone or color of a surface and the amount of light falling upon that surface from one or more light sources. Shadows when sharply defined tend to suggest another viewpoint and improves surface definition. When controlled, shading can also emphasize particular parts of the drawing. If techniques for the automatic determination of chiaroscuro with good resolution should prove to be competitive with line drawings, and this is a possibility, machine generated photographs might replace line drawings as the principal mode of graphical communication in engineering and architecture.

A picture strictly rendered in chiaroscuro defines the scene in a dark and light area pattern, colored or in tones of grey and no lines are made. Rembrandt and Reubens were masters of chiaroscuro. In order to simulate the chiaroscuro of a photograph many difficult problems need to be solved such as the effect of illumination by direct and diffuse lighting, atmospheric diffusion, back reflection, the effect of surface texture, tonal specification, and the transparency of surfaces. At present, there is the additional problem of hardware for display of the calculated picture. Devices presently available use lines or points as the principal pictorial element and are not comparable to oil paint, or wash, or crayon in the ability to render the subtle changes in tone or color across an area. The best we can hope to do is to simulate the halftone process of printing.

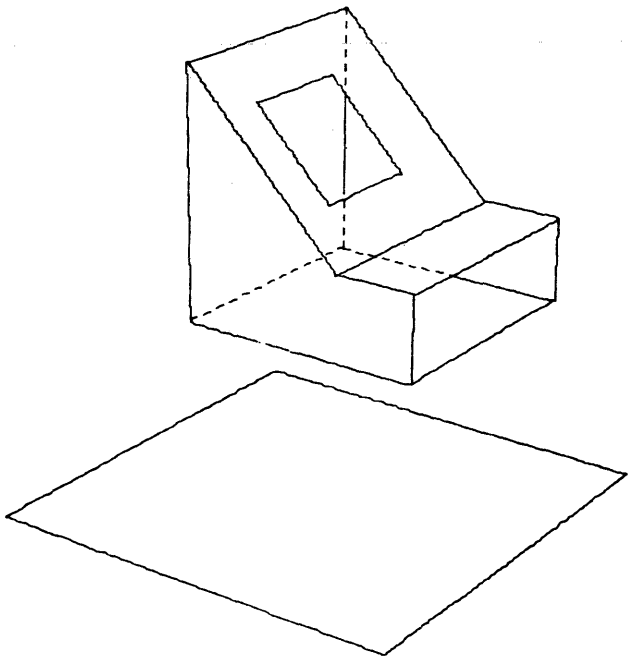


Figure 1 – A machine generated line drawing of an electrical console and an arbitrary plane in space

This paper presents some recent experimental results in the automatic shading of line drawings. The purpose of these experiments was to generate pictures of objects consisting of flat surfaces on a digital plotter and to evaluate the cost of generating such pictures and the resultant graphical quality.

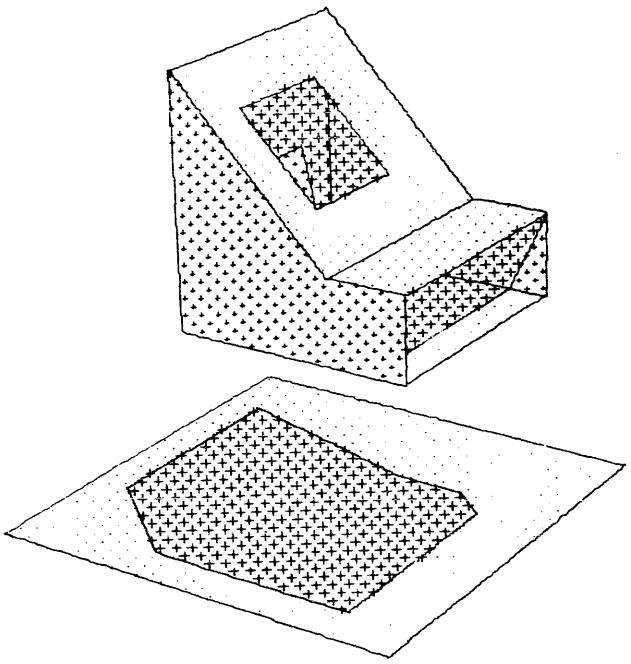


Figure 2—A shaded line drawing of the scene in Figure 1

Previous work

Considerable work has been done in the digitizing of photographs.[7,8] Especially successful are the pictures transmitted from spacecraft.[9] The significance of this work is the demonstration of the quality of digitally generated pictures.

L. G. Roberts has accomplished the converse of the problem being discussed in this paper by developing techniques for the machine perception of solids which are assemblies of convex polyhedra modules.[10] His work suggests the possibility that it may be more useful to analyze the contents of a photograph and to create a mathematical model of the scene. This analysis can be used to generate any view of the scene with greater graphical control.

G. Lasher has written a program which can be used to generate three dimensional graphs of mathematical functions which are unique for values of X and Y. This program, which was used to illustrate an article in theoretical physics,[11] generates contour curves of the surface, constant coordinate curves and renders only those curve segments that are visible in a perspective projection. A shading effect occurs in these pictures because the projection of the surface rulings tend to concentrate as the surface becomes tangent to the line of sight. This effect contributes significantly to the vividness of the renderings.

J. L. Pfaltz and A. Rosenfeld have applied their notions on encoding plane regions to shading two dimensional maps.[12] Their notion of skeleton representation is that a region can be specified by a list of points on a plane and a radius; all parts of the plane within the radius of the point are within the region described. For shading, a set of parallel straight lines are generated and those portions of the lines which be within the region are rendered. The angle and spacing of the set of parallel lines can be varied and other textures can be generated.

A vivid automatically shaded picture of a polyhedron was generated by a subroutine written by B. Archer for an article by A. T. Cole.[13] The shading is accomplished by varying the spacing of parallel lines. The spacing of lines on a particular surface is proportional to the illumination. No attempt was made to determine the shadow cast by the polyhedron and the methods described are inadequate for drawing more than one convex polyhedra at a time.

Recently, C. Wylie, G. Romney, D. Evans, and A. Erdahl[13] published an algorithm for generation of half-tone pictures of objects described by assemblies of triangular bounded planes. Their results are toned pictures generated with calculation time competitive with line drawings. However their scheme sets the source of illumination at the viewpoint, and since a point light source cannot see the shadows it casts, no shadows are rendered.

Previous work in the automatic determination of chiaroscuro demonstrates how the computer can improve the level of graphics designers can work with. The primary limitation has been neglect of shadow casting from arbitrarily located light sources. Also no work has been done on the control of the toned picture to take into account the surface tone or color of an object. Any system for rendering in chiaroscuro should solve economically at least these two problems. It can be seen from previous results that toning an area by varying the spacing of parallel lines is not entirely satisfactory. This technique is economic but has several disadvantages. The lines when widely spaced do not fuse to form a continuous tone. The viewer does not then perceive the object but is distracted by the two dimensional pattern. Depth perception is reduced. These lines also tend to suggest a surface finish which may not exist. A good standard for evaluating toning mechanics can be the ben-day pattern used in printing. This pattern enables a great range of dark and light with good tonal fusion. The half-tone process uses many small dots arranged in a regular array. The size of dots are varied to create a degree of grayness, small dots where white predominates are light and as the dots increase in size such that

they eventually blend together the toned region becomes darker. In order for the dot pattern not to be distracting, the dot spacing should be at least seventy dots to the inch. The large dot density required for toning indicates that calculation schemes for toning should be as resolution independent as possible. For an algorithm to be resolution independent it must enable perfect resolution. It may not be possible for contemporary hardware to take advantage of such an algorithm but this should be the goal.

Toning on a digital plotter

A great many experiments were conducted to evaluate the quality of various toning techniques that would be applicable to digital plotting. A simulation technique tested was to shoot random light rays from the light source at the scene and project a symbol from the piercing point on the first surface the light ray pierced. These symbols would concentrate in regions of high light intensity, and a negative print of the hard copy could be made which would approximate a photograph if enough light rays are generated. Even for about 1000 light rays results were splotchy. Generating light rays in regular patterns improved the graphic quality but did not allow economic tonal control. During these experiments, various symbols were evaluated for graphic quality and speed of plotter generation. The plus sign or a small square were to give best results. Eventually the best technique from graphic and economic considerations for toning was found to be plus signs arranged in staggered rows with shadows outlined. This arrangement is most easily seen in Figure 2. The size of plus signs were to be rendered proportional to darkness required.

Ignoring atmospheric diffusion, the intensity of light incident upon a unit plane area from a point light source is:

$$I = S\ (\text{Cosine } L)/D^2$$

where S is the intensity of the light source, L is the angle of the normal to the plane and direction of light at the illuminated area, and D is the distance from the illuminated point to the light source. For experimental purposes it was assumed that the light source is so far from the objects being illuminated that variations in L and D are insignificant. L need be calculated only once for each surface. Also since we are interested in simulating illumination only to the extent that comparative light and darkness of surfaces are displayed and also because the range of toning on the digital plotter is limited, we did not concern ourselves with the actual intensity of the single light source. The comparative intensity of illumination of a point on a plane then is proportional to Cosine L. The apparent illumination of a flat surface then will be constant over the surface. The digital plotter does not generate light as a cathode ray tube but makes a dark mark. The size of this mark should indicate an absence of light. So the degree of darkness at a point or the size of plus sign rendered is

$$H^* = 1 - \text{Cosine } L$$

For simplicity it can be assumed that if a point is in shadow the largest allowable mark will be rendered on that point. For point j then, the size of symbol H_j is the maximum symbol H_s. H_s is proportional to the dot spacing. During early experiments of the size of symbols rendered on a point not in shadow was

$$H_j = 1 - \text{Cosine } L$$

However it was found that results were confusing; it was difficult to detect the difference between surfaces in shadow and surfaces almost parallel to the direction of light. In actual viewing of a solid, surfaces almost parallel to the direction of light reflect a considerable amount of light due to the surface roughness, but as soon as the surface faces away from the light source, no light can be reflected and the apparent illumination changes sharply. In order to simulate this effect a contrast factor, h, usually .8, was introduced. So then

$$H_j = h\ H_s\ (1 - \text{Cosine } L) \qquad (1a)$$

if the surface point j is on faces the light source and

$$H_j = H_s \text{ if point j is in shadow.} \qquad (1b)$$

We are faced now with essentially at least four programming problems:

1. Given a viewpoint and a mathematically described scene what is the point in picture to point in scene correspondence? This problem is to determine what visible point, if any, on the objects being rendered project onto a particular point in the picture plane.
2. Given one or more light sources what is the intensity of light falling on a point in the scene? This problem includes the determination of which regions are in shadow.
3. Given a light source what are the boundaries of the shadow cast and how much of this cast shadow can be seen? If the picture could be rendered large and/or if symbol density could be large outlining the shadows could be dispensed with.
4. How can the tone or color of a surface be specified and how should this specification affect the tones rendered?

Economic solutions to the first two problems are most critical. If an economic point to point correspondence technique could be found that would permit dense symbol packing, the problem of casting shadow outlines could be eliminated. The problem of

determining how much light falls on a flat surface not in shadow is trivial, and even for curved surfaces this is not difficult, but economically determining exactly what regions of the scene are in shadow is a very difficult problem.

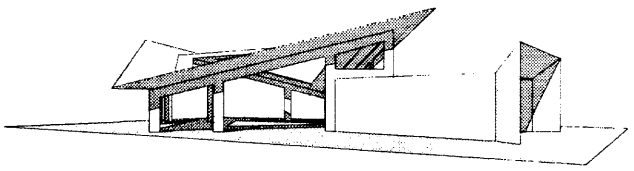


Figure 3 – An assembly of planes which make up a cardboard model of a building

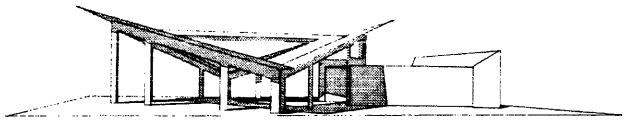


Figure 4 – Another view of the building

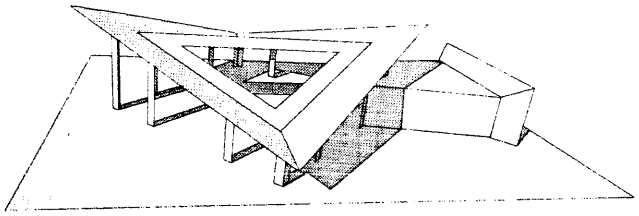


Figure 5 – A higher angle view of the building. 7094 calculation time for this picture was about 30 minutes.

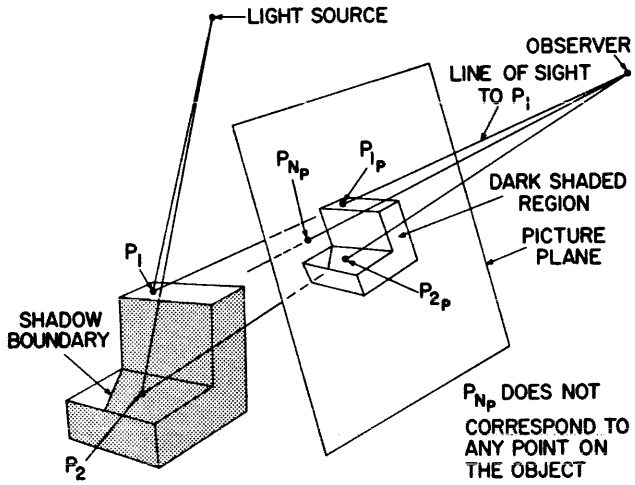


Figure 6 – Point by point shading

Point by point shading

Point by point shading techniques yield good graphic results but at large computational times. These techniques are docile, require the minimum of storage and enable easily coded graphical experimentation. Figures 3, 4, and 5 are examples of point by point shading. Referring to Figure 6, the technique in generating these pictures was as follows:

1. Determine the range of coordinates of the projection of the vertex points.
2. Within this range generate a roster of spots (P_{ip}) in the picture plane, reproject these spots one at a time to the eye of the observer and generate the equation of the line of sight to that spot.
3. Determine the first plane the line of sight to a particular spot pierces. Locate the piercing point (P_i) in space. Ignore the spots that do not correspond to points in the scene (P_{np}).
4. Determine whether the piercing point is hidden from the light source by any other surface. If the point is hidden from the light source (for example P_2) or if the surface the piercing point is on is being observed from its shadow side, mark on the roster spot the largest allowable plus sign H_s. If the point in space is visible to the light source (for example P_1) draw a plus sign with dimension H_j as determined by Equation 1.

This method is very time consuming, usually requiring for useful results several thousand times as much calculation time as a wire frame drawing. About one half of this time is devoted to determining the point to point correspondence of the projection and the scene. In order to minimize calculation time for point by point shading and maintain resolution, techniques were developed to determine the outline of cast shadows. Outlining shadows has the advantage that all regions of dissimilar tones on the picture plane are outlined. Even when projected shadows are delicate, and symbol spacing is large, the shadows are specified and the discontinuity in tone is emphasized.

The strategy for point by point determination of shadow boundaries is as follows: (Referring to Figure 7)

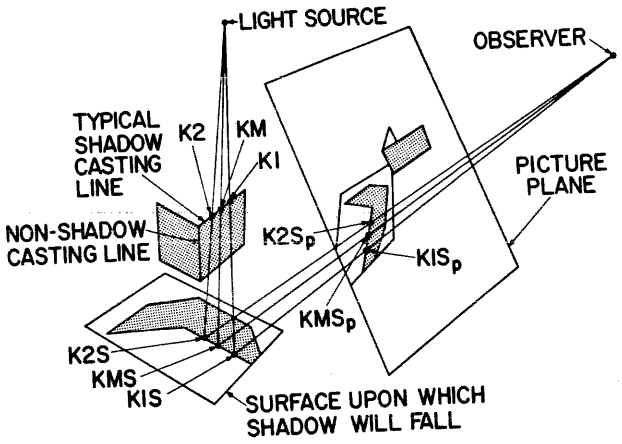


Figure 7 – Segment by segment outlining of shadows

1. Classify all surface line boundaries into shadow casting and non-shadow casting. A shadow casting line is from the viewpoint of an observer at the light source a contour line. For assemblies of flat surfaces, a contour line along which all surfaces associated with this line appear on only one side of this line.
2. Determine whether the observer is on the shadow side or lighted side of all surfaces.
3. Subdivide all shadow casting lines, one at a time, into small segments (K1, K2), usually .005 units, and determine the midpoint of this segment (KM).
4. Generate a light ray to the midpoint of the segment (KM). If any surface lies between KM and the light source go on to the next segment. Determine the next surface behind KM that the light ray to KM pierces within its boundary. If no surface lies behind KM go on to the next segment. A point can cast only one shadow. Project K1, KM, and K2 onto the surface to obtain K1S, KMS, and K2S, the shadows of K1, KM, and K2. If KMS lies on a surface which is seen from its shadow side go on to the next segment. This particular shadow boundary is invisible. Also a shadow cannot fall within a shadow.
5. Test KMS for visibility. If KMS is hidden from the observer go on to the next segment.
6. If KMS is visible project the line (K1S-K2S) onto the picture plane and draw the projection.

As can be expected, determining the outline of shadows by this described strategy is very time consuming usually requiring as much time as a point by point line visibility determination.

Methods of quantitative invisibility

In a previous report, the notion of quantitative invisibility was discussed as the basis for rapidly determining the visibility of lines in the line rendering of polyhedra.[4] P. Loutrell has implemented several tactical improvements for this application.[15] Quantitative invisibility is the count of all surfaces, seen from their spatial side, which hide a line from an observer at a given point on the line.

The methods of quantitative invisibility are useful because techniques for detecting changes in quantitative invisibility along a line are more economical than measuring the visibility, absolute or quantitative, at a single point. These techniques are applicable only to *material lines* which are lines that have specific end points and that do not pierce any bounded surface within its boundary. Objects that are manufactured contain only material lines. A *contour line* is a line along which the line of sight is tangent to the surface of the solid. For polyhedra, given a specific viewpoint, a contour line is a material line which is the intersection of two surfaces, one of which is invisible. For a given viewpoint the quantitative invisibility of a material line can change only when it passes behind a contour line. Figure 8 illustrates how quantitative invisibility varies as a line passes behind a solid. Notice that only surfaces which are viewed from the spatial side affect the measurement of quantitative invisibility. In determing line visibility for line drawings only those segments of the line for which quantitative invisibility is zero are drawn. For this application only the quantitative invisibility of vertex points are stored and changes in quantitative invisibility along a line are measured and discarded as soon as commands to the graphic device are generated. The methods of quantitative invisibility can be applied to shading a picture if the changes in quantitative invisibility of a line from the light sources and the observer are stored and compared.

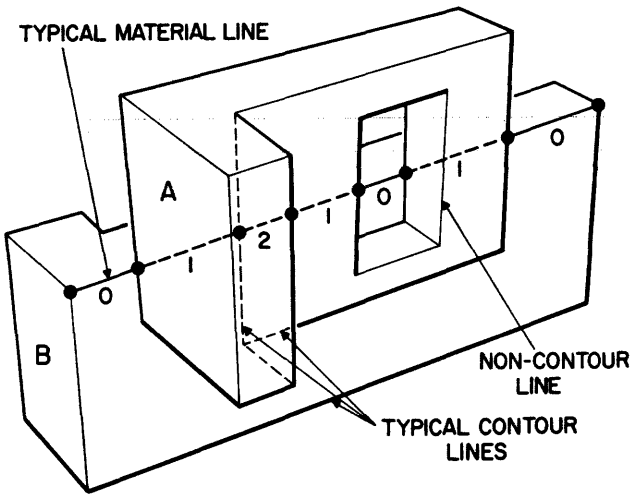


Figure 8—Changes in quantitative invisibility. Object A is in front of and does not touch object B

The method of cutting planes

In descriptive geometry, the intersection of simple quadric surfaces is determined by passing carefully chosen planes through the quadric surface to determine the intersection curve of the quadric surface and the plane; and from these first and second degree surface intersections the intersection curve of one or more quadric surfaces can be deduced. This procedure is time consuming but does solve a problem difficult for most mathematicians. This technique of manual rendering is the inspiration for the *method of cutting planes* for shading machine renderings of solids. Point by point shading techniques are expensive because

it is difficult with good resolution to correlate the shading of adjacent spots on the picture plane. With simple codings, the method of cutting planes enables such correlation in one direction, with more elaborate coding the correlation can be in all directions.

The basic concept of the method of cutting planes is that when the intersections of a plane that passes through the observation point and assemblies of planes which can enclose one or more polyhedra are projected onto the picture plane, these projected intersections are colinear. In detail, as illustrated in Figure 9, the strategy is:

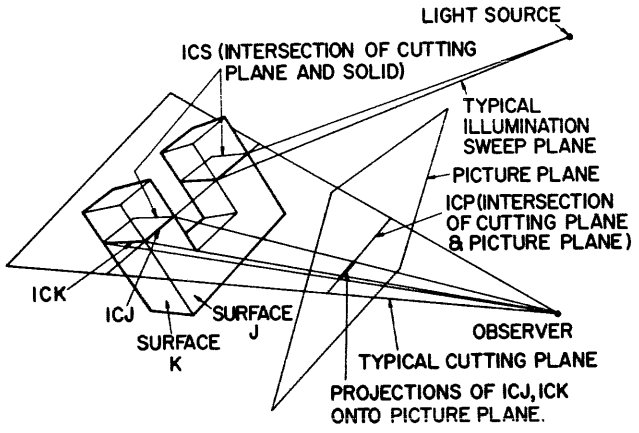


Figure 9—The method of cutting planes

1. Generate a cutting plane which passes through the observation point.
2. This cutting plane will intersect the picture plane along a specific line (ICP).
3. This cutting plane will cut or pass through the surfaces of the polyhedra and generate the intersection ICS which is a string of three or more line segments. Each of these segments is a material line (ICJ).
4. All the ICJ of the polyhedral faces and a particular cutting plane will project colinear onto the picture plane. This colinear projection is the line ICP.
5. These intersections ICJ for a particular cutting plane can then be measured for changes in quantitative invisibility by techniques previously reported[4,15]. Those intersections ICJ which prove to be completely invisible can be quickly determined and need be analyzed no further. We have now determined a correspondence between a line on the picture plane and a series of lines in the scene to be rendered.
6. Those lines of intersection ICJ which are on surfaces which face away from the light source can be rendered with no further analysis. These lines are completely in shadow and along their visible projected length plus signs of the maximum size (H_s) can be generated.
7. Lines of intersection ICJ which are on surfaces which face toward the light source can be analyzed to determine changes in quantitatives invisibility from the viewpoint of the light source. Those projected portions of the lines which are hidden from the light source are rendered by a series of plus signs of size H_s. Those portions which are visible to the light source are rendered by a series of plus signs whose size is determined by Equation 1.

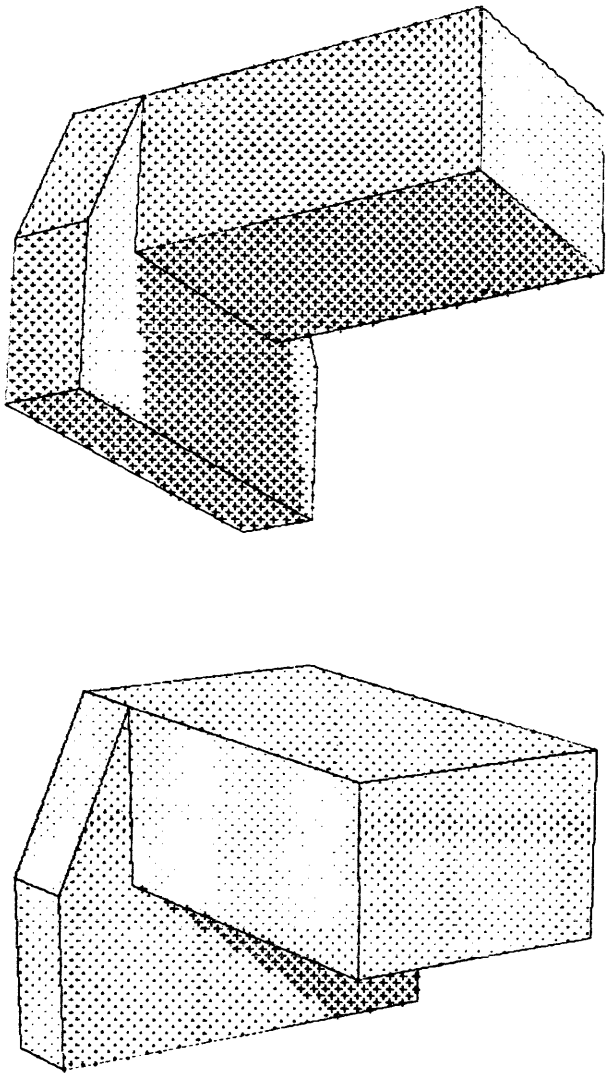


Figure 10—Two views of a machine part where the light source is moved relative to the object

The resolution of shading by the method of cutting planes is no longer limited by the spot to spot spacing on the picture plane but by the spacing of the cutting planes intersections with the picture plane. For comparable resolution the calculation time for shading by cutting planes is slightly less than the square root of the time for shading by point by point methods.

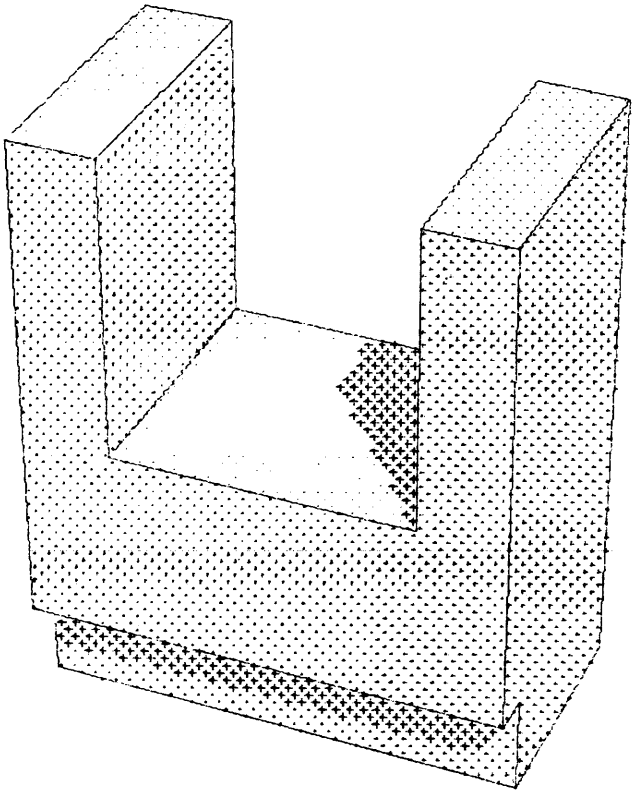


Figure 11 – Another machine part. 7094 calculation time for this picture was about 30 seconds

The speed of calculation is very dependent on how effectively the measurements of quantitative invisibility of all the lines in the scene are stored and correlated. This list is the basis for determining visibility along cutting plane intersections. The first version of the Fortran IV program used to generate Figures 10 to 14 was experimental and is not as fast as theoretically expected. Pictures were plotted on an IBM 1627 (Calcomp). A faster version which can take into account more than one light source is under development for operation on a 360/67. The larger core, greater data storage capacity and time sharing capability of this machine will be utilized. The method of cutting planes which enable rapid correspondence of projected points to real points in the scene certainly includes the illumination of the object by more than one light source of differing intensities. The size of

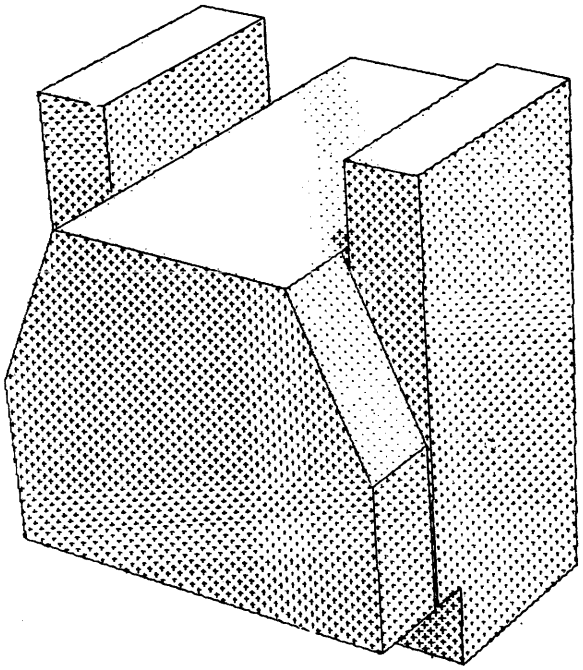


Figure 12 – Assembly of the two previously drawn machine parts. 7094 calculation time: about 50 seconds.

plus signs drawn on a particular spot can be the sum of the shadow intensity from all the light sources.

$$\text{HDRAWN} = \sum H_i \qquad (2)$$

where H is the shadow intensity from a particular light source.

The more intense the light source, the more intense the shadow it causes when a point cannot see this source of light,

$$\text{I TOTAL} = \sum I_i \qquad (3)$$

where I is the intensity of the light source

$$H_{si} = I_i/\text{I TOTAL} \qquad (4)$$

Where H_{si} is the shadow intensity in the absence of light from light source i.

$H_i = H_{si}$ when a point is in shadow

$$H_i = H_{si}\,(1\text{-CosineL})\,h \qquad (5)$$

when a point is seen by light source i.

It is also possible to exercise tone control for emphasis while generating the half-tone picture. A list of comparative surface tones can be entered which will describe the basic tone of each surface. For example, if a scene consists of object A with four sur-

faces and object B with six surfaces and object A is lighter than object B the surface tone list would be (.5, .5, .5, .5, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0). The size of the shading symbol can then be determined by

$$HT_{ik} = (H_i \times FLIGHT + FTONE)\, TONE_k \quad (6)$$

where FLIGHT and FTONE are influence factors of light and surface tone, and $TONE_k$ is the relative tone of surface k. FLIGHT and FTONE enable the control of highlighting. Master copy for the preparation of color process plates for letterpress printing have been generated using mathematical models similar to equation 6. It is obvious that once the basic problems of determining how light falls on the object are solved, considerable artistic freedom is possible.

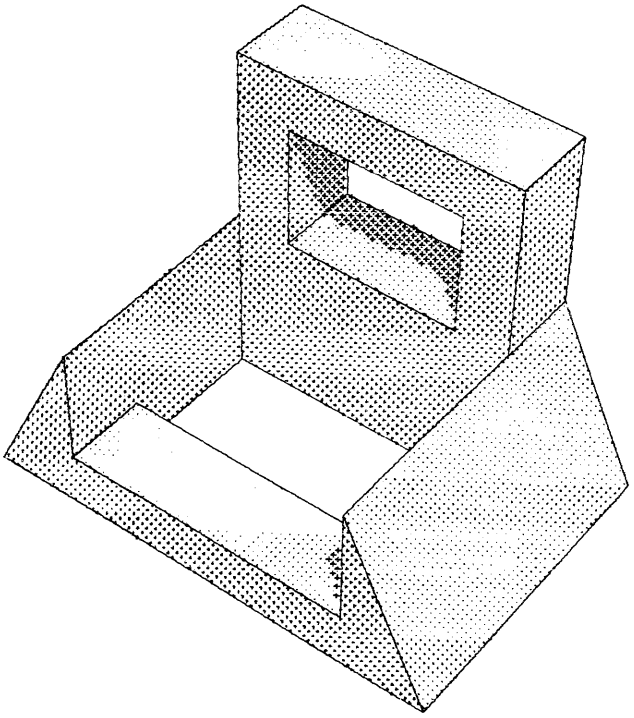


Figure 13 – Another machine part

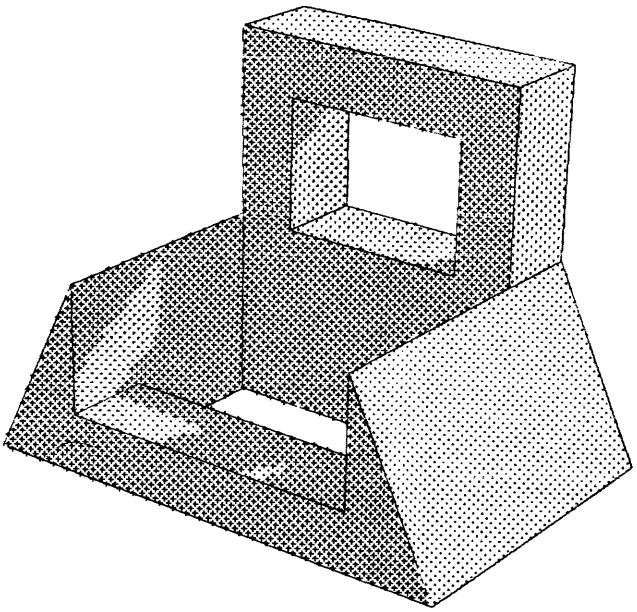


Figure 14 – Another view of the machine part shown in the previous figure. The light source has been moved relative to the object. Notice the light passing through the opening in the object

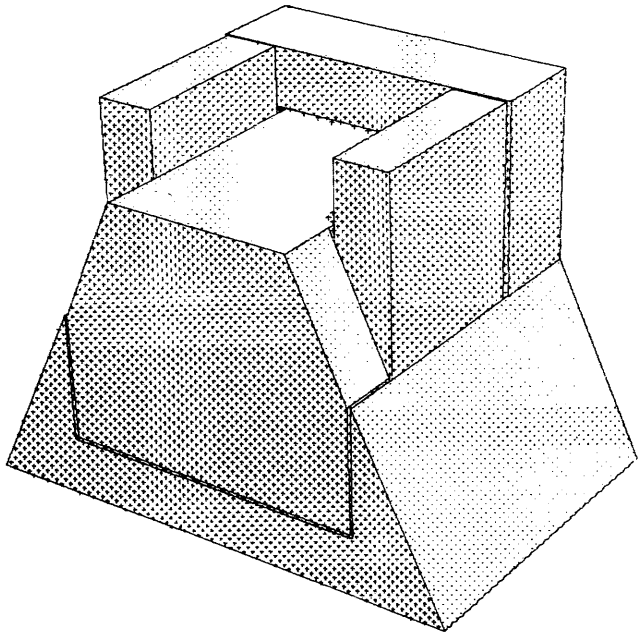


Figure 15 – Assembly of the three machine parts. 7094 calculation time for this view was about two minutes

ACKNOWLEDGMENTS

The author is grateful to G. Folchi, Dr. G. Lasher, and Dr. P. Loutrell for some helpful discussions. R. L. Ennis, J. J. Gordineer, J. A. Mancini, Mr. & Mrs. E. P. McGilton, W. H. Murray, and G. J. Walsh among others were helpful with plotter output and other machine problems. The author is deeply indebted to J. P. Gilvey and F. L. Graner for their continuing support and encouragement of this project.

REFERENCES

1 T E JOHNSON
Sketchpad III: A computer program for drawing in three dimensions.
Proc AFIPS 1963 Spring Joint Computer Conf Vol 23 pp 347-353

2. A APPEL
The visibility problem and machine rendering of solids
IBM Research Report RC 1618 May 20 1966

3 P LOUTREL
Determination of hidden edges in polyhedral figures: convex case
Technical Report 400-145, Laboratory for Electroscience Research NYU September 1966

4 A APPEL
The notion of quantitative invisibility and the machine rendering of solids
Proc ACM 1967 Conference pp 387-393

5 R A WEISS
BE VISION a package of IBM 7090 Fortran programs to draw orthographic views of combinations of plane and quadric surfaces
JACM 13 April 1966 11 194-204

6 H R PUCKETT
Computer method for perspective drawing journal of spacecraft and rockets
Vol I No 1 pp 44-48 1964

7 W S HOLMES H R LELAND G E RICHMOND
Design of a photo interpretation automaton
AFIPS Conf Proceedings 1962 Fall Joint Computer Conference Vol 22 pp 27-35

8 R W CONN
Digitized photographs for illustrated computer output
AFIPS Conference Proceedings 1967 Spring Joint Computer Conference Vol 30 pp 103-106

9 Lunar Orbiter Surveys the Moon Sky and Telescope Vol 32 No 4 October 1966 pp 192-197

10 L G ROBERTS
Machine perception of three-dimensional solids
Technical Report No 315 Lincoln Laboratory MIT May 1963

11 G LASHER
Mixed state of type-I superconducting films in a perpendicular magnetic filed
The Physical Review Vol 154 No 2 pp 345-348 Feb 10 1967

12 J L PFALTZ A ROSENFELD
Computer representation of planar regions by their skeletons
CACM Vol 10 No 2 February 1967 pp 119-125

13 A J COLE
Plane and stereographic projections of convex polyhedra from minimal information
The Computer Journal

14 C WYLIE G ROMNEY D EVANS A ERDAHL
Half-tone perspective drawings by computer
AFIPS Conf proceedings 1967 Fall Joint Computer Conference Vol 27

15 P LOUTRELL
Phd Thesis NYU September 1967
NYU September 1967

A system for interactive graphical programming*

by WILLIAM M. NEWMAN**
Harvard University
Cambridge, Massachusetts

INTRODUCTION

A system is described in this paper for developing graphical problem-oriented languages. This topic is of great importance in computer-aided design, but has hitherto received only sketchy documentation, with few attempts at a comparative study. Meanwhile displays are beginning to be used for design, and the results of such a study are badly needed. What has held back experimentation with computer graphics has been the difficulty of specifying new graphic techniques using the available programming languages; the method described in this paper appears to avoid this difficulty.

Defining a problem-oriented language

Notation

Any description of an interactive process must define the response of the system to each input. For this reason it is convenient to describe graphical problem-oriented languages in terms of *actions* and *reactions*. An *action* is simply an input which may produce a response; the corresponding *reaction* defines this response, and in addition any unmanifested effect of the action on the state of the machine. The same action may cause a different reaction on different occasions: for example, movement of the light pen may affect the display in a number of different ways. It is therefore convenient to treat the system as a finite-state automaton, and to say that the reaction is determined by the *state* of the program as well as by the action. In other words, the actions are *inputs* to the automaton, which cause it to change state; reactions are the *outputs*.

Just how convenient this is for describing interactive processes is illustrated by the following example. A 'rubber-band' line[1] can be created by means of a light pen and one push-button, in a sequence of five operations:

1) press button to start pen tracking;
2) track pen to starting point of line;
3) press button to fix starting point;
4) track pen to end point;
5) press button to fix end point and stop tracking.

The 'rubber-band' effect is created by displaying a line joining the starting point to the pen position throughout stage 4.

Figure 1 shows a state-diagram representing this sequence. Each branch represents an action, and the resultant reaction is specified in the "arrowhead." Only valid actions are included; for example, pen movements are meaningless in state 1 and are therefore omitted. The inclusion or exclusion of an action may add semantic properties to the diagram. This is shown by the 'pen movement' branches on states 2 and 3, which imply pen tracking during those states and make explicit reference to tracking unnecessary.

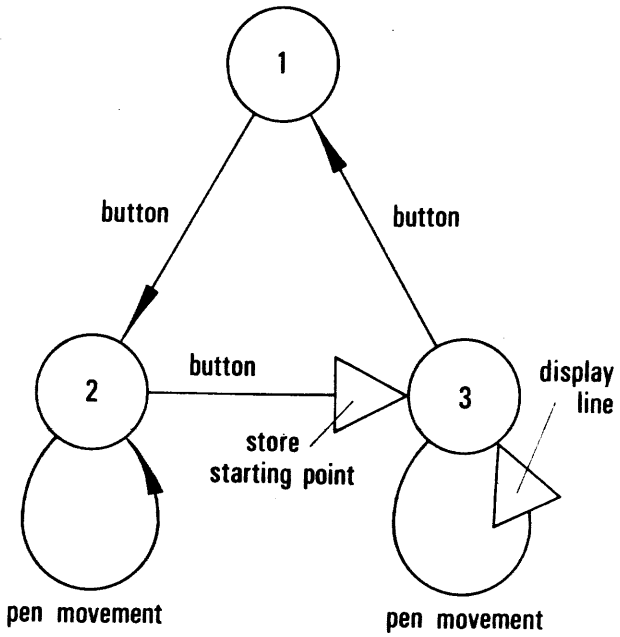


Figure 1 – A state-diagram representing rubber-bank line-drawing

*The work described in this report was supported by a Science Research Council Contract, No. B/SR/2071, "Computer Processing of Three-Dimensional Shapes."

**Formerly at the Centre for Computing and Automation, Imperial College, London.

The state-diagram has been used in this way as the basis of a method for defining problem-oriented languages. A particular advantage of this technique is the way an *immediate* reaction can be associated with each action in a sequence; this is of great importance in graphical programs. On the other hand the state-diagram offers no direct method of attaching semantic functions to groups of actions, and is therefore of little use for describing phrase-structured grammars. This is less of a drawback than it seems. An interactive problem-oriented language need not possess a complex structure to function efficiently, and benefit can often be gained from simplifying the language as much as possible. Roos, for example, has noted the difficulty experienced by some engineers in using the relatively simple languages of the ICES System.[2]

The basic function of the state-diagram, as illustrated in Figure 1, is to indicate the actions which may validly occur during each state, and the reactions and changes of state which they will cause. A number of additions have been made to this basic notation. Normally, branching takes place when a user action matches an action defined in the diagram. Branching may however be over-ridden by the result of a *test routine* included in the branch definition. Furthermore, branching may be initiated by the program itself by means of *system actions*: thus the result of a procedure may determine to which of several states the program will branch. A procedure of this kind is called a *program block* and is attached to a state rather than to a branch; it is executed every time the corresponding state is entered. Program blocks need not terminate in a system action, but may instead be used to provide some sort of continuous background activity.

These are the essential additions to the notation. One other has been included for convenience in programming 'conversational' systems, in which each input message produces a predetermined output message. This message can be coded within the reaction procedure or program block, but it is convenient to be able to state it separately as an output string or *response*. States therefore possess a response as well as a program block, and reactions are similarly defined as two components, a response and a procedure. The procedure is represented as an *instruction for execution* or *IEX*.

The suggested form of these additions to the notation is shown in Figure 2; a somewhat similar notation has been used by Phillips[3] to describe real-time control programs. Figure 2 illustrates how the first example could be entended to permit the removal of lines and initialization of the program. These two functions are controlled by the commands DELETE and RESTART; as explained below, commands may be typed at the console typewriter, or may be arranged to appear on the screen as light-buttons. After giving the command DELETE, the user points the light pen at each line to be removed. Deletion is carried out by a test routine DLAST, which also tests whether any other lines remain on the screen. When none remains, or the user gives the command DRAW, the program changes state. RESTART causes the program to enter the initial state 4, execute its program block PBGO and return to state 1 when initialization is complete.

The language

A *Network Definition Language* has been developed so that problem-oriented languages, defined in the form of state-diagrams, can be compiled into interactive programs. W. R. Sutherland[4] has shown that programs can be described directly to the computer in graphical form, and this technique has obvious applications to the input of state-diagrams. However, much of the information in these diagrams is in character form, and would be difficult to describe in purely graphical terms. For this reason, and because it is more suited to off-line preparation, a character-based language was preferred. The following remarks and examples are intended to give a general impression of this language, which is described elsewhere in some detail.[5]

The state-diagram is described by defining each state in turn; each such *state definition* is followed by a list of the branches from that state and their properties. The ordering of the state definitions, and of the *branch definitions* within a list, is immaterial. State and branch definitions are constructed from *statements,* each defining one property as in the following examples:

```
RESP   PRESS BUTTON
PB     PB22
IEX    REPROG
```

These define a response "Press button," a program block called PB22 and an instruction for execution named REPROG, respectively.

Each state has three properties (name, response and program block) and each branch has seven. For convenience, however, the language permits certain statements to be omitted if the value of the property is null, meaningless or zero. The only statements which are syntactically necessary are those defining the names of states and the actions of branches. The remaining statements in a state or branch definition must follow these, but can be given in any order.

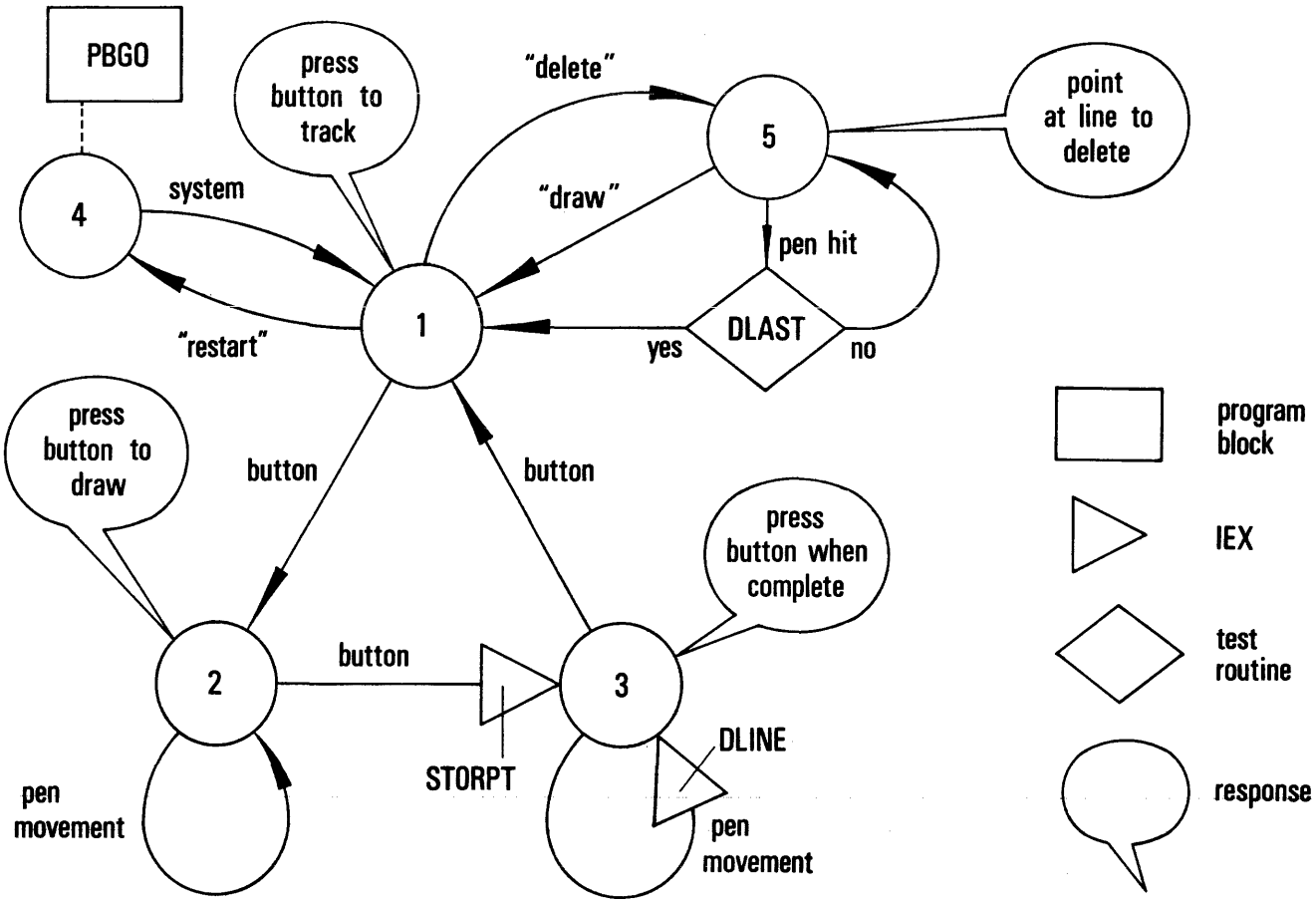


Figure 2—An extended diagram including responses, and with provision for drawing and deleting lines and for initialization

Some care has been taken to avoid explicit references to peripherals in the language. As a result, state-diagrams are largely *device-independent* and compile into similarly device-independent programs. For example, a 'pen movement' action may originate as movement of a light pen or tracker ball, as a pair of typed coordinated, or even as two numbers read off a tape. Any such compatible set of actions is called a *category*, and the definition of an action must include the category name in the first statement. Some actions, such as typed commands, require a further property to define the message content. A command "restart" would be defined thus:

```
ACT  0
MES  RESTART
```

The first statement indicates an action of category 0, which includes light-button hits as well as typed commands; the second defines the message content. Numerical names have been used for categories so that extensions to the category list can be made more easily.

Table I shows the state-diagram of Figure 2 coded by means of the language into a *network definition* and illustrates the use of the *state entry* to define a change of state. If no state entry is mentioned in a branch definition the state remains unaltered.

Compiling and executing an interactive program

Bilingual programming

The Network Definition Language contains no facilities for coding the procedures named in state diagrams. It is intended rather to be used in conjunction with a procedure-oriented language, each language being used for the tasks to which it is most suited. Some readers may disagree with this approach, which requires the programmer to be bilingual. The fact remains that procedure-oriented languages on to which powerful control facilities have been grafted rarely make for easy programming. It therefore seems reasonable that interactive programs should be written in two languages, one procedure-oriented and the other *control-oriented.*

```
STAT  1                          Comment:  State definition, state 1
RESP  PRESS BUTTON TO TRACK                State 1 response, "Press button to track"
 ACT  0                                    Branch definition, action of category 0 (command)
 MES  RESTART                                Message "restart"
 SE   4                                      State entry, i.e. branch leads to state 4
 ACT  0                                    Branch definition; command "delete" leads to state 5
 MES  DELETE
 SE   5
 ACT  10                                   Branch definition, category 10 (button)
 SE   2                                      Pressing button leads to state 2

STAT  2                                    State 2 definition
RESP  PRESS BUTTON TO DRAW                 State 2 response
 ACT  7                                    Branch definition, category 7 (pen movement)
 ACT  10                                   Branch definition; pressing button leads to state 3
 IEX  STORPT                                 STORPT stores pen position as starting point when button is pressed
 SE   3

STAT  3                                    State 3 definition
RESP  PRESS BUTTON WHEN COMPLETE
 ACT  10                                   Branch definition; pressing button leads to state 1
 SE   1
 ACT  7                                    Branch definition, pen movement
 IEX  DLINE                                  DLINE computes and displays fresh line at every pen movement

STAT  4                                    State 4 definition
INIT                                       Initial state, program starts here
PB    PBGO                                 Program block PBGO, executed on entering state 4
 ACT  5                                    Branch definition, category 5 (system)
 SE   1                                      Completion of PBGO leads to state 1

STAT  5                                    State 5 definition
RESP  POINT AT LINE TO DELETE
 ACT  0                                    Branch definition; command "draw" leads to state 1
 MES  DRAW
 SE   1
 ACT  6                                    Branch definition: category 6 (pen hit)
 TEST DLAST                                  Test routine DLAST deletes indicated line
 SE   1                                      If last line, branch to state 1
END
```

TABLE I: The example of Figure 2 coded into Network Definition Language

This bilingual approach permits an interactive program to be created as three separate components, namely the *control component, procedure component* and *supervisor.* The supervisor contains routines for handling interrupts and maintaining the display; at its nucleus is a program which analyses and interprets inputs. This program, the *Reaction Handler,* is basically a table-driven syntax analyser.[5] The tables to which it refers are ring-structures and include a model of the state-diagram, created by compiling the network definition with a *Network Compiler.* These tables form the control component of the program. They contain references to the test routines, program blocks and instructions for execution, which constitute the procedure component and are compiled separately.

The decision to use ring-structures for the Reaction Handler tables was made for a number of reasons. It permitted null-valued entries to be omitted from the tables, and others such as message and response definitions to be of variable length. It meant that a package of routines would be available for building data structures for computer-aided design. It also allowed a much more flexible approach to the design of the Reaction Handler, since the tables could be easily extended or rearranged. The ring-processing package, which has been described in another paper,[7] was based on the ASP language of J. C. Gray.[8] It differs from the earlier ring languages of Roberts[9] and others in the ease with which dynamic alterations can be made to the structure. In particular, elements can be attached to rings or removed from them without altering the element size, since the connections are made by *ring starts* and *associators* which need not be contiguous with the element. This flexibility has made it possible to write an on-line, incremental Network Compiler.

The network compiler

The essence of an incremental compiler, as described by Lock,[10] is that each statement is independently compiled into executable form, and can later be modified without complete recompilation. Normally this means assigning a number to each statement so that it can be referenced. In the Network Compiler it was found sufficient to assign names to the states and branches; individual statements could then be referenced by the property name. Branch naming has since been discarded in order to save space, and it is therefore no longer possible to refer back to individual branch definition statements. Users have not found this to be a great disadvantage.

The first statement in a state or branch definition causes the Network Compiler to set up a corresponding ring element; the two classes of element are called *state elements* and *branch elements*. An extra word is provided in the state element to hold the state name. Each further statement in a definition has the effect of attaching to the element a *definition ring* defining the named property. The manner in which rings define properties is shown diagrammatically in Figure 3. The value of any property of an element can be found by selecting the definition ring with the appropriate *attribute* in the associator, and ascending it to the ring start; the element attached to this ring start contains the value.

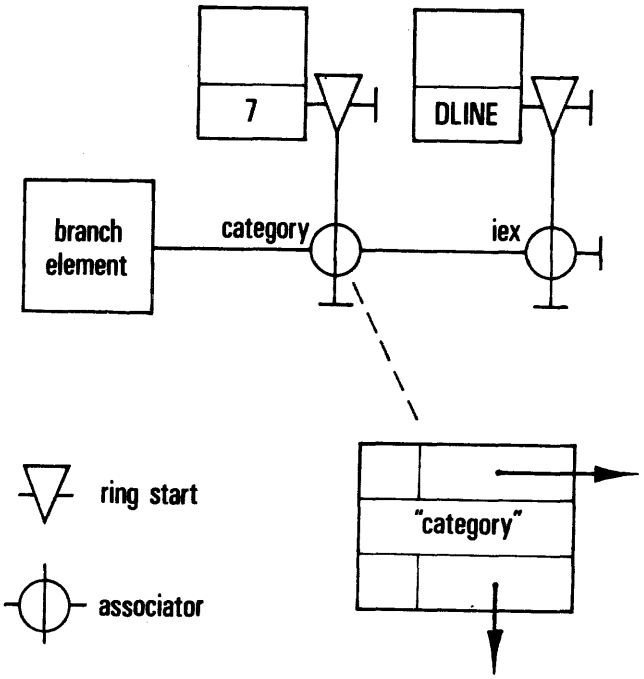


Figure 3—Defining properties by means of definition rings. This shows the ring structure defining a branch of category 7 with an IEX called DLINE. An assocator is shown in detail

Statements defining state entries are treated in the same fashion: the ring to which the branch element is attached leads to the named state element. Each state element has such a *state entry ring*, whose constituent elements define the set of branches leading to this state. A second ring, the *permitted action ring*, starts from each state element, and defines the set of branches leading from that state; this set includes branches which return to the same state and therefore possess no state entry. Figure 4 shows part of the ring-structure resulting from compiling the network definition of Table I.

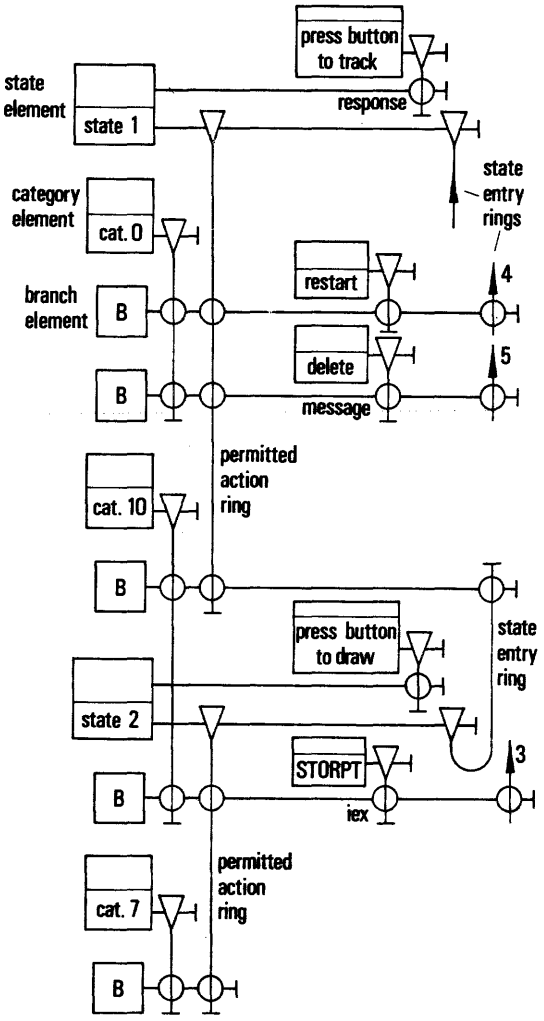


Figure 4—Part of the ring-structure resulting from compiling the network definition of Table I

As mentioned above, the Network Compiler is designed for on-line use, and may be operated from the teleprinter or from the display using light-buttons. The teleprinter has proved the more convenient for on-line compilation, but the light-buttons and displayed responses provide a valuable aid, particularly to the novice. The compiler will also accept paper

tapes prepared off-line. It has error-checking facilities, and will halt at any erroneous command on the tape until the correct version is typed.

The reaction handler: modes

A program under the control of the Reaction Handler may be in one of three modes: these are *interrupted mode, reaction handling* (or *RH*) *mode* and *waiting mode. Waiting mode* does not imply inactivity, but that the program has reached one of the states in the state-diagram and is ready for an action to occur. It may therefore be engaged in computation (user waiting for computer) or looping on a dynamic stop)computer waiting for user).

During waiting mode any action by the user will cause the program to switch to *interrupted mode*, and a *stimulus* to be passed to the Reaction Handler. The stimulus specifies the device involved and may also refer to a block of data or *message,* such as a teleprinter character or a pair of light pen coordinates. Device name and message address are stored in a five-word element which forms the head of a queue of stimuli.

The program then enters *RH mode* and the Reaction Handler processes the first stimulus in the queue. Stimulus processing is a form of syntax analysis, whose goal is to match the stimulus to an entry in the appropriate table. If this goal cannot be reached, the stimulus represents an ungrammatical action. If however the goal is reached, the table may specify a fresh goal to be attained. Eventually the process stops, either when it fails to reach a goal or when it arrives at an *ultimate goal* where no further goal is specified. The Reaction Handler then fetches the next stimulus from the queue and processes it; when the queue is empty the program returns to waiting mode.

At any time the Reaction Handler may be interrupted by a user action, and a further stimulus may be added to the end of the queue. To prevent the queue from growing uncontrollably, software flip-flops or *latches* are used to govern the rate at which each device generates stimuli. The light pen latch, for example, is set when a pen stimulus enters the queue and cleared when it has been processed; while it is set, all fresh pen positions are ignored.

Stimulus processing

The first task of the Reaction Handler on receiving a fresh stimulus is to establish what category of action, if any, the stimulus represents. This it does by referring to a *category table.* Once an action has been recognised in this way, it is matched against descriptions in a *branch table* of the branches leading from the current state. This second phase determines what reaction and change of state should occur, and it is convenient to describe this phase first.

The branch table is in fact the ring structure created by the Network Compiler, as described above and illustrated in Figure 4. The Reaction Handler's first goal is to find a branch of the appropriate category which belongs to the current state. This can be done by searching in parallel the permitted action ring and the category definition ring. If any branch elements belong to both rings, the next goal is to find among them an element whose message definition matches the stimulus message. A series of message comparisons therefore takes place; before carrying out a comparison on a branch element, the test routine is executed.

If a matching element is found, the corresponding reaction takes place: the reaction response is displayed, and the IEX is executed. If no match can be achieved, the Reaction Handler will accept any branch element with a 'null' message. The final goal is the branch's state entry. If none exists, there is no change of state. If on the other hand the branch element is attached to a state entry ring, the new state's response is displayed, and the program block address is used as a transfer address when the program eventually returns to waiting mode. This address may be modified by a *jump displacement* included in the branch definition: this is a method of achieving multiple entries to a program block. If the state has no program block, the program transfers to a standard dynamic stop address.

The most time-consuming part of this process is the parallel ring search, which becomes particularly undesirable when repeated actions such as pen movements are taking place very frequently. The parallel search is therefore avoided by scanning through the permitted action ring at every change of state. At each branch element on the ring an *activity bit* is set which allows a simple search through the category definition ring to be carried out whenever an action of that category occurs.

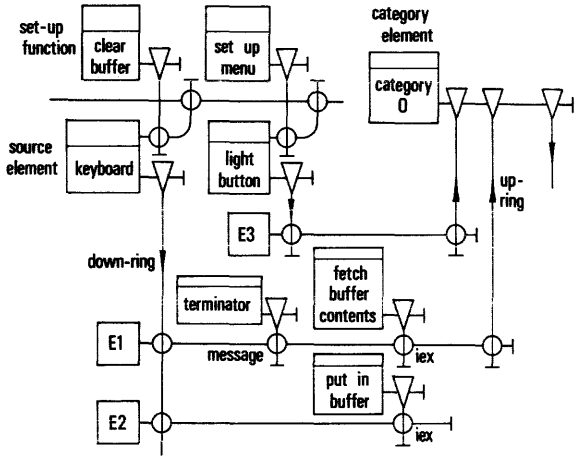


Figure 5 – A category table ring-structure, capable of dealing with typed commands and light-button commands

The first phase of the reaction handling process, in which an input stimulus may be recognized as a particular category of action, is carried out in a similar fashion to the second. During this phase the Reaction Handler uses the *category table*, a fragment of which is shown in Figure 5. When a stimulus is received, it is first matched with a *source element* containing the same device name as the stimulus. A search is then carried out along the *down-ring* from this element, in an identical fashion to the second-phase search along the category ring. If a matching element is found on this ring, its IEX is executed. The next goal is to find an *up-ring* from this element, leading to a *category element*: this ring is treated in the same fashion as the state entry ring in the second phase. The category element forms a link between the category table and the branch table, and the down-ring which starts at this element is in fact the category definition ring used in the second-phase search.

The first phase of reaction handling performs two important functions. It is capable of concatenating stimuli so that the combined input string can be treated as a single action by the second phase; it is also responsible for grouping together actions of the same category. Both functions are illustrated by Figure 5, which depicts the ring-structure for dealing with input commands. Typed commands originate as a string of characters, each of which matches with element E2 and is stored in a buffer. When the terminating character is typed, the IEX of element E1 exchanges the single-character stimulus message for the complete buffer contents, and the second phase of reaction handling commences from the "Category 0" element. Light-button hits produce stimuli containing the complete command as an input message. These match with element E3 and lead immediately to the second phase.

When the program changes state, the old activity bits must be cleared and the new ones set. The Reaction Handler must also carry out various *set-up functions,* defined as properties of the source elements. These functions include such operations as clearing buffers, starting pen-tracking and setting up the light-button 'menu.' Source elements are themselves held on a ring, whose members define the set of active devices. The display is treated as a number of different 'devices': light pen interrupts are separated into light-button hits, tracking interrupts and so forth, and passed to the Reaction Handler under different device names.

In its general layout, the category table closely resembles the branch table, and is set up by a very similar compiler. This *Category Compiler* is also incremental, and accepts table descriptions written in the *Category Definition Language*,[5] which differs only slightly from the Network Definition Language. In general, a complete category table suits most programs, but it is convenient to be able to edit out unwanted categories with the aid of the Category Compiler in order to save space.

CONCLUSION

At the time of writing, the Reaction Handler system has been in use for only a few months. Nevertheless it has during this brief period demonstrated a number of valuable features. In particular, the Network Definition Language provides a very efficient means of writing graphical programs, and simple experiments with graphical techniques can now be carried out in a matter of hours instead of weeks. Both the language and the underlying state-diagram concept are extremely simple, and can be used by those with very little programming experience.

The adoption of a bilingual approach has undoubtedly helped to make this possible, and it is interesting to compare other systems of a similar nature. The use of a separate language to define a program's control sequence has been proposed before, but it is rare to find explicit reference to the need for two languages in interactive programming. The ICES System employs what amounts to a bilingual method, in which a *Command Definition Language* is used to define the control sequence. The language is designed around the use of card-image input, however, and is not particularly suitable for interactive programming. *Command Flow Graphs* are used in a similar fashion to state-diagrams, but the concept of program states is not employed.

A much more powerful facility for treating problem-oriented languages of a very general nature is provided in the AED System.[11] Language syntax can be described by means of the AEDJR Command Language,[12] the extreme generality which this system permits is attractive, but is probably unnecessary in graphical programs. The Command Language is very complex, and its efficient use obviously requires considerable experience.

The processing of basic characters by the AED System is carried out by the RWORD System. This system is particularly interesting, as it employs the concept of representing programs as finite-state automata. It possesses many of the features of the Reaction Handler, but avoids the explicit definition of program states, a feature which has been found valuable in practice. RWORD instead uses a very neat regular-expression language for defining vocabulary words, and avoids the use of tables in order to speed up program execution. It is clearly capable of producing more efficient programs than is possible using the

Reaction Handler's ring-structured category network.

Nevertheless, the Reaction Handler has performed quite satisfactorily as a real-time supervisor. It provides a fast response to all types of user action, including pen movement where a good response is essential. It does so at the expense of a high system overhead, which may reach as much as 20% during pen-tracking. In a display processor, which is idle most of the time, this is quite acceptable.

Less acceptable is the space consumed by the supervisor. The system was developed on an 8K DEC PDP-7 computer and Type 340 display, and in this machine the supervisor occupies nearly 4K. Besides the Reaction Handler, this includes the ring-processing package, a full set of interrupt-handling and output routines, and a software character generator. Some difficulty was experienced in coding the ring-processing routines as pure procedures, due to the lack of index registers on the PDP-7. It seems likely that the size of the supervisor could be greatly reduced by using a machine equipped with index registers and a hardware character generator.

ACKNOWLEDGMENTS

I wish to thank Mr. C. B. Jones for his extensive assistance in programming the system. I am also grateful to Mr. Alan Tritter for suggesting including the test routine; and to many members of staff of the Centre for Computing and Automation, Imperial College, and of the Cambridge University Engineering and Mathematical Laboratories, including Professor W. S. Elliott and Messrs G. F. Coulouris, C. A. Lang and R. J. Pankhurst, for their advice and encouragement.

REFERENCES

1 I E SUTHERLAND
Sketchpad: a man-machine graphical communication system
Proceedings of the 1963 Spring Joint Computer Conference

2 D ROOS
ICES system design
MIT Press Cambridge Massachusetts 1966 p 25

3 C S E PHILLIPS
Networks for real-time programming
Computer Journal Volume 10 May 1967 p 46

4 W R SUTHERLAND
On-line graphical specification of computer procedures
MIT Lincoln Laboratory Technical Report No 405
Lincoln Laboratory Lexington Massachusetts

5 W M NEWMAN
Definition languages for use with the reaction handler
Computer Technology Group Report 67/9 Imperial College London October 1967

6 T E CHEATHAM K SATTLEY
Syntax-directed compiling
Proceedings of the 1964 Spring Joint Computer Conference

7 W M NEWMAN
The ASP-7 ring structure processor
Computer Technology Group Report 67/8 Imperial College London October 1967

8 J C GRAY
Compound data structures for computer aided design: a survey
Proceedings of the ACM 20th Anniversary Conference 1967

9 L G ROBERTS
Graphical communication and control languages
Information System Sciences Spartan Books 1964

10 K LOCK
Structuring programs for multiprogram time-sharing on-line applications
Proceedings of the 1965 Fall Joint Computer Conference

11 D T ROSS
The AED approach to generalized computer-aided design
Proceedings of the ACM 20th Anniversary Conference 1967

12 D T ROSS
AEDJR: An experimental language processor
MIT Electronic Systems Laboratory Memorandum 211, 1964

Automation in the design of asynchronous sequential circuits*

by R. J. SMITH, II, J. H. TRACEY, W. L. SCHOEFFEL and G. K. MAKI
University of Missouri at Rolla
Rolla, Missouri

INTRODUCTION

Sequential switching circuits are commonly classified as being either synchronous or asynchronous. Clock pulses synchronize the operations of the synchronous circuit. The operation of an asynchronous circuit is usually assumed to be independent of such clocks. The operating speed of an asynchronous circuit is thus limited only by basic device speed. One disadvantage of asynchronous circuit design has been the complexity of the synthesis procedures for large circuits.

This paper describes a computer program[1,2] which automatically generates the complete set of design equations for asynchronous sequential circuits. Many of the algorithms employed are new and have been shown to be much more practical than classical techniques for the synthesis of large circuits.

Minimum or near-minimum variable internal state assignments are generated using two of the Tracey algorithms.[3] An evaluation procedure predicts which of several codes generated will most likely yield the least complex design equations. Next-state equations, including don't-cares, are then produced without constructing transition tables. Output-state equations are also generated. Finally, simplified normal form design equations containing no static hazards are produced. The program is capable of designing circuits much too large to design manually.

The operation of a sequential circuit is often described by means of a flow table.[4] An example is shown in Figure 1. The columns of a flow table represent input states, while the rows represent internal states assumed by the circuit. Each flow table entry specifies the next internal state and output which result from the given input and internal states. When the next-state equals the present state, that state is said to be stable and is customarily circled. Unstable states correspond to transitions within a flow table column.

	I_1	I_2	I_3
1	①/10	-	4
2	3	4	②/01
3	③/01	③/00	-
4	1	④/11	④/00
5	1	⑤/10	2

Figure 1 – Flow table

A sequential circuit is operating in fundamental mode if the inputs are never changed unless the circuit is stable internally. If, in addition, each unstable state leads directly to a stable state, the circuit is said to be operating in normal fundamental mode. The computer program described in this paper automatically generates design equations for asynchronous sequential circuits operating in the normal fundamental mode. Circuit specifications are conveniently input in flow table form. Input state binary codes are specified by the program user. A summary flow chart of the procedure followed by the synthesis algorithm is shown in Figure 2.

*The research reported in this paper was supported in part by the National Science Foundation through Grant GK-820.

The program used to implement the design procedure suggested above is written in PL/1. This language was chosen because of its bit-string data format, Boolean operations, and the controllable storage feature. The program consists of about 2000 PL/1 statements divided into 8 subroutines. The system was designed to run on an IBM 360/40 but could be run on a somewhat smaller machine.

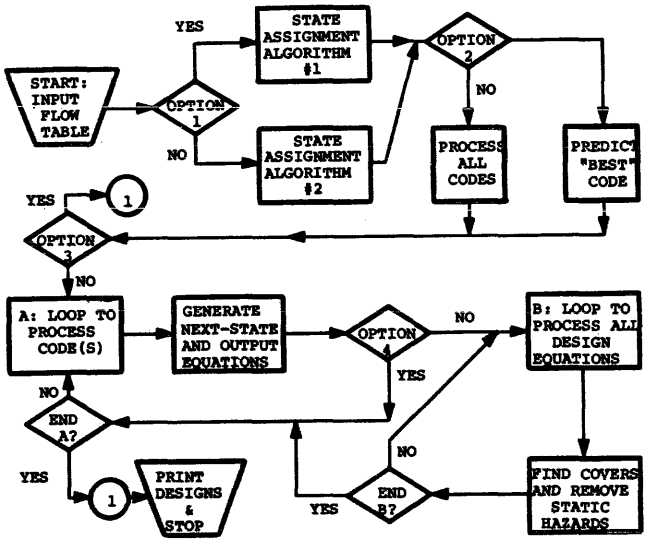


Figure 2 – The programmed synthesis algorithm

State assignment algorithm

The internal state assignment procedures employed are a modification of those described by Tracey.[3] Either completely or partially simplified flow tables may be input to the program. Flow table simplification is not presently included in the synthesis procedure, but will be included in a later version of the program.

The Tracey state assignment algorithms are based on the following theorem which is reproduced without proof. "A row assignment to a flow table which allots one internal state per row is satisfactory for the realization of normal fundamental mode flow tables without critical races if and only if for every transition (S_i, S_j), a) if (S_m, S_n) is another transition in the same column, then at least one internal state variable partitions the pair $\{S_i, S_j\}$ and the pair $\{S_m, S_n\}$ into separate blocks; b) if S_k is a stable state not involved in any transitions in the column, then at least one internal state variable partitions the pair $\{S_i, S_j\}$ and the state S_k into separate blocks; and c) for $i \neq j$, S_i and S_j are in separate blocks of at least one internal state variable partition."

Constraints generated as a result of applying the above theorem may be listed in Boolean matrix form, with each row corresponding to a partially specified state variable. Consider, for example, the constraint list generated by the flow table of Figure 1. The constraints shown below are generated on a per-column basis in satisfying theorem parts a) and b):

Constraint List	Boolean Matrix 12345
(14; 23)	0110-
(15; 23)	011-0
(3; 24)	-101-
(24; 5)	-0-01
(25; 14)	10-10
(1; 4)	0--1-

The program forms all partitions associated with the topmost stable state of column 1, then all irredundant partitions due to the second state, and continues until all stable states in the column have been examined. The process is then repeated for the remaining columns.

Note that for the above example, none of the column-generated constraints partitioned flow table rows 1 and 4; the constraint (1;4) was thus included in the list to satisfy theorem part c). The program checks all pairs of flow table rows, and generates additional partitions as required by c).

The state assignment problem now becomes one of finding a minimum number of internal state variables satisfying all of the constraints just generated. This problem has been shown to be analogous to the generation of maximal compatibles by the Paull-Unger algorithms for flow table simplification. A set of completely specified state variables, at least one of which covers each constraint, corresponds to the maximal compatibles. These completely specified state variables will be referred to here, therefore, as maximal constraints. The selection of a minimum number of maximal constraints, and hence minimum number of internal state variables is similar to the covering problem in the Quine-McCluskey method for simplifying Boolean equations. Details of these algorithms are available in the literature and will not be given here.[3,5,6]

As in the Paull-Unger Method A,[5] maximal constraint generation may begin with the assumption that no constraints exist. Then each constraint is examined for contradictions to this assumption and all implied partitions are generated. After each constraint has been so examined, one is left with the set of maximal constraints. In another approach, Paull-Unger Method B, one begins with the list of constraints and seeks

to enlarge each through the complete specification of the corresponding state variable until all enlargements are found that cover one or more of the original constraints. Both procedures were programmed and the second was found to be nearly a factor of two faster than the first in this application.

As stated previously, the selecting of a minimum number of maximal constraints is similar to the covering problem in Boolean equation simplification. A branching method has been used which is capable of producing all irredundant minimum-variable assignments. Operating speed of this algorithm is increased by reorganization of the maximal constraint list, based on the idea that those maximal constraints including the largest number of the original constraints would most likely be members of a minimal cover. Internal representation of each maximal constraint is restructured in such a manner that the covering problem cannot be further simplified using column dominance techniques.

It is obvious that either of the two assignments below satisfy all of the constraints shown above:

Assignment #1						Assignment #2					
	1	2	3	4	5		1	2	3	4	5
y_1	0	1	1	0	0	y_1	0	1	1	0	1
y_2	0	1	1	0	1	y_2	0	1	1	1	0
y_3	0	1	0	1	0	y_3	0	1	0	1	0

Furthermore, these are the only two significantly different minimum assignments which successfully code the Figure 1 flow table.

It has been found that even with certain look-ahead provisions in the branching routine, generation of minimum variable assignments becomes a time-consuming problem for typical flow tables of 12 rows or more. A second and much faster algorithm has been programmed. It is an approximate method, and generates near-minimum variable codes.

The fast algorithm reduces the Boolean matrix corresponding to the maximal constraints through the use of an approximate reduction technique. A constraint is constructed which seems to include a large number of matrix rows. The included matrix rows are then removed. This process is then continued until all rows of the original matrix are included in at least one of the generated constraints. This reduced matrix corresponds to a near-minimum variable state assignment.

The fast Boolean matrix reduction program usually produces satisfactory assignments having less than 1⅓ times the minimum number of variables. Assignment generation times for large flow tables may be reduced by two orders of magnitude using this approximate procedure. Near-minimal assignments have been efficiently generated for flow tables having up to 75 specified next-state entries and 150 constraints with approximately 15 minutes computer time on an IBM 360/50. Many satisfactory assignments are often generated. One of these may be selected by a test routine or chosen by the designer. The test routine, to be discussed below, chooses a "good" assignment for reduced hardware realization.

Design equations

As the example above illustrates, the code generating algorithms frequently produce several satisfactory assignments. Generated codes may be evaluated by a procedure due to Maki,[7] which selects that assignment most likely to have simple next-state equations. Consider, for example, the assignments and next-state equations shown in Figure 3. Note that the next-state equations for column I_1 Assignment #1 are much less complex than those for Assignment #2.

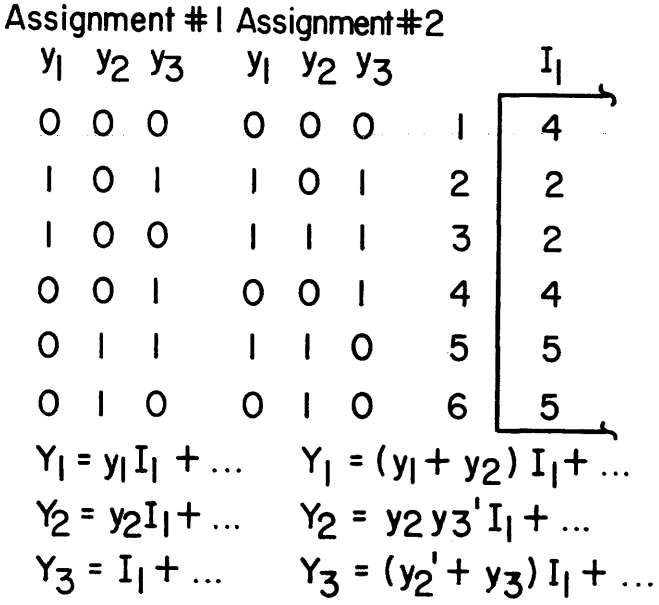


Figure 3 – Partial flow table and assignments

The test procedure searches for that assignment with a maximum amount of reduced dependency in the next-state equations. Two types of reduced dependency are easily detected from the assignment. First, observe that Y_3 is dependent only on the input in the given example. This can be predicted by noting that y_3 has the same value, 1, for all stable states in the column. A second observation is that Y_1 is dependent only on the input and the present state of y_1. Similarly, Y_2 is a function only of y_2 and the input. This can also be predicted by simply noting that y_1 and y_2 are never excited to change state for any transition under input I_1. In other words, one need simply observe that y_1

and y_2 have the same value for state pair (1, 4), state pair (2, 3) and again for state pair (5, 6). Observe the increased complexity of the next-state variables in Assignment #2 of Figure 3 as a result of its failure to insure reduced dependency. The programmed routine based on this method will evaluate each generated state assignment for reduced dependency in just a few seconds.

Maki has also described a procedure for obtaining next-state equations without construction of the traditional excitation matrix.[7] An algorithm derived from his method is presented here.

Each internal state transition may be associated with a p-subcube of the n-cube defined by the input and internal state variables. Furthermore, all of the next-state entries of p-subcubes associated with a single stable state will be identical, and equal to the row code of the stable state. Consider, for example, the application of Assignment 1 to Column I_1 of Figure 3, as shown in Figure 4.

In the transition between rows 2 and 3, all states in the p-subcube y_1y_2 ($y_1 = 1$, $y_2 = 1$) must have the same next-state entries, namely that of stable state 3, 110. A tabular form of p-subcube generation may be illustrated as follows:

y_1	y_2	y_3	
1	1	0	③ Stable Row Code
1	1	1	3 Unstable Row Code
1	1	-	P-Subcube Resulting from Transition

The transitions from rows 4 and 5 to stable state 1 define the remaining two p-subcubes listed in P_{I_1} of Figure 3. Note that the Boolean sum P_{I_1} of these terms represent all next states requiring specified entries under input I_1.

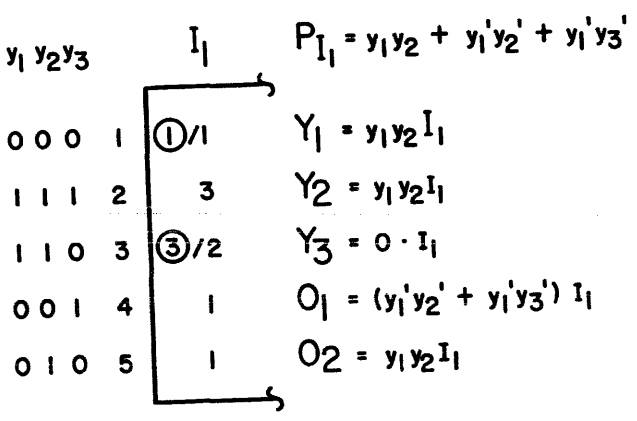


Figure 4 – Partial flow table, specified p-subcubes and 1-sets

Notice that if y_i is 1 in the stable state row code, then next-state variable $Y_i = 1$ for *all* states in p-subcubes associated with that stable state. For example, since in row 3 $y_1 = 1$, all states in the p-subcube y_1y_2 will have next-state variable $Y_1 = 1$. In other words, all p-subcubes associated with transitions to a stable state will appear in the Boolean sum-of-products next-state equation Y_i *if digit i of the stable row code is one.*

As the p-subcubes are generated by the computer program, they are added to the appropriate next-state 1-set lists only if the corresponding next-state variable is 1 in the subcube (see Figure 3). The final results are (partially simplified) Boolean equations representing the 1-cells of the next-state variables.

The synthesis program also generates the output equations of the sequential circuits. The output corresponding to a given stable state is also associated with all unstable states leading to the stable state. All p-subcubes generated previously are grouped according to stable state. If an output variable is 1 for a particular stable state, the associated p-subcubes become a partial list of 1-sets under the corresponding input. The output 1-sets for Column I_1 in Figure 1 are shown as sums in Figure 3.

To permit further simplification of the design equations generated above, it is desirable to compute the unspecified entries for all equations. Fortunately, unspecified p-subcubes are common to all the design equations. A Boolean equation for don't-care entries is generated by simply taking the complement of the available equation for specified entries (see Equation P_{I_1} in Figure 3).

Complementation of a Boolean sum-of-products expression may be performed by complementing the expression, multiplying out the result, then simplifying the resultant sum-of-products expression to obtain the solution. The procedure used here is a modification of that method. Simplification illustrated by

$$A\cdot(A + B + C) = A \qquad \text{and}$$
$$A\cdot(A' + D + E) = A(D + E)$$

is performed both before and during the multiplication of the product-of-sums expression. Redundant terms are also deleted. A brief example will perhaps illustrate the method employed. Figure 5 shows complementation of the sum of p-subcubes shown as P_{I_1} in Figure 3.

A normal-form Boolean equation for each next-state variable may be obtained by combining the don't-care terms found above with the appropriate next-state 1-sets. Since the output associated with don't-care internal states may be assumed to be unspecified, the output-state equations also include the same don't-care terms.

$$P_{I_1} = y_1y_2 + y_1'y_2' + y_1'y_3'$$
$$P'_{I_1} = (y_1' + y_2') \cdot (y_1 + y_2) \cdot (y_1 + y_3)$$
$$= y_1(y_2') + y_2 + y_3 \cdot (y_1' + y_2') \cdot (y_1 + y_2)$$
$$= y_1y_2' + y_1 + (y_1'y_3 + y_2'y_3) \cdot (y_1 + y_2)$$
$$= y_1 + y_1'y_2y_3$$

Figure 5 – P-subcube complementation and simplification

The program then finds prime implicants of each design equation produced above. A conventional consensus algorithm is used and will not be presented here.

A covering algorithm is used to find simplified, but not necessarily minimal design equations. Instead of covering the 1-cells of a design equation the program covers the 1-*sets* originally generated from flow table columns. (Recall that a 1-set is a subcube containing one or more vertices or 1-cells for which the expression is 1.) The problem of generating and covering a large number of 1-cells is thus avoided. More importantly, it can easily be shown that by covering the 1-*sets*, all static hazards associated with vertical flow table transitions are eliminated from combinational circuit outputs. A static hazard exists when there is a transition between a pair of adjacent states having the same output, during which it is possible for a momentary improper output level to occur. Using two-level AND-OR synthesis, if each product (prime implicant) covers only 1-sets, all transitions within that 1-set are static-hazard-free; static hazards may only be caused by input-state changes which correspond to horizontal transitions on the flow table.

A procedure for eliminating the remaining "horizontal" hazards has been included. It is based on the restriction that only one input-state variable at a time changes. All pairwise combinations of a design equation's products (prime implicants) are examined for horizontally adjacent 1-sets. If such an adjacency is found, a static hazard exists. Since a horizontal transition may only originate at a stable state, the static hazard cannot possibly cause a malfunction unless one of the 1-sets includes a stable state.

Consider, for example, the illustration shown below, which is the simplified design equation for Y_2 of Figure 1, using Assignment 1:

$$Y_2 = y_3'v'w + y_2v + y_1w'$$

(where the input variables are v w)

Note that the horizontally adjacent 1-sets $y_3'v'w$ and y_1w' appear as the first and third terms. If any stable state has a code in the subcube $y_1y_3'v'$ then a static hazard exists which may cause a malfunction. Note that stable state 3 in columns I_1 and I_2 both satisfy

$$Y_2 = y_3'v'w + y_2v + y_1w' + y_1y_3'v'.$$

Program performance

Execution times obtained using the program described here depend on hardware and software efficiencies, as well as the complexity of the input flow table. The solution times stated here were obtained using an IBM 360/50 computer and the IBM Release 13 PL/1 Compiler.

Simplified design equations for flow tables of 6 rows by 4 columns (24 cells) have been produced in 45 seconds to 4 minutes, depending on problem complexity. Eight row by 4 column tables usually are solved in 1.5 to 8 minutes. Three assignments for a 12 × 4 (48 cell) flow table have been produced in about 8 minutes, with next-state equations (unsimplified) generated in 3 minutes per assignment. Two satisfactory codes for a 18 × 4 (72 cell) flow table were found in 15 minutes. Computation times for large problems have been found to be extremely problem-dependent.

SUMMARY

A description of a programmed algorithm for the synthesis of normal fundamental mode sequential circuits has been presented. The program permits the logic designer to input his asynchronous sequential circuit specifications in the form of a flow table and obtain all next-state equations and output equations in the form of simplified sum-of-products. Two internal state assignment algorithms are available to the designer. One will generate a minimum-variable assignment but may be lengthy to execute while the other will execute much faster but guarantees only a near-minimum variable solution. A testing routine is then available to aid the designer in deciding which of several satisfactory state assignments will tend to reduce the complexity of the design equations. A "good" assignment will be selected and simplified next-state and output equations will be generated based on the selected assignment. The complete program has been written in PL/1 and is running on an IBM 360/50 computer at the University of Missouri at Rolla. Flow tables with up to 75 specified next-state entries have already been run and much larger flow tables will soon be generated for experimentation purposes.

REFERENCES

1 R J SMITH II
A programmed synthesis procedure for asynchronous sequential circuits

Masters Thesis University of Missouri at Rolla 1967
2 W L SCHOEFFEL
Programmed state assignment algorithms for asnychronous sequential machines
Masters Thesis University of Missouri at Rolla 1967
3 J H TRACEY
Internal state assignments for asynchronous sequential machines
IEEE Transactions on Electronic Computers Volume EC-15 pp 551-560 August 1966
4 D A HUFFMAN
The synthesis of sequential switching circuits
Journal of the Franklin Institute vol 257 pp 151-190 and 275-303 March and April 1954
5 M C PAULL S H UNGER
Minimizing the number of states in incompletely specified sequential switching functions
IRE Transactions on Electronic Computers vol EC-8 pp 356-367 September 1959
6 E J MC CLUSKEY
Minimization of Boolean functions
Bell System Technical Journal pp 1417-1443 November 1956
7 G K MAKI
Minimization and generation of next-state expressions for asynchronous sequential circuits
Masters Thesis University of Missouri at Rolla 1967

Interpretation of organic chemical formulas by computer *

by ALBERT N. DeMOTT
Computer Research
Rockville, Maryland

INTRODUCTION

Over the last few years, a frequently discussed problem in the area of chemical information systems has been the need for some means by which chemists could communicate with the system in terms of their normal chemical language, the structural formula, rather than requiring them to use special, machine-oriented notations. The Walter Reed Army Institute of Research (WRAIR), as part of its Chemical Structures Storage and Retrieval System,[1] has developed an economical and effective computer program to analyze structural formulas as normally written by chemists, producing as output a detailed description, in machine-oriented format, of the atoms in the molecule and their connections to each other. In principle, any trained chemist can prepare compounds for entry in the system master file, or questions for searching it, without any special training in the WRAIR system. The program is now being used in daily operations and is, we believe, the only operational program capable of performing this function without major restrictions on the formulas which can be accepted. As a special case of the general problem of the man-machine interface, the program may well be of interest outside the chemical field, particularly since many of the techniques used have no essential relation to chemistry.

The operational cost of this facility compares favorably with the cost of preparing the connection tables,[2] fragment lists, systematic names,[3] or other special notations[4] required by many chemical retrieval systems. Execution time on an IBM 7094 computer averages about five to seven minutes per thousand compounds. In the environment in which the program is currently operating, preparation of input to the program requires one and a half to two minutes of clerical time per compound, and about five minutes of 7094 time per thousand compounds. The program accepts well over 95% of the chemically-correct structures presented to it, and the accuracy of interpretation of those accepted (excluding compounds for which warning messages are issued) closely approaches 100%. The only limitations on the freedom of the chemist in writing formulas are the following: (1) Organic, rather than inorganic, chemical conventions must be followed where the two systems differ. (2) The structural formula must be given in enough detail to resolve any ambiguities which might normally be resolved by the context of discussion. (3) A few specialized types of compounds (such as polymers, coordination compounds, and stereo isomers) cannot be handled. (4) In a few cases of variant usage, the chemist is restricted to one of the options normally open to him; in general, however, the program will handle all or most of the conventions commonly used.

Background

The number of known organic compounds has increased rapidly over the last ten or fifteen years, creating a critical need for rapid means of retrieving information about compounds related to the compound a chemist may be currently studying. For example, an urgent problem in the medical field, at present, is to find new anti-malarial drugs which will be effective against the drug-resistant strains of the malaria parasite which have appeared recently in southeast Asia. When a research worker finds a compound with some effectiveness in treating the disease, his first need is to obtain information about the biological activity of known related compounds, as a guide to determining what modifications to the molecule might offer promise of increasing the activity of his potential drug. A manual search of files containing several hundred thousand compounds is impractical, no matter how well cross indexed they may be, since the portion of a molecule which is relevant for one search is likely to be irrelevant for nearly all others. In the example just cited, in fact, the question of which portion of the molecule is relevant

*This paper is contribution No. 330 from the Army Research Program on Malaria.

is precisely the question the search is intended to help answer.

To solve this problem, WRAIR has developed a computerized storage and retrieval system which allows the user to specify any chemically valid structure or portion of a structure. The system will then retrieve all compounds in the file which contain that structure as a part of their molecules. Retrieval is on the basis of a successful mapping of the structure of the question into the structure of the compound on file. In order to permit such a mapping the file entries must contain, and input must provide, a specification of the characteristics of each atom in the molecule and a specification of all the pairs of atoms which are bonded directly to each other (with the nature of the bond given in each case). On the other hand, the structures of new compounds for the file, whether obtained from published catalogues, submitted by the chemists who have synthesized them, or gotten from some other source, will nearly always be specified originally by means of conventional chemical formulas. The formulas can be, and in the past have been, converted to an atom-by-atom and bond-by-bond format by manual methods, but this is a tedious task which must be performed by trained chemists. Furthermore, the original formula is lost in this process and is not available at retrieval time. The program which is the subject of this paper was created to bridge this gap. The preparation of input can be entrusted to relatively unskilled clerical personnel, since their only task is to copy the chemist's original drawing. Furthermore, since the original formula is provided to the system in a binary coded form, it can be conveniently included in the master file entry for the compound and printed on a line printer at retrieval time. It can also be provided to the user during initial processing in conjunction with rejection messages and warnings of ambiguities and suspected errors.

In the current operational environment of the program, input is prepared by typing on a chemical typewriter.[5] The paper tape output from the typewriter is converted to magnetic tape and processed by computer into line-by-line order. Output from the program consists of a fairly conventional connection list. The details of this input and output are beyond the scope of this paper, however, since they do not affect the logic of the program. Relatively minor coding changes would suffice, in fact, to provide output in other formats (such as a connectivity matrix) or to accept input prepared in other ways, provided the input represents a line-by-line image of the formula and preserves the original geometric relations.

The problem

Organic chemical formulas are a language which has grown up over the past 100 years with little attempt at standardization. Its "rules of grammar" have never been codified, and must be deduced from the actual practice of chemists. In principle, a formula is a conventionalized picture of a molecule as projected on a plane, but the emphasis must be on the word "conventionalized." Each atom is represented by an element symbol consisting of one or two letters, and the connections between atoms are indicated by straight lines (single, double, or triple according to the nature of the bond) connecting two element symbols. In practice, however, the name "structural formula" is misleading. Only the major outlines of structure are shown by means of bond lines—the details must be inferred by the reader. For example, the characters "SO_2" in an organic (but not in an inorganic) formula mean that two oxygens are each double bonded to a sulfur atom and that the sulfur atom in turn is single bonded to each of two other atoms in the molecule. (The bond lines for the latter bonding may or may not be written.) If you ask a chemist why "SO_2" represents this structure and no other, the answer will be, in substance, "Because it does." Chemically it is perfectly possible for two oxygens to be single bonded to a sulfur atom with each oxygen single bonded in turn to some other atom, and such structures do in fact occur. They are never, however, represented by "SO_2." Most "structural" formulas include lengthy strings of element symbols, subscripts, parentheses, and brackets. Their structure is obvious to a chemist, but not at all apparent to a layman.

The problem of interpreting chemical formulas is therefore twofold: First, the program must be able to trace the chains and rings formed by bond lines, occasionally in extensive patterns resembling chicken-wire. Second, it must be able to determine the structures implied by strings of symbols which give no explicit indication of the mutual relations of the atoms represented.

Basic procedure of the program

Input to the program consists of structural formulas whose individual characters have been arranged in line-by-line order, including all blanks within each line. One formula is read in, stored in a two-dimensional matrix, and a starting node is chosen arbitrarily. For the purposes of the program, a node may be a string of symbols, but for simplicity let us assume that all nodes in the structure are single atoms with or without attached hydrogens. The atoms and its characteristics (including the number

of attached hydrogens, if any) are recorded. A dot at the corner of a ring structure is interpreted to represent a carbon atom with enough attached hydrogens to make up its full valence of four. Next, all adjacent matrix cells are checked for bonds pointing to the node. Each bond found is traced and the matrix location of the atom at its far end is entered in a table of unprocessed nodes. This entry also records the nature of the bond and identifies the atom at the node currently being processed. The location of the node table entry is then stored in the matrix cell at the far end of the bond.

When all bonds pointing to the node have been traced, the valence of the atom at the node is checked to make sure it agrees with the total of the bondings shown. A new starting point is then chosen by taking an entry from the node table, and the new node is processed in the same manner. If a matrix cell containing a bond pointing to the node is found to have a node table reference in it, the information in the entry is used to record the bonding between the atoms at the two nodes. The table entry is then erased. Processing of the molecule is complete when no entries remain in the node table.

Interpretation of strings of symbols

When a node consists of a string of element symbols, subscripts, brackets, and parentheses, the problem becomes vastly more complicated. The bulk of the coding in the program is devoted to handling this problem.

Two basic approaches to the problem of interpreting such strings were considered in designing the program. The first approach would be to analyze strings entirely by program logic, taking each element symbol separately and inferring the relation of its atoms to the other atoms in the string. In terms of an analogy with natural languages, it represents interpreting a sentence word by word, allowing for all the changes in meaning of a given word which can be produced by changes in context, and for the variation in the relation between two words which results from changes in their relative positions and the presence or absence of other words in the sentence. In the case of chemical formulas, one of the major difficulties in this approach is the fact that most chemical elements can take on any one of several different valences (i.e., have different number of bonds).

A second approach to interpreting strings would be, using the linguistic analogy, to analyze the sentence in terms of phrases instead of individual words. Chemically, this would mean defining a set of glyphs (i.e., groups of element symbols and auxiliary characters), each of which would represent one and only one arrangement of atoms and bonds. Many such glyphs can, in fact, be defined, and the approach offers obvious advantages from the standpoint of simplicity of programming. The approach was used with considerable success by E. B. Gasser and C. W. Gregory at Colgate-Palmotive Company in designing and implementing for a small computer an experimental predecessor to the present program.

The final decision, however, was in favor of using program logic exclusively, and experience has confirmed that the decision was a good one. First, it was found that a number of lengthy glyphs would need to be defined, with many glyphs being subsets of longer ones. This would require lengthy table searches and repetitious processing of element symbols as overlapping fields were tested successively against the glyph table. Program execution would be slow. Second, the number of glyphs to be defined would be large (probably on the order of 1,000), and few would be used by chemists with absolute consistency. To require a chemist to consult such a glyph list to ensure that his structure would be interpreted correctly would defeat the primary goal of the program, and would open wide opportunities for errors. Third, and most vital, a study of a set of representative formulas led to the conclusion that it would in fact be possible to abstract a set of rules simple enough to be practical from a programming standpoint, and universal enough to insure reliable operation of the program. Furthermore, it appeared possible to define criteria for identifying genuinely ambiguous formulas and either rejecting them or issuing a warning to allow the chemist to check the program's interpretation. The flexibility of the program logic approach appeared to be more valuable than the definiteness of the glyph approach.

The original set of rules turned out, not unexpectedly, to be thoroughly inadequate; but progressive refinement as problems became apparent has produced, with no changes in the basic logic, a program which comes very close to meeting the goals originally defined. At present, two typewritten pages are sufficient to specify the conventions which chemists must observe in writing formulas for input to the system.

A complete description of the rules used to interpret strings of element symbols is beyond the scope of this paper, but the basic principles are as follows:

1. Since Western languages are written from left to right (and most chemists are Westerners) strings are usually written, and can usually be analyzed, from left to right.

2. The bondings on each atom will exactly equal one of its normal valences, unless another valence

has been specified in the formula. Except for oxygen groups (see rule 5 below), the valence will be the lowest compatible with the valences of surrounding atoms.

3. A string, since it resembles a chain in appearance, will normally represent a chain structure (with or without side branches) and, except in connection with oxygen groups, it will not contain any rings. A straight chain should be preferred over a chain with branches, where both are possible.

4. Each string represents a single molecule, or portion of a molecule, and each atom in the string must be bonded directly or indirectly to every other atom in the string.

5. Oxygen, particularly subscripted oxygen, is most likely to be bonded as a side atom, rather than as part of the main chain, even if this requires assigning the atom to which it is bonded a valence higher than its lowest normal valence.

6. Triple bonds are rare, and a pattern of one double and one single bond is preferred over a triple bond.

In processing a string, the general procedure is to take each atom in turn (treating subscripted symbols other than oxygen as if an equivalent number of symbols had been written side by side). First, any written bonds approaching the atom vertically or from the left are traced and their value is subtracted from the valence of the atom. (Bondings are made or entries added to the bond table in the same way as for structures shown in full detail.) Next, the valences of the atom are used to satisfy all unsatisfied valences remaining on previous atoms in the string, unless all atoms in the string so far have the same valence and no written bonds are present on any of them. In the latter case, the atom will be left unbonded. Last, any bonds approaching the atom from the right will be used to satisfy valences on whatever atom still has unsatisfied valences. This will not necessarily be the atom being processed, but may well be an earlier one. The program then requires that unsatisfied valences be left on at least one atom in the string, unless the end of the string has been reached. In the latter case, all valences must be satisfied. If at any stage of processing the valences on the atoms are such that the above rules cannot be followed, the valence of one of the atoms involved is raised to its next higher value.

In addition to the main processing routine described above, three special routines are provided to deal with (1) oxygen groups, (2) alkane chains of the form C_nH_m, and (3) groups inverted from their natural order becuase they occur at the left end of a string.

Parentheses and brackets

Although several special usages occur, and are provided for by the program, brackets and parentheses are most often used in one of two ways: (1) The parenthetic group may represent one or more branches from the main chain of the string. This is referred to as "fanwise bonding." If the parentheses carry a subscript, all the groups represented will be bonded to an atom or atoms in the main chain. If an oxygen group precedes, each parenthetic group will be bonded to a different atom within the oxygen group. Otherwise, all groups must be bonded to the same atom. (2) The group may represent a unit which is repeated as part of the main chain. This is called "chain bonding." The groups are bonded to each other in a chain, with the first group bonded to a preceding atom in the string and the last group bonded to an atom which follows.

The two usages are distinguished by counting what might be called "handles" on the group. If the group has only one handle, it is bonded fanwise. If it has two handles it is chained. Handles are counted by first processing the group as if it were a string in itself, reserving one valence on the first atom in the group if the group is not at the beginning of the whole string. A handle is then defined as (1) a reserved valence, (2) unsatisfied valences on any one atom in the group, or (3) a written bond extending to some atom outside the group.

Parentheses and small brackets are expanded and processed when the end of the group is reached in normal processing of the string. Large brackets (which may enclose substructures rather than single strings) are processed when interpretation of the structures within them has been completed.

SUMMARY

The program described in this paper meets a need which has been recognized for a number of years, by allowing communication between chemists and computers in terms familiar to all trained chemists. Certain limitations still exist, but our experience has been that when a formula must be rewritten to meet these limitations the result is nearly always a formula which chemists consider better chemically. Since these same formulas are used as part of the output for searches, this can be a distinct advantage. The program is economical in operation, and some two years of use have shown it to be reliable and subject to progressive refinement.

ACKNOWLEDGMENTS

All work on this program has been supported by the

U. S. Army Medical Research and Development Command.

The original design work was performed by the author at the Service Bureau Corporation and was initially implemented by him and by other employees of the Corporation working under his supervision. Some modifications were made by the author while at Computer Applications Incorporated, and a major revision was carried out by him at Computer Research.

REFERENCES

1 D P JACOBUS D E DAVIDSON A P FELDMAN J A SCHAFER
Experience with the mechanized chemical and biological information retrieval system
Presented before the Division of Chemical Literature American Chemical Society Chicago Illinois September 1967

2 W S HOFFMAN
An integrated chemical structure storage and search system operating at Du Pont
Presented before the Division of Chemical Literature American Chemical Society Chicago Illinois September 1967

3 G G VANDERSTOUW I NAZNITSKY J E RUSH
Procedures for converting systematic names of organic compounds into atom-bond connection tables
Journal of Chemical Documentation vol 7 no 3 1967

4 E HYDE F W MATTHEWS L H THOMSON
Conversion of Wiswesser notation to a connectivity matrix for organic compounds
Presented before the Division of Chemical Literature American Chemical Society Miami Beach Florida April 1967

5 A FELDMAN D B HOLLAND D P JACOBUS
Automatic encoding of chemical structures
Journal of Chemical Documentation vol 3 no 4 1963

A simulation in plant ecology

by RAYMOND E. BOCHE
Texas Technological College
Lubbock, Texas

INTRODUCTION

The purpose of this paper is to present some results from a preliminary study investigating the application of the computer sciences to problems in plant ecology. Results include a model which simulates the growth of a forest in a particular time dependent environment and an implementation of that model using a digital computer and assumed data. At present the model is somewhat restricted in level of detail and range of applicability. It is, however, believed to be a pioneer in plant sciences, and further study will surely suggest directions for refinement in level of detail. The range of applicability is constrained by factors not modeled to a young, growing forest of natural occurrence in a particular environment. Validity is anticipated, not for individual trees, but for the entire forest presented as an ecological system.

Purpose and scope

The general purpose of the model described here is to investigate the feasibility of computer simulation of plant growth processes. The specific model developed devotes unusual attention to adaptability and flexibility in order to provide ready means of incorporating improvements and modifications suggested by plant physiologists or plant ecologists. This approach recognizes the strongly qualitative and descriptive aspects of these sciences and makes allowance for the shortage of quantitative knowledge and relationships required by the model.

The specific model, a young, growing forest of natural occurrence in a particular environment, will allow us to predict changes in composition of a forest as influenced by the ecological interactions. The model includes and accounts for three of the five principal limiting factors in plant ecology, light, temperature, and moisture. The exclusion from consideration of the remaining two factors, soil fertility and soil type, limits the application of the model to forests of natural origin; thus assuring some degree of soil type compatibility for species present. Excluding soil fertility limits the scope of the model to relatively short periods of time during which soil fertility remains relatively constant and uniform in its effect on species present.

It should be emphasized that the model is for growth in a natural environment; consequently, it does not at present encompass such considerations as probability of seedlings started, human intervention, or other extraordinary influences such as fire.

Feasibility

Before embarking on this project, it is appropriate to investigate its feasibility. Toward that end, it would seem that three major areas must prove amenable to computation if the model is to satisfy our general purpose.

First, growth must be predictable from environment.
Given a plant of known heredity and of known previous history, it is possible, by applying the principles of plant physiology, to predict with considerable certainty the physiological reactions which will be evoked in that plant upon its exposure to a given complex of environmental conditions.

Second, ecological interaction must be predictable given a particular flora subjected to a particular environment.
Ecological measurement has been sufficiently perfected to give material aid in predicting the hazards to be encountered in critical areas under various types of land use and management.

Third, the interactions must be "modelable" and "computable."
Important in plant ecology is the principle of limiting factors, which says in effect that the least favorable of conditions present will prove epistatic. In particular, photosynthisis or plant metabolism, will be controlled by the least favorable of soil fertility, soil type, light, temperature, and moisture.

The three above definitive statements were abstracted from the Encyclopaedia Britannica and, together, give strong indications of feasibility for the proposed study.

Basic model structure

1. General flow

A forest, for our purposes, will be defined as two or more established trees that interact ecologically with one another. The first step in the model is to initialize a particular forest (thru observation or assumed data) and measure its initial composition.

The second step is to apply an amount of growth resources (temperature, light, and moisture) determined as a result of a simulated period of climate.

At each time step of the model, the moisture added is allocated to individual trees. The temperature, light, and moisture that would otherwise be available during the climate period are then modified by the influence of neighboring trees. The actual moisture availability determined is based upon moisture evaporation rates influenced by light and temperature, previous moisture present, and potential losses to the subsoil during the period.

Finally, growth takes place at a rate determined by moisture, temperature, and light. Growth rates used vary with individual species and give account of current season and present state of maturity. Moisture used in growing is removed from the soil.

Composition is measured, as requested by input data; the processing of successive climate periods continues until completion.

2. Simplifications and analogs

The particular sample of a forest selected or assumed will be a straight line. In the case of observed data, a strip of some appropriate, but as yet undetermined, width will be selected; all trees within that strip will be assumed to occur on a single straight line. This simplification is believed to be essential to the computational feasibility of the project and causes no significant detraction from realistic representation of the physical world. To simplify calculations the line is chosen in a north-south direction. Thus we will be modeling a "two-dimensional" forest. It should be recalled that we are interested in the ecological system, rather than individual trees. Such a simplification may cause loss of information concerning individual trees, but the loss of information to the overall model should prove to be well within the bounds of significance in the best climate sub-model we can hope to construct, or in the best plant growth models we can conceive.

An important model analogue used is the depicting of shading by an angle, (determined by species, season, and latitude) with shade occurring to the north during the daylight hours within the area of the right triangle, specified by the height of a tree and its shading angle. (Figure 1) The effects of shade on temperature and light (and hence on evaporation and growth) are modeled by analogy. The single angle selected will result in less area of shade than actually exists during the early morning and late afternoon hours; but in so doing, account is taken of the much reduced heat and light intensities occurring at those hours.

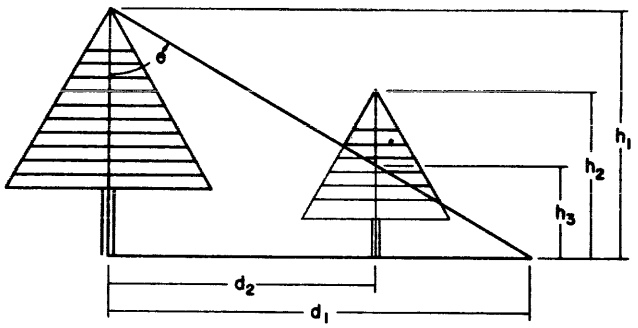


Figure 1 – Shade angle

Other simplifications and analogies are, in general, less crucial to the model and are encountered primarily in level of detail.

3. Level of detail

Every reasonable attempt has been made at every point to select a level of detail commensurate with the return of enlightening information and to provide flexibility in model construction and implementation, allowing modular adaptability over a broad range of detail levels. We are unlikely to have achieved an optimum level of detail in this early model. Also, since a major purpose of the study was to investigate feasibility, it has been appropriate in some cases to quite consciously avoid detail that would tend to improve validity but have little bearing on feasibility. An example is that the present model considers all moisture below the surface as a single number for each tree. The number indicates a total amount available without regard to root zones or soil stratification.

The climate model is, at present, a yearly cycle of temperature, moisture, and light means, each quantified as a single number occurring per time period. The time period selected was one month with model runs extending for ten years of simulated time.

Principal model features

1. Climate

During each period of simulated time the climate determines the quantities of additional resources made available for growth. The "model" of climate used here is simply a table depicting typical precipitation, temperature and light during each month.

Precipitation is moisture added in centimeters during the month. Light is the approximate number of hours of daylight per day reduced slightly during months normally experiencing considerable cloud cover or fog. Temperature was entered as a single number determined for each month as follows: Mean daily high and low temperatures for the month were assumed to occur at 1 p.m. plus 20% of time until darkness and 1 a.m. plus 90% of time until daylight respectively. It was assumed that temperature rises and falls linearly from high to low during the day. The positive portion of the above temperature function reduced by 50°F. was integrated between the limits, daylight and darkness, to determine the single temperature input for the month.

2. Shade

During the daylight hours shade occurs to the north of each tree. As noted above, an angle, θ, is used together with the tree height to determine a triangular area in which shade occurs. The shade angle, θ, will generally be smaller for evergreen than for deciduous trees except during those periods of dormancy in which the shade angle is reduced for the leaf shedding deciduous species. If any other tree exists wholly or partially within the shaded area, its light and temperature environment, and hence, its growth rate, is modified. The "shade factor," a number between 0 and 1 indicating the percent shaded, is determined for each tree at each time step by the following procedure.

After resetting all shade factors to zero, perform the following computation for each tree in the system, working from south to north (Figure 1).

1. For tree n, $d_1 = n_1 \tan \theta$.
2. Find d_2 for the next tree to the north.
3. If $d_1 \tau e_2$, begin again at step 1 with tree n + 1.
4. $h_3 = \tan \theta/(d_1 - d_2)$
5. $h_3 = \min(h_3, h_2)$
6. Shade factor = $\max(h_3/h_2$, present shade factor).
7. Return to step 2 above to see if more than one tree to the north is shaded by the present tree.

During each time period, loss of moisture added occurs due to evaporation from the soil at a rate dependent on whether or not the ground is shaded at the center of the "moisture zone."

3. Moisture Zones

The extent of a tree's root system limits the range or distance in each direction in which soil moisture will be available to it. The model assumes that, in the absence of conflicts, all moisture which falls within a distance equal to a tree's present height in either direction becomes available to its roots directly or through capillary action. This distance, or "natural moisture zone," is illustrated in Figure 2 with 45° triangles. (Other angles could be used and varied by species and state of maturity.)

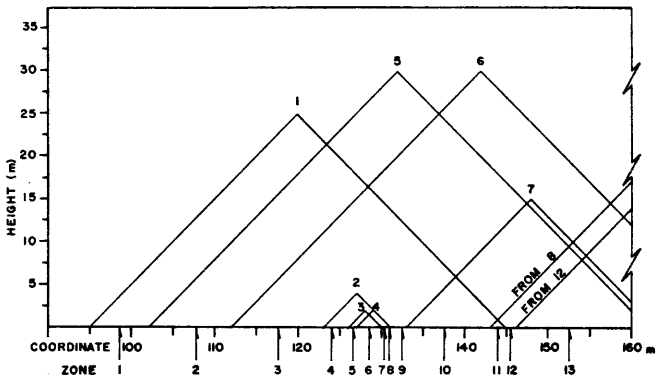


Figure 2 – Moisture allocation

When natural moisture zones overlap, as is generally the case, the trees must compete for the moisture that falls in the overlapping zones. The moisture zone algorithm begins by preparing a table of the starting and stopping co-ordinates of each tree's natural moisture zone. The entire forest is then divided into moisture zones. Associated with each zone is a list of trees competing for moisture in that zone, (Table 1). First we search the table of natural moisture zones to find the tree with the smallest starting co-ordinate. The co-ordinate and tree are entered as the first line of Table 1. We continue by selecting, for each line, the next smallest co-ordinate from the natural moisture zone table and entering it in Table 1. The co-ordinate entered terminates the preceding zone and begins a new one. If a starting co-ordinate is selected, the list of trees competing in the preceding zone is copied and the new tree added to the competition. If a tree's stopping co-ordinate is selected, the list is copied with that tree deleted. Zones of zero length will be subsequently ignored.

4. Moisture allocation

The moisture that falls in each moisture zone will be allocated to the trees competing in that zone by some combination of the following rules. (A weighting factor has been provided as input.)

TABLE 1

MOISTURE ALLOCATION

MOISTURE ZONES*	STARTING CO-ORDINATE	STOPPING CO-ORDINATE	TREES CONFLICTING IN ZONE
1	95	102	1
2	102	112	1 5
3	112	123	1 5 6
4	123	126	1 5 6 2
5	126	127	1 5 6 2 3
6	127	130	1 5 6 2 3 4
7	130	131	1 5 6 2 4
8	131	131	1 5 6 4
9	131	133	1 5 6
10	133	143	1 5 6 7
11	143	145	1 5 6 7 8
12	145	146	5 6 7 8
13	146	162	5 6 7 8 12
14	162	163	6 7 8 12
.	.		
.	.	.	. . . . .
.	.		

* At Time Zero

Rule A allocates moisture in direct proportion to the heights of the competing trees.

Rule B projects with the 45° angle the heights of all competing trees to the center of the zone and then allocates moisture in proportion to the projected heights.

Rule C allocates moisture in inverse proportion to distances from tree basis to the center of the zone.

Results presented later were obtained using Rule C.

5. Growth

Each species has associated with it an ideal growth rate which is a function of height. The growth that actually occurs during a climate period will be determined by reducing the ideal growth by some amount for each unfavorable environmental condition encountered. Ideal conditions for each species, moisture usage rates, and sensitivities, are input parameters.

A tree is selected, and its ideal growth for the period is determined as a function of species and height. If the tree is deciduous and dormant in the present climate period, its projected growth is reduced by an input factor.

The absolute difference in the temperature number that occurred and the ideal temperature is divided by the sensitivity. The sensitivity is the temperature differences that reduces growth by 10%. Thus, from the quotient of the above division, we determine a new growth rate.

The growth is multiplied by the hours of light per day in the current period divided by twelve. If a tree is entirely shaded (shade factor equals one) the growth rate is reduced by the "zero light rate" input factor for that species. If the tree is partially shaded, the growth rate is reduced to a proportionate rate between the zero light rate and the last determined rate.

The moisture needed for growth is determined first, from a usage rate multiplied by height, and then by one plus the projected growth in meters. A subsoil loss is determined as a percentage of moisture present. If the available moisture is greater than that needed for growth and subsoil loss, the moisture is removed and growth takes place. If adequate moisture is not present, the actual growth will be that fraction of the projected growth for which moisture is available.

Preliminary results

At the time of this writing no field data collection has taken place. Therefore the results obtained so far have been the outcome of qualitative model testing. Primarily, this testing has been the operation of the model under extreme conditions such as total absence of some necessary growth resource. Other tests have included such things as operation with and without ecological interference as controlled by spacing of individual plants. A sensitivity analysis has also been performed in order to insure "reasonable" responsiveness to major factors modeled. Validity of basic model structure has now been assumed, since behavior under the tests indicated above is of a form and direction anticipated by initial assumptions.

The output from a typical model run is included below (Table 2 and Figure 3). The forest in this particular example was obtained by random shuffling of data cards representing a deciduous and an evergreen species. (Similar model runs with more than 100 trees have been made.) In Table 2, co-ordinates and heights are in meters. Shade factors and moisture present are described above. The trees at each end are generally in a superior competitive position, and hence "less representative." The third and fourth trees illustrate markedly the effect of changing environment with time. Others illustrate such effects as declining growth rates caused by the increased moisture requirements accompanying increased heights.

This particular, somewhat abstract, forest has been run many times with many variations of model parameters. As a result of the adaptability and responsive-

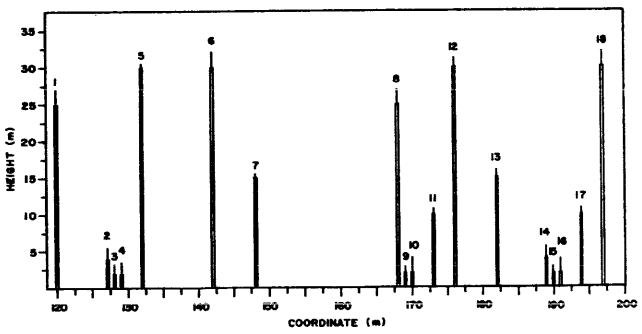


Figure 3 – Results

TABLE 2

RESULTS

TREE NO.	CO-ORDINATE	SPECIES	INITIAL HEIGHT	HEIGHT 5 YRS.	HEIGHT 10 YRS.	SHADE* FACTOR	MOISTURE* PRESENT
1	120	Evg.	25	26.96	28.93	0.00	1259.9
2	127	Dec.	4	5.41	6.76	0.80	58.2
3	128	Evg.	2	3.36	4.12	0.32	0.0
4	129	Evg.	2	3.45	4.11	0.04	0.0
5	132	Dec.	30	30.47	31.02	0.00	60.6
6	142	Evg.	30	31.94	33.57	0.00	50.2
7	148	Dec.	15	15.54	16.25	0.81	31.4
8	168	Evg.	25	26.90	28.52	0.00	51.6
9	169	Dec.	2	2.74	3.16	1.00	0.0
10	170	Evg.	2	4.05	5.45	1.00	0.1
11	173	Dec.	10	10.61	11.36	1.00	41.5
12	176	Evg.	30	31.18	32.51	0.03	0.0
13	182	Dec.	15	15.71	16.39	0.75	65.1
14	189	Dec.	4	5.46	6.56	0.00	70.4
15	190	Dec.	2	2.70	3.12	0.19	0.0
16	191	Evg.	2	3.81	5.15	0.00	0.2
17	194	Dec.	10	10.62	11.25	0.00	228.1
18	197	Evg.	30	32.04	34.08	0.00	1820.6

* At Five Years

ness demonstrated, we are able to conclude that the model will prove valuable in its present form and help to open the gates to a large number of computer studies and applications in the plant sciences.

The future

A step preliminary to future development will be an extensive data collection effort. Data collected over extended periods of time and taken from many different geographic regions for many different species and forest densities will be necessary to properly adjust, or "fine tune," model parameters.

Additional levels of detail may be considered. A stratified soil model seems to be a most promising possibility and would allow for finer accounting of moisture availability, soil type, and fertility in the root zone of individual trees.

A less difficult, but logical, next step will be the ascribing of measures of economic importance to the forest. Such a measure will allow the present model to be used as a "laboratory" for the investigation of many forest management practices. For example, quantitative results concerning the value of thinning to reduce shading could be determined by simply applying the model repeatedly with various thinning criteria applied to the same forest. Such a study made in the physical world, by experimentation, would not only take many years to complete, but would depend on the critical assumption that each forest or subforest under study was completely equivalent and subjected to the very same environmental influences during the entire duration of the study period.

The adaption of the principals and algorithms of the present model to crop plants has already begun. The emphasis, here, must change from one of reaction to natural influences to that of reaction to soil management practices. Also, our attention must focus on the fruit of the plant rather than the plant itself. However, the many similarities make the present model an important first step in this direction.

REFERENCES

1 F W WENT
The experimental control of plant growth
Chronica Botanica Company Waltham Massachusetts 1957

2 *R E BOCHE*
Some algorithms for allocation of environmental resources determining plant growth
Symposium on Physiological Systems in Semi-Arid Environments. Sponsored by the University of New Mexico and the National Science Foundation Albuquerque New Mexico November 1967 Proceedings to appear 1968

A major seismic use for the fast-multiply unit

by ROBERT D. FORESTER*
Digital Seismic Corporation
Houston, Texas
and
TIM J. HOLLINGSWORTH and JAMES D. MORGAN
Petty Geophysical Engineering
San Antonio, Texas

No matter how much speed or capacity you add to your computer system, programmers will develop software which will tax it to its limits. Programmers at Petty Geophysical Engineering are no exception. Two years ago a fast-multiply unit which can multiply and add 2,000,000 times per second was added to their CDC 3200 installation. Soon afterwards they developed a sophisticated program for enhancing seismic signals which depends heavily on the unit's great speed. The program is called "APE", which is an acronym for Automatic Phase Editing. It has proved to be a valuable tool in the search for oil.

In order to see how the APE program fits into the scheme of seismic exploration for oil, we will broadly describe how seismic data are gathered in the field and processed in the laboratory.

Figure 1 shows a diagrammatic cross section of how seismic data are collected in the field. Dynamite is loaded at the bottom of shotholes drilled through the earth's weathered layer. Sound energy emitted by shooting the dynamite spreads downward and is reflected upward by discontinuities in the velocity or density of the earth's layering. Ray paths symbolize the directions along which the energy travels. The returning echos are sensed by a surface array of geophones and recorded on magnetic tape in analog or digital form. The weathered layer tends to be irregular and can cause reflected events to mismatch in time from one geophone to the next.

Figure 2 shows a photographic plot of multichannel, seismic data recorded on magnetic tape. Energy for a single shot was recorded by an array of 24 geophone groups, each geophone group representing a seismic trace. The horizontal sub-surface coverage of the geophone array was about ½ mile. In this example, the time extent of the recording shown is 1.9 seconds. The maximum time length of the original magnetic recordings is usually about 6 seconds. For most digital processing purposes, each trace is sampled 3000 times; thus for a 24-trace record, the computer must handle 72,000 samples of data. Our center processes tens of thousands of records like this each month. In terms of digitized data samples, this amounts to an input of over a billion pieces of information per month.

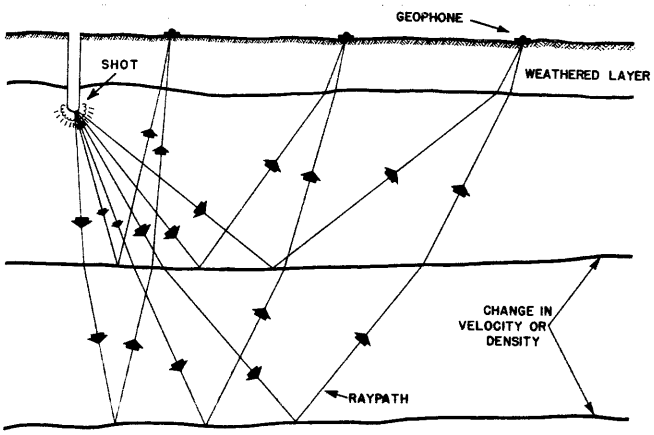


Figure 1 – Cross section of seismic reflection shooting

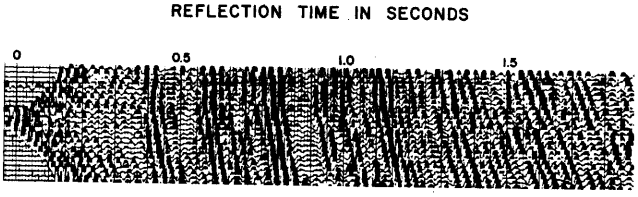


Figure 2 – A 24-trace seismogram

*Formerly Vice President-Service Development, Petty Geophysical Engineering, San Antonio, Texas.

The processed records are assembled into cross sections like that shown in Figure 3. This section comprises 6 records representing a horizontal sub-surface coverage of about 3 miles. The depth penetration is about 10,000 feet. The reflection pattern extending across the section depicts the configuration of strata within the earth. This type of presentation is widely used by geophysicists to detect structural and stratigraphic anomalies in the earth which may contain oil. (To emphasize the reflection patterns, the positive halves of the trace swings are filled in solid by an electronic plotter.)

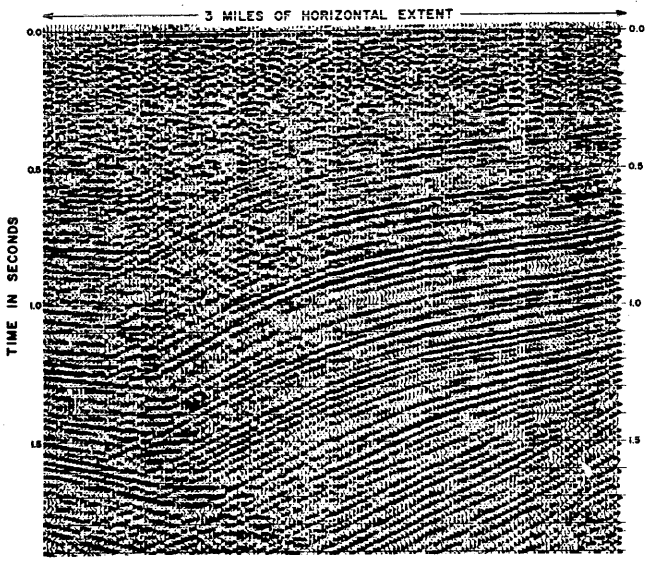


Figure 3 – A seismic cross section

It would be so simple if there were no noise, but mother nature is not that kind. There are many varieties of noise, both random and organized, to plague the seismic interpreter. In order to cancel noise, or build up the signal-to-noise ratio, it's now a common practice to shoot repeated coverage in an area and then add or "stack" the data together. Such shooting of repeated coverage has added greatly to the store of data to be processed in the last five years. Many computing centers have sprung up to keep up with the increased load.

Figure 4 shows a diagram of a 12-fold repeated coverage field setup. The shotpoints and geophones are placed so that the 12 ray paths converge symmetrically toward a common reflection point at each reflector. Travel times along each slanted ray path are corrected to what they would be, had the ray path been vertical. The corrections involve a nonlinear time stretching which is nicely done by a computer. If the weathered layer is highly variable in

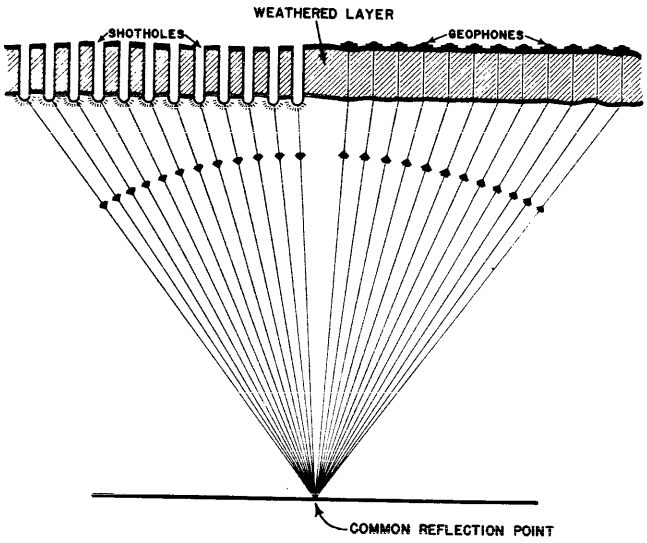


Figure 4 – 12-fold common-reflection-point setup

velocity and thickness, reflections may show severe displacements from one trace to the next.

In Figure 5 there are 9 sets of 12-fold common-reflection-point[1] traces. Each set is to be reduced to a single trace by addition. Owing to weathering irregularities and noise, the reflections are not aligned and have a ragged appearance. Stacking events which are badly out-of-phase would do little toward building up the signal-to-noise ratio.

In the past, our geophysicists used to determine time shifts which would line up reflections by visual correlation of reflection character. To straighten up the reflection bands shown in Figure 5 would be a laborious task taking most of a working day. The APE computer program will align reflections automatically. Figure 6 shows the effects of applying APE to the data in Figure 5. The computer took only a few minutes to generate this display of smoothly aligned reflections.

We will now describe the basic principles which are utilized in the total APE process, each step of which makes use of the fast-multiply unit.

Figure 7 shows the correlation process in the time domain. The correlation process is commonly used to search for a correspondence between traces. In this example we are searching for a replica of the search signal in the field trace. As the search signal is moved in incremental steps past the field trace, the amplitude values are cross multiplied and summed. The sums constitute the amplitudes of the output trace at the bottom of the Figure. Three sums, corresponding respectively to positions A, B, and C of the

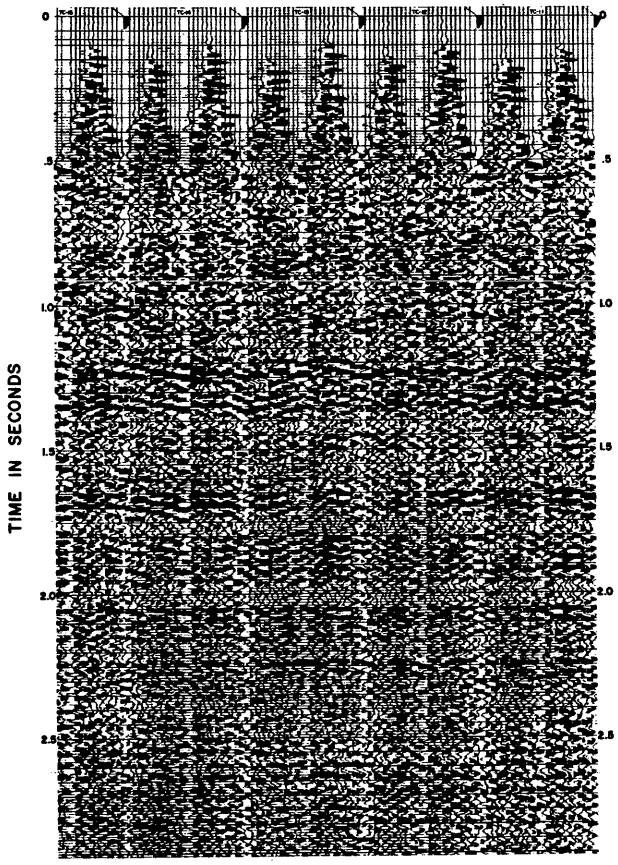


Figure 5 – 12-fold trace collections before APE

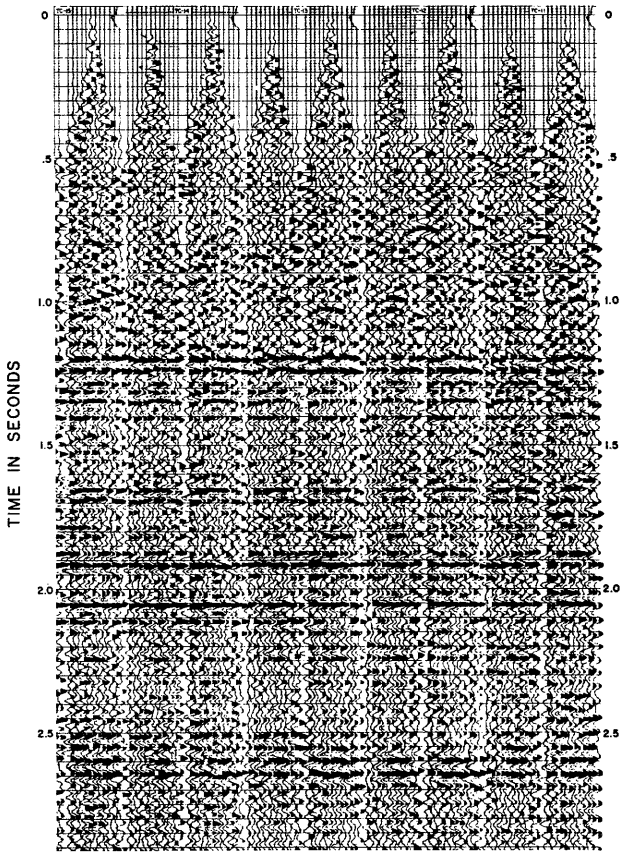


Figure 6 – 12-fold trace collections after APE

search signal, are marked on the output trace. In position B the search signal is lined up exactly with the pulse of identical shape on the field trace. The output trace shows an amplitude peak corresponding to position B. The arithmetic for position B, which is noted on the right side of the Figure, could be done by the fast-multiply unit in just 2.5 microseconds.

In the correlation process, the phases of the two traces are subtracted from each other. Since the search signal in this example is identical to that of the field trace, the phases subtract out to zero for all frequencies and a symmetric output waveform is obtained.

At one time we tried to estimate the corrections needed to bring traces into time alignment by simply noting displacements among correlation peaks. Experience showed that these peaks were generally too dull to be used for the precise determination of such corrections. This simple approach was abandoned in favor of the APE process, which employs cross-correlation only as a part of a more sophisticated scheme.

Convolution, which is another part of the APE process, is depicted in Figure 8. Convolution is popularly thought of as the mathematical equivalent of filtering. For digital filtering applications, the sliding waveform represents the impulse response of the filter to be used. The numerical technique of summing cross multiplications is identical to that used for the correlation process. The difference between convolution and correlation is that the sliding waveform is time-reversed in the convolution process.

The arithmetic corresponding to position B of the sliding waveform is shown on the right. This arithmetic could also be done with the fast-multiply unit in 2.5 microseconds.

The convolution process causes the phases of the harmonic components of the two traces to be added to each other. Since the shape of the field trace is identical to that of the impulse response, the output waveform can be thought of as an impulse response produced by double filtering. Note that the output waveform is not symmetric.

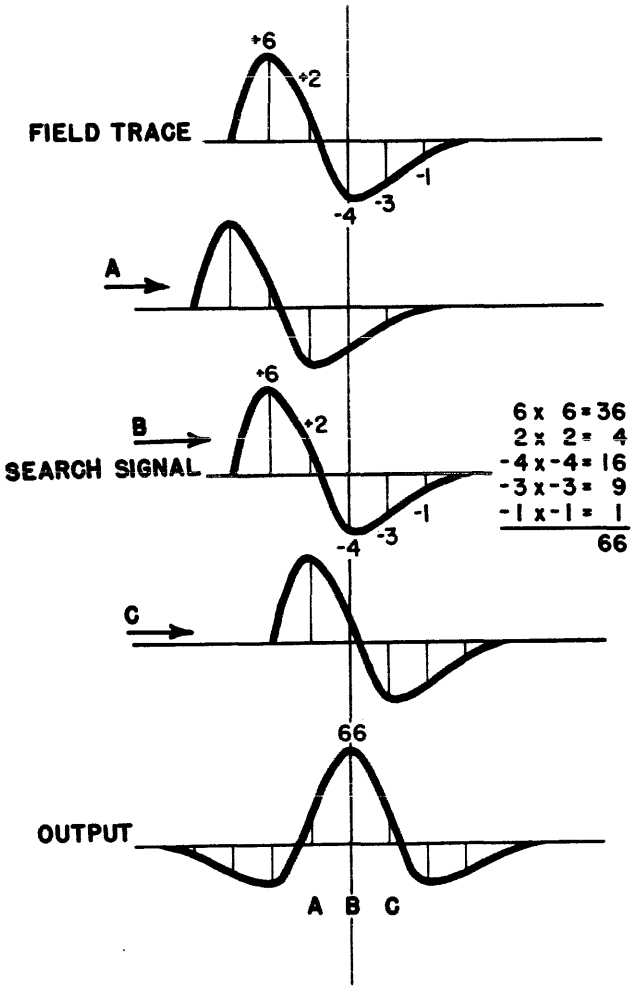


Figure 7 – Correlation in the time domain

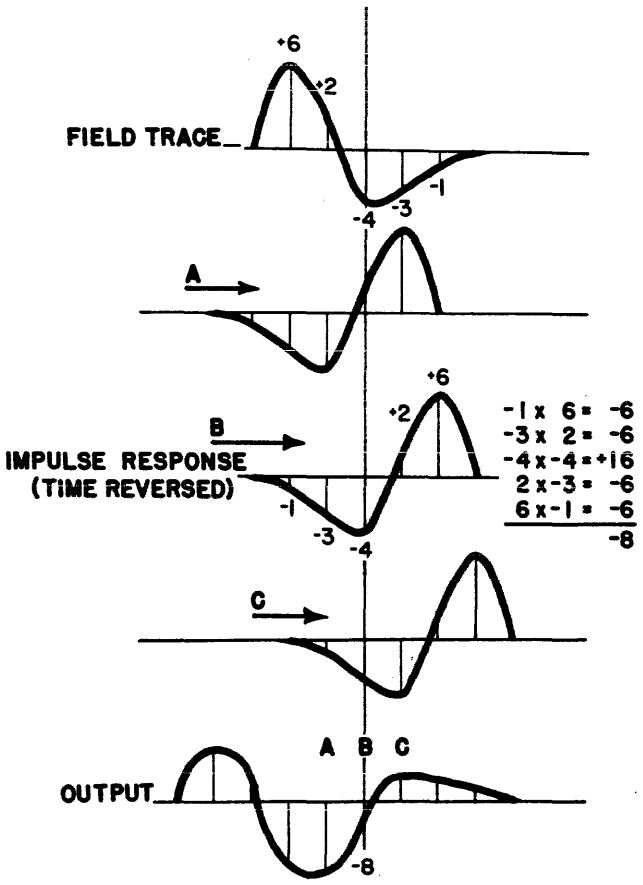


Figure 8 – Convolution in the time domain

Figure 9 shows how convolution is used in the APE process. For simplification, operations involving only a single field trace are shown. The top trace is the field trace; the middle trace, the autocorrelation of the field trace; the bottom trace, the convolution of the field trace with its autocorrelation. The higher the order of the cross multiplication involving a given shaped signal, the more the final wave shape will ring or oscillate. Note that the convolved autocorrelation is much more oscillatory than the original field trace. Such induced ringing would be bad for seismic interpretation because it can cause discrete reflection wavelets to overlap each other. Thus the APE process carried only this far would result in a loss of resolution.

To get rid of the ringing, we resort to a process called "deconvolution.[2]" Figure 10 depicts the successive stages of the deconvolution process. The ringing trace is first autocorrelated. Using Z-transforms, an inverse filter is designed which when convolved with a ringing trace will shrink the reflections into spikes. In a spike trace, all frequencies contain equal amplitudes. Because the low and high frequencies brought up by the deconvolution process may contain excessive noise, it is generally necessary to apply to the deconvolved data a bandpass filter having a broad peak. To accomplish bandlimiting, the deconvolved trace is convolved with a symmetric impulse response to give the trace at bottom. The bandpass-filtered trace can be thought of as having been obtained by replacing each spike by a replica of the impulse response of the bandpass filter; the magnitude, polarity, and time position of each replica being determined by the spike it replaces.

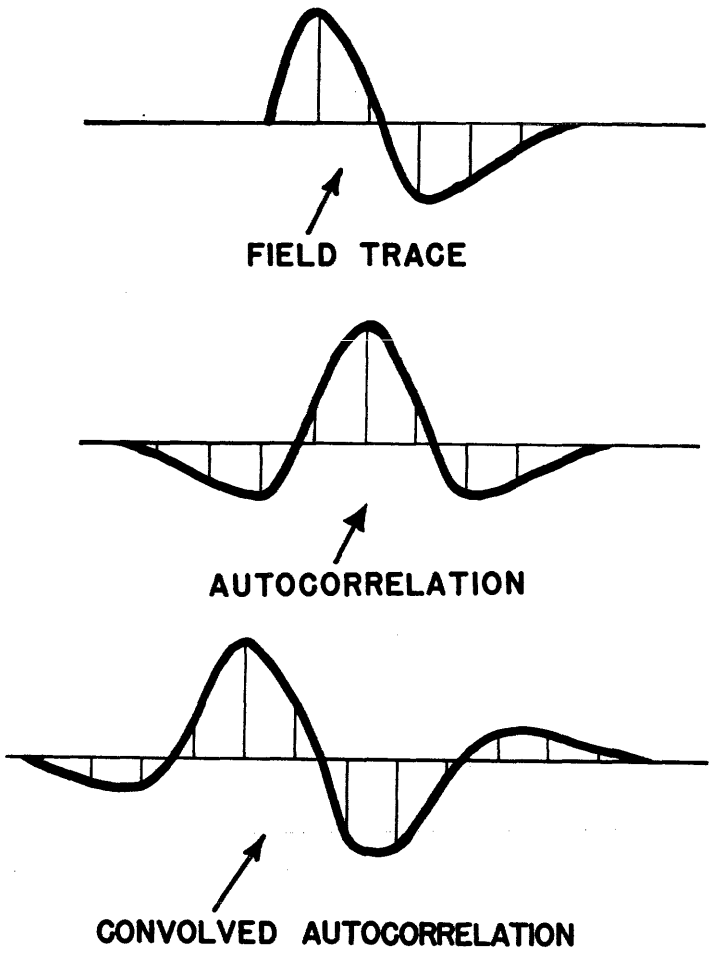


Figure 9 – Convolution of an autocorrelation

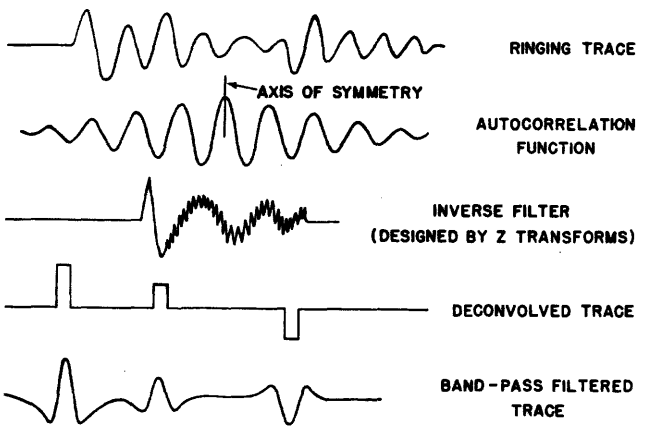


Figure 10 – Deconvolution process

We are now ready to describe the total APE process (Figure 11). The goal is to line up traces 1 and 2 with respect to the reference trace. Manipulations involving trace 1 are shown as dashed lines; those for trace 2, as solid lines. The reference trace is first cross-correlated with traces 1 and 2, respectively. Because phases are subtracted in the correlation process, time lead and lag relations become reversed. The correlation function for trace 1 leads the reference trace by the amount trace 1 lags it; and the correlation function for trace 2 lags the reference trace by the amount trace 2 leads it.

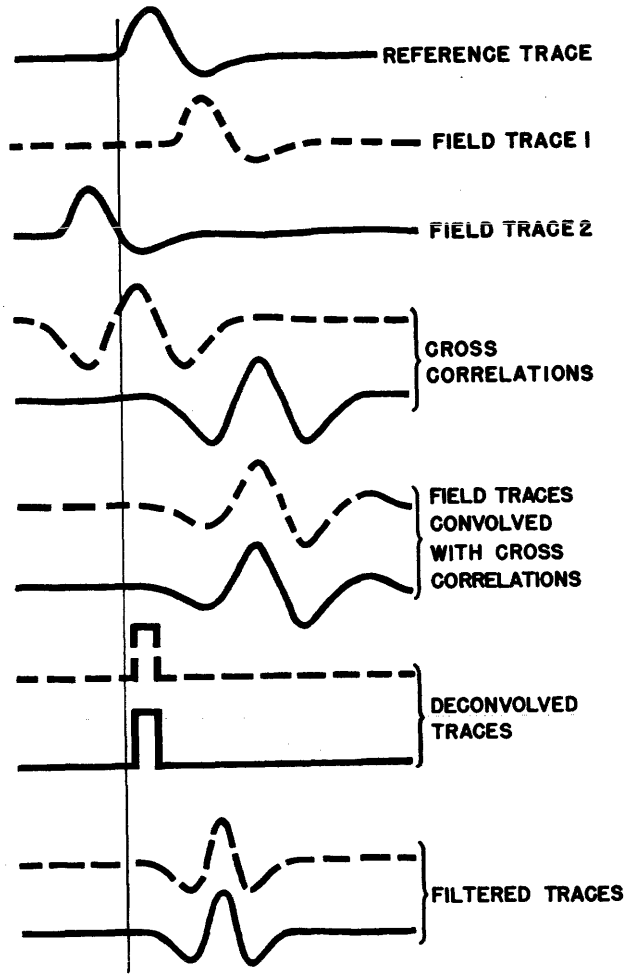


Figure 11 – Steps in the Automatic Phase Edit process

Traces 1 and 2 are convolved with their respective cross-correlations with the reference trace. The resultant convolutions line up in phase, but ring excessively. The two functions are than autocorrelated and inverse filters designed. The inverse filters are then convolved with the ringing functions to restore resolution, as signified by the lined-up spikes. These spike traces are then convolved with a bandpass filter to yield aligned traces which are seismically realistic in appearance.

In the time domain, APE can cause various frequencies to shift by different time amounts. In other words, APE can produce non-linear phase shifts. This is demonstrated in Figure 12 where APE is applied to synthetic pulses of two different frequencies. The raw data in the top panel consists of identical suites of low-frequency pulses on the left side and high-frequency pulses on the right side. Not only are

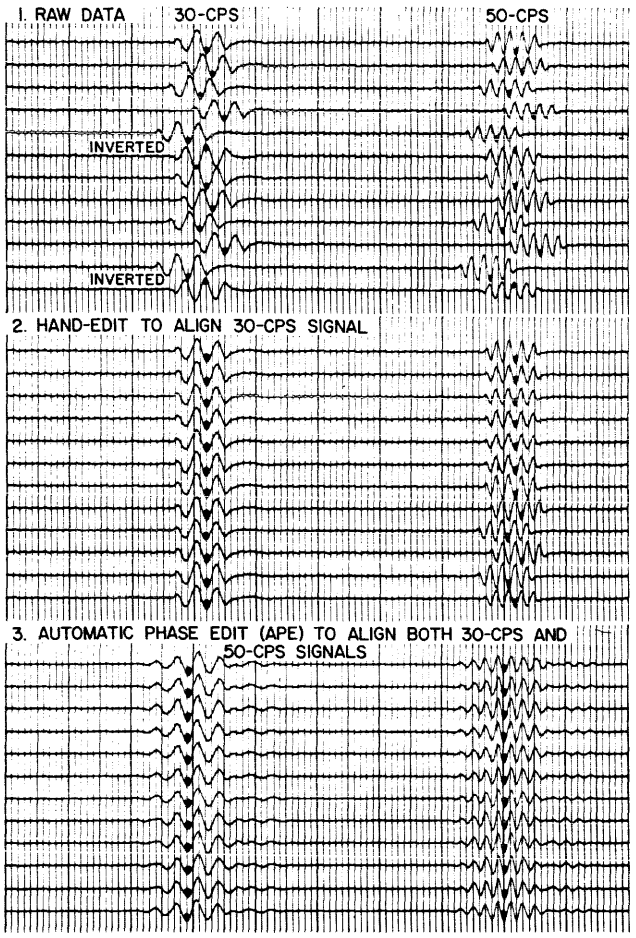


Figure 12 – Using APE to align synthetic pulses of two different frequencies

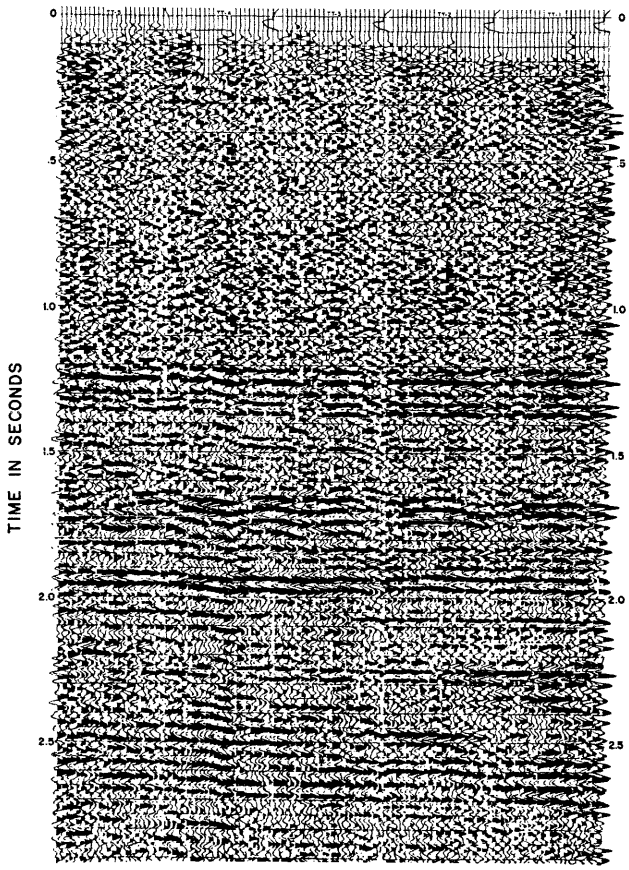


Figure 13 – 12-fold stack before APE

the pulses statically shifted relative to one another, but the interval between them is not uniform. Two of the traces are inverted in polarity to symbolize accidental reversal of terminal connections in the field. In the middle panel, the low-frequency pulses have been lined up by manual editing. Note that the high-frequency pulses are still misaligned because of the non-uniform interval between pulses on each trace. The bottom panel shows the results of APE processing. Both low- and high-frequency pulses have been lined up, and the traces reversed in polarity have been automatically inverted. (In this example, ringing is yet to be removed by deconvolution.)

APE's power of producing non-linear phase shifts can compensate for distortion in wavelet character occurring from one trace to the next. Areal variations in the filtering properties of the weathered layer usually account for most of such distortion. Manual determinations of time-alignment corrections are limited by the assumption that the filtering properties of the weathering do not change much throughout an area.

Now let us again examine the effects of APE on field data. We have noted that APE transformed the ragged-appearing reflections in Figure 5 into the smoothly aligned reflections shown in Figure 6. In addition there is an improved signal-to-noise ratio as indicated by the fact that the intervals between the reflections appear cleaner and quieter. This happened because the APE process builds up only those frequencies showing trace-to-trace coherence and cancels those that lack coherence.

The 12-fold trace collections in Figures 5 and 6 were stacked to yield the results in Figures 13 and 14, respectively. The reflections on the stacked APE section are cleaner, stronger, and sharper than those on the stacked non-APE section. Because reflection character has been stabilized from trace to trace, residual variations in character should be more reliably diagnostic of changes in lithology. Because the phase shifting has been done objectively by a machine, structural anomalies will not be suspected as being due to human bias, as is often the case in manual

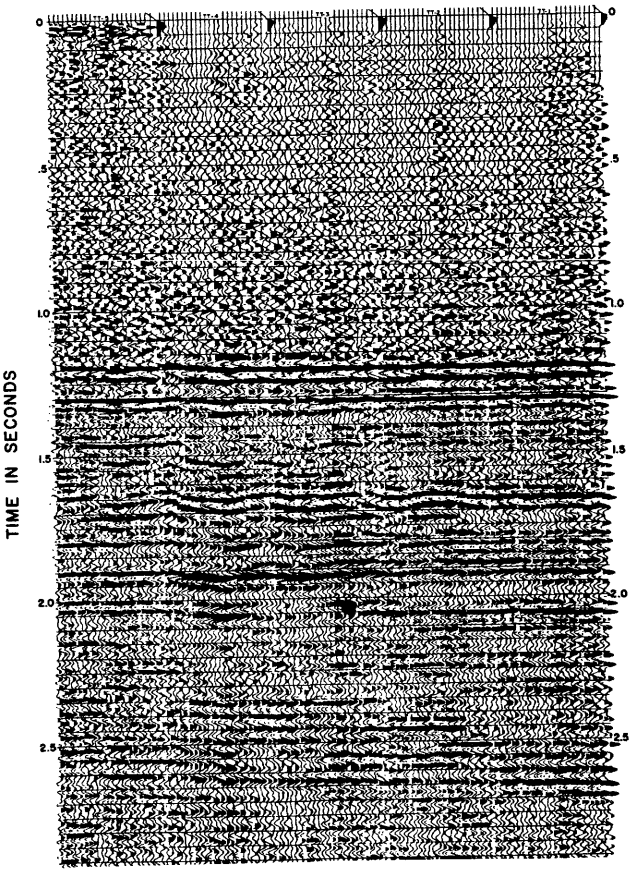


Figure 14 – 12-fold stack after APE

editing. All these improvements effected by APE can help the geophysicist make more reliable seismic interpretations in the search for oil.

SUBPROCESS	MULTIPLICATIONS AND ADDITIONS
1. CROSS-CORRELATION	7,200,000
2. CONVOLVED CROSS-CORRELATION	7,200,000
3. DECONVOLUTION	
A. AUTOCORRELATION	7,200,000
B. CONVOLUTION WITH INVERSE FILTER	7,200,000
C. CONVOLUTION WITH BAND-LIMITING FILTER	3,600,000
TOTAL	32,400,000

Figure 15 – Tally of multiplications and additions required to APE process a 24-trace seismogram

Without the fast-multiply unit, APE processing would not be economically feasible. Figure 15 tallies the number of multiplications and additions required to APE process a single record of 24 traces, each trace having been sampled 3000 times. The number of multiplications and additions required to APE process one record amounts to a grand total of 32,400,000. The fast-multiply unit speeds through the computations in just 16.2 seconds. For APE processing, a client currently buys 16,200 multiplications and additions for a penny. APE processing represents a big seismic use for the fast-multiply unit.

REFERENCES

1 W H MAYNE
Common reflection point horizontal data stacking techniques
Geophysics vol XXVII no 6 pt II December 1962 pp 927-938

2 *The MIT geophysical analysis group reports*
Special issue of Geophysics vol XXXII no 3 June 1967 pp 411-521

A generalized linear model for optimization of architectural planning

by RODOLFO J. AGUILAR and JAMES E. HAND
Louisiana State University
Baton Rouge, Louisiana

INTRODUCTION

In the realm of architectural planning there exists a type of problem with which designers are frequently confronted where financial return is the most appropriate measure of the system's effectiveness. In this category can be included all rental and speculative housing (single and multiple family dwellings), office buildings, warehouses, stores, many industrial facilities, etc., and building complexes which combine some or all of these to provide comprehensive services to the tenant.

Traditionally, the problem of planning for capital investment has been handled in a semi-empirical way, where the data which serve as basis for decision making may be more or less reliable and up to date, but where the processing of these data, to arrive at the optimum allocation of space for maximum expectation of financial return, is entirely intuitive and, consequently, unreliable.

The writers have studied the allocation problem in detail and have concluded that once the pertinent data are collected, the most profitable design configuration can be ascertained through the optimization of a linear model which incorporates the most salient features of the real-life, planning situation.

The allocation of rental housing (single type facility) has been modeled by Aguilar.[1,2] However, a mix of various types of uses has not been investigated previously.

The model developed in the following section takes into account multi-use facilities and considers realistic design constraints such as zoning regulations, parking requirements, etc. An example problem is also presented to illustrate application of the theory.

The model

Definition of terms

The design of buildings and other architectural facilities for financial return is constrained by many internal and external factors such as budget, market restrictions (construction costs, rentability, individual preferences), parking regulations, lot coverage, etc., in addition to environmental factors, many of which defy quantification.

The procedure described in this section attempts to consider as many as possible of the quantifiable factors in optimizing architectural planning within the framework of economics; return on invested capital.

With the information extracted from the model, (a type of *simulation* model in many respects), rational decisions can be formulated based upon knowledge of the most recent tax-depreciation structure, maintenance costs, alternative investment opportunities, etc.

Let x_{ij}^{k} be the *floor area* of facility type i, at level j, of architectural quality k.

The "type i facility" refers to the use made of the space considered; offices, apartments, stores, warehouses, laboratories, industrial facilities, classrooms, etc.

The "j level" denotes the location of the facility; first, second,...,n^{th} floor level.

The "k quality" refers to the degree of architectural refinement of the space; types of finishes, environmental control, and many other considerations, the effects of which are reflected in the cost of construction and in the rentability of the facility.

Similary, Let

c_{ij}^{k} = *cost* per unit of area of facility i, at level j, of architectural quality k.
r_{ij}^{k} = *rent* per unit of area, per unit of time, from facility i, at level j, and architectural quality k.
p_{ij}^{k} = *probability of renting or selling* facility i, at level j, and architectural quality k.
and,
q_{ij}^{k} = *probability of not renting or selling* facility i, at

level j, and architectural quality k. Hence, $p_{ij}^k + q_{ij}^k = 1$.

It will be assumed that the market is large, and that for the time period under consideration, it is in a steady state condition, such that the introduction of additional facilities for rent or sale will *not* affect significantly the parameters defined above.

Mathematical formulation

The total expected rent per unit of time can be expressed as follows:

$$E(R) = \sum_{k=1}^{L} \sum_{j=1}^{m} \sum_{i=1}^{n} p_{ij}^k r_{ij}^k x_{ij}^k. \quad (1)$$

This is the objective function to be maximized subjected to constraints of the types described below.

1. *Market*
   a. A survey may reveal that, within a specified geographic region, there exist vacancy rates for some or all of the proposed facilities. These vacancy rates are in fact the $q_{ij}^k = 1 - p_{ij}^k$ which were already considered in the formulation of the objective function. Nevertheless, the q_{ij}^k should be compared with the upper limits set on them by banks and other funding institutions. If these limits are exceeded, the project may be very difficult if not impossible to finance. Therefore,

$$q_{ij}^k \leq (q_i^k)\ \max, \text{ for all i and k.} \quad (2)$$

   b. *Market preferences* data may indicate that the area of each facility type and quality must not exceed a certain fraction of the total building area. Let f_i^k be upper bounds to the area ratios; then, the constraints can be written,

$$\sum_j x_{ij}^k \leq f_i^k \left(\sum_{k=1}^{L} \sum_{i=1}^{m} \sum_{j=1}^{n} x_{ij}^k \right) + e_i^k, \quad i = 1, 2, \ldots, m; \quad k = 1, 2, \ldots, L. \quad (3)^*$$

In general,

$$\sum_{k=1}^{L} \sum_{i=1}^{n} f_i^k > 1 \quad (4)$$

because of the requirement that the f_i^k be upper bounds to the area ratios.

The e_i^k are small positive constants introduced into the constraint equations and inequalities to avert degeneracy.

*The range of the sum on the left hand side of inequality (3) varies for each facility type and quality and for this reason it is not given explicity. This convention will be used for the rest of this paper.

2. *Zoning Regulations* such as:
   a. *Maximum building coverage* of total lot area which can be mathematically expressed as follows,

$$\sum_k \sum_i x_{ij}^k \leq A_j, \quad j = 1, 2, \ldots m, \quad (5)$$

   where A_j = maximum allowable building area of floor level j.
   b. *Height restrictions* which could be overall building restrictions that $j \leq m$ or restrictions on each individual facility; merchandising space, for example, should be located on lowest floors, etc.
   c. *Off-street parking* regulations, usually given as the number, n_i, of parking stalls per building area, a_i, of facility type i. In addition, the area, a, required for each car for parking, drives, etc., can be easily computed and the off-street, surface parking restriction expressed as,

$$a\left[\sum_{i=1}^{n} \frac{ni}{ai} \left(\sum_{k=1}^{L} \sum_{j=1}^{m} x_{ij}^k \right) \right] \leq A_t - \sum_k \sum_i x_{ij}^k, \quad (6)$$

   where A_t = total buildable lot area (total lot area minus area required for landscaping, street rights of way, utility easements, set back restrictions, topographically unsuitable land, etc.).
   Note that $\left(A_t - \sum_k \sum_i x_{ij}^k \right)$ is the site area available for parking (A_t minus first floor area of building).

3. *Design decisions* (massing studies). These decisions are made by designers for aesthetic and other reasons and are often arbitrary. Nevertheless, their effect upon the economic health of the system can be measured by comparing all optimal solutions from models with different sets of design constraints to one having *no* design constraint. The model without design constraints yields a solution that in this paper will be called the *optimum optimal solution*. The massing associated with the optimum optimal solution may not be aesthetically and/or structurally acceptable and it is at this point that design constraints must be introduced. The constrained model will, in general, yield a lower value of the objective function. Thus, if $E(R)_0$ is the optimum optimal rent and $E(R)_c$ is the expected rent when the problem is constrained by design considerations,

$$E(R)_c = E(R)_0 - C_c, \qquad (7)$$

where C_c is the cost of the design decisions.

It should be realized that aesthetic values may affect the four parameters c_{ij}^k, r_{ij}^k, p_{ij}^k, and q_{ij}^k as well as others. If their effect is known, the parameters should be modified accordingly and a new $E(R)_c$ computed.

The ability to ascertain the effect of design decisions upon the system's economic health is a most valuable characteristic of the model, for it measures indirectly the "cost" of beauty and of other aesthetic factors; it represents an attempt to quantify some of the intangibles of architecture and urban planning. Design constraints may take different mathematical forms depending upon the massing restrictions. If, for example, the planner decides that the building should take the form of a tower with a spread out, s story base, b times or more larger than each of the tower's floors (Lever House type of building), the constraints would be written, thus,

$$\sum_k \sum_i x_{i(s+1)}^k \leq \frac{1}{b}\left[\sum_k \sum_i x_{i1}^k\right] + e_{s+1}, \qquad (8)$$

$$\sum_k \sum_i x_{ij}^k = \sum_k \sum_i x_{i1}^k + e_j, j = 2, 3, \ldots, s, \qquad (9)$$

$$\sum_k \sum_i x_{ij}^k = \sum_k \sum_i x_{i(s+1)}^k + e_j, j = s+2, s+3, \ldots, m. \qquad (10)$$

The e_j, $j = 2, \ldots, m$, are, again, small positive constants introduced to avert degeneracy. Many other types of design decisions can be similarly formulated.

4. *Budget,* generally expressed as a fixed amount of money, B, which must not be exceeded by all fees and construction costs, excluding the cost of land.

Construction costs for each facility type and quality are, in general, functions of the total number of floors. Specifically, as the number of floors in the building increases, the costs per unit of area could decrease or increase. This variability is not usually too sensitive and can be conveniently expressed as a cost multiplier, step function of the total number of floors, m. Let $\beta_{ij}^k(m) \geq 0$ be such step function. Graphically, it could be typically mapped as in Figure 1. The figure shows that $\beta_{ij}^k(m)$ decreases for $(\alpha_{ij}^k)_2 \leq m < (\alpha_{ij}^k)_3$. It further decreases for $(\alpha_{ij}^k)_3 \leq m < (\alpha_{ij}^k)_4$. Then, it increases in the interval $(\alpha_{ij}^k)_4 \leq m < (\alpha_{ij}^k)_5$, and the increase is even greater for $(\alpha_{ij}^k)_5 \leq m \leq (\alpha_{ij}^k)_6$. These latter increases would reflect the higher costs of foundation and vertical transportation, for example.

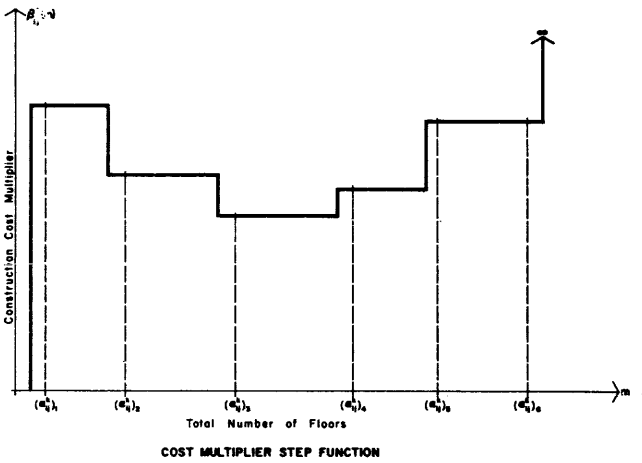


Figure 1 – Cost multiplier step function

When $m > (\alpha_{ij}^k)_6$, the cost multiplier becomes infinite, denoting that $(\alpha_{ij}^k)_6$ is the upper bound to the number of floors due to zoning restrictions or other considerations. The variation shown in Figure 1 is, of course, only one of an infinite number of possibilities, but all of them will exhibit the same general shape.

In general, there will be $n \times m \times L$ construction cost multipliers with a proportionate number of $(\alpha_{ij}^k)_q$ nodes where changes in the costs occur. Let the nodes be ordered sequentially for all $\beta_{ij}^k(m)$ and renumbered γ_q, $q = 1, 2, 3, \ldots, r$. Then, a typical cost multiplier, step function of m, would appear as given in Figure 2.

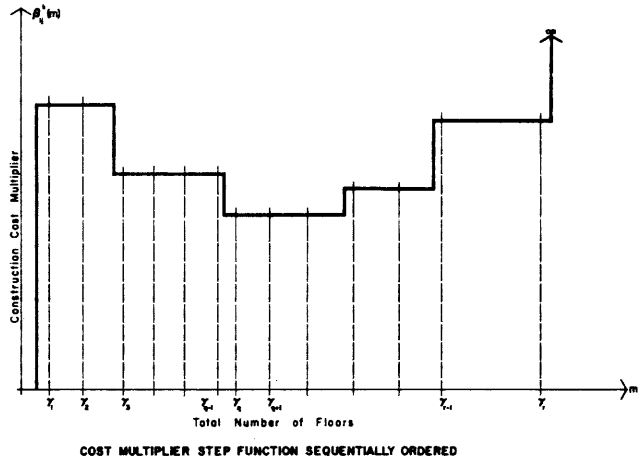


Figure 2 – Cost multiplier step function sequentally ordered

This relabeling is necessary to show, for each $\beta_{ij}^{k}(m)$ multiplier, all points at which construction costs change for any type, level and quality of facility; thus, affording complete control upon the optimization process.

The cost multiplier functions, however, introduce a further complication in the model because the number of floors may not be known a priori. This is, in fact, a non-linear characteristic of the system, that, in this context, exhibits recycle loops. A method will be proposed to handle the non-linearity in a satisfactory, linear manner. Additional items of cost and cost coefficients must be defined, for example:

a. *Cost of sub-surface exploration*, soil investigation, foundation recommendations, and report. Usually given as a lump sum, b_s.
b. *Architectural-Engineering fees*, usually expressed as a percentage of total construction costs by a decimal coefficient, c_a.
c. *Fund for contingencies*, also expressed as a percentage of total construction costs. Let c_c be the decimal equivalent of that percentage.
d. *Cost of movable furniture*, expressed as a percentage of construction cost for each facility type i and quality k. Let c_i^k be the decimal equivalents of those percentages.
e. *Supervision of construction costs*, generally computed as a lump sum, b_c, which is the total salary or salaries of one or more supervisors for the anticipated duration of the construction phase of the building. b_c is assumed constant for a given project.
f. *Cost of parking facilities per unit of area*, c_p.

Support space, such as halls, elevators, toilets, etc., is assumed to be included in the x_{ij}^k and for this reason, its cost must be apportioned among tenants and/or buyers (as it always is).

From these considerations, the following constraint inequality can be readily formulated:

$$(1+c_a+c_b)\left\{\sum_{k=1}^{L}\sum_{i=1}^{n}(1+c_i^k)\left[\sum_{j=1}^{m}\beta_{ij}^k\, c_{ij}^k\, x_{ij}^k\right]\right\} + c_p\, a\left[\sum_{i=1}^{n}\frac{ni}{ai}\left(\sum_{k=1}^{L}\sum_{j=1}^{m}x_{ij}^k\right)\right]+b_s+b_c \leq B. \qquad (11)$$

The constraint given above presupposes that the total number of floors, m, is known in advance. On this basis, the β_{ij}^k multipliers are obtained without difficulty. In performing an optimization study, however, design constraints would be relaxed at some point, and the total number of floors would become one of the unknown parameters. This, in turn, makes the β_{ij}^k be undetermined and a non-linear, recycle loop could be generated. To avoid this complication, the following algorithm is proposed:

Step 1. Assume that the maximum number of floors γ_r will be built (see Figure 2). Therefore, $m = \gamma_r$ and the corresponding β_{ij}^k multipliers are used in inequality (11) and in the formulation of the mathematical model. A solution is then obtained using a linear programming algorithm.

Step 2. Suppose that the solution generated under these conditions yields $m = \lambda_{s1}$, and that $\gamma_r \geq \gamma_{s1} \geq \gamma_{r-1}$. Then, the β_{ij}^k used were the correct ones and this is an optimal solution for the constraints imposed.

If on the other hand, $\gamma_{s1} < \gamma_{r-1}$, the β_{ij}^k used are incorrect and the solution is *not* consistent with its associated construction costs. Proceed to step 3.

Step 3. Now, assume that the total number of floors $m = \gamma_{r-1} - 1$, i.e., the maximum value on the next lower interval. Use this value of m to select the β_{ij}, formulate the model, and solve using linear programming. Two possible results follow:

(1) The solution yields $m = \gamma_{s2}$ and $\gamma_{s2} < \gamma_{r-2}$. In this case, repeat step 3 for the next lower interval.
(2) The solution yields $m = \gamma_{s2}$ where $\gamma_{r-1} > \gamma_{s2} \geq \gamma_{r-2}$. Then, the optimal solution for the model has been obtained and $m = \gamma_{s2}$.

5. Many other constraints can be imposed upon the objective function to make the problem conform to reality as much as possible. In writing the constraints, however, one must keep in mind that they must be linear, if the problem solution is to be obtained with a linear programming algorithm, and one must be careful not to write lineary dependent and/or redundant constraints, for they can be a source of error and inefficiency in carrying out the solution.

Optimization procedure

The model developed in the previous section is a linear one in that a linear objective function is subjected to linear constraint inequalities and equations. It fits within the framework of problems which can be optimized with "Linear Programming" methods,[3,4] for which computer programs are generally available.

It can be shown that, after the introduction of slack and artificial slack variables, the matrix of the structural coefficients must have the same rank as the matrix formed by augmenting the previous matrix with the column vector of stipulations, for the system to have basic solutions. This is a good check on the constraint equations, and should be performed before embarking upon the task of solving the problem.

CONCLUSIONS

The architectural planning model presented in this paper allows the designer to make rational decisions based on the information provided by the optimal solutions generated. It must be emphasized that the approach proposed by the writers asserts the importance of the role played by the architect-planner in shaping the urban environment. Even with a systematic method such as the one described, he must be, at all times, deeply involved in the formulation of the model, especially at the "design constraint" level, for the linear model presented is a full-fledged "simulation machine," sensitive to the settings the designer gives it and responsive to the reactions of the system's economic health. A simple, but descriptive, functional example problem is given in the Appendix.

REFERENCES

1 R J AGUILAR
Decision making in building planning
Computers in Engineering Design Education College of Engineering The University of Michigan Ann Arbor Vol III pp III-26 to III-33 April 1 1966

2 R J AGUILAR
The mathematical formulation and optimization of architectural and planning functions
Division of Engineering Research Louisiana State University Baton Rouge Bulletin No 93 1967

3 W W GARVIN
Introduction to linear programming
McGraw Hill Book Company Inc New York 1960

4 R W LLEWELLYN
Linear Programming
Holt Rinehart and Winston New York 1964

APPENDIX

A functional example

An individual wishes to develop a 250 ft. × 400 ft. commercial piece of property located at the intersection of 2 major traffic arteries.

A. Market studies indicate that for a development to be rentable, available office space must not exceed 2 floors, stores must not exceed 1 floor and living units, 3 floors for a "walk-up" facility.

B. The studies further indicate that within the sector of the city where the property is located, offices have a 20% vacancy rate, stores a 5% vacancy rate and apartment units a 10% vacancy rate (occupancies must not be less than 70%, to obtain financial support of a funding agency).

C. Data on market rental preferences reveal that facility areas of individual types and qualities should not exceed the following percentages of total building area:

TYPE	QUALITY	% OF TOTAL BUILDING AREA
1. Office	1	10
	2	40
	3	25
2. Stores	1	30
3. Apartments	1	40
	2	30

TABLE I – Rentability

D. Zoning regulations for off-street parking are:
   a. Offices – 1 parking space/ea. 200 sf of bldg. area.
   b. Stores – 1 parking space/ea. 1,000 sf of bldg. area.
   c. Apartments – 1 parking space/ea. 800 sf of bldg. area.

   Further – building coverage shall not exceed 30% of total lot area. Allow 400 sf/car for parking space, drives, etc., and a total of 5,000 sf for landscaping.

E.

TYPE	LEVEL	QUALITY 1	2	3
1. Offices	1. Ground Fl'r	4.50	4.00	3.50
	2. Second Fl'r	4.00	3.50	3.00
2. Stores	1. Ground Fl'r	3.00	--	--
3. Apartments	1. Ground Fl'r	3.00	2.50	--
	2. Second Fl'r	3.00	2.50	--
	3. Third Fl'r	2.50	2.00	--

TABLE II – Rent ($/sq.ft./yr.)

F. The $(\beta_{ij}^{k} c_{ij}^{k})$ coefficients are given below in tabular form.

TYPE	QUALITY	TOTAL NUMBER OF FLOORS IN BUILDING: 1 LEVEL			2 LEVEL			3 LEVEL		
		Ground Fl'r	:	:	Ground Fl'r	Second Fl'r	:	Ground Fl'r	Second Fl'r	Third Fl'r
1. Offices	1	20.00	--	--	20.00	19.00	--	19.00	18.00	--
	2	18.00	--	--	17.00	16.00	--	17.00	15.00	--
	3	15.00	--	--	14.00	14.00	--	13.00	13.00	--
2. Stores	1	13.00	--	--	12.00	--	--	11.00	--	--
3. Apartments	1	20.00	--	--	19.00	18.00	--	18.00	17.00	17.
	2	17.00	--	--	17.00	16.00	--	16.00	15.00	14.

Cost of parking area = $0.75 per sq. ft.

TABLE III – Construction cost, including support space ($/sq.ft.)

G. Miscellaneous Costs:
   a. Architectural-Engineering fees – 6% of total construction cost.
   b. Contingencies – 1.5% of total construction cost.

c. Movable furniture –
   1. Offices – 2% of total construction cost.
   2. Stores – 1% of total construction cost.
   3. Apartments – 4% of total construction cost.

d. Soil's report – $3,000

e. Supervision of construction – $2,000.

H. Budget: Shall not exceed $800,000, excluding cost of land.
Question: What types of units and how many square feet of each shall the investor develop to maximize the rent under present market conditions?

The model

Let:

$$i = \begin{cases} 1 \rightarrow \text{Offices} \\ 2 \rightarrow \text{Stores} \\ 3 \rightarrow \text{Apartments} \end{cases}$$

$$j = \begin{cases} 1 \rightarrow \text{Ground Floor} \\ 2 \rightarrow \text{Second Floor} \\ 3 \rightarrow \text{Third Floor} \end{cases}$$

$$k = \begin{cases} 1 \rightarrow \text{Quality 1} \\ 2 \rightarrow \text{Quality 2} \\ 3 \rightarrow \text{Quality 3} \end{cases}$$

Then, from A and E,

$$x_{13}^k = 0, \quad k = 1, 2, 3;$$
$$x_{21}^k = 0, \quad k = 2, 3,;$$
$$x_{2j}^k = 0, \quad j = 2, 3; \qquad k = 1, 2, 3,;$$
$$x_{3j}^3 = 0, \quad j = 1, 2, 3.$$

From B,

$$q_{1j}^k = .20 < .30, \quad j = 1, 2, 3; \quad k = 1, 2, 3,. \quad \text{OK.}$$

$$q_{2j}^k = .05 < .30, \quad j = 1, 2, 3, \quad k = 1, 2, 3. \quad \text{OK.}$$

$$q_{3j}^k = .10 < .30, \quad j = 1, 2, 3; \quad k = 1, 2, 3. \quad \text{OK.}$$

From B, E and Equ. (1),

OBJECTIVE FUNCTION:

Max E(R)

$$= .80\,(4.50 \times_{11}^{1} + 4.00 \times_{11}^{2} + 3.50 \times_{11}^{3} + 4.00\, x_{12}^{1} + 3.50\, x_{12}^{2} + 3.00\, x_{12}^{3}) + .95\,(3.00 \times_{21}^{1}) + .90\,(3.00 \times_{31}^{1} + 2.50 \times_{31}^{2} + 3.00 \times_{32}^{1} + 2.50 \times_{32}^{2} + 2.50 \times_{33}^{1} + 2.00 \times_{33}^{2}). \qquad (12)$$

CONSTRAINTS:

I. The lowest construction cost is, from F, $11.00 per sq. ft. Therefore, an upper bound to the total building area is 800,000/11.00 ≅ 74,000 sq. ft. Therefore,

$$x_{11}^1 + x_{11}^2 + x_{11}^3 + x_{12}^1 + x_{12}^2 + x_{12}^3 + x_{21}^1 + x_{31}^1 + x_{31}^2 + x_{32}^1 + x_{32}^2 + x_{33}^1 + x_{33}^2 \leq 74{,}000.$$

When slack variable x_1 is introduced, obtain,

$$x_{11}^1 + x_{11}^2 + x_{11}^3 + x_{12}^1 + x_{12}^2 + x_{12}^3 + x_{21}^1 + x_{31}^1 + x_{31}^2 + x_{32}^1 + x_{32}^2 + x_{33}^1 + x_{33}^2 + x_1 = 74{,}000. \qquad (13)$$

Notice that the total building area is 74,000–x_1.

II. Market preferences.
From C, and introducing slack variables x_2 through x_7, obtain,

$$x_{11}^1 + x_{12}^1 + .10x_1 + x_2 = 7{,}400 \qquad (14)$$

$$x_{11}^2 + x_{12}^2 + .40x_1 + x_3 = 29{,}600 \qquad (15)$$

$$x_{11}^3 + x_{12}^3 + .25x_1 + x_4 = 18{,}500 \qquad (16)$$

$$x_{21}^1 + .30x_1 + x_5 = 22{,}200 \qquad (17)$$

$$x_{31}^1 + x_{32}^1 + x_{33}^1 + .40x_1 + x_6 = 29{,}600 \qquad (18)$$

$$x_{31}^2 + x_{32}^2 + x_{33}^2 + .30x_1 + x_7 = 22{,}200 \qquad (19)$$

III. Site building area.
From D, in Eq. (5), and with slack variable x_8,

$$x_{11}^1 + x_{11}^2 + x_{11}^3 + x_{21}^i + x_{31}^i + x_{31}^2 + x_8 = 30{,}000. \qquad (20)$$

IV. Parking.
From D, in Eq. (6), and introducing slack variable x_9, form,

$$3.00\,(x_{11}^1 + x_{11}^2 + x_{11}^3) + 2.00\,(x_{12}^1 + x_{12}^2 + x_{12}^3) + 1.40\, x_{21}^1 + 1.50\,(x_{31}^1 + x_{31}^2) + .50\,(x_{32}^1 + x_{33}^2 + x_{33}^1 + x^2 + x_9 = 95{,}000. \qquad (21)$$

V. Design Decisions (Massing Study).
(All floors must be approximately the same size)
First floor area approximately equal to second floor area.

$$x_{11}^1 + x_{11}^2 + x_{11}^3 + x_{21}^1 + x_{31}^1 + x_{31}^2 = x_{12}^1 + x_{12}^2 + x_{12}^3 + x_{32}^1 + x_{32}^2 + e_1.$$

First floor area approximately equal to third floor area,

$$x_{11}^1 + x_{11}^2 + x_{11}^3 + x_{21}^1 + x_{31}^1 + x_{31}^2 = x_{33}^1 + x_{33}^2 + e_2.$$

Let $e_1 = e_2 = 1000$ to avert degeneracy. This means the second and third floor areas will be within 1000 sq. ft. of the first floor area. One can write:

$$x_{11}^1 + x_{11}^2 + x_{11}^3 + x_{21}^1 + x_{31}^1 + x_{31}^2 - (x_{12}^1 + x_{12}^2 + x_{12}^3 + x_{32}^1 + x_{32}^2) = 1000. \quad (22)$$

and,

$$x_{11}^1 + x_{11}^2 + x_{11}^3 + x_{21}^1 + x_{31}^1 + x_{31}^2 - (x_{33}^1 + x_{33}^2) = 1000. \quad (23)$$

Artificial slack variables must be introduced into eq.'s (22) and (23) before proceeding to optimize.

VI. Budget.

Under the assumption that the total number of floors in the building will be three (3), the budget constraint, from F, G, H, ineq. (11) and introducing slack variable x_{10}, can be written as follows:

$$1.075\{1.02(19\,x_{11}^1 + 17\,x_{11}^2 + 13\,x_{11}^3 + 18\,x_{12}^1 + 15\,x_{12}^2 + 13\,x_{12}^3) + 1.01(11\,x_{21}^1) + 1.04(18\,x_{31}^1 + 16\,x_{31}^2 + 17\,x_{32}^1 + 15\,x_{32}^2 + 17\,x_{33}^1 + 14\,x_{33}^2)\} + .75\{2.00(x_{11}^1 + x_{11}^2 + x_{11}^3 + x_{12}^1 + x_{12}^2 + x_{12}^3) + .40(x_{21}^1) + .50(x_{31}^1 + x_{31}^2 + x_{32}^1 + x_{32}^2 + x_{33}^1 + x_{32}^2)\} + x_{10} = 795{,}000. \quad (24)$$

The model is complete and a linear programming maximization of objective function (12) subjected to constraint equations (13) through (24) can now be performed. Because there are only 12 constraint equations, any basic feasible solution, including the optimal one, cannot contain more than 12 non-zero variables. This limitation could be removed by introducing additional constraint conditions.

In accordance with the budget algorithm given in the paper, if design decision constraints are removed and the optimal solution shows that

$$x_{33}^1 = x_{33}^2 = 0, \quad (25)$$

the total number of floors could not exceed two (2). Therefore, set $x_{33}^1 = x_{33}^2 = 0$ in the model and formulate the budget constraint equation as follows:

$$1.075\{1.02(20\,x_{11}^1 + 17\,x_{11}^2 + 14\,x_{11}^3 + 19\,x_{12}^1 + 16\,x_{12}^2 + 14\,x_{12}^3) + 1.01(12\,x_{21}^1) + 1.04(19\,x_{31}^1 + 17\,x_{31}^2 + 18\,x_{32}^1 + 16\,x_{32}^2)\} + .75\{2.00(x_{11}^1 + x_{11}^2 + x_{11}^3 + x_{12}^1 + x_{12}^2 + x_{12}^3) + .40(x_{21}^1) + .50(x_{31}^1 + x_{31}^2 + x_{32}^1 + x_{32}^2)\} + x_{10} = 795{,}000. \quad (26)$$

In a similar manner, if the new optimal solution shows:

$$x_{12}^1 = x_{12}^2 = x_{12}^3 = x_{32}^1 = x_{32}^2 = 0, \quad (27)$$

the building can have only one (1) floor. Hence, set $x_{12}^1 = x_{12}^2 = x_{12}^3 = x_{32}^1 = x_{32}^2 = 0$ in the model and the budget constraint becomes,

$$1.075\{1.02(20\,x_{11}^1 + 18\,x_{11}^2 + 15\,x_{11}^3) + 1.01(13\,x_{21}^1) + 1.04(20\,x_{31}^1 + 17\,x_{31}^2)\} + .75\{2.00(x_{11}^1 + x_{11}^2 + x_{11}^3) + .40(x_{21}^1) + .50(x_{31}^1 + x_{31}^2)\} + x_{10} = 795{,}000. \quad (28)$$

The budget algorithm decribes the procedure to follow in handling cost coefficient multipliers.

When design constraints are relaxed, the *optimum optimal solution* is obtained.

SOLUTION

The first run, *with no design constraints,* yielded the following data.

At a gross expected profit of $140,570 per annum, build:

12,536 sq. ft. of Quality 3, ground floor
OFFICES
16,700 sq. ft. of Quality 2, second floor
OFFICES
15,043 sq. ft. of Quality 1, ground floor
STORES
5,865 sq. ft. of Quality 1, second floor
APARTMENTS

This 2=level solution was obtained using construction costs for a 3=level complex. It is therefore necessary to change the cost coefficients in the budget constraint. The third level variables were assigned large, positive cost coefficients in order to drive them out of solution.

The second run, *with no design constraints,* yielded:

At a gross expected profit of $132,640 per annum, build:

4,348 sq. ft. of Quality 2, ground floor
OFFICES
11,660 sq. ft. of Quality 3, ground floor
OFFICES
12,711 sq. ft. of Quality 2, second floor
OFFICES
13,992 sq. ft. of Quality 1, ground floor
STORES
3,929 sq. ft. of Quality 1, second floor
APARTMENTS

This two-level configuration constitutes the optimum-optimal solution with which all other optima (subject to all design constraints) must be compared.

It is important to note that the optimum-optimal solution established a building area of 30,000 sq. ft. on the ground floor and 16,640 sq. ft. on the second floor.

The third run, for which a design constraint was introduced into the model (that the ground floor area must be approximately equal to that of the second floor), yielded the following space allocation:

For a gross expected profit of $132,126 per annum, build:

9,856 sq. ft. of Quality 3, ground floor
OFFICES
2,506 sq. ft. of Quality 1, second floor
OFFICES
18,712 sq. ft. of Quality 2, second floor
OFFICES
1,671 sq. ft. of Quality 3, second floor
OFFICES
14,035 sq. ft. of Quality 1, ground floor
STORES

Due to *one* decision, the designer is forced to relinquish approximately $500/annum in profit and to re-allocate spaces so that a local optimum may be achieved subject to the constraint imposed upon the building configuration.

Various other design decisions can be incorporated with the same ease. This completes the presentation of the example problem.

Standards for user procedures and data formats in automated information systems and networks

by JOHN L. LITTLE
National Bureau of Standards
Washington, D. C.
and
CALVIN N. MOOERS
Rockford Research Institute
Cambridge, Massachusetts

INTRODUCTION

At the present time, a low-cost, suitably connected teletypewriter, and a telephone call, is the "passport" that can permit a person to make direct contact with any of more than two dozen computer-based information storage and processing capabilities within academic and research establishments scattered across the country. By the same method of access, one can in addition make contact with at least as many commercial services offering similar capabilities. By any reasonable estimate, there are now (Spring 1968) in operation more than two thousand such teletypewriter units which are being used by students, faculty, scientists, engineers, secretaries, and administrators. New accessible computer facilities, both academic and commercial, are being announced with regularity. In addition, large projects, both privately and governmentally sponsored, are under way with the purpose of creating vast topical information stores with associated processors. An important part of some of these plans is the linking of the stores into large networks, both for the exchange of information among the stores, as well as for presentation of the information to, and service to, the directly-connected ultimate user.

The rate of growth of such facilities is very high, and it will continue to be high because of the exceedingly favorable user response to this mode of service. As recently as 1961, essentially no facilities of this kind were in operation. Yet, during just the past three years, the number of accessible processors, and the number of users, has increased by at least thirtyfold. Such a rate of growth represents a tenfold increase every two years. Because of the evident enthusiasm of the users, the size of the untapped population, and the potentials of this kind of service, we can probably expect this rate of increase to continue unabated for at least the next five years, before the saturation effects begin to take over. If these predictions are fulfilled (and they may be exceeded), then by 1972 there will be in the order of 15,000 accessible automated storage and processor complexes, both large and small. Also, there will be in use something in the order of 300,000 on-line terminal devices of various kinds, each permitting connections to automated remote systems over spans ranging from one room to the next, up to long-haul transoceanic connections by radio satellite.

These numerical predictions may be compared to the more than 4 million electric office typewriters now in use; about 80,000 commercial teletypewriter instruments operating in business point-to-point communications; and about 50,000 electronic computers, both large and small, now installed.

This happy picture of such widespread user acceptance and the growth of a new technology is seriously flawed by the Babel of differing languages and control methods in use. Once a telephone connection to a remote automated storage and processor unit has been established, the user is absolutely helpless unless he is thoroughly familiar with the particular keyboard rituals and incantations required to elicit performance from the specific remote machine. It is safe to estimate the knowledge of at least 30 to 40 different languages and rituals would now be required for operation of all the 50 or so presently accessible automated systems. In each contact, the poor user must know which language and ritual to employ before he can establish even the most minimal communication with the re-

mote machine. For example, should he type "HELLO" or should he press the BREAK key to get the computer's attention? Should he use the CARRIAGE RETURN or the ALT MODE key to request the computer to act? And so on, for something like a dozen basic control functions.

The present chaotic situation is rapidly growing worse as more systems are brought into operation. For each new system, the local computer programmers take it as their prerogative to concoct yet another unique language and method of control. This is a direct result of lack of standards subscribed to by the users to provide compatibility, as well as a lack of administrative guidance. If such proliferation continues, unchecked by any plan or guidance, it is predictable that by 1972 there will be hundreds, or more likely thousands, of differing control methods and languages. If so, a substantial part of the great advantage of widespread communications and of direct access to the automated information systems will have been needlessly destroyed.

This is not a unique situation. History is merely repeating itself. The North American railroads could not merge into a true system until they had undone a great proliferation of track gauges, couplers and brakes. Likewise, the power companies could not link together until they had abandoned a variety of powerline frequencies. Telegraph and telephone followed a similar course. A nationwide highway system depended upon undoing many local signalling and road marking conventions.

In 1968 we are confronted with the ingredients of information systems, and again, the existing confusion must be undone to some extent, and a set of interface and operating standards must be agreed upon, in order to provide a viable system of information networks.

The purpose and orientation of this paper

The purpose of this paper is to attempt to find a way out of this chaotic situation by the development of standards for user control procedures and standards for data formats to be used in automated information systems and networks. A goal is to make it possible for a person sitting before the keyboard of his on-line teletypewriter or display console to use a standard method for establishing communication with, and using, a remote automated information storage and processing system. Another goal is to provide him with a standard method for control of format and the interpretation of the output of such systems.

The method of approach is to examine the complex environment confronting the person making contact with and using a remote automated information processing system, and from this examination to discover the fundamental functional characteristics of the situation. From the study of the logical properties of these functional characteristics, it has been possible to formulate a proposed course of action that can lead to workable standards for use in automated systems.

Two main areas are treated. The first area has to do with the problems of elementary user control procedures and methods for gaining access to and control of automated data store and computer systems. It is primarily concerned with the possibility of standardizing a set of certain elementary keyboard actions. It stops short of undertaking a detailed study of all user command languages (e.g., BASIC, QED, or QUICKTRAN).

The second area is concerned with the format in which data and other information are presented to the user, or in which it may be interchanged between cooperating automated information systems.

An important constraint has been to formulate a course of action which would provide users with a set of basic control methods and procedures, and methods for dealing with data formats, yet which would not destroy the investment in locally-established procedures or data stores. The goal of satisfying this constraint appears to have been achieved.

The results of the study in a nutshell

In brief, the results of the initial part of this standardization study are as follows. The fantastic variety of different keyboard control actions presently being used is so great that any effort of standardization by seeking a consensus is out of the question. However, it is found that there are only about twelve logically distinct elemental control actions which are of crucial importance to the user when he initially enters an automated information system. These elemental logical actions can be standardized as to function and can be given standard keyboard assignments. Then the user, by invoking these standard actions, can have access to the full control resources of the rest of the system. These twelve control actions are shown in Appendix I, along with suggested keyboard assignments.

The necessity for user group action

Because the technology of automated information systems is still so very new, the number of people who are now actually using on-line terminals and automated information systems is still very small. However, they already begin to provide a cross section of the great number of users of the future. These present users—since they can now realize what some of the problems are—hold a great obligation to the users of the future. It is their obligation to begin immediately to take the steps that are necessary to protect the vital

interests of themselves and future users, and to do so before the present chaotic situation grows larger or becomes permanent (as it could!).

Historically—that is, since computers first came into being—any layman user of a computing machine found himself cast in a role which was inferior to that of the machine. The layman users have had to conform to the requirements set by those in charge of the machines. When questions arose affecting the interests of the users, it was the convenience of the machine (speed and efficiency, or inherent limitations) which were cited as providing the basis for the decisions taken. The users had essentially no recourse, since the machines were in fact rare, expensive, somewhat limited in capability—and most important, under the control of another group.

A change in orientation is now justified. Processing machines are now available in abundance. They have speeds and power of a magnitude only hoped for ten years ago. Auxilliary facilities (disk stores, displays, communication) have had a tremendous improvement. As a result, we should now take a new look at the ways these machines are applied. More important, we should reconsider the manner in which we think about ourselves in respect to the machines.

In comparison to the machines, the rare and valuable commodity has now become the person doing a job. The acknowledged purpose of the large scale automated information system is to provide service to people—to provide them with information, and to arrange and rearrange the information that they bring to the system.

The on-line terminal and automated systems

Our thinking about an automated information system should begin with the "user," the person sitting down at the on-line terminal keyboard. When not involved with an active connection to some automated system, the on-line terminal behaves much like an ordinary typewriter. One makes it go by touching keys, and the result is line after line of printing on the page, or display on the screen.

An on-line terminal can be thought of as a typewriter that permits one to "do things" at the keyboard—to do things of a most general sort. One can store text, edit it, arrange it, print it out, search for and retrieve information, perform computations, transmit text to another location, and do many other things—limited only by the range of facilities accessible to the system.

Access to automated information systems will also be performed by a wide variety of devices with keyboards, in addition to typewriter-like devices. Thus consideration must also extend to such things as TV-type presentations or displays, to various photographic processes and printing methods, various high speed methods of printing, and the like. However, it is probable that typewriter-like devices will be the most widely used devices in the predictable early future. For this reason, and also for the reason that most persons are familiar with the typewriter, the typewriter will be used as the most useful point of departure in considering standards for user procedures in automated information systems.

The user system interface

The confrontation between the person and the automated system comes at the user's fingertips and at the user's eyes. The user is concerned with what he must do with those fingers to perform the actions that he desires to accomplish. Similarly, he is concerned with the interpretation to be given to the text which is presented on a page before his eyes.

The confrontation at the user's fingers encompasses the area of "standardization of user procedures," since it is through such user procedures that the action to be taken by the system is determined.

The confrontation at the user's eyes encompasses the area of "standardization of data formats," since the form and arrangement in which the text is presented on the page is most important for his understanding of what any particular passage of text means.

It is at this user-system interface that we are particularly concerned: first with the logical or functional characteristics of the system, i.e., what the system can do for the user; and second with the manner in which the user tells the system to do something for him. Throughout, we must view the behind-the-scenes technology (whether electronic, computer, or communication) as being in a *secondary service role* to the user, whoever and wherever he may be.

Focus upon details or upon principles?

Most standardization efforts, in the ordinary course of affairs, are concerned with achieving and stating a consensus with regard to a relatively large set of particular details, technical aspects, physical dimensions, codes, numbers, material compositions, procedures, and the like. This course cannot be followed here. There is no basis for such kind of consensus based on overt detail. Already there are several dozen data storage and computer projects, and they are characterized by vastly different details in their techniques for accomplishing even the simplest (which are the most important) things. If one has even the briefest conversation about these matters with a custodian of one of the presently operating automated systems, it will disclose how completely he realizes the impossibility (and undesirability) of achieving a useful

standard based upon a consensus of details as practiced at the various automated systems.

This kind of impediment to standardization prevails in both parts of this standards study. In the "user procedures" area, consensus is impossible because of the great variety of control methods used for character delete, transfer of control, call for help, and other actions. In the "data formats" area, the possible number of usable formats for data is infinite. Thus again, consensus based upon detail is not possible.

The only basis for achieving the consensus required for cooperation and standardization is through the focus of attention to another level, a level at which consensus is possible.

In user procedures, we must focus our attention upon the various *elements of logical control* which are accomplished, and not upon the detailed key action, button pushing, or other ritual used to initiate each element of control action. Most of the elements of logical control are present in all systems, in one form or another. Therefore it is possible to discover, isolate, and identify these basic logical elements. Then, with a list of such recognized logical control elements available for discussion, it is reasonable to expect that a consensus upon them can be reached, and that a given set of them can be chosen as being required and necessary for user control of automated systems. After a consensus is available on such a common set of logical control elements—then and only then—will it be appropriate to consider choices for standard keyboard methods of accomplishing the different logical control actions.

A surprising outcome of this study is that it appears that only about twelve basic logical elements are sufficient when used in the proper environment.

A similar situation prevails in the area of data formats. There is no hope for any success through the cataloging of formats to be used, and seeking a consensus upon them. Yet, it does appear that a consensus can be developed for a standard method of describing formats for data. Each specific use of data can then have a standard and commonly understood description of its format. Through this description, the meaning, or purpose, of each segment of text or data can be understood by the user. The same standard data format descriptions can also be the basis from which the automated systems will be able to manipulate and rearrange the data and place it in more useful forms.

User procedures vs. programming languages

In this study of standardization, it is necessary to have a clear distinction between "user procedures" and "programming languages." By "user procedures" we are concerned with such elemental control actions as: how to correct typing errors while communicating with the system; how to get the attention of the remote automated system; how to get assistance in the use of the automated system; how to take overriding control to stop undesired actions; and so on. We are also concerned with the procedural environment in which these commands are given.

In contrast, "programming languages" are concerned with formal languages systems, such as FORTRAN, ALGOL, or COBOL, which are used for writing out a step-by-step description of the series of actions to be taken by a computer in carrying out some computation or manipulation. A complete prepared description of this kind is what is known as a "program." Associated with programming languages will often be means used by the programmers and technicians for writing, correcting, filing, and maintaining such programs.

Programming languages were developed for an entirely different purpose than that of serving the person working at an on-line terminal communicating with the ordinary automated information system. Such programming languages are mainly for the technicians. For this reason, the conventions and habits that have been built up around these formal computer programming languages do not necessarily have any direct application to user control procedures. Indeed, some of the conventions of these languages are inherently bad for our purposes. Therefore we should be most cautious about importing conventions from these languages (without careful examination) into standards for user procedures.

There is an intermediate class of languages which deserves careful consideration as to how they might fit into this program. These languages are in three groups: (1) the on-line numerical computational languages specifically oriented to layman users, exemplified by BASIC, JOSS, and QUICKTRAN; (2) the on-line editorial languages for text preparation and correction, exemplified by QED, TYPESET and RUNOFF, and ATS; and (3) the general on-line teletypewriter control languages, such as TRAC. Each of these languages is of tremendous interest to the user. For various reasons, it is undesirable to attempt to include them at this time within this standardization study. Later, it may be very desirable and appropriate to nominate certain of these languages for consideration for future standardization. Until then, we should observe how the proposed standards for "user control" and "data formats" impact upon these specific on-line languages. As a result of such impact, it may be desirable to formulate requirements for minor modifications of the nominated languages in order to make their control features more compatible with the standard user control procedures.

In further explanation of what is meant by "user procedures," the following situation should be contemplated: A person (meaning a person not expert in either programming or in computers) comes to an online terminal keyboard. This keyboard may be different from the one with which he is familiar. The system into which he makes a connection may be different from the one on which he has had his training. Frequently he will be connected into a remote automated system which he has never used before. Such is the situation. In seeking standards for user procedures, we desire to provide a standard course of action which will allow such a person to use the new keyboard, to use the local facilities, and to use the remote automated facilities to perform the work he needs to do. Attention must therefore be focussed upon all of these aspects of the situation.

The concern must be with respect to what the user will do, and how he can be guided in his actions in using the individual elements of the environment provided to him. We are overwhelmingly interested in the logical requirements of the user's task (and not of the logic of the machine). We are concerned with how the user's actions may be delimited by his human physical or mental capabilities and limitations. If there are discrepancies or shortcomings (as there may well be) in the presently available hardware or in present software facilities, these discrepancies should be discovered and examined. Future hardware and software must accommodate the user's logical requirements. Biological evolution is simply too slow to make the user accommodate!

Formats—external vs. internal

Data formats and file structures have long been a very esoteric matter. Part of their complexity and mystery arose from the historic development of computer hardware and storage media. Data formats and file structures were based upon the artifacts of particular computers and upon the 80-column IBM tabulating card. Other aspects of the complexity can be explained by the early limitations, both in size and generality, of the available equipment. These limitations led the programmers to develop many complex and ingenious techniques to accomplish their tasks at hand.

The limitations in hardware, from these early days in computer history, are now rapidly disappearing—particularly the limitations in lack of storage space and speed of operation of the processors. Unfortunately, the programmers have not in general used these technological advances to make things simpler. Quite the contrary. Contemporary programming and use of the systems seems now to involve more complexity than ever before. What is worse, some of these elements of complexity often obtrude into the user area, limiting the user in what he can do, and requiring the user to know unnecessary details about the internal structure of the system. This requirement on the user extends into the area of data formats and file structures.

In this study of standards, in its concern with data formats, a thorough-going attitude is taken of separating the *external* representation of the data from all matters of *internal* representation. The external representation is of paramount interest to the user. It is the one which he sees upon the piece of paper, or he sees displayed on the TV screen. The external representation is also the one that he must employ when he is entering new data (text or numbers) into the system. It is also the representation that he must picture in his mind when he operates on (edits, selects, or rearranges) the stored data by means of his keyboard actions.

In contrast, the internal representation of data is concerned with the manner in which particular computers and other hardware store and move the data around inside the system. This is not the user's province, and there should be no imposition upon him as to how it is done. For example, it should be of no concern to him whether the hardware represents his data by means of 6-bit, 8-bit, or 64-bit units within the computer. He should have no concern with "packing" and "unpacking" of characters within the computer words. He should not be troubled with physical file units. These are all aspects of the supporting technology, i.e., of the hardware and the programs.

By "external representation" is meant the manner in which the data is presented to the user—the arrangement of the data in lines or columns, or the like. Thus the user is concerned with the various separators (such as spaces, or new lines) that determine the format of presentation. When data is arranged in a known format, it is easy to know what a data element means (e.g., whether the name of a person or the name of a street). In general, the format of presentation may include indentations, columnar arrangements, punctuation, special markers, and the like. Because files may be extensive, the user must also be able to command the display (or printout) of only a part of some file. In this case, he must be able to give a command in terms of the external representation of the data. His command may depend either upon format (e.g., "show the next line") or upon the data element (e.g., "type the address"). A technique for data formats must therefore be provided which will permit a user to give such commands for the handling of the data.

The techniques for external representation will have a wider application than merely facilitating what the local user sees and does. They can provide a standard

format description and control method for the *general* interchange of data between automated information systems. In data interchange between automated systems, the machine at the receiving end is faced with some of the same problems of interpretation and control that a user faces. Thus the solution of the problem for the human user—as a byproduct—provides a *machine-independent* solution for the interchange of data between systems.

In this sense, it is appropriate to consider that "external data representation" should deal with all matters of data *representation for the user* when the data are written on any display medium or interchange medium. By "interchange medium" we must include wire transmission, paper tape exchange, magnetic tape exchange, or any other form of machine readable record interchange. For many of these interchange media, standards of some sort already exist—but they are not standards which apply to the format of the data as the user sees it. Instead, the standards for the "internal representation" in general permit any user "text" to be written, and the user is concerned only with the manner in which such text can be structured and put in format for his external use. In consequence, these two areas—"internal" or machine representations and formats, and "external" or user text representations and formats—can be considered quite separate. In fact, it is important that such separation be preserved, and that separate standards be developed for the two distinct domains. If this separation is maintained, then the work on external data formats will not collide in any way with the requirements and standards for message transmission, for magnetic tape records, or for other media representations.

Standard description for external data formats

Since data formats have a potentially infinite variety, it is futile to consider the standardization of the formats themselves. This is not attempted. The effort is directed first to discovering what constitutes the necessary logical structure behind formats in general. To be specific, an external data format, in the sense used here, consists of text segments separated by various spacers or syntactic markers. These provide the clues to the meaning to be attached to the different text segments. The markers do not act in isolation: their meaning must be stated somewhere. The formal statement of the manner of the use of the syntactic markers, and how they are used to create the total format and to specify the content of the text segments is called a "data description."

Data descriptions *can* be standardized, and this is one of our goals. A logical treatment of the elements of format generation has been developed, and from this suggestions for a standard method of description of external data formats is outlined in Part IV of the Reference Reports.

ACKNOWLEDGMENT

This paper is based in part upon the series of referenced reports prepared under contract by C. N. Mooers for the National Bureau of Standards in 1967. That contract was inspired in part by the insight and dynamic persuasiveness of Dr. Chalmers W. Sherwin.

REFERENCES

C N MOOERS
Standards for user procedures and data formats in automated information systmes and networks
Zator Co Cambridge Mass
Part I - *The need for standardization and the manner in which standardization can be accomplished*
July 5 1967 45 pp
Part II - *The standardizable elements of user control procedures and a unified system model*
Aug 10 1967 39 pp
Part III - *A suggested standard keyboard assignment for the elemental user control actions*
Aug 1 1967 21 pp
Part IV - *A standard method for the description of external data formats*
Aug 28 1967 32 pp
NOTE: The above reports were prepared under Contract No. CST-382 to the National Bureau of Standards

APPENDIX I—TABLE OF ASSIGNMENTS ELEMENTAL USER CONTROL ACTIONS

Description	*Mnemonic*	*Suggested Assignment**
Dialog Signals		
User Signal	*US*	; (semicolon)
Typewriter Signal	*TS*	(carriage return) (line feed) or (new line)
Stop	*ST*	(break)
Single-Character String Commands		
Delete Character	*DC*	#
Delete Line	*DL*	&
Literal Prefix	*LP*	"
Argument Separator	*AS*	, (comma)
Help	*HE*	?
Commands to The System		
Standard Mode	*SM*	SM or STD
Revert One Level	*RV*	RV
Restart (or begin)	*RS*	RS
Exit System	*EX*	EX

*These are merely suggestions for purposes of discussion. The assignments suggested in this chart are from Part III of the References.

Procedures and standards for inter-computer communications

by ABHAY K. BHUSHAN and ROBERT H. STOTZ
Electronic Systems Laboratory
Massachusetts Institute of Technology
Cambridge, Massachusetts

INTRODUCTION

In recent years there has been considerable discussion on computer communication networks,[1,2,3] information service systems[4,5] and the computer utility.[6,7] Sophisticated displays maintained by small processors connected to remote multi-access computers are also being developed.[8] Such applications involving interaction and exchange of information between computers are increasing in number and pointing to the widening need for reliable computer-to-computer communications. In order to allow communication between many arbitrary computers, a uniform agreed-upon manner for exchanging information is needed. This requires the establishment of a standard message format (i.e., character code structure and message syntax) and common communications protocol procedures (i.e., the agreed-upon manner of exchanging messages).

This paper is an attempt towards defining the needed format and protocol for inter-computer communications. A standard message format, based on USASCII* which appears to be flexible enough to cover the entire range of inter-computer communications is proposed. For the protocol procedures, only guidelines are presented because a single standard may not be feasible due to the wide differences in hardware, software, and capabilities of various machines, and the differences in user requirements and nature of operation. (It should be noted that the procedures described in this paper are recommended for general inter-computer communications and may not necessarily be the best for some particular non-generalized applications.[9])

Computer-to-computer communications

Computer-to-computer communication differs from communcation to teletypewriters and remote terminals attended by a human operator in several important respects. These come about because of the absence of the interactive human operator from the environment. When a teletypewriter is the recipient of a message, the human operator acts as a highly sophisticated interpreter and error detector/corrector. On the other hand if a computer is to act on a message received, it must perform these functions. The computer must be told the beginning and end of messages and it must do careful error checking, since its acting on a bad message can have disasterous effects. Generally, this implies retransmission systems with schemes for block error detection and message acknowledgment,[10,11] which are straightforward techniques for machines with data storage and programmable processing capabilities.

Another difference between inter-computer communications and communication to terminals attended by a human operator is that a man can seldom take advantage of very high data rate lines. A computer by contrast is very good at making decisions rapidly so that it can usually interact effectively over high-speed communication lines. In fact, communication at high speed is economical, efficient and often essential for an adequate interaction between machines. The cost of high-speed data transmission between computers is likely to be drastically reduced in the future with the introduction of digital transmission into the communications network. Transmission systems carrying all types of communications in a digital pulse stream are gradually being introduced into the Bell System.[12,13] Eventually they will be connected together in a digital hierarchy to form a nationwide digital communcations network.

Inter-computer communication also differs from communication between a computer and such devices as tapes, discs, and printers. Computers have processing capabilities that data storage devices do not possess. Thus communication need not be on a mes-

*USA Standard Code for Information Interchange.

sage-by-message basis. The rules for inter-computer communication need to be and can be more elaborate, and the communications can be made more efficient and interactive.

The present-day public facilities for data communications include the Bell System and Western Union wideband facilities.[14,15] The Bell System 303 series data sets provide communications at speeds up to 230.4 kbps on private leased lines. The data transmission facilities may be synchronous or asynchronous and full duplex or half duplex.* Synchronous transmission is more efficient as it obviates the need for start and stop elements with each character. For this reason, most computer I/O terminals have only the synchronous adapter option available for high-speed links.

The message format and communications protocol procedures described herein do not depend on the nature of the transmission network. They are recommended for full-duplex as well as half-duplex, and synchronous as well as asynchronous, transmission at speeds typically ranging from a few hundred bits per second to well in the megabits per second range.

Standardization and flexibility

Our intent in the design of the message format and communications procedures is to provide as much consistency with standards as possible, without unduly losing flexibility in operation, overall efficiency, and convenience of transmission. It is essential for this objective that all communicators employ a common standard code and be able to use the existing facilities at little or no extra cost. The obvious choice for this code was the USA Standard Code for Information Interchange[16,17] (USASCII or ASCII).* In fact most communication carriers and computer equipment manufacturers have accepted this standard proposed by the U.S.A. Standards Institute and are manufacturing equipment based on it. The general message format described in this paper follows closely the draft proposed USA Standard Communication Control Procedures,[18,19] and is intended to be compatible with equipment offered by the communication carriers and the computer industry.

There have been some questions regarding the usefulness of USASCII in a computer network environment. Alternative solutions to message formatting (incompatible with USASCII) using fixed block lengths and rigid bit coding have also been proposed.[3,9] It has been argued that it is not desirable to conform to USASCII, if more efficient alternatives can be found.

It is our feeling that besides being a proposed USA standard, USASCII represents a very good way of transmitting information. Though USASCII may not be the best for one particular application, it appears to be the most reasonable for a large variety of applications. Also, it can be used with intelligence to provide a high degree of flexibility. An alternative coding scheme will not be very much more efficient than USASCII and still be able to provide the degree of flexibility and reliability that is needed for inter-computer communications. It may be uneconomical as well as unnecessary.

In our recommendations we have adopted the basic framework provided by USASCII, and have extended it to suit the many requirements of inter-computer communications. For certain applications (such as very efficient binary text transmission) we have felt USASCII to be somewhat inadequate. Alternatives have been suggested wherever necessary and possible. It is our hope that the proposed USA standards may be modified for the inter-computer communications applications, or at least not made so rigid as to make suitable alternatives incompatible.

The message format

For clarity we shall first define some terms:

A transmission - a group of one or more *messages* which are transmitted continuously without interruption.

A message - a sequence of characters arranged for the purpose of conveying information from an originator to one or more destinations. It includes all *text* and the appropriate *heading*.

A heading - a sequence of characters which constitute the auxiliary information necessary to the communications of a text. Such auxiliary information may include, for example, characters representing routing, priority, security, message numbering and associated separator characters.

A text - a block of data that is to be transmitted as an entity from the sender to the receiver(s).

A typical message is illustrated in Fig. 1. Each character in the above message (with the exception of BCC) is a 7-bit ASCII character followed by one bit of odd parity (an odd rather than even parity is chosen to

Synchronous (or character synchronous) transmission means characters (and bits) are transmitted at a fixed rate. *Asynchronous* transmission means the interval of time between characters can vary arbitrarily. *Full duplex* is the ability to transmit simultaneously in both directions while *half duplex* is the ability to transmit in both directions, but not simultaneously.

*USASCII X3.4, 1967. The standard code is referred to both as the ASCII code and the USASCII code.

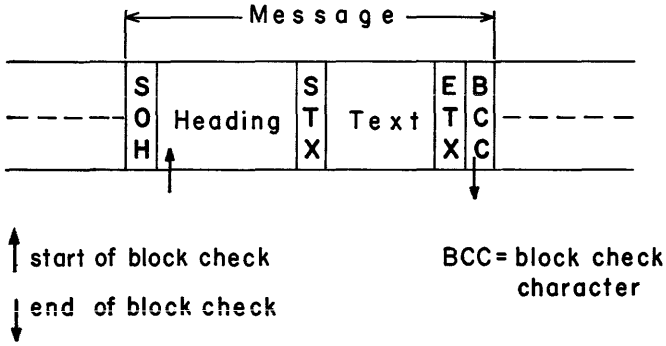


Figure 1 – Basic message format

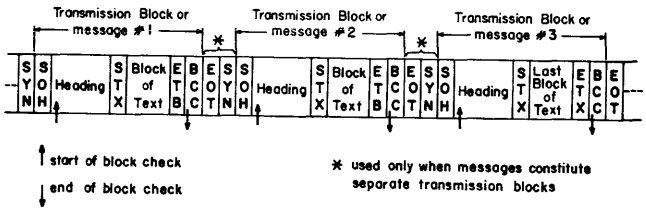


Figure 2 – Blocking of text into separate messages

facilitate bit synchronization). The least significant bit is transmitted first and the parity bit is transmitted last.[20] The characters SYN, SOH, STX, ETX, ETB and EOT are all standard ASCII communication control characters and have a fixed code value and meaning.

The message starts with an SOH (Start of Heading) followed by the heading (messages that have no heading information may start with an STX but are of questionable value in an inter-computer environment). An STX (Start of Text) signals the end of heading and the beginning of user's text. An ETX (End of Text) indicates the end of user's text and of the message. A block check character (BCC) immediately follows the ETX. (Current USASI proposals recommend the longitudinal block parity check.*)

In many systems there are limitations on maximum message (or transmission block) length due to such factors as error rates or buffer capacities. If the size of a text provided by the user exceeds the maximum message (or transmission block) size, the text may be blocked into several messages (or transmission blocks) as shown in Fig. 2. An ETB (End of Transmission Block) is used to indicate the end of all text blocks except the last (which ends with an ETX), and is also immediately followed by a block check character. The error check includes the ETX or the ETB, which ends the checked sequence, but does not include the SOH (or STX) which starts it as shown in Figs. 1 and 2.

For the purpose of message block identification and other communication protocol needs, each of the messages (or transmission blocks) should be preceded by a separate heading. Note that this is a modification of USASI standards which propose that only the first of these blocks should be preceded by a heading. Thus messages (or transmission blocks) which contain only a part of the text should be regarded as separate messages insofar as communications is concerned.

On start of transmission, SYN (synchronous idle) characters (the number may vary with equipment needs) are supplied to provide the communcations adapter with correct synchronization and indicate the start of transmission. These may also appear within the message (if required for synchronization), and may be stripped from the message on reception. An EOT (End of Transmission) signals the end of a transmission which may have contained one or more messages, as shown in Figs. 2 and 3. For half-duplex service the originator may go into the receive mode (temporarily relinquishing its right to transmit) after the transmission of an EOT. (Proposed USASI standards recommend this 'EOT' function on ETB and ETX's also. This would prohibit multiple message transmission. We recommend that only the EOT character should be used for this purpose.)

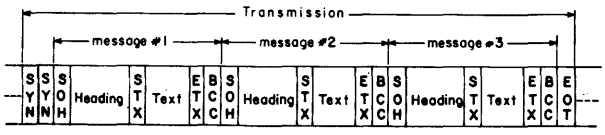


Figure 3 – Multiple messages in a single transmission

Multiple message transmissions of the type shown in Fig. 3 are important towards making the communications more convenient and efficient. A message need not be acknowledged before the next is sent, and messages can be sent continuously without interruption. For multiple message transmission capability it is important that all messages containing text and requiring an acknowledgment (for error control)

*This is an 8-bit character generated by taking a binary sum, *without carry*, on each of the 7 individual bits of the transmitted code (resulting in an even longitudinal parity). The eighth bit of the BCC is a character parity on the BCC itself, and is in the same sense as the rest of the characters (odd in our case).

contain a message identification in the heading. Messages with identifications may now be acknowledged with their identification numbers. Non-text messages containing only heading information (such as acknowledgments) may have the format shown in Fig. 4. These messages start with an SOH and end with an ETB. A BCC follows the ETB so that the heading message is error checked. For extra error protection, other messages containing text may also have their headings errors checked by the use of ETB (see Fig. 4).

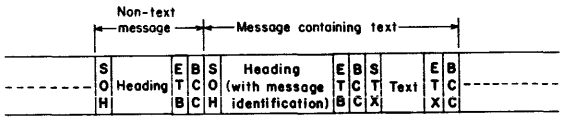


Figure 4 – Non-text messages and error checking of headings

Heading information

Information in the heading of the message is used only to aid the communications program to transmit, receive, sequence and route the messages efficiently and securely. The heading information is never seen by the user, and all information regarding the text portion of the message (e.g., should the ASCII characters in the text be interpreted as ASCII or binary) is contained in the text portion of the message.

Our scheme for coding information into the heading is to separate the ASCII characters into three categories: control characters, key characters and argument characters. As shown in Fig. 5, control characters are all those characters with octal values from 00 through 37, key characters are those with octal values 40 through 77, and argument characters are all those with octal values greater than 77.

The control characters are ASCII control characters assigned meaning only in accordance with proposed USASI standards. The key characters have a predefined meaning and may be followed by a string of characters from the third category called the argument of the key. The argument of a particular key is interpreted by the computer in accordance with the predefined meaning assigned to the particular key. The key completely defines the interpretation of the information contained in the argument. Additional separator characters are not needed, as key characters themselves act as information separators.

The key characters together with some control characters serve as heading control characters. When used for the purpose of heading control information, the standard meaning of the ASCII control characters must be preserved, thus communication control characters like ACK, NAK and ENQ are not used in the heading because there exists computer/communications hardware that is sensitive to these. Note also that proposed USASI standards disallow the use of these characters within a message. Additional key characters may be generated by using an escape character (a specified key or control character). The escape character followed by an argument assumes the meaning of a new key character. If a particular key character is not recognized by the computer software (or device hardware), it is to be ignored (together with the argument which may follow it) until the next recognizable key or control character appears.

The concept of key character with arguments provides a large capacity for expansion and great flexibility in operation, together with simple implementation. The scheme, a simple extension of the proposed USASI standards, represents an attractive alternative to hardwiring particular bits in a heading. Its advantage over the 'Escape' convention outlined in USASCII is the greater efficiency permitted by use of single characters instead of character sequences.

For illustration, the heading control characters have been divided into three classes, representing communication requirements for different applications. For example, some small machines may require only a limited class of heading control characters. Other small machines (e.g., satellite computers for display application) may require unbuffered transmission. Table I illustrates a possible assignment of key characters and their meanings. This list may be modified and extended as requirements are better understood. In Table I, the character n stands for a binary number represented by the six least significant bits of the argument character. A message of size n contains no more than 2**n characters. Since messages can be of variable length, n serves primarily as an upper bound on the maximum size of the message.

The basic Class I heading control characters include only the acknowledgment, identification, inquiry (not USASCII control characters) and a means to interrupt and quit. Messages containing identification may be acknowledged with that identification. Also, a limitation on the maximum size of the message can be indicated. Such a bound may be necessary, as some machines may have limited buffers.

Class II heading control characters are intended primarily for unbuffered transmission between machines (i.e., machine A can send data to machine B directly into its proper location in B). This is particularly important when a large time-sharing computer communicates with a small satellite computer. In

Bit Positions b7 b6 b5 →	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
b4 b3 b2 b1 ↓								
0 0 0 0	NUL	DLE	SP	0	@	P	\	p
0 0 0 1	SOH	DC1	!	1	A	Q	a	q
0 0 1 0	STX	DC2	"	2	B	R	b	r
0 0 1 1	ETX	DC3	#	3	C	S	c	s
0 1 0 0	EOT	DC4	$	4	D	T	d	t
0 1 0 1	ENQ	NAK	%	5	E	U	e	u
0 1 1 0	ACK	SYN	&	6	F	V	f	v
0 1 1 1	BEL	ETB	'	7	G	W	g	w
1 0 0 0	BS	CAN	(	8	H	X	h	x
1 0 0 1	HT	EM	)	9	I	Y	i	y
1 0 1 0	LF	SUB	*	:	J	Z	j	z
1 0 1 1	VT	ESC	+	;	K	[	k	{
1 1 0 0	FF	FS	,	<	L	\	l	|
1 1 0 1	CR	GS	-	=	M	]	m	}
1 1 1 0	SO	RS	.	>	N	^	n	~
1 1 1 1	SI	US	/	?	O	_	o	DEL

First category (standard USASCII definitions)

Control characters

Communication control characters

Second category designated as Key characters for additional controls

Third category designated as argument characters (includes the USASCII control character DEL)

Control character DEL included in the argument set

Figure 5 – Proposed designation of USASCII characters for message control in inter-computer communication

general, satellite computers will have a limited amount of memory, and it is desirable not to have to provide a large message buffer in this machine. On the other hand, if a small message buffer is provided, it will take multiple calls on the time-shared computer to get a large message across, which is also undesirable. Direct transmission is very attractive in this case. Note that the "where to" and "number of characters in this message" information must come before the text, i.e., in the heading.

a complete overhaul when a change has to be introduced.

In the long run, the key character scheme may even turn out to be more efficient than fixed bit coding for some applications, since it would not be necessary to code all of the information all the time. For example, the acknowledgments would not have a message id, priority may be assumed normal unless otherwise indicated and acknowledgments, repeat acknowl-

TABLE I – Assignment of key characters

Class	Key character		Interpretation
I	#	arg	argument is message identification
	+	arg	acknowledge message #arg (message O.K.)
	−	arg	negative acknowledge message #arg (message in error)
	?	arg	repeat acknowledgment of message #arg
	.	n	send transmissions no longer than size n
	"		interrupt (get processor's attention)
	!		quit (stop transmission of messages)
II	:	arg	sender's status
	;	arg	sender's interpretation of receiver's status
	/	arg	number of characters in text of message
	&	arg	argument is core memory address of incoming text
III	=	n	make standard buffer size n
	<	n	"=n" request rejected
	>	n	"=n" request accepted
	*	n	size n blocks of text only
	(	arg	routing information (sender)
	)	arg	routing information (receiver)
	'	arg	escape sequence to obtain additional key characters

Class III heading control characters are intended primarily to provide buffer allocation and message routing information in a network environment. Thus information such as standard buffer size, message block size and identification of source and destination computers may be included in the heading by using the appropriate key characters.

Alternative schemes suggested for coding heading information have been on a bit-by-bit basis. We feel that even though bit-coding the information may be slightly more efficient (in that there is no overhead for key characters and some information may be less than 6 bits), it will not have the flexibility of coding that the key character scheme provides. As the size and capacity of a computer network increases, or if additional functions are called for, the key character scheme can easily accommodate these by use of longer arguments and/or additional key characters. A rigid bit-coding scheme on the other hand, will need

edgments, etc., may or may not be included within the heading of the message.

Text information

The manner of transmitting text information recommended herein conforms with USASCII and the proposed USASI standards. USASCII however is principally geared towards the transmission of alphanumeric symbols, not binary data which is required in general inter-computer communication. Also, it is often desirable to be able to mix binary data and ASCII alphanumeric data within a single message.

The mechanism we have chosen to transmit binary data is to preface it in the text stream with a Group Separator (GS) character (ASCII octal 035), and to escape (back to alphanumeric text) with a File Separator (FS) character (ASCII octal 034). The binary data is encoded as the six least significant bits of ASCII characters with bit 7 always a 'ONE.' The binary mes-

sage is then treated simply as a stream of ASCII characters. It should be noted that the 64 permissible "binary" characters are identical with the argument set and none of them fall in the category of key or control characters.

The positioning of GS and FS characters in a text stream is completely independent of the message itself. Thus a single message may contain both binary and USASCII information, or any combination of the two as shown in Fig. 6. The start of text is always assumed alphanumeric unless indicated otherwise by a GS.

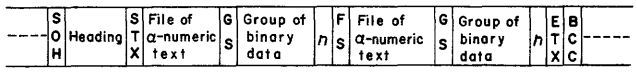


Figure 6—Transmission of binary and alphanumeric text within a message

Characters with 'ZERO' in their most significant bit may occur within binary mode of transmission and be interpreted usefully. These may be control characters or key characters. The USASCII control characters are to be used in accordance with USASI recomendations. The key characters occurring within binary mode may be used to further define the binary information that follows it and indicate what it is about. The meaning assigned to the key characters in text may be different from that assigned in the heading. Thus the text may be preceded by its own leader information set by user conventions. Another example of the possible use of key characters within binary text may be to indicate the end of binary string if it is not an even multiple of 6 bits. An ASCII numeral 'n' (1 through 5) may follow the last binary character of the string to indicate the number of meaningful information bits in that character (see Fig. 6).

Transparent text

Many people have expressed a need for a "Transparent Text" mode of operation which would disable communications system sensitivity to control codes and permit an unrestricted coding of data. If unrestricted transmission of full 7-bit binary information (e.g., all 128 ASCII characters, "pure" 7-bit binary data or encrypted information) is allowed, messages containing the ASCII codes EOT, ETX, ETB, ACK, NAK and similar control characters could be transmitted intact without affecting the transmission system. Similarly, if the parity bit is also ignored in the transparent mode, it would be possible to send full 8-bit binary data. As mentioned previously, the ability to transmit binary data is extremely important in computer communications, and many computers use 8-bit bytes.

The main difficulty with transparent text transmission is that there is no standard way to indicate the end of this mode. If all 128 (256 if the parity bit is also used for information) codes are allowed for data, there can be no reserved "end transparent mode" code. For transmitting 7-bit transparent text, USASI has proposed a technique that makes use of the communication control character DLE (Data Link Escape).[17] The proposal is that the sequence DLE STX initiates the transparent text mode and DLE ETX terminates it. If a bit pattern equivalent to DLE appears within the transparent data, it is replaced by the sequence DLE DLE to permit the transmission of DLE as data. In addition, other control sequences using DLE (such as DLE ETB and DLE SYN) are available to provide active control characters within transparent text as required (see Fig. 7).

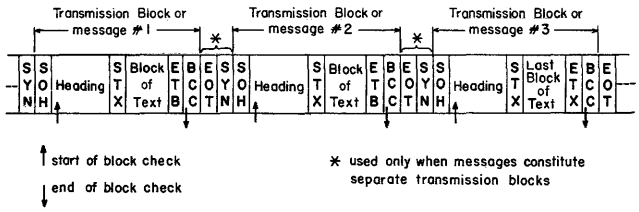


Figure 7—Proposed USASI technique for transparent text transmission

We see several difficulties with this technique proposed by USASI. First, the detection (and generation) of the special DLE control sequences must be done in hardware to be practical. This implies that the channel hardware of all the computers in a network must be built to conform to these rather complex rules for transparent text. Further, the technique will be ineffective (or disastrous!) if any of the equipment along the communication line is ignorant of these rules and is sensitive to ASCII communication control codes. Also, this method makes error recovery a more difficult task. For example, if the character that indicates the "end transparent mode" is in error there will be no simple means for escaping from the transparent mode. This is because standard ASCII communication control characters are ignored once in transparent mode. Finally there is the price that has to be paid for inserting extra DLE's and extracting them (each character in the transmitted and received messages has to be examined as to whether or not it is a DLE).

Our approach for transmitting transparent or binary text is to provide a 6-bit binary mode within ordinary text by using the Group and File Separator characters, as suggested earlier. This has the advantage that it is uniform, USASCII conforming, easy to implement, allows mixing ASCII symbols with binary in a single message, and keeps aside the standard control character set for error control and recovery protocol. Further, it is independent of code-sensitive equipment, and does not require insertion and deletion of DLE's in the transmitted and received texts. The price paid is the reduced effective bandwidth, since only 6 bits out of each 8-bit character carry information.

Computer companies with machines based on 8-bit characters, have a compelling motivation for providing an 8-bit binary "transparent" mode. To accomplish this IBM, for example, has extended the USASI approach by using the character parity bit for information and thus allowing all possible 256 codes.[21,22] The longitudinal block parity check recommended by USASI is inadequate for error control in the absence of character parity, thus IBM has resorted to the use of the more powerful cyclic redundancy check[23] CRC-16. This permits the checking of any code set using the checking polynomial $X^{16} + X^{15} + X^2 + 1$. Although we recommend our 6-bit transmission for sending binary data, we feel the USASI standards should allow the cyclic redundancy check as an alternative for adequate error control in those cases where 8-bit transparent transmission is necessary.

Since we feel that it is desirable to always avoid any problems of transmission system sensitivity to communication control codes, such as exist in the above two approaches, we have devised an alternative scheme for 7- or 8-bit transparent text. In this scheme, the ten ASCII communication control characters (with correct parity) always retain their fixed meaning and are set aside for error control and recovery protocol. The transparent mode is initiated by DLE STX. Whenever a communication control character code appears within the binary stream, the DLE character is inserted to precede it as in the previous schemes, but bit 7 of the control character is inverted (changed from 0 to 1) to change it to a non-control (and non-key) character code. Normal communication control characters retain their ASCII meaning, thus ETB and ETX signify the end of text and SYN is interpreted as synchronous idle. The price paid in this scheme is the slightly less than four-per cent overhead necessary in transmitting the DLE's (assuming a random distribution of binary numbers) and the hardware/software required to implement it.

There are several advantages to this scheme for 8-bit transparent text transmission. First, it retains a single meaning for the ASCII communication control characters, which simplifies error-recovery procedures. Secondly, the communications hardware does not get involved at all in the rules for DLE's, since the control codes retain their meaning. The only place that this has to be done is at the transmitting and receiving computers, and it can be done either in software or in hardware. In a computer network (of the store-and-forward type), each node in the network can be oblivious of the fact that the message it is handling is in transparent mode.

Communications protocol

Communications protocol here refers to the uniform agreed-upon manner of exchanging messages between computing machines. This includes data link control,[24] acknowledgments and error recovery procedures. It may also include message buffering and routing techniques required for communication of messages. As pointed out earlier a single standard protocol procedure may not be practical because of varying needs of different systems and discrepancies in their hardware/software characteristics. Operation could be synchronous or asynchronous, full duplex or half duplex, point-to-point or multi-point, centralized or decentralized, and over private line, switched network, or a general network environment employing store-and-forward routing techniques. In many of the above cases, the protocol needs would differ, and hence the procedures adapted would vary.

Even though the ASCII communication control characters (such as ACK, NAK and ENQ) and other communication control character sequences starting with DLE (such as DLE ?, meaning wait before transmit, and DLE EOT, meaning mandatory disconnect) may be sufficient for communications protocol needs, our recommendation for general inter-computer communications is to provide these functions in heading type messages by use of key characters, as described earlier. There are several important reasons for doing this. First of all much greater error protection is provided since each of the heading messages is followed by a BCC and is error checked (undetected errors in control character sequences can have disastrous effects). Secondly, multiple messages and message blocks can now be reliably acknowledged with their particular identification number, and enquiries can be sent asking for particular messages. (This feature when implemented with control character sequences is limited and clumsy.) A third reason is that key characters in general will not cause any hardware action along the communications line (communication control

characters on the other hand may cause line turn around etc.). This should result in a great improvement in operation, since it is easier and more efficient to have error recovery in software rather than hardware for the more sophisticated computers. Finally, in a general network environment all the messages (including acknowledgments and enquiries) will need routing information. It will be difficult to provide this routing information without the use of heading messages.

To start transmission, a number of SYN characters are first sent to establish the bit and character synchronization. The originating station may then send an enquiry to request a response from the remote station. This is a heading message included within an SOH and ETB followed by block check character and an EOT. The message may include (besides an enquiry), station identification, station status and other desirable information. The receiving station will respond with its own heading message to establish the link. Messages can now be exchanged between the computing machines.

Full-duplex operation allows simultaneous communication of messages in both directions and thus obviates the need for transmission reversals. It also completely avoids the issue of line control, and the possible confusion arising thereof (i.e., both communicators trying to transmit simultaneously over a single line). In half-duplex operation, however, suitable protocol procedures must be adopted to assign line priority to the stations and avoid such confusion. To avoid "hung" conditions (confusion in half-duplex operation), and accidental loss of messages or acknowledgments, the transmitter is always responsible for getting the message through and acknowledged. If the expected acknowledgment is not received within a specified time (exact time may be fixed by each communicator depending on the requirements), an enquiry is sent by the originating station to repeat last acknowledgment.

Error recovery procedures

Error recovery procedures refer to correction of conditions such as "hung" (in half-duplex lines), incorrect receipt of messages or acknowledgments and loss of messages. These procedures are relatively simple if operation is in a message-by-message manner. In the general network environment, it is desirable to allow multiple message transmission. This causes a few problems to arise. If a message is received in error, we cannot assume anything about it. It may have been an acknowledgment, a retransmission or a regular message. Our recommendation is to transmit multiple messages continuously (for reasons of greater efficiency) until an error occurs. When an error condition (loss of message indicated by irregularity in id numbering, acknowledgments not received in specified time, or erroneous message indicated by error checks) is detected, the continuous multiple message process is to be interrupted and error recovery procedures should go into effect. These error recovery procedures will be more efficient and simple on a message-by-message basis. On detection of an error condition, the receiving station can ask for retransmission of messages and outstanding acknowledgments by suitable heading messages. The transmitting station will send the retransmissions desired with the highest priority. When the error condition is corrected, the normal multiple message transmission process may be resumed.

If for some reason the communicators are unable to get through after repeated attempts, they should be able to resort to additional error recovery mechanisms. One such back-up procedure may be a request to exchange line and station status information. The status message may point out the future course of action to the communicators, and thus make error recovery simpler in exceptional circumstances.

CONCLUSION

In this paper we have attempted to define the message format and protocol procedures required for inter-computer communications. A great emphasis has been laid on flexibility, compatibility, efficiency and convenience throughout. We have suggested the use of the framework provided by USASI standards and USASCII. Our scheme for coding heading information by use of key characters appears reasonably efficient and provides flexibility and convenience. Transmission of 6-bit binary within USASCII text is recommended for binary messages (ideal for machines whose word lengths are based on 6-bit characters). Alternative schemes for transparent text have been discussed and their relative merits pointed out. For 8-bit transparent text, without character parity, the longitudinal block parity check seems inadequate for error control and the use of the more powerful cyclic redundancy check is recommended. Finally, we have discussed the acknowledgment and error recovery procedures required for a general network environment. Use of heading messages rather than communication control character sequences is recommended for communications protocol. An approach we recommend is to operate on a message-by-message basis for error recovery (the normal operation being multiple message transmission). This would simplify

error recovery and make it more efficient. Further experimentation in form of real-time or simulation studies is required to define the error recovery procedure more completely.

We hope that the communications and the computer industries, and various groups attempting inter-computer communications, would devote attention to the problem of compatibility and standards. Recognition and acceptance of such standards and guidelines would be an important step forward in the direction of compatible computer networks and information utilities of the future.

ACKNOWLEDGMENTS

The authors are grateful to the Messrs. J. E. Ward and R. G. Mills for the many discussions that led to the current paper. Appreciation is also due to Mr. J. E. Ward for his editorial and technical suggestions.

Work reported herein was supported in part by Project MAC, an M.I.T. research program sponsored by the Advanced Research Projects Agency, Department of Defense, under Office of Naval Research Contract Number Nonr-4102(01). Reproduction in whole or in part is permitted for any purpose of the United States Government.

REFERENCES

1 L G ROBERTS
Multiple computer networks and inter computer communications
Conference of ACM on Operating Principles
Gatlinburg October 1967

2 A K BHUSHAN R H STOTZ J E WARD
Recommendations for an inter–computer communication network for MIT
Project MAC Memorandum MAC M 355 July 1967

3 DAVIES BARTLETT et al
A digital communication network for computers giving rapid response at remote terminals
Conference of ACM on Operating Principles
Gatlinburg October 1967

4 J B DENNIS
A position paper on computing and communications
ibid

5 R G MILLS
Communications implications of the project MAC multiple-access computer system
1965 IEEE International Convention Record

6 D F PARKHILL
The challenge of computer utility
Addisson Wesley 1966

7 E E DAVID R M FANO
Some thoughts about the social implications of accessible computing
AFIPS Conference Proceedings Vol 27 Part I Spartan Books 1965 pp 243-247

8 D T ROSS R H STOTZ et al
The design and programming of a display interface system integrating multi-access and satellite computers
ACM/SHARE 4th Annual Design Automation Workshop
Los Angeles California June 1967

9 P BARAN
On distributed communications
Rand Corporation Memoranda August 1964 RM 3420 PR

10 R J BENICE A H FREY JR
An analysis of retransmission systems
IEEE Transactions on Communication Technology
December 1964 pp 135-146

11 F E FROEHLICH R R ANDERSON
Data transmission over a self contained error detection and retransmission channel
BSTJ January 1964

12 R A KELLY
An experimental high speed digital transmission system
Bell Labs Record pp 65 February 1967

13 D F HOTCH
Digital communications
Bell Labs Record February 1967

14 R T JAMES
High speed information channels
IEEE Spectrum April 1966

15 W E SIMONSON
Data communications the boiling pot
Datamation April 1967

16 *Proposed revised American standard code for information interchange*
Communications of the ACM Vol 8 No 4 April 1965 pp 207-214

17 *Code extension in ASCII (an ASA tutorial)*
Communications of the ACM Vol 9 No 10 October 1966 pp 758-762

18 *Control procedures for data communications–an ASA progress report*
Communications of the ACM Vol 9 No 2 February 1966 pp 100-107

19 Ninth Draft proposed USA Standard Communication Control Procedures for the USASCII
July 1967

20 *Character structure and character parity sense for serial-by-bit data communication in the American standard code for information interchange*
Communications of the ACM Vol 8 No 9 September 1965 pp 553-556

21 IBM Systems Reference Library Form A27 3004 0 General Information Binary Synchronous Communication

22 IBM Technical Newsletter No N27 3011 Re Form A22 6468 1 on SDA II

23 W W PETERSON D T BROWN
Cyclic codes for error detection
Proceedings IRE Vol 49 pp 228 235 January 1961

24 H F ICKES
Data link control procedures: what they are and what they mean to the user
Proceedings ACM National Meeting
August 1967 pp 521-525

An error-correcting data link between small and large computers

by SYPKO W. ANDREAE and ROBERT W. LAFORE, JR.
Lawrence Radiation Laboratory
University of California
Berkeley, California

Operating environment

The need for a data-link connecting small data-acquisition computers to a central computer with great analysis power arose in a particular context at the Lawrence Radiation Laboratory in Berkeley. Both the type of high-energy physics experiments being performed and the operation of the available large computer, a CDC 6600, posed unusual design problems.

Experimental requirements

A comparison of two important approaches to the recording of data from high-energy physics experiments is instructive in providing a background for the operation of the data link.

The bubble chamber approach uses a photographic process to record tracks made by particles in a nuclear event. Afterwards the photographs are examined by elaborate man-machine scanning systems to yield data in the appropriate digitized form. This data may then be analyzed by computer. Bubble chamber pictures in general contain much more information than can be abstracted from them during the first such analysis. It is thus possible to scan the same pictures many times to digitize the data relating to different phenomena.

By contrast, the counting approach uses a technique in which the data from the experiment is digitized directly. In the past, electronic counters were the principle devices used to record the data; today there is a variety of such direct-digitizing devices, including spark chambers and photomultiplier tubes. The success of a counting experiment depends largely on how correct the physicist is initially in assuming which phenomena are to be expected. The experiment is specifically aimed at one or perhaps a few phenomena, and if well-aimed, will provide the required data. But if the physicist's assumptions were not accurate, there is then no opportunity to re-examine the data for other phenomena, as with bubble-chamber photographs. The entire experiment must be rerun with the equipment set up to record a different phenomenon.

Computer availability

The physicist therefore requires rapid feedback in the form of completely analyzed data to permit him to alter his experiment's configuration during a run if required. The capability of the small computers, such as the PDP-8, commonly used for on-line data acquisition at the experiment are too limited to allow this type of analysis, although they can perform simple checks to determine whether the experimental equipment is working normally and whether the data looks reasonable in a general sense. The only way the experimenter can use the large computer is to hand-carry magnetic tapes from the small to the large computer, a process resulting in a turnaround time of hours or even days.

To provide feedback within a useful time the experimenter thus requires sufficient computing power to provide him, on-line, with a detailed analysis of his data in terms of physics. A suitable computer is available at the Radiation Laboratory, a CDC 6600 located several thousand feet from the Bevatron and 184-inch cyclotron where high-energy physics experiments are conducted. Private telephone lines are also available, running more or less along the desired routes. The data link was conceived to provide a reliable high-speed transfer medium between the small and large computers.

The CDC 6600 provided several unusual problems in the design of the data link because of its construction and the way it is normally used at the Radiation Laboratory. The 6600 is designed to protect the central processing unit (CPU) as much as possible from the interference of input/output (I/O) devices. The CPU may be thought of as being surrounded by a protective layer of central memory (see Fig. 1), which

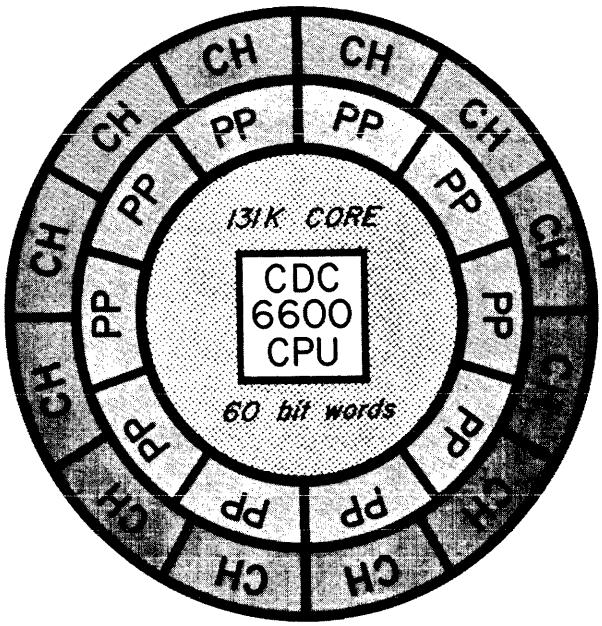


Figure 1 – CDC 6600 organization

is in turn surrounded by 10 peripheral processors (PP) which communicate with I/O devices via 12 data channels. The only way the CPU can communicate with the outside world is via the 131K core memory. Nearly all the PP's are involved in I/O communications. Their tasks are assigned by a controlling PP on the basis of availability, which makes for very efficient use of the PP's. In this sense there is a kind of "time sharing" within the system, an approach used in many areas of 6600 design. This kind of time sharing results in a very fast computer, but not one which is easily used for time sharing in the more usual sense of devices outside the computer. For example, there is no facility for interrupts, so that a PP is forced to continually check a flag from a particular device to determine when service is required. This is satisfactory for such devices as tape units. However, dedicating a PP to watch a flag from the data link would result in a prohibitively heavy demand on the pool of available PP's, since many minutes might elapse between calls for service from the experiment. The data link cannot therefore be treated as a normal I/O device.

Another possibility is to alter the operating system to assign a PP to check at infrequent intervals if the link is requesting service. However, any such modifications are difficult to make on the 6600 operating systems and would result in degradation of the batch processing throughput. It is therefore necessary to include the 6600 operator in the interrupt chain; he is in fact the only part of the 6600 system which can be interrupted.

Design Objectives

From the situation described above, the following design objectives evolved for the data link:

1. No hardware modifications were to be made to the 6600. Modifications to the software operating system were to be minimized to avoid degradation of the normal batch processing.
2. A reasonably high data rate was required, preferably exceeding that of the high-speed tape units already used by the 6600.
3. Error-correction facilities were to be kept as much as possible within the data link itself, to avoid complicated software checking by both the small computer and the 6600. Also, since the data were to be carried by twisted-pair phone lines, error-detection capability needed to be quite powerful.
4. The link would need to be used only occasionally, no more often than every half hour or so, for the transmission of 20 to 30,000 words of data.
5. Rapid response by the 6600 to the link was unnecessary: a lapse of several minutes from the time the link requested service from the 6600 until the 6600 was able to respond was acceptable.
6. The link was to be kept as general-purpose as possible to enable it to be used with other types of computers if the need arose.

Design philosophy and evolution

The approach to the design of the data link was governed by one important consideration: since a relatively small amount of data handling would require use of the link, cost was to be minimized, and existing equipment was to be used as much as possible. This eliminated immediately such alternate approaches as using a medium-sized computer on-line or substituting coaxial cable for the already existing phone lines.

Synchronization

The use of twisted-pair lines, however, raises fairly serious noise problems, since the route through which they run contains many powerful noise-producing devices, such as spark chambers, magnets, and the rf field of the cyclotron. Associated with the noise problem is one of synchronization: how to provide communication between two synchronous devices each running independently on its own clock.

The simplest solution to the synchronization problem is to transmit an echo, or "received your last

word" signal from the receiver to the transmitter, and send the next word only when the echo is received. In this way either the receiver or the transmitter can halt the flow of data when the words cannot be obtained from, or accepted by, the respective computer. The alternative approach, that of sending all words without interruption at a rate slow enough for the slowest computer to handle even during the worst case, would actually have required a slower transmission rate.

Error correction

In order to match the format of both the PDP-8 and the PP, data are transmitted in words of 12 parallel bits, with a 13th bit indicating whether the word is data or function/status (see below). Various error-detection schemes were considered. The most common, the addition of a parity bit in parallel with the data word, has decreasing utility as the number of bits of the transmitted word increases. In this case, with 13 parallel bits and the possibility of powerful noise wiping out entire words, parity was considered too weak a system. Another common error-detection method, the checksum, suffers from a similar difficulty in that if the average error rate is high enough for several errors to occur in each block of data, many retransmissions of each block may be required before the record is received with the correct checksum. Also, either storage devices capable of holding an entire data block must be provided at each end, which is prohibitively expensive, or software in the two computers must intervene in the case of an error to cause the data block to be retransmitted.

A method which both solves the synchronization problem and provides a powerful error-detection technique is to echo each word in its entirety back to the transmitter for a complete bit-by-bit comparison before the next word is sent. This of course increases the time to transmit each word, but the increase is not as large as the propagation delay, since reading and writing data between the computers and the link may be overlapped with transmission. However, for the line lengths in use, the propagation delay is the limiting factor: one typical line of 4000 ft has a round-trip delay of about 12.5 μsec.

Line transmission and reception

Each individual bit of a word is transmitted over its own line. Each line is a twisted pair, transformer-isolated from its associated transmitter and receiver. A full duplex approach is used, and therefore 26 twisted pairs are dedicated to word transfer. An additional 8 lines are used for the control signals, making for a total systems requirement of 34 twisted pairs. These are privately owned telephone lines.

In order to keep the cost low, it was desirable to design transmitters and receivers using simple circuitry but still capable of rejecting a significant amount of noise. Each transmitter consists of two power-nand integrated circuit gates which drive two transistors connected in a push-pull configuration. This produces a bipolar pulse approximating a square wave in the secondary of the output transformer. Two monostable vibrators control the width of the positive and negative areas of the bipolar pulse for all 13 data transmitters in one synchronizer.

Both areas of the bipolar pulse are, in general, equal in width. For reasons of convenience, the width is chosen in this design to maintain an amplitude attenuation of a factor of 4; for instance, 1.3 μsec for a 4000-ft line.

Our initial receiver design actually proved unnecessarily complex. It required the positive portion of the incoming pulse (now resembling a sine wave due to the filtering effect of the lines) to pass through a height window much like a single-channel analyzer. In tests over the actual phone lines, however, we found that, even with the window set quite wide, many more errors were caused by failure to detect an existing bit than by triggering on unwanted noise. The final receiver design therefore requires only that the positive portion of the incoming signal exceeds a certain threshold.

The maximum word-transfer repetitive rate of the link is 80,000 words per sec for a 4000-ft line (12.5-μsec round-trip delay). Obviously the lines of the link are operating with a bandwidth far in excess of the bandwidth of normal telephone lines. In contrast to the latter, the link lines are limited to a length of a few miles and do not run through switch circuits, line amplifiers, etc.

Implementation and operation

Levels of communication

Communication over the link system actually takes place on several different levels, as indicated somewhat idealistically in Fig. 2. In a general sense the link may be thought of as a medium of communication between the experiment and the mathematical analysis of the experimental data. This data, and the results of the analysis, are transformed into 12-bit data words by the small-computer program and the FORTRAN program in the CPU of the 6600.

The programs however are not restricted to sending data. They can also communicate instructions to each other. For example, the experimenter, via the small computer, can request different analysis approaches or output format, while the FORTRAN program may

instruct the small computer to modify a display program depending on the results of analysis. Thus the experimenter has available to him in a limited way an extended on-line processing capability that includes some control over the analyzing process.

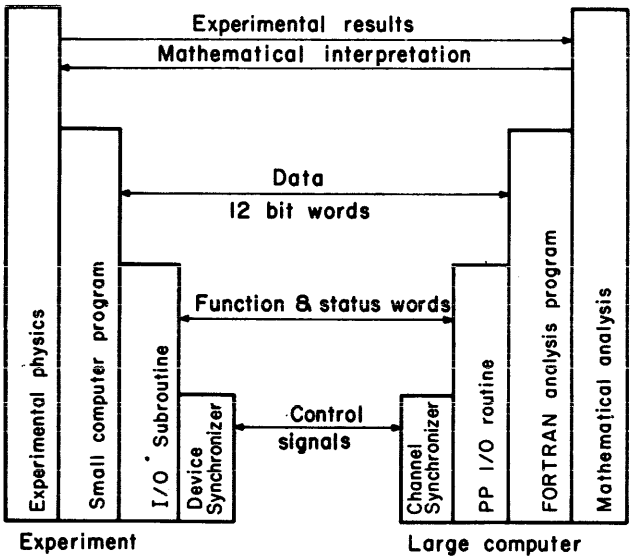


Figure 2 – Levels of communication

At a lower level, it is necessary for the I/O routine in the PDP-8 and the PP's I/O program to communicate certain information involving the timing and format of the data records to be sent. This is accomplished by the use of words distinguished from data by their 13th bit being set to 0, instead of 1, as with data. Borrowing from CDC terminology, these words are called *function words* if they originate at the 6600 end and travel toward the experiment, and *status words* if they originate at the PDP-8 end and travel toward the 6600. The 13th bit also permits detection of an all-zero data word, which would otherwise cause no transmission.

On a still lower level, the device synchronizer (DS), which is the portion of the data link at the experiment end of the lines, and the channel synchronizer (CS) at the 6600 end, must communicate information concerning the words being sent. The control signals which transmit this information are critical to the operation of the system, in that any error occurring in their transmission will result in irretrievable failure of the system. This is true because these control signals are responsible not only for initializing and terminating each record sent, but also because they convey information concerning any errors that may occur in the data record.

It is always possible to increase reliability at the expense of increased time, and this was the technique applied to the control signals. Instead of consisting of a single bipolar pulse, as the 13 parallel bits of the data words do, they consist of a train of 16 bipolar pulses. The receivers require that at least half of these pulses arrive: the remaining half may drop out anywhere along the pulse train.

Error rates

In practice it was found that the error rate was lower than expected. On the 4000-ft line tested, errors (either the dropping of data bits or the addition of noise bits) occurred once in 10^7 transmitted words. However, this rate may well prove to be significantly worse in longer lines. Also, it is difficult to extrapolate the results of error rates from one location or time to another, since the addition of new equipment may result in unexpected large-scale increases in noise.

Tests were run in which the number of times the control signals were transferred was several orders of magnitude larger than would be transferred during the probable lifetime of the link. Not a single error was found during these tests.

The link "conversation"

Before an actual transfer of data can take place via the link, the I/O routines in the computers at each end of the telephone lines must exchange information regarding the format of the data to be transmitted, and also specify the timing. This is done using function and status words as defined above.

An initial status word from the small computer is used to signal the operator of a service request, via the console light, and a function word carries back his push-button response (see Fig. 3). Later, when the operator has loaded the Analysis program, the program itself initiates the sending of a function word to the small computer to inform it that the data may now be transmitted.

Since the Fortran analysis program in the 6600 requests only *logical* records of a certain length, the PP I/O routine must obtain the *physical* record length from the small computer: This is sent as a single data word. To read in this word (as to read in any status or data words) the PP first sends a function word, in this case "GO-WORDCOUNT," whose only purpose is to cause the status word to be transmitted from the channel synchronizer to the PP.

Following the transfer to each physical record, the small computer informs the PP that either (a) more physical records must follow to complete the logical record, in which case an "end of physical record" status word is transmitted; or (b) the logical record has been completed, in which case an "end of logical record" is sent.

Data output (from the 6600) is analogous to data input, except that at the beginning of each record the PP must inform the small computer whether the record will be composed of data or only an end of file.

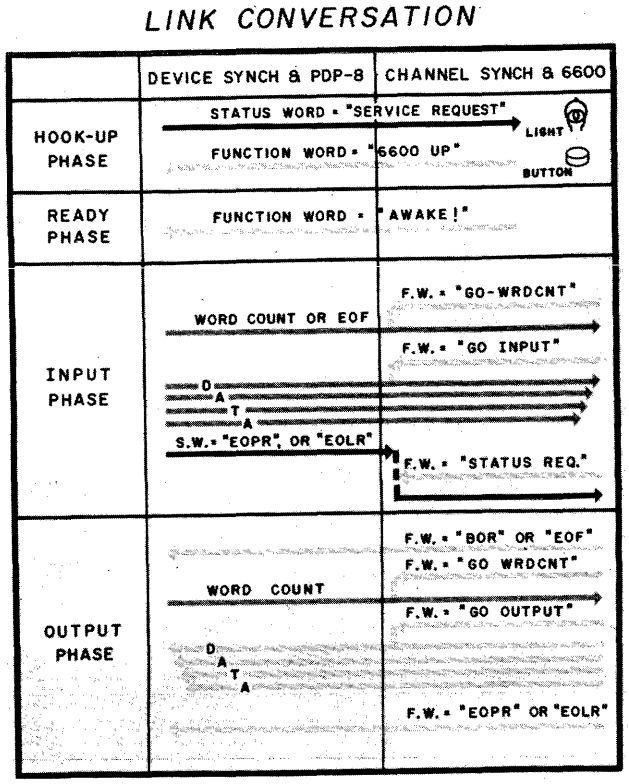


Figure 3 – Conversation procedure

Since the status and function words are to a large extent initiated and interpreted by software, the "elements of the conversation" described above can be altered to meet future changes in the 6600 operating system or even to provide the interface for entirely different computers at either end of the link.

The carrousel

The four registers connected directly to the long-line receivers and transmitters are called the "carrousel" and constitute the heart of the error checking and correcting system. (See Fig. 4. The register names are arbitrary).

During normal data transmission (for example, during input to the 6600) a word flows out of the small computer memory buffer into the BR in the device synchronizer, then into the CR, and finally the DR, where it is both stored and transmitted to the channel synchronizer. In the CS it is received in the CR and then shifted to the DR, where it is stored and retransmitted to the DS. Arriving in the CR of the device synchronizer, this echo is compared with the original word still in the DR. If the comparison succeeds, both CR and DR are cleared. During the round trip of this first word from DS to CS and back, a second word will have shifted from the small computer

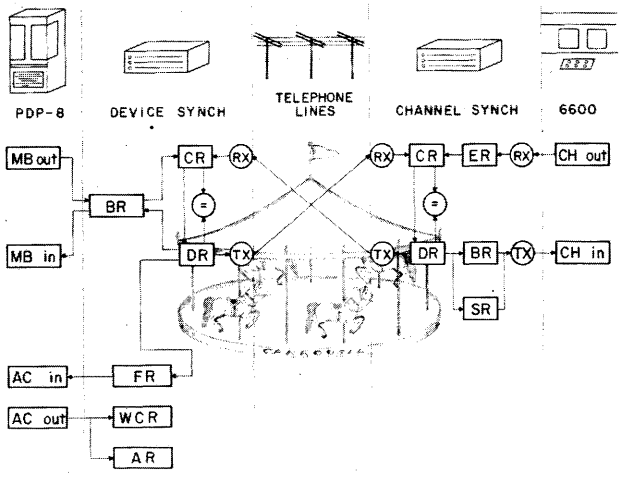


Figure 4 – Register arrangement.

memory to the BR, where it awaits a successful comparison of the previous word. Once both CR and DR are cleared, it will be free to shift into the CR, and the sequence will be repeated again. A similar process takes place for data output from the 6600.

Since I/O operations between the computers and the link are concurrent with the transmission and echoing of the previous word, the cycle times of the computers cease to effect the data rate of the system (assuming line length is the limiting factor). Also, if the computer on the receiving end is not able to accept the data fast enough, the process will simply pause until the received word is finally read in, thus preventing loss of data or the necessity for complex synchronization and timing devices.

Error correction

If in the transmitting synchronizer the comparison check on the echoed word should fail, only the CR is cleared, and a control signal is sent to the receiving synchronizer to warn that the previously received word was in error. This clears the DR in the receiving system and permits the same word to be retransmitted as before. The same word may be sent many times until a correct echo is finally received. However, if it should cycle for too long a time the link assumes that a serious fault has occurred and sends function and status words to this effect to the two I/O programs.

While the link is waiting for the computers or the 6600 operator to give it further instructions, it arranges that both the synchronizers are in transmit mode. This ensures that any noise word received at either end will be compared with the zeros in the empty DR and erased, as with an ordinary error. (If the synchronizers were left in receive mode they would retransmit any noise word arriving in them and eventually fill up all registers with replicas of the noise word.)

CONCLUSION

In the special situation for which it was designed, the data link provides an effective technique for communication of data between computers. Its strong points are:

1. It is relatively inexpensive.
2. It provides a very strong error-correction capability, necessary in its noisy environment.
3. It requires very little modification to the existing large computer, either hardware or software.

On the other hand, the techniques used in the data-link system are not immediately applicable to data transmission over more than a few miles, because the round-trip propagation delay then becomes excessively large, or over commercial phone lines where the bandwidth is restricted to about 3kHz.

ACKNOWLEDGMENT

The authors wish to acknowledge the contribution of Alan Oakes to the design and development of the receivers and transmitters.

This work was done under the auspices of the U. S. Atomic Energy Commission.

Graphical data processing

by E. J. SMURA
Xerox Corporation
Rochester, New York

There is a need for faster and more efficient production of high quality graphic images. Since the information in graphic images can be coded into data bits, we should be able to adapt the techniques of high speed electronic data processing to graphic image processing. Accordingly, we have constructed an experimental graphic data processor. Its design is based on the same logical design principles as an electronic data processor, but we have modified the system to accommodate the special features needed for processing graphical data. We will describe here our approach to the design, present some details of the system, and show some of the processed images. We will begin with a short discussion of how graphic data is reproduced today, outline some of the advantages of automating the process, and then talk about our own experience.

Most of you are familiar with the techniques now used to process and present graphic data. We are talking, of course, about such things as typing, drawing, photography, printing, copying, duplicating, and CRT presentations. We at Xerox believe the present techniques (which we will describe briefly) fall somewhat short of satisfying the present needs of graphic communications and will certainly be absolutely inadequate in the very near future. The graphic data processor described in this paper is a step toward faster and more efficient, thus more economical, graphic image generation, processing, and display.

Consider, for example, the printed pages making up a book or a technical journal. The pages may utilize several different fonts including even special fonts of Greek or Roman alphabets, italics, boldface, or special mathematical symbols. The page may also include illustrations such as circuit schematics, drawings, or photographs. There are, in fact, an almost infinite number of variations and combinations of alphabets, special symbols, drawings, and photographs which could appear—consider only the possibilities made available through varying type sizes.

Our general approach to better graphical data processing and presentation was to take a detailed look at each step of the usual practice followed in producing graphic images. We then studied existing electronic data processing equipment to see if it could be adapted for graphic data processing. Finally, we looked at the fundamental descriptors of graphic images to see if they could be used in developing a new system which would make optimum use of this existing equipment. We began with a look at how a manuscript is processed into a page in a typical technical journal.

Let us assume that the text has been written and edited, that rough sketches have been made of line drawings and schematics, and the necessary photographs have been taken. We also assume that the finished page will require the use of several fonts and that the sizes and positions of the illustrations will be different on each page.

The first step is to give the text (the alphanumerics) to a typist (or typesetter) and the drawings (the line art) to an artist or illustrator. The photographs are given to another specialist who converts the continuous tones of the photographs into dot patterns for halftone reproduction.

The typist can type only those alphanumerics provided by her machine. Most typewriters have only a few of the hundreds of possible fonts; and even if all fonts were available, there would be a time delay either in switching fonts or switching machines. The artists, illustrators, and photographers also face problems. For example, they must know the sizes of the finished illustrations, the locations of the illustrations on the page, and the type of system to be used in reproducing the page, because each of these factors will influence the artist's technique in preparing the illustrations. For complicated publications, a coordinator must be used to resolve these many questions on format, layout, and style. Note how often the "human factor" is required in this normal publication process.

We can mention only briefly some of the many other factors involved in setting up a page for printing. Some examples are: are the paragraphs indented?; are the

right-hand margins "justified" (i.e., no ragged edges on the right-hand margin)?; how are the pages numbered?

The job of pasting up the page—that is, making a dummy—takes a long time even when done by experts in publications. The printer must then assemble the various components (type, photoengravings, etc.) and then provide a proof for editorial revision. Even when modern phototypesetting techniques are used, preparation of a finished made-up page is a laborious process involving the stripping of film and/or paste-up of velox prints.

Th inefficiencies involved in processing the draft through to the finished copy indicated to us some benefits of automating the graphical data processing and recording system.

For example, with automated graphical data processing, a typist would simply type the book, report, or paper. No special attention would be needed for character font, size of the page to be produced, hyphenation and justification, and the insertion of line graphics and halftones; these functions would be provided by the graphical data processor. In addition, an editing function could be made available to provide for easy additions, changes, or rearrangement of the material within the book or paper.

Graphical data processing could allow a draftsman to produce all his line drawings in one standard size. In addition, he might only be required to indicate the beginning and end-points of any line with a computer check for fit and tolerance into the assembly drawing. Finally, by pressing a button, the draftsman could produce multiple copies of the finished drawing at any magnification. Conceivably, graphical data processing could allow an industrial photographer to produce aesthetically pleasing results with a printing press by providing the ability to vary tonal range within a halftone picture, considering at the same time the type of paper and ink being used on the press. In addition, the photographer could vary hues and tonal range in a printed composite color photograph.

We have already mentioned the wide variety of information which we might need to insert on a page. Obviously, there would be advantages if we could reduce the variety of "kinds" of information which our equipment had to process. This led to an investigation of what we call the fundamental descriptors of graphic data.

An analysis of typical publications shows the pages usually contain one of three basic types of graphic information or a combination of the three. We call these three basic data presentation methods, 1) the alphanumeric mode, 2) the line copy mode, and 3) the halftone mode.

By the *alphanumeric mode,* we mean using sets of standard type such as alphabets, mathematical symbols, and chemical symbols. Characteristically, these graphic symbols have special meanings (in context) regardless of the variation of form (font) employed.

By the *line copy mode,* we mean drawing images using lines only, for example, outline drawings and circuit schematics. The images are considered to be generated by a set of vectors in the image plane, and we usually refer to the line copy mode as the *vector generation mode.* Note that the alphanumeric mode is actually a special case of the vector generation and (later) we will show how we used this fact in an interesting variation of our experimental graphical data processor.

By the *halftone mode,* we mean creating images by arranging black dots on a white background or, more rigorously, forming an image by combining black and white dots to reproduce various gray levels.

We can, of course, logically define other modes, for example, continuous tone and color. For the system which we will describe in this paper, however, such other modes can be considered as extensions, variations, refinements, or combinations of one of the three modes described above. *Thus, the problem of designing a graphical data processing system reduces to the problem of designing a system to generate, process, and reproduce in these three modes.*

We at Xerox have been looking at the three fundamental steps required to reproduce a graphical image in the three modes: 1) generating the input, 2) processing the input data, and 3) presenting and reproducing the output. Specifically, we have placed our major emphasis, to date, on processing, display, and reproduction with some preliminary work on the fundamental requirements of input data and format. Later, we will show you some samples of the graphical data produced with our equipment.

In developing our graphical data processor, we started with the state-of-the-art digital computer and its peripheral devices. We analyzed present-day equipment to determine how best to adapt them to graphical data processing. Some very serious deficiencies were found in the present input/output devices available.

Each form of graphic output usually requires a separate output device; in other words, for alphanumerics, we may use an impact printer; for line drawing, a mechanical plotter; and for viewing the two modes, a vector display. A vector display from which hard copy may be generated presently requires significant manual intervention. The principal limitation of this type of equipment is a lack of control over a document which contains the three basic forms of

graphical data; i.e., alphanumerics, line drawings, and halftone pictures. The limited style and quality and the need for different forms of output—i.e., fan-fold sheets, cards—prevent generation of a graphical record in material, style, and format compatible with today's manual methods. Even with some of the new phototypesetting equipment, generation of halftones and graphics is not practical because of the burden placed upon the processor and the need for complex software programs. Another serious limitation of electronic typesetters, even in generating alphanumerics only, is the extreme difficulty of accomplishing multi-column work. The line-at-a-time printing mode usually employed necessitates full-page storage of the alphanumerics and a software routine to do the necessary sorting. Finally, the signals which operate output devices in a standard EDP operation cannot be edited—that is, do not have the monitoring capability which includes ability to format, change format, and edit. While light pens and CRT's permit this control, they require large software packages in order to use them. To require them to accomplish the functions of a GDP would place such massive demands on the software that the GDP would be limited in its application to only those with very large computer installations.

For graphical data processing, we need the specialized storage and manipulating capabilites shown in Fig. 1. We need the latter means to handle in one machine the three basic graphic modes and also to handle format—that is, margins, justification, hyphenation, editing, and change of mode. Finally, storage is necessary to hold the fonts—the alphanumeric character signatures.

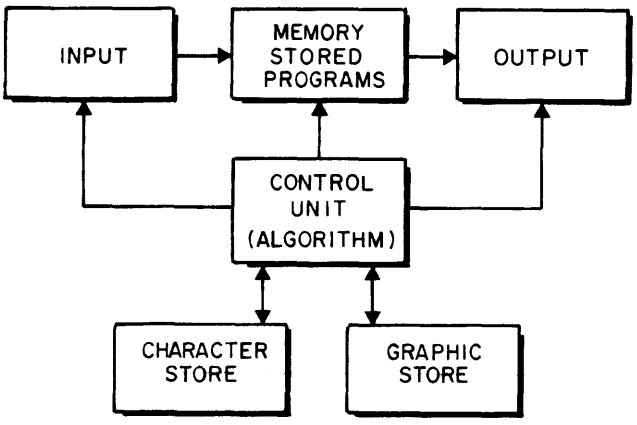


Figure 1 – Functions needed for graphical data processing

Significant advantages could accrue if the two basic storage capabilities normally provided could be increased to four self-contained storage areas, i.e., character signature storage, format instruction storage, program storage, and graphic data storage. With this arrangement, parallel communication is possible, editing can be done efficiently, and the requirements of the system monitoring program are reduced. In addition, improved communication liks between these storage areas and the output peripheral devices could reduce, significantly, the need for computation. For example, it might be possible, using the format storage and its associated communication registers, to communicate easily to the output device, the left-hand margin of a particular column of text. If it were desired to change this margin, an alteration could be made to the stored data in the format memory and the corrected data relayed to the output device to update the margin register. This, in effect, would move the column to be written to the desired physical location. We are familiar with the same operation that could be performed if a margin register did not exist, i.e., adding a fixed distance to the incoming alphanumeric data prior to relaying to the output device. The disadvantages of the latter method are that it requires a program, it must be stored, time is wasted in computing, and the processor might be required to alternate between numerical and graphical programs for a more difficult topology change; i.e., image rotation.

Thus, the status of present equipment and the functions desired indicated that special equipment should be designed to secure speed, automation, and convenience in converting ideas to an understandable graphic form. This equipment must be capable of 1) transforming the three forms of data into electrical signals defined as input, 2) manipulating, storing, and processing the electrical signal, and 3) transforming the electrical signals into the desired image—that is, it should have an output capability. This equipment would be a Graphical Data Processor (GDP) because it produces true graphics.

We decided to begin our GDP experiments by constructing an adequate output device. We then selected a modern, general-purpose digital computer as part of our experimental processor basing our choice on the need for sufficient interrupt capabilities and rapid communication with any equipment we might develop. We realized that speed, efficiency, and communications would be limited, but we felt confident that we could achieve our aim of optimizing software and hardware to perform the basic experiments. As we were concentrating on output for this first phase, we accepted the standard equipment found in an EDP installation for input and later supplemented it with a simple, opaque gray-level scanner. We then began our construction and experiments on this equipment and developed processing capability to generate the three basic forms of graphical data. Our principal initial objective was to meet the needs

for image generators producing a natural and familiar style and format, with compatibility between image generators and duplicators.

Our experimental graphical data processor consists of, 1) an SDS 930 digital computer into which graphical data are entered and stored in digital form, 2) a graphical recorder which uses several cathode-ray tubes to present the graphical data for recording or direct view, and 3) an interface between the computer and recorder for interpreting data from the computer and converting it to analog form suitable for presentation to the output device.

Figure 2 shows our experimental equipment; we used this equipment to generate the graphic images in all of the samples included in this paper. The 930 computer is in the background, the digital input devices are on the left, and the interface is to the right of the magnetic tape unit; two display devices can also be seen.

Figure 2—The 930 computer and display equipment used in the experiments

The equipment can be operated in any one of the several graphical data modes described previously. An alphanumeric mode may be used for justified and hyphenated generation of text, a vector mode for the generation of line-type drawings, and the dot mode for the creation of halftone pictures. Combining these modes of operation with computer programs, we can edit, and, in general, regulate the output format. For example, a line drawing can be inserted into a page of hyphenated-justified; multi-font text in a multi-column format.

Graphical data can be inserted into the experimental GDP system through such standard EDP equipment as a computer keyboard, punched card, paper tape, or magnetic tape. More recently, we have added a capability to introduce line and continuous tone images through a full-page opaque, gray-level scanner. Sometimes this is very inefficient; however, it has allowed us to attain the goals we sought with this experimental system. The command and control programs are permanently stored on a drum for convenient and rapid access. Storage of graphical data within the system can be in core for random access, magnetic drum for high volume semi-random access, or magnetic tape for serial access.

Digital data are directed from the computer to the interface under both interface and computer program control. The interface decodes and converts the data to analog form for presentation to the output device. There are three principal output devices, each of which is a cathode-ray tube. The first is a high-speed, real time, closed circuit television display. The second is a real time, high-speed storage tube which retains data until operator erased. The third real-time, high-speed, high resolution cathode-ray tube is used for generating hard copy by either a xerographic or a photographic process.

We can given you some idea of how this graphical data processing equipment operates by considering in greater detail the three modes—alphanumeric composition, dot composition, and vector generation. We also have examples of the end product of each mode and examples of operations where we combined several modes.

First, let us describe how an alphanumeric character is placed in the GDP memory and then follow the character from an instruction to print in the input stage through its final generation in hard copy. First, the character is hand typeset, scaled in size and located in a grid for easy reduction to digital form. As we digitize the character, we remove redundancy by coding or digitizing only the black portions of the character leaving the white surrounding portions uncoded. The black portions are further coded in an optimal form. The digitized character is then stored in the computer as part of a full font of some 80-90 other characters and special symbols. A typical font consists of capital and miniscule letters, digits, ligatures, periods, commas, brackets, and other special symbols. A wide variety of such fonts may be stored on the drum. Also stored on the drum are composition programs which furnish instructions on format and page make-up (for example, hyphenation and justification).

To generate a page of alphanumeric characters, an alphanumeric composition program is called from the drum and set up in a section of the core in the computer set aside for use by the interface. Suppose we want to print an A. Upon command, the computer sends the digital information describing the position of the A in storage to the interface. The interface retrieves the A from storage and converts the digital

information into analog form for presentation to the output device. At the same time, the same alphanumeric composition program determines several other variables and constants of the composition process and sends the necessary data to the interface. These other instructions may include:

(1) digital address of the position of the A on the cathode-ray tube
(2) the velocity at which the A should be written
(3) the number of times it should be overwritten
(4) the character zoom or size in horizontal or vertical direction or both
(5) the intensity at which it should be written
(6) spot resolution at which it should be written

Repeating the same process, we can generate a page using any or all letters, numerals, or special symbols contained in the stored font, and we can program the GDP to change fonts and mode of operation. Figure 3 shows a full page where two fonts have been used—that is, several words have been italicized. The fonts are Bodoni 12 point Roman and Italic, and the text has been hyphenated and justified. We used the Bodoni font because it illustrates the very fine lines and serifs that can be obtained in full-page recording using CRT technology and the photographic process.

The characteristics of zooming are shown in Fig. 4. (Zooming is a change in size on the output document over what was initially used for character digitizing.) Along any one of the rows, only horizontal zoom has been used. Down any one of the columns, only vertical zoom has been used. Thus, the diagonal of this page represents both horizontal and vertical zooming and, therefore, a change in point size. All of the characters on this page have been produced by various zooms of an original 12 point character set. (A point is approximately 1/72″; most office typewriter characters are 10 or 12 point.) The smallest possible increment of zoom upward or downward in the system (in this case) is 1/4 point. This is only an illustration of the versatility of the zoom feature in producing letters or special symbols of various point sizes and aspect ratios.

Figure 4 – Changing character size by zooming

The vector mode of the experimental GDP system is equally versatile. There are two basic ways to draw a vector between any two points, A and B. The first is to draw the vector between A and B at constant velocity. The second is to draw the vector between A and B in constant time, regardless of the distance from A to B. A constant velocity vector is used to control exposure in the xerographic process or in the photo-

THEORETICAL DISTRIBUTIONS

of rectangle *ABCD* is approximately equal to the shaded area under the continuous curve. Since the area of rectangle *ABCD* represents (is proportional to) the probability of getting 3 heads in 10 flips of a bal-

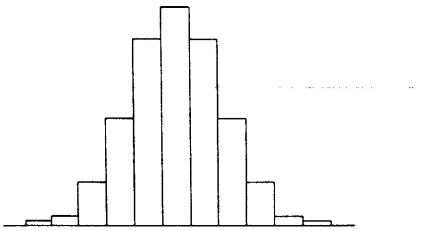


FIGURE 7.4

anced coin, we can say that this probability is also represented by the shaded area under the continuous curve which approximates the histogram. *More generally, if a histogram is approximated by means of a smooth curve, the frequency, percentage, or probability, of any given class is represented by (proportional to) the corresponding area under the curve.*

If we approximated the distribution of 1953 family incomes in the United States with a smooth curve, we could determine the proportion of incomes falling into any given class by looking at the corresponding area under the curve. By comparing the shaded area of Figure 7.6 with the total area under the curve we find that in 1953 roughly 17 per cent of the families had incomes of $7,500 or more. It can, similarly, be seen from Figure 7.6 that 30 per cent of the families had incomes of $4,000 or more. We obtained these percentages by (mentally) dividing the corresponding areas under the curve by the total area under the curve, which, after all, represents 100 per cent of the families.

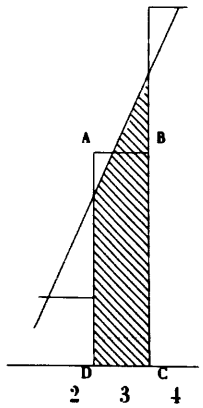


FIGURE 7.5

Had we drawn the curve of Figure 7.6 so that the total area under the curve is equal to 1, the proportion of the families belonging to any (income) class would have been given directly by the

Figure 3 – Justified and hyphenated page of mixed graphics and alphanumerics produced by the graphical data processor

graphic process because this constant velocity insures uniform exposure of every point along the vector. While our equipment can operate at either constant velocity or constant time, we will describe here only constant velocity vector operation.

Data for vector mode operation are entered into the system and stored in much the same manner as for alphanumeric mode operation. In the vector mode, however, only beginning and end points of the various vectors are entered and stored. From the variable data—that is, the beginning and end points of each vector—the computer automatically computes the velocity for each direction (x and y), and the information is transmitted to the interface for generation of signals which direct the recorder to physically generate the vector. Through the use of a special program, certain constants are inserted into the computer for use in generating any vector. These constants include the size of the page, the velocity at which the vector should be drawn, the intensity at which the vector should be drawn, and the resolution of the vector. Once these constants are entered, they remain fixed throughout the composition of a line image unless an alteration is required.

The velocity of writing may vary from 3 inches/second to as high as 10,000 inches/second. On a 10″ × 10″ page, vector line thickness may be as small as 2 mils with vector position being specified by a 13 bit word.

Figure 5 illustrates the versatility of the vector generation program. The slide shows a series of ellipses generated about one another. It is interesting to note that our equipment does not have curve generation capabilities at the present time; the ellipses you see here were generated by a large number of short vectors.

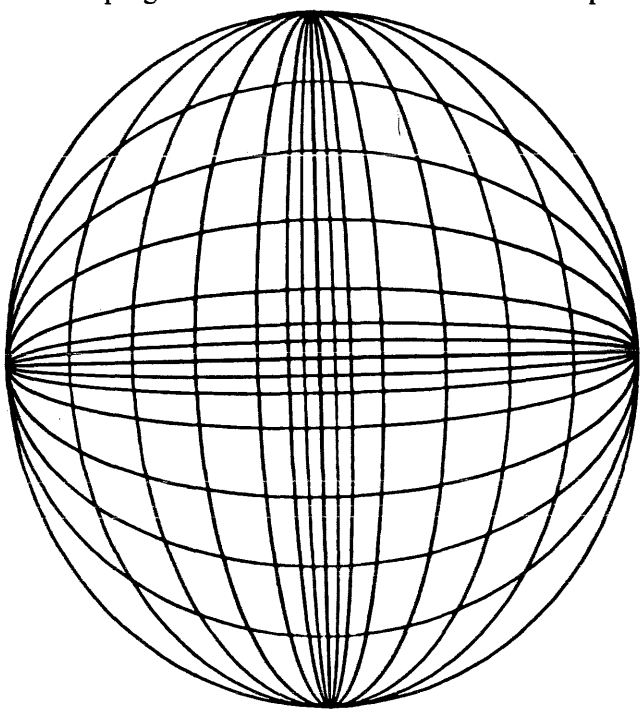


Figure 5—Ellipses generated by the graphical data processor operating in the vector generation mode

The dot mode of the graphical data processing equipment may be used to generate halftone pictures. The machine can generate any desired dot density and practically any resolution up to a maximum of 666 dots per inch in either field direction. The dot mode of the recorder is controlled by a dot mode program which controls such variables and constants as page size, resolution of the dot, intensity of the dot and screen density.

Figure 6—Halftone picture generated by the graphical data processor

A "picture" consisting solely of $\frac{1}{2}$ million dots may not be very interesting because it may be simply a black wall. However, if those $\frac{1}{2}$ million dots are intensity modulated, the picture might become very interesting, as in this picture (Fig. 6) of a pretty girl. To reproduce this picture, the screen size was set at 200 × 200 dots per inch and 16 levels of intensity modulation were used.

To illustrate some of the power of this combination graphical recorder and digital processor, let us consider what would happen if we were to ask the computer to separate out one intensity level in the picture of the girl; for example, the 16th intensity level (Fig. 7). Note that only the hair and eyes are principal contributors to this level. Our graphical data processor can print out any other (given) intensity level

Figure 7 – Reproduction of only the 16th intensity level in Figure 6

from the original picture, and, in addition, can combine any ordering of intensity to produce different types of output. Another example is shown in Fig. 8. Here we see a picture generated by calling out the least significant bit of a 4 bit intensity level code. This picture looks something like a numbered painting. Note also that it is a very "busy" picture compared to the original.

Figure 8 – Reproduction of the least significant bit of a 4-bit intensity level (of Figure 6)

We are continuing our study of the structure of halftone images, aiming at reducing the redundancy of data in an effort to speed up processing time and to minimize storage requirements.

These samples indicate that generating the three basic types of graphical data is technically feasible. It is also feasible to mix the operating modes and computer programs to create some interesting graphical images. For example, the sample page of justified and hyphenated text shown in Fig. 3 also contains graphics. All of the alphanumerics in the text, title, and labels on the graph, together with the line drawings, were totally composed by the graphical recorder in a moment's time. This picture illustrates that it is possible to generate the types of pages usually found in reports, periodicals, and technical journals.

To further illustrate flexibility, Fig. 9 shows a complete printed board master made by a graphical recorder with a stored font containing geometrical shapes instead of alphanumeric characters. This figure shows that engineering drawings of high quality and precision can be created, thus relieving the draftsman of many tedious and tiresome tasks.

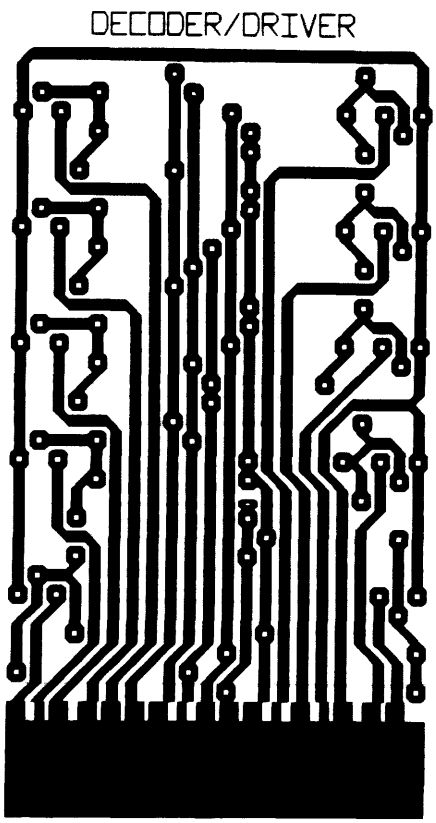


Figure 9 – Engineering drawing produced by the graphical data processor using both alphanumeric and vector generation modes

Finally, we decided to see if our graphical data processor could do the job normally performed by an impact printer tied on-line to a computer. In this application, speed is more of an asset than quality, so we tailored a font of alphanumerics to minimize digital character description, interface processing time, and recorder writing time. A typical print-out is shown in Fig. 10. The equipment which generated this page is the same, unaltered equipment that generated the previous examples. The recorder, using the same basic programs, assumed the role of a computer printer.

```
00355  0 46 00002      51          CLB
00356  0 67 00013      52          LSH     11
00357  0 35 01231      53          STA     LHAR
00360  0 76 15026      54          LDA     =5
00361  0 75 15041      55          LDB     =MSGY
00362  0 43 00525      56          BRM     WRTMSG
00363  0 43 00435      57          BRM     CKSIGN
00364  0 46 00002      58          CLB
00365  0 67 00013      59          LSH     11
00366  0 16 15032      60          MRG     =04
00367  0 35 01232      61          STA     LVAR
00370  0 76 15035      62          LDA     =6
00371  0 75 15042      63          LDB     =MSGG
00372  0 43 00525      64          BRM     WRTMSG
00373  0 46 00002      65          CLB
00374  0 71 15043      66          LDX     =-4
00375  0 67 00003      67          LSH     3
00376  0 43 00544      68          BRM     READCH
00377  0 16 00561      69          MRG     C1
00400  0 41 00375      70          BRX     =-3
00401  0 35 15024      71          STA     PMTC
00402  0 76 15044      72          LDA     =BB1
00403  0 35 00200      73          STA     B200
00404  0 02 20002      74          EIR
00405  0 71 15030      75          LDX     =3
00406  0 02 30000      76          EDM     30000
00407  2 13 01233      77          PDT     LPDR+3, 2
00410  0 01 00410      78          BRU     $
00411  0 41 00406      79  BB1     BRX     $-3
00412  0 76 15045      80          LDA     =BKPT
00413  0 35 01214      81          STA     IADD
00414  0 20 00000      82          NOP
00415  0 20 00000      83          NOP
```

Figure 10 – Alphanumeric "print-out" from the graphical data processor operating in the alphanumeric generation mode

We generated the full page of text shown here at a speed comparable to presently available standard computer impact printers. In addition, we have the option of choosing our page size. For example, we can elect to print the same data on a tabulating card simply by inserting the proper information into the GDP. As a matter of fact, we can print the full page of text on a tabulating card much faster than we can print it on the $8\frac{1}{2} \times 11''$ format because the total exposed black area is much smaller in the card format (remember that the GDP is using a photographic or xerographic print-out process).

In conclusion, the experimental system we have described takes digitally coded input data and, through optics, electronics, cathode-ray tubes, and photosensitive materials, creates in some cases very high quality images. This particular graphical system is comprised partially of standard electronic data processing equipment to allow us to do experimental graphical data processing related to recording. We call this equipment arrangement a graphical data processor.

We have illustrated that some day it will be technically feasible for the secretary, the draftsman, the industrial photographer, and others to use such equipment to generate their graphical images with greater flexibility, speed, efficiency, and quality. The graphical data processor, which sees all graphical data as combinations of alphanumerics, vectors, or dot patterns, can create and manipulate pages of graphical data and reproduce the data in a wide variety of formats, sizes, styles, and on many different kinds of materials. To date, our studies and experimental data show that the system, at least, through trade-offs, say speed *vs* quality, could replace, in some cases, present-day computer peripheral devices.

We are working at improving hardware and software designs to provide rapid and convenient data-input devices. We are studying how to optimize processing steps and storage requirements related to graphical data manipulation, so that equipment size might be reduced and speed of the operations increased. In short, we hope to fulfill all the processing needs of those concerned with the manipulation of graphical data. The ability to thus create, manipulate, and reproduce graphical images will enable faster, more efficient document creation, distribution, communication, alteration, and storage and retrieval.

ACKNOWLEDGMENTS

The author is deeply indebted to the project leaders, L. Blumberg, L. Mailloux, J. Rosenthal, and C. Smoyer and their staffs for their response and encouragement in reducing an idea to hardware and also for their significant contributions to graphical data processing. A special note of recognition goes to Arlene Bencin and the programming staff for software development and to Al Lawson, our consultant from Rochester Institute of Technology, Rochester, New York, for typographic guidance and helpful judgment of graphic arts quality.

The advancing communication technology and computer communication systems

by SIDNEY J. KAPLAN
Western Union
Mahwah, New Jersey

The time factor involved in the transportation of information is assuming more importance in data processing. Examples of systems that illustrate the growing dependence of the computer industry on communications are: computer complexes, interactive terminals, computer utility, real time processing and management information systems. In these applications the computer hardware and software costs, as well as computer throughput, have been found to be critically dependent on the data communication operation.

The established communications services are regarded by some as inefficient for computer system application. It is notable that the only systems originally designed for data communications are the telegraph systems which have evolved over five generations. These systems have experienced considerable growth in recent years. However, to satisfy the new data processing communication requirements, the telegraph and new data communication systems can be expected to evolve more rapidly in the future.

The more recent advances in communications technology are discussed in this paper, along with the probable new and improved communication services that the new technological advances will allow. A broad discussion of aspects of the rapidly changing computer system communication requirements will also be given. The evolution of data communications is dependent on more than just technology. Factors such as regulation and pricing policies are important in stimulating systems to grow to their most efficient form. These factors are not discussed in this paper.

Data communication requirements

Much has already been said about the need for low cost data systems, particularly in such industries as securities, banking and airlines. "Management Information Systems" for these industries requires data communication connecting diversely located divisions of a corporation with a computer system where services are provided for manufacturing, sales and planning, and other support departments such as accounting, purchasing and inventory control. The telegraph industry that represented data communications until ten years ago fulfills important requirements for message traffic in computer systems. However, large volume high speed transfers of data and the man-machine dialogue over interactive terminals represents new requirements

The representation in Figure 1 is designed to characterize data communication types as a function of transmission rate and data volume. Four specific groupings of data communication can be seen and they include the telegraph message group in addition to three others that are required in computer applications.

Data Communication Category I is shown to represent the telegraph applications. The average telegraph message is approximately 50 words or 300 characters in length, including about 25 address words. At a 75 baud telegraph speed the message transmission time is approximately one half minute. Telegraph systems have been designed so that the rate of transmission is approximately equal to the rate of manual message generation. Terminal speeds of 50, 75 and 110 baud are the most popular. Higher speeds, up to 180 baud, are expected to be more common in the near future. The 180 baud operation is the highest speed that can be carried on a class D-TTY circuit. It is well above typing speed and will require data storage for operation at its highest rate.

Category II of data communications in Figure 1 represents the interactive terminal applications. Short messages are generated by humans and computers and there is a relatively long waiting period between messages. At times the computer message might be lengthy. In general, long messages to or from the computer run at the maximum terminal speed since they are read from a tape reader to computer memory. Experience in existing time sharing services indicates that data is only transferred between 2% and 4% of the time that the terminal is operating with the computer.

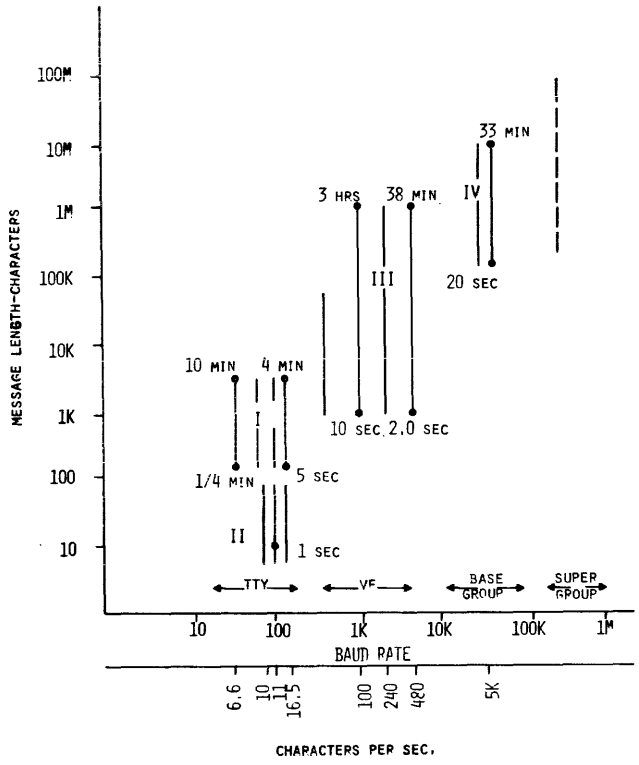


Figure 1 – Categories of data communications

Message headings or endings are not included with each transmission since in present systems there is generally a dedicated line or circuit switch connection set up between the terminal and the computer.

Two other classes of data communications are also shown in Figure 1 which represents the large volume data messages that are transferred between high speed terminals and computer memory. Each category is characterized by the transmission facilities over which it is used. Category III is associated with voice line transmission and makes use of high speed paper tape and card readers operating in the region of 100 character/sec. Newer terminal equipment is designed to operate at higher speeds, e.g., low cost magnetic tape terminals now operate at 2400 baud. Modems operating at 4800 baud are also coming into common use. One million characters of data trasmission would take approximately 3 hours at 1000 baud. At 4800 baud, however, it would take approximately 38 minutes.

Category IV in Figure 1 represents the much larger volume of data sent on base group transmission facilities. The data transmission speed of operation is in the area of 50 K baud, depending on the modem used. It takes approximately 3.6 minutes to transport one million characters at these speeds, neglecting the time for terminal interaction and acknowledgements.

While Categories III and IV represent data transfer between computers and terminals, another category of data communications might enable real time operation of remote computers in parallel. The transmission over super group facilities at 200 Kb/s can be utilized for this purpose.

An important application for high volume data communications are remote computing, load sharing, financial reporting, order entry and inventory control. Based on user surveys among those performing remote computing, the average transmission to a center or from a center is 160,000 characters. Twenty such jobs may be processed daily for a total of 3 hours of on-line time where the speed limitations are card read and print speed operation. These surveys also indicated a wide spread in opinions that potential users have on the amount of usage the typical decentralized terminal would receive on a daily basis. The opinions thought to be conservative indicated that half of the users expected over 2 hours of use a day.

Communication technology

A brief review of the growth of communications can add perspective needed to help understand the impact of the more recent advances in communication technology.

It was one hundred and thirty years ago that Morse perfected his telegraph system which led to what is regarded as the first digital communication system. Later advances in terminal equipment led to the development of teletypewriters which automated the functions of coding, decoding and printing. About ninety years ago, a voice system was born when Alexander Bell perfected the telephone. The telephone system used larger bandwidth channels and more expensive transmission, but because of the phone terminal simplicity and automatic dialing the labor costs to provide phone service were minimized. The modernization of the telegraph system also allowed economies, i.e., automatic cross office, store and forward reperforator systems, automated and improved service. However, in the public telegraphic systems the manual effort still required in the acceptance, transcription, record keeping, and delivery of messages precluded any lowering of costs with the growing labor expense over the years. While there were no competing systems for the telephone traffic, the mail offered competition to the telegraph services because of the increasing speed of delivery made possible by the advances in aircraft technology. Only in recent years have public automatic dial circuit

switching telegraph systems such as Telex and TWX been introduced. Like the voice system, much of the common carrier manual labor has been eliminated and the burden of responsibility for the message content has been left with the subscriber.

The high speed solid state electronics that made possible the accelerated growth of the computer industry has placed new expanded requirements on the communications industry. It is this same high speed solid state electronics that will allow the communication industry to meet these requirements. The existing communication technology is shown in relation to the new emerging technology in Table I. These new technology areas and improvements are discussed in the following sections in light of their impact on the data communications systems to be.

Transmission technology

The transmission technology has developed over the years in areas associated with the categories of distance over which it is applied. Local loop transmission connects subscribers to the common carrier wire and repeater centers and local exchanges up to 15 miles in distance. Tributary transmission spans up to 150 miles to larger centers or tributary exchanges. Medium haul transmission extends over distances spanning continents. Long haul covers up to 5,000 miles of transmission between continents.

The advancing transmission technology has made improvements possible in all areas of short and long distance transmission. Reliable transistor technology has made possible more than one alternative to increasing wire line capacity. The T-1 system uses digital repeaters located along the length of a twisted pair and permits up to 1.544 megabits for 24 digital voice channels on the wire. This T-1 system is planned to be one of a class of cable T-systems that will allow in its largest configuration the capability to pass 281 megabits. These systems are expected to enable a significant reduction of the digital traffic error rate. They also have the capability to carry analog traffic as efficiently as analog facilities.

The microwave technology for tributary and medium haul transmission has permitted increases in data carrying capacity through the use of extended radio spectrum and digital modulation methods. Digital baseband microwave systems are capable of transmitting data time division modulated at a rate more than an order of magnitude greater than conventional microwave frequency division modulated. As an example, a modern type microwave system operating within a 30 MHz channel bandwidth in the 6 GHz carrier band has the approximate capability as shown in Table II. The microwave analog transmission system has a greater capacity for analog signals.

The transmission technology in the past was involved with frequency stability and linearity. The transmission technology today is concerned with exploiting the wideband transmission methods. It is involved with time and phase linearity using digital equipment and is ideally suited for digital communications. Associated with this trend toward high capacity is a lowering in the cost of facilities. Over the past twenty years the capital costs of terrestrial transmission have come down by almost two orders of magnitude.

The digital transmission facilities available for use in the data communication systems are listed in Table III. The digital systems have a decided advantage because pulses are regenerated in the transmission and multiplex equipment, minimizing the possibility of noise and distortion accumulating to cause data errors. In the past there were not enough users to justify a complete digital facility. However, since T-system application studies have shown that analog signals can be sent over digital plant as efficiently as on an analog facility, the new data applications, as well as the conventional digitized analog communications applications, can act as the spur for the rapid growth of this technology.

While long distance transmission costs are an important factor in data communications, if they were reduced to zero one would still be left with a significant cost for the service. Local loops, multilexing, switching facilities and operational functions cost would make up the remaining cost for the service. Advances in these technologies are discussed next.

Multiplexing technology

While the FDM subdivision may not be optimum for the data communication plant in the future, at the present time the data communications systems are bound to use these channels. It has therefore been necessary to establish a spectrum of data channels that correspond to the capacities of the present FDM plant. The favored high quality channels have always been reserved for data over the analog frequency division multiplex equipment. It was convenient in the past to favor data in this way because data was a small percentage of the total. However, with the growing demand for data and the attendant shortage of high quality channels it will be necessary for a transition to time division multiplexing.

Only in rare cases in the past—and these were with special electromechanical equipment—was time divi-

sion multiplexing ever implemented. In recent years, however, with the introduction of high speed, low cost, solid state elements time domain technology has appeared promising. There are a number of time division techniques used to concentrate data signals onto a high speed data line. Important differences among the various multiplex method are involved with the manner of clocking the data out of the terminal and the number of transmission bits per data bit. It is essential at TTY speeds and it is desirable at high data speeds to use the multiplex method that allows the input data to run at its own clock speed rather than be dependent on the multiplexer. This form of multiplexing allows data signals to be multiplexed through tandem stages of multiplexing in a common carrier plant. The interleaving of data streams is, however, complicated by allowing the data terminal clocks to be independent.

The Dalcode is a multiplex system in operation for over a year. It is a Western Union acronym that stems from the equipment use as a *da*ta *l*ine *co*mbiner and *de*multiplexer. Similar types of equipment have been produced by other manufacturers. Basically the system accepts asynchronous characters from TTY units and higher speed terminals and multiplexes them onto one synchronous data stream. It may be regarded as a wired program type store and forward system that operates on a single character per input line rather than on a message basis.

Another class of multiplex equipment that is not code sensitive is termed a Transparent Multiplex System. It involves less efficient transmission since several transmission bits are required for every data bit. The period of the data bit is quantized and coded. Faithful reproduction of the relative crossover point positions are important in transparent mux systems since this allows the data stream to operate at any code and speed. The multiplex method is not economical in the use of transmission bits and therefore might be only considered for use on T-systems where the large transmission bit rate capacity is available at low cost.

Synchronous Time Division Multiplex Systems are those that provide clock for the individual input terminals. The system is efficient for data transmission as well as multiplexing; however, signals cannot be routed through tandem multiplexers unless a common clock system is used universally in the carrier's facility. The technique of a common clock has proven awkward in the past. Also it is inconvenient to synchronize the TTY electromechanical distributor from a clock different than the TTY unit local power source.

Other multiplex methods, such as Pulse Stuffing, are designed to overcome the problem of independent data clock operation of synchronous very high speed data signals. This form of multiplex operation has several variations and some require several transmission bits for each data bit.

Because of the phenomenal growth of data and the trend to use data processors as message switch controllers, the number of circuits in a network terminated in a single center has increased. This means that there are a large number of low speed circuits transmitting over long distances. To minimize this proliferation of low speed circuits, Western Union has employed the Dalcode. The Dalcode was designed for two applications, as a back-to-back time division carrier system and as a computer subsystem. It is used to combine traffic from asynchronous communication lines having the same or different speeds and variable character lengths which it transmits on a time division basis to a remote location over a voice grade channel synchronously at 2400 baud. For a back-to-back configuration, as shown in Figure 2, another Dalcode will dechannelize the signal at the receiving end at the same speed and character length at which it was sent. Operation in the reverse direction is identical.

The Dalcode, as a computer subsystem, is capable of operating directly into a computer without the need of a back-to-back configuration. The computer requires a 2400 baud interface and software necessary to decommutate the Dalcode signals. Private wire systems and such Western Union computer communications systems as SICOM are using Dalcode time division networks.

The Dalcode system is only a forerunner of the time division technology of the future. Time division technology is flexible enough to be used for the same purpose as existing older systems and in most present applications it is simply used as a substitute. In itself, however, it has advantages that have yet to be exploited. A major advantage is its capacity to handle large numbers of channels. With 2400 baud modems, 21 TTY 110 baud channels can be handled on a single voice line. With the use of modems at 4800 baud operation, 45 TTY 110 baud channels are possible on a single voice line. Modems with even higher baud rates are promised, making the comparison with the F.D.M. equipment capacity even more favorable. A comparison of the characteristics of the two technologies are shown in Table IV. Notable in the comparison is that the Dalcode system will allow error control. Also, with some modification that involves allocation of time slots (discussed later) the system can be used as a combination of switch concentrator and multiplexer with great economies possible.

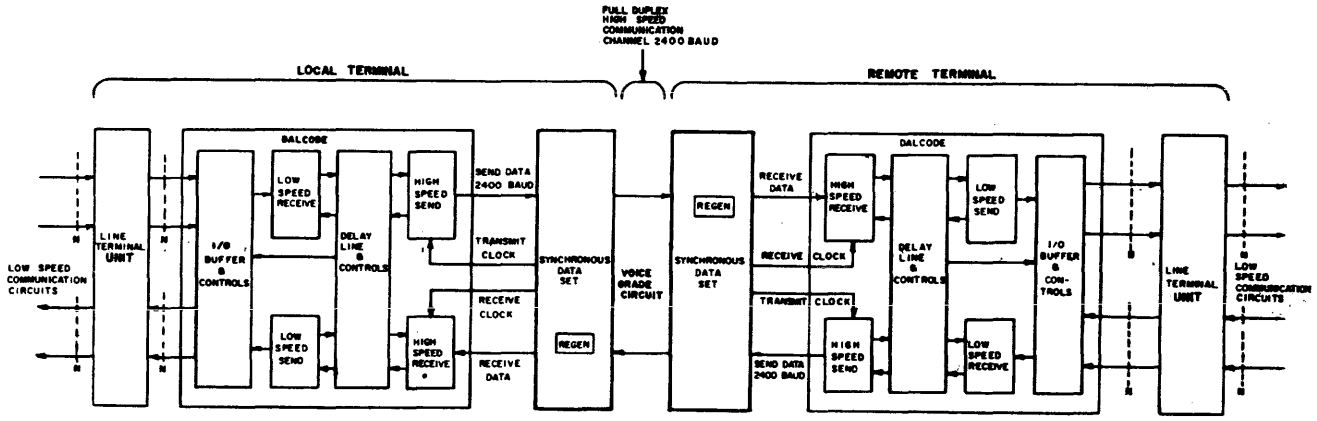


Figure 2 – Back-to-back configuration of two dalcodes

Switching systems

Achieving a cost effective data communication system requires making efficient use of transmission facilities which insures that the cost per unit of data transmitted will be a minimum. Data switching systems are designed to make efficient use of the relatively expensive long distance transmission facilities. Two standard classes of switching systems, message and circuit switching, are designed to share the use of long distance transmission facilities and enable connections to be made between these users.

The circuit switching systems enable direct connection to be made between subscriber terminals, allowing real time communication. The circuit switch system is generally designed so that there is a high probability that a circuit is available at the time the service is required. To make efficient use of these transmission trunks and also provide a good grade of service, it is necessary to have a system with a large number of users and large trunk bundles. The circuit switch systems are designed to provide a high probability of connection during a busy hour. During other periods the trunk utilization is low. Thus, unless circuit swithch systems employ special load leveling techniques, the overall transmission efficiency will be lower than that in message switch systems. In general, circuit switch service is economical for the subscriber who has a light data communication load since he is billed only for his usage time. If a customer uses his data system over two hours each day he is often better off obtaining dedicated transmission facilities.

Message switching systems operate by accepting a message without delay in storage directly from a subscriber terminal. Since stored messages may be queued, the message switch system makes excellent use of transmission facilities; however, the storage control and accounting functions needed to keep track of the message involve equipment that is relatively expensive. The message switching system can be used efficiently in large scale applications such as Public Message system or Autodin or small dedicated systems where storage is in the terminal in multipoint way system operation. It is also evident that if high volume data messages were to be sent over a classical message switch system, large expensive memory would be necessary.

Current message and circuit switching systems are shown in Table V and they are grouped according to categories of private dedicated system or public shared system. It is noteworthy that there are no examples of dedicated data circuit switch systems and this is explained by the fact that they only make efficient use of trunk facilities if there is a very large number of users.

Subscribers on switching systems generally have very specialized communication needs which have been adapted to the services offered. The format, quantity and timeliness of the data is determined by the particular subscriber operation. How the new technologies will influence the classes of communications systems are considered below in terms of the fundamental parts: transmission, switching and terminal systems.

Shared store and forward systems

The technology of transmission, switching and terminals for the Telegraph Public Message system is represented in Figure 3. The electromechanical automatic paper tape reperforator switching system has been in operation for 20 years. One of the characteristics of the Telegraph Public Message System is the relatively large amount of manual work involved in the acceptance, transcription and delivery of mes-

sages. To reduce these costs the new C.R.T. editing terminals might possibly be used as a means to eliminate much of the manual handling before the messages enter the switching and transmission system. The electromechanical switching equipment which had advantages over the torn tape centers might be replaced by electronic computer systems with advantages of speed, reliability, and flexibility. Accounting functions can be performed as well as specialized real time services for dedicated users.

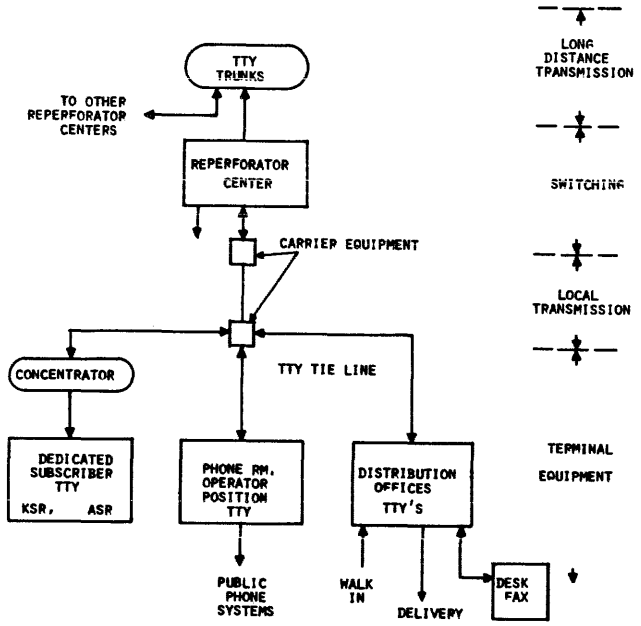


Figure 3 – Block diagram – shared store and forward system, public message

Since the store and forward switching centers are relatively complicated, it is more cost effective to have relatively few processing centers and many multiplexing concentration points that funnel the traffic to the processing centers. These smaller centers can use the more efficient time division methods for funnelling the lower speed traffic into higher speed trunk bundles. As already discussed, it is less expensive to transmit each element of data on a large trunk bundle than a smaller one. Figure 4 is a representation of the existing and new time division concepts being considered for implementing store and forward systems. The newer approach shows fewer centers and employs high capacity trunks. The routing of traffic may be less direct extending over longer distances, but the economies achievable because of the use of the larger trunk bundles might make up for it. Forward error correction methods can be employed economically on the high speed T.D. lines rather than retransmission methods that consume processing and transmission capacity.

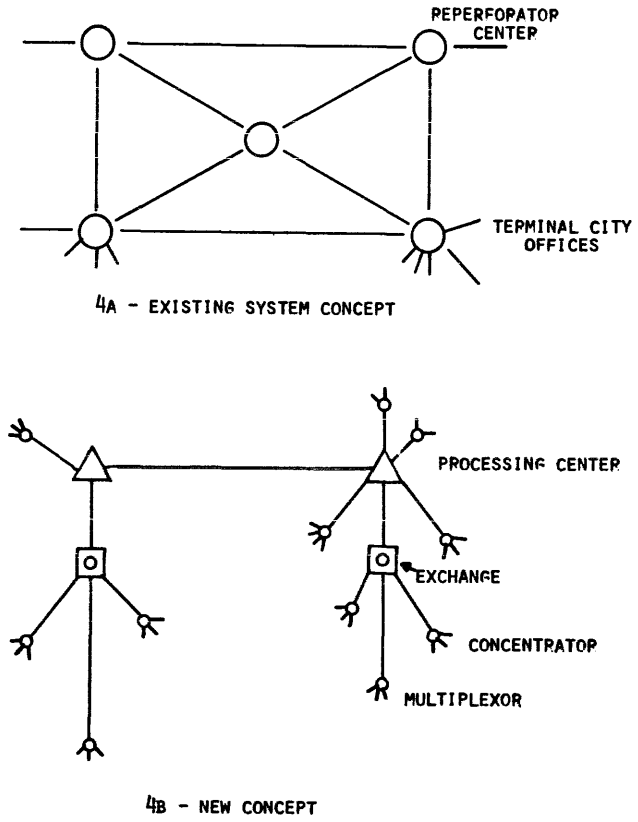


Figure 4 – Old and new concepts – shared store and forward system

Private wire store and forward systems

Private Wire Systems are communication systems designed for corporations with a communication requirement between two or more terminals. Generally the terminal usage is very high and the economies are such that the organization employs dedicated communication facilities rather than shared facilities. The facilities might be Full Duplex or Half Duplex so that data can be sent in both directions simultaneously or in only one direction at a time. When more than two terminals exist on one facility it becomes necessary to coordinate or control the flow of traffic to obtain full efficiency from the dedicated lines. Private wire systems are generally high volume and the limit for the maximum number of terminals on one circuit is generally between five and twenty, depending on the volume of traffic. To coordinate the traffic, selectors and controllers are used in a polling and answer back operation. These operating procedures allow all the terminals to make efficient use of transmission. Each terminal is generally a ASR type of TTY unit where messages on paper tape are leaded in to a tape reader. The terminal unit may be polled or contend for the line before the waiting message is sent.

The selector systems, in effect, are a store and forward system governed by a control unit. When a large number of multipoint way circuits, each with many drops, are to be governed from one point, a computer system is generally programmed to act as a controller for all circuits. The complexity of the computer system real time control program depends on the number of drops and the control functions to be performed. Figure 5 is a block diagram representation of a Private Wire Selector System.

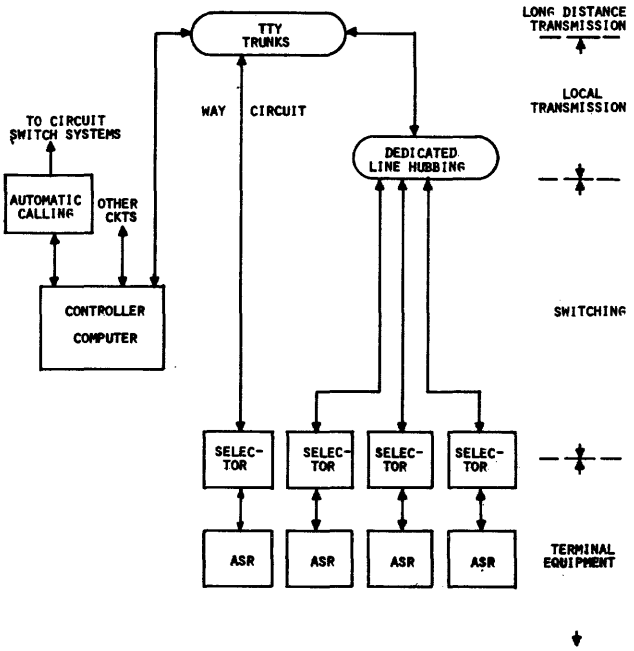
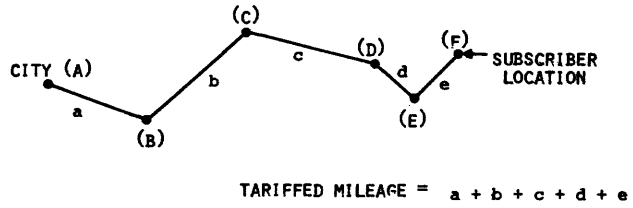


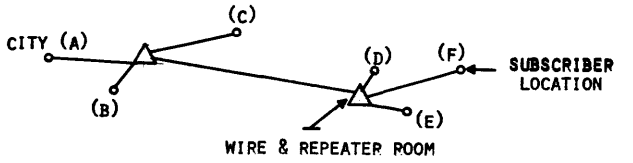
Figure 5 – Block diagram – dedicated private wire systems

The advancing new technology has already had considerable effect on private wire systems. Computer systems used extensively in industry have been enlisted to perform real time communication control functions. Selector systems have also been implemented with solid state electronics and further advances in reduced size and system flexibility are being made with the new microcircuit technology. In the new high speed selector systems presently being installed more complex selector functions are required with the use of the new high speed tape terminals operating at 1200 and 2400 baud. These more complex operations involve the use of error checking functions on data block transfers.

Figure 6 illustrates the point to point, tariffed mileage between terminals in a Private Wire multipoint way system. In actuality, the lines connect wire and repeater room locations where transmission facilities are interconnected. Hubbing repeaters are used at these locations to connect the individual subscribers. Centralizing the selector equipment in W & R rooms will allow placing selector functions on a single W & R room computer that operates on a time shared real time basis for all selectors in the area. This centralization arrangement will also facilitate trouble diagnosis.



TARIFFED MILEAGE IN WAY SYSTEMS



ACTUAL TRANSMISSION ROUTES IN WAY SYSTEMS

Figure 6 – Way system connectivity

A service called Infocom is presently planned for use in Private Wire Systems. The service involves the use of a message store and forward system under control of the processing center computers in the configuration shown in Figure 4-B. The processing center stores the private wire family of addresses for each subscriber and only allows communication between users in the family. While transmission is more indirect, the low cost of shared trunk bundles more than makes up for the additional mileage. In addition, the center can perform message processing functions for the family of terminals.

Circuit switch system

Circuit Switching Systems for data communication users, such as Telex, TWX, Broadband and Data Phone, enable connection between any two of a large number of generally low usage lines. Figure 7 shows the basic block diagram for the generalized circuit switching system. Low usage generally involves 6 to 12 minutes during busy hours with a half hour per day total service requirement. The number of trunks between switches is in the range of 25% of the number of data subscribers to provide a high probability of connection when requested during the busy

period. Dial signals preceding the data flow are used to set up a connection. The dial information is decoded in registers and matrices are switched with appropriate signalling between calling and called party for the connection to be made.

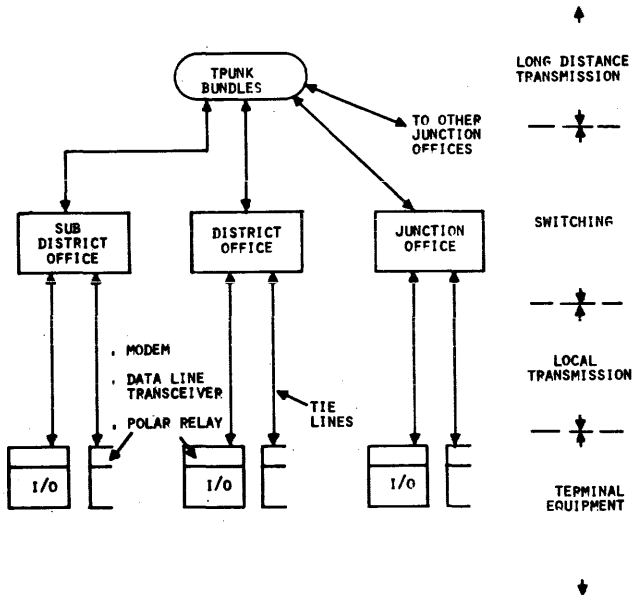


Figure 7 – Block diagram – circuit switching systems

The service is billed on distance and timed usage. It is relatively expensive for high usage long holding time traffic. Technology can offer improvements in circuit switching services by reducing cost of switching equipment, making the services more reliable and allowing the introduction of special billing and load leveling services. Table VI is a list of the new services that programmable common control systems will allow. All of these services are designed to facilitate the setting up of connections, provide reliable service in the case of unusual traffic or transmission conditions, and allow special classes of services for special subscriber requirements. One capability thought to be desirable is one which will allow subscribers the capability to operate at different speeds interchangeably. This is thought to be feasible by the integration of the various circuit TTY, V.F. and Base Group trunks into common exchanges, and this will allow one subscriber to initiate, as the need arises, low bit rate TTY traffic as well as data rates up to 50 K baud. Arrangements to switch 40.8 kb, 50 kb and 200 kb traffic are presently under consideration.

Time division switching systems

Time division multiplexing has been described as having great promise for configuring cost effective communication systems. However, there are many other time division opportunities that can be taken advantage of to improve the communication system operation. The subsystem building blocks employing time division technology from which a data communication system may be configured, are first defined below. These blocks are then configured to make up systems for carrying the four categories of data communications discussed earlier.

Digital transmission facilities (DTF)

The transmission facilities available for digital communication discussed previously are tabulated in Table III. A more cost effective data communication system will result if the higher capacity facilities are employed and efficiently utilized. In system configurations this often requires concentrating traffic in trunks directed to a relatively few switching or processing centers.

Time division multiplexing (TDM)

Time Division Multiplex systems like the Dalcode efficiently channelize asynchronous signals into V.F. facilities. They have the potential of an order of magnitude more channel capacity than an FDM system. TDM systems are also employed to channelize high data rate synchronous signals.

Time division concentrator (TDC)

The digital multiplexer TDM systems always allow a time slot channel of transmission for every terminal. A TDC system would be one which has more data lines than time slot positions on the high speed transmission link. The concentrator therefore functions to allocate a time slot position as the data users require it. A particularly useful mode of operation for the TDC enables the concentrator equipment to remember the request for service and ring back or establish the connection as soon as the time slot becomes available. This operation affords both improvement in trunk efficiency and is a convenience for the user.

Time division exchanges (TDX)

The TDC and TDM systems described are envisaged as routing traffic over a single V.F. data facility. Several of these V.F. lines would terminate in a higher level TDX system that will further multiplex or concentrate the traffic on higher speed trunk bundles. This higher level equipment will have the capability to identify routing requests on the individual

time slot channels. The equipment, by redistributing data from one time slot to another in response to the routing service request, provides the functions of a switching exchange. Other functions, such as alternate routing and priority requests, might also be provided

The TDX, as has been defined here, is different than any other exchange previously built, or to the writer's knowledge, planned. Conventional systems require that each individual signal be dechannelized to a specific line and then the switching equipment services each line independently. The conventional switching operation may involve either space division or time division sampling switching techniques. The TDX will not require independent demultiplexing equipment for the switching operation. Switching would take place by the reallocation of a time slot for a given user in response to a routing requirement. In fact, the operation might approximate that of a computer store and forward system with the capability to store no more than one or two characters at one time. The TDX system therefore functionally approximates both a circuit switch and a message switch. The system is expected to be low cost and reliable since multiplex equipment terminations are not needed and since the switching operation does not involve any electro-mechanical equipment. Since the signals are transmitted and switched at the high data rates and regenerated through logic in the system, there will not be any signal distortion or noise interference in the switching operation.

Time division way systems (TDW)

The TDC previously described involved line terminations located at one point. Some systems involve a distribution of terminals at varying distances along a circuit. A system called a TDW would have characteristics of a TDC with distributed terminations as in a multipoint way wire system. A voice line, rather than a single teletype channel, would connect the users. The time division signal on the voice line would be made up of the individual user data inserted in the appropriate time slot at the point of connection to the line. One central point would control the synchronization. The system may also be put into a high speed mode of operation with the terminals operating as high as 2400 baud. In this case, since only one subscriber can send at a time, polling and answer back operation would convert the system to function as a high speed way wire system.

Processing center (PC)

A processing center performs all those message processing functions not connected with the direct transmission of messages. The center will store the messages and initiate their transmission at the appropriate time. Services such as code translation, message editing, multiple message control, billing and other on line and off line functions are performed in the processing center.

The building blocks of data communications systems defined above may be used to implement the four categories of data communications shown in Figure 1. Category I involves the telegraph message. The configuration of the building blocks to perform this data communication service corresponds to the message store and forward operation in its modern form as described in Figure 4-B. In general, it would involve the use of a TDM funnelling traffic into a processing center. For a large system and to increase efficiency of line utilization, TDC's and TDM's would direct traffic into TDX's and from these to PC's. The main function of the TDX's would be to concentrate or multiplex traffic onto a high capacity trunk bundle. The switching operation would involve rerouting of traffic over different transmission paths for purposes of load leveling or fallback.

Category II involves the interactive terminal type of traffic. As described previously, it consists of a small number of characters in the form of a short message between an individual and a computer. In both cases, when the messages are either originated by the human or the computer, it is desirable that the communication system does not block the operation. Dedicated lines or direct circuit switch connections commonly in use for this application are a significant cost factor for the service, particularly over long distances. Since the central processor unit is addressed on the average only 2% to 4% of the time in the period of interactive terminal connection, most of the time the communications are in an idle condition. The most economical system would be one which establishes the connection every time the interactive terminal mesage is sent. However, the operation of setting up the connection with conventional circuit switches is a relatively long manual dialing effort. The TDC and TDX can be designed with the capability of instantaneously setting up the connection every time the terminal is addressed, since they do not contain electromechanical equipment. Thus, using this equipment, the transmission economies of circuit switch operation can be attained at no loss in operating convenience.

At present blockage might develop in the system because of the limitations of the central processor unit to handle simultaneous user queries. When systems get larger and the communications systems become more efficient the communications system

blockage might be integrated with the computer limitations of handling traffic. It may prove efficient to enable the time division equipment to buffer a block of characters, if an interactive terminal message cannot be accepted for transmission immediately. The time division system would then operate as a store and forward system, allowing greater economies in transmission as well as greater computer processor throughput.

Data communication Category III represents the volume data transfer now sent on voice lines and Category IV represents that sent on base group facilities. In each case the traffic is sent on either circuit switches or over a dedicated connection. The switching system configuration that will allow reduced costs will be one which makes more efficient use of transmission. One way of performing this would be to enable the data to be sent in a simplex mode where data are sent in one direction only. The return transmission normally associated with a duplex operation can be assigned another function. In present systems the return loop is sometimes used in retransmission type of error control. However, time division high speed circuits can efficiently employ forward error correction equipment, eliminating the need for a return loop. Another method for improving transmission efficiency involves the use of scheduled calling or ring back mode of operation for load leveling purposes. This, in effect, takes advantage of the storage facilities of the computer subscriber and allows the time division communication system to operate with the computer terminal hardware, achieving store and forward operation and its attendant economies.

At present the major categories of data transmission are segregated on groups of channels according to their unique transmission speed requirements, and never mixed. There is no reason to continue this form of separation on the high speed digital transmission trunks. It should be possible to utilize the same trunk bundle for different mixes of data channels. Thus the high speed data channel during evening hours might use the TTY time division slots. The rearrangement of channels can be accomplished automatically in a TDX under program controls in a modernized transmission control center that replaces the wire and repeater rooms.

Inter-computer communication system

One concept being discussed which will allow more efficient use of computer is to tie computer systems together even across the country via a communication network. This would allow all computer users in the enlarged family to have access to the diversity of programs, make use of unique computer characteristics in particular places and distribute loads. Aside from the computer system planning needed to define the services made possible by this inter-connection, there are unique communication system problems to solve as well.

The characteristics of the communication for interconnected computer systems are such that the information will be transmitted in real time with a minimum of delay and it shall have characteristics of store and forward systems since it deals with message flow. Ideally, the system should have the best features of both message and circuit switching systems: negligible waiting time that favors the high priced human, and an efficient message interface to favor the high priced computer.

Figure 8 represents a possible geographic arrangement of computer systems. Note that these computer systems might appear in clusters around city locations. The most efficient communication plan for these computer systems might involve all combinations of the different types of communication services—dial circuit switch systems or dedicated line systems or even shared message systems. Messages might go directly to the computer from an I/O device or via a way circuit system. The general purpose computer systems might be programmed to control the input lines or a special communication interface designed for this purpose might be used, as is shown in Figure 9. Long distance routing of traffic between computer systems might also be done in the computer or a specialized communication interface. This intercomputer communications function might take on many automatic operations which involve decisions regarding which computers in the system would more efficiently perform the service for operational or load leveling reasons. In addition, the communications function might be made up of a mix of transmission services and, for reasons of cost effectiveness, the system could have the capability to choose the correct transmission facility at the moment. For example, if all dedicated lines are in use, it might delay transmission or make use of the automatic dial system or use the public shared store and forward system. The design of this system can become very complex, involving software, hardware, transmission and throughput trade-offs.

CONCLUSIONS

Four categories of data communication have been defined and they include: the TTY message, the interactive terminal message, and the high volume data messages. The high volume messages are categorized according to their transmission on either voice lines or on base group facilities. The established

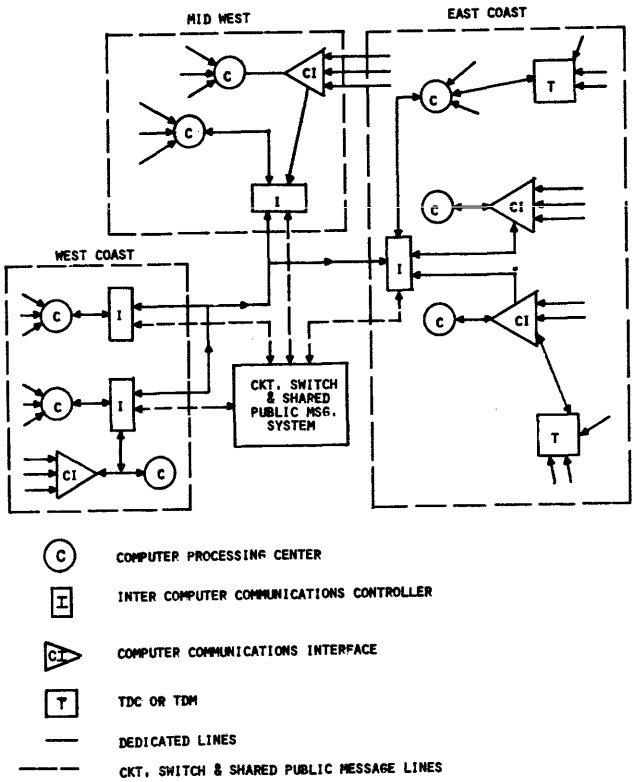


Figure 8 – Block diagram – inter computer communication system

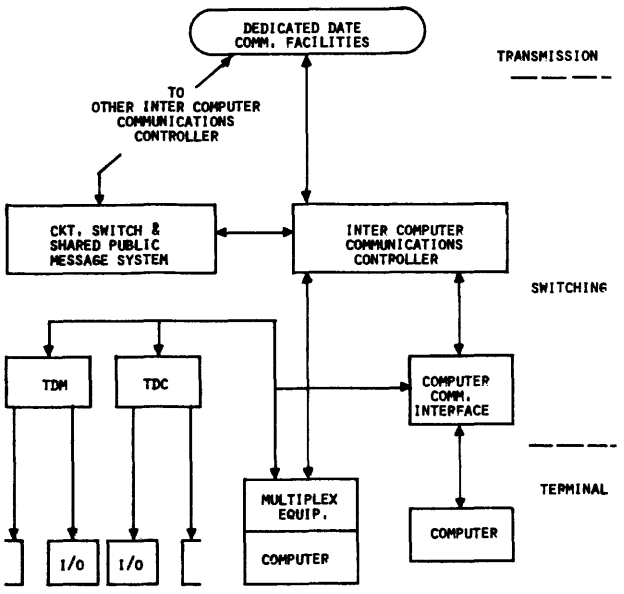


Figure 9 – Block diagram – inter computer communication system

communication systems of today are expected to go through an evolutionary process to meet the data communication requirements in the above categories. The changes in communication services should come about as a result of more efficient transmission systems and time division multiplex equipment, and the use of lower cost flexible switching equipment that will allow high transmission efficiency on trunks.

The advancing technology in transmission has been in the direction of large capacity digital baseband systems. Digital baseband microwave and T-systems are important advances, allowing greater than an order of magnitude increased data transmission capacity over conventional frequency division subdivided channels. The high data rate systems should contribute significantly to reduced transmission error rate and costs. The application of these wideband data systems ought to be accelerated because analog transmission can be included as economic users on some of these high data rate facilities.

The advancing digital multiplexer and modem technologies are making it possible to more efficiently use the existing standard voice, base group, and super group carrier channels for data communications. The data communication switching systems of the future are expected to be a hybrid combination of a computer controlled circuit switch and a message store and forward system. Efficient time division multiplexing systems can be extended to perform concentration and switching, obviating the need for complex, relatively unreliable electromechanical hardware in the data communication exchanges. The store and forward operation, depending on the speed and length of message, might involve storage for one bit or one or two characters, in which case the system will function as a circuit switch, or it might store a block of characters or the entire message itself. Programmable common control equipment can be used to flexibly provide the convenience and load leveling services.

Considering the advancing technology the computer industry can look forward to the time when long distance data communication will be low cost and flexible enough to suit their spectrum of application. The system advances reported here are presently within the state of the art of technology and yet the system advances will take many years to implement. The problems to be tackled will involve aspects of compatibility, investment, regulation and standardization. At present both computer applications and communications industries are learning and adapting to each other. Only with proper coordination, planning and leadership in these industries will the data communication services of the future be effective.

TABLE I – Data Communication Technology Factors

	Existing	*New*
Transmission Technology	• Wire & Cable	• Satellite
	• Radio - H.F.	• Digital Microwave
	• Microwave	• T-System
	• W & R Repeater Rm. Operations	• Computer controlled Transmission Centers
Multiplex System	• FDM	• TDM -Synchronous Asynchronous Code Sensitive
		• High Speed Modems
Terminals	≃ TTY KSR	• CRT's
	• TTY ASR	• High Speed Terminals: Card, Magnetic Tape
	• Paper tape terminals	
Switching - Ckt. Switching	• Electromechanical Matrices	• Time Division Space Switching
	• Electromechanical Common Control	• Time Division Multiplex Line Control
- Message Switching	• Reperforator Cross Office	• ESS 7 Computer Message Switch
	• Electromechanical Way System	• Micro circuit Way Systems

TABLE II – Micro Wave System Capacity

Digital System Options	*Analog System Options*
1 - 46 megabit channel	3 master groups
7 - 6 megabit channels 28 - 1.5 megabit channels	30 super groups
672 - 50 kilobit channels (digital voice capability)	150 base groups 1800 voice channels

TABLE III – Data Communication System Data Transmission Channels

FDM Transmission	TTY	50, 57, 75, 180 bits/sec.
	V.F.	1200, 2400, 4800, 7200 bits/sec.
	Base Group	40.8 or 50.0 K bits/sec.
	Super Group	200 K bits/sec.
Digital Base-band Microwave	6 MHz Carrier	46 M bits/sec.
T systems	T-1 V.F. Channel	64 K bits/sec.
	T-1 system	1.544 M bits/sec.
	T-2 system	6 M bits/sec.
	T-4 system	281 M bits/sec.
Satellite Systems	Master Group	46 to 92 M bits/sec.

TABLE IV – Comparison of TD & FD Multiplexer System Characteristics

	FDM System	TDM System
Channel Capacity	10 - 110 baud TTY Channels	21 to 45 - 110 baud TTY channels
Error Control Potential	none	Error Detection Error Correction
Codes	Transparent	Fixed - but can be changed by change of a card
Speed	Variable up to a maximum baud rate	Variable up to a maximum baud rate
Delay	none	One character
Regeneration	None without special regenerators	Regenerated automatically
Switching System Compatibility	Suitable for Space Division	Suitable for Space and Time Division switching, No extra multiplex equipment necessary for high speed time slot switching.
Computer Interface	Requires per line equipment	Can be terminated directly. Demultiplex function performed by Computer.

TABLE V – Data Communications Switching System Categories

	Public or Shared	Private Dedicated System
Circuit Switching	Telephone System	
	Broadband System	
	W.D.S.	----
	Telex	
	TWX	
	GSA - ARS	
Message Switching	Telegraph Public Message	Corporation Dedicated Systems
	Info-com	Reservations Systems
	GSA - ARS	Multipoint Way Systems
	Autodin	
	SICOM	

TABLE VI – Services Possible with Programmable Common Control

- Camp-on - Ring back
- Answer Back
- Traffic Measurement
- Priority
- Distinguish Classes of Services
- Quick Line and Abbreviated Dialing
- Restricted Access
- Network Rerouting
- Interface with Other Services
- Conferencing

REFERENCES

1 J E CALDWELL
Computer controlled subsystems
Western Union Technical Review January 1967

2 E H MUELLER et al.
Analog and digital transmission capability of microwave systems
Western Union Technical Review August 1967

3 D F HOTH
Digital communication
Bell Laboratories Record February 1968

Analog computer simulation of semiconductor circuits

by PHILIP BALABAN and JOHN LOGAN
Bell Telephone Laboratories, Incorporated
Holmdel, New Jersey

INTRODUCTION

This paper describes new simulation techniques which are being used in the analysis and design of integrated circuits. They have also proved advantageous in the characterization of semiconductor devices. Two approaches are available; the breadboard method, a method based on retaining circuit topology, and the analog method which is based on traditional analog computer programing techniques.

The techniques are based on a proposal by Gummel and Murphy[1] in which they suggest using the low frequency component of the actual junction current as a measure of the charge stored in a semiconductor junction. This idea has subsequently been named the Separation Principle since it allows the nonlinear dc characteristic to be separated from the transient behavior of the device.

The breadboard method uses the transistors themselves as computing elements in the simulation much as in the real circuit, thereby maintaining topological similarity. The analog method relies on a simple representation of a semiconductor junction, using a diode to provide the junction nonlinearity. Transistors are then programmed as two interacting semiconductor junctions allowing access to internal device parameters which are not available in the breadboard method. Topological identity, however, is generally lost.

The need for these methods arises from the fact that other computer aids to the design of complex circuits are not entirely satisfactory, particularly if realistic models of the active devices are required. In analog simulation, circuit size has previously been limited by the available number of computing elements. In digital computation storage capacity and excessive solution times have imposed the limitation. These solution times rule out optimization routines when many runs have to be made.

The present approach provides adequate models in an efficient manner, the programming is relatively simple, and parameters can be readily varied. Many designs of reasonably sized circuits can be explored quickly, sensitivity investigated and the design optimized. Thus it is felt that a significant step has been made in reducing circuit design turnaround time.

The separation principle for diodes

Theory

Charge control theory for diode operation shows[2] that the diode current can be written as

$$i_d = \frac{q}{\tau_f} + \frac{dq}{dt} + C_j \frac{dV_d}{dt} \tag{1}$$

where i_d is the instantaneous diode current
q is the minority carrier excess charge stored in the device
τ_f is the minority carrier lifetime
C_j is the junction transition-region capacitance
V_d is the voltage across the junction.

It is convenient to define

$$i_R = \frac{q}{\tau_f} \tag{2}$$

At low frequencies, as used on the analog computer, only the current i_R is observed since the derivative terms are many orders of magnitude faster. Equation (2) shows that the low frequency current i_R is a measure of the charge stored in the device. Thus Equation (1) can be rewritten as

$$i_d = i_R + \tau_f \frac{di_R}{dt} + C_j \frac{dV_d}{dt} \tag{3}$$

Time scaling may be applied such that

$$T = \alpha_t t \tag{4}$$

where α_t is the time scaling factor. Equation (3) then becomes

$$i_d = i_R + \alpha_t \tau_f \frac{di_R}{dT} + \alpha_t C_j \frac{dV_d}{dT} \tag{5}$$

The effective life time is $\alpha_t \tau_f$ and the time scaled transition region capacitance is $\alpha_t C_j$.

The breadboard method for diodes

If a small sensing resistor r is placed in series with the diode as shown in Figure 1, then the low frequency current i_R flows through this resistor developing a voltage ri_R.

A differential amplifier having gain A is connected across the sensing resistor r to give an output voltage

$$V = Ar\, i_R \tag{6}$$

This voltage is applied to capacitor C connected as shown in Figure 1. If the voltage across the diode and sensing resistor is small compared with V, then the current through the capacitor is

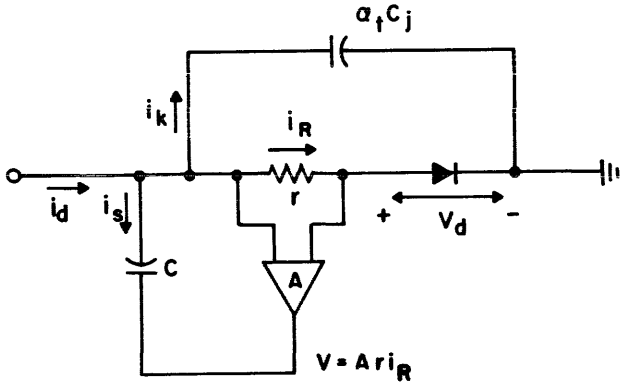


Figure 1 – Breadboard method for applying separation principle to diodes

$$i_s = C\frac{dV}{dT} = CAr\frac{di_R}{dT} \tag{7}$$

If $CAr = \alpha_t\tau_f$, then i_s gives the second term on the right hand side of Equation (5) and the time scale factor is given by

$$\alpha_t = \frac{CAr}{\tau_f} \tag{8}$$

A capacitance of magnitude $\alpha_t C_j$ is then connected as shown in Figure 1 and assuming that

$$ri_R < V_d,$$

the voltage across this capacitor may be taken to be V_d.

Typical values of r $= 10\Omega$
$i_R = 1$ ma and
$V_d = .7V$
gives $ri_R = .01V$ which is much less than V_d as required. The current through the capacitor is

$$i_k = \alpha_t C_j \frac{dV_d}{dT} \tag{9}$$

This gives the third term in Equation (5) and the situation in Figure 1 therefore represents a time scaled simulation of the diode, satisfying Equation (4).
Typical values for the parameters are

$$r = 10\Omega$$
$$A = 10^3$$
$$C = 1\mu F$$
$$\tau_f = 10^{-8}\text{sec}$$

giving the time scale factor, $\alpha_t = \frac{CAr}{\tau_f} = 10^6$

The simulation speeds are one million times slower than the response of the diode itself indicating that any (real time) transient effects in the diode are completely swamped by the external elements C and $\alpha_t C_j$. The current i_R is therefore the low frequency current as required by Equation (2) and the transient currents flow through the external elements showing that the low frequency and transient effects have been separated. An excellent dc representation is provided since the diode acts as its own dc model.

The analog method for diodes

This method is also based on the assumption that the diodes used as models are so fast compared with scaled computer time that only the nonderivative terms of Equation (3) are measured by the analog computer. Therefore, if a voltage V_d is applied to the input of the circuit in Figure 2, the output voltage $V_0 = -i_R R_d$ is proportional to the low frequency or conduction current i_R. Using this relationship, Equation (3) can be implemented using standard analog computer techniques. Taking the Laplace transform of Equation (3)

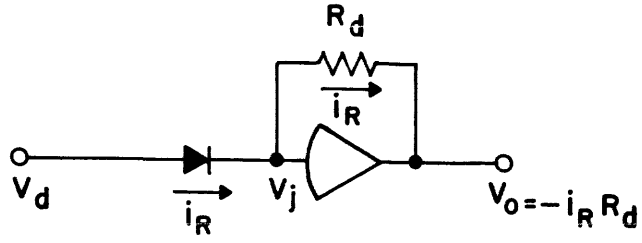


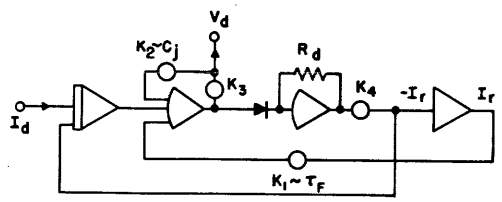
Figure 2 – Low frequency diode simulation

$$i_d(s) = i_R(s) + s\tau_f\, i_R(s) + sC_j\, v_d(s) \tag{11}$$

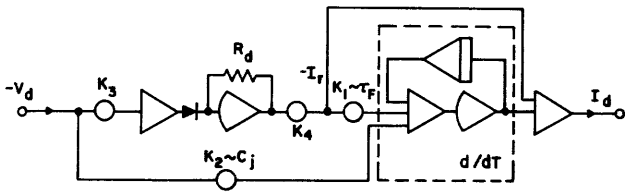
or

$$v_d = \frac{1}{C_j}\left[\frac{i_d(s) - i_R(s)}{s} - \tau_f i_R(s)\right] \tag{12}$$

The diagram in Figure 3a represents the simulation of the diode Equation (12) and Figure 3b the simulation of Equation (11).



(a)



(b)

Figure 3 – Simulated diodes

The scale factors for those simulations are

Voltage $V_d = v_d$
Current $I_d = R_d i_d$

Transition region

$$\begin{aligned} &\text{capacity} && C_j = K_2 R_d \alpha_t C_J \\ &\text{Lifetime} && \tau_F = K_1 \alpha_t \tau_f \end{aligned} \qquad (13)$$

Control of DC parameters

The dc current-voltage relationship of an idealized diode is given by

$$i_R = I_s(e^{\lambda v_d} - 1) \qquad (14)$$

$$\lambda = \frac{q}{Mk\Theta}$$

where I_s = saturation current (a constant)
q = magnitude of electronic charge
k = Boltzmann constant
Θ = absolute temperature
M = a constant which depends on the type of diode used.

The output voltage I_R in Figure 3b is then

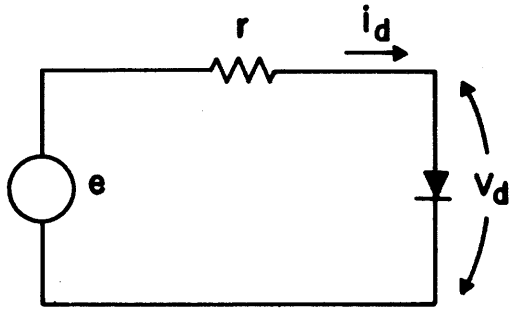
$$I_R = i_R R_d K_4 = (I_s K_4)\, R_d \left[e^{\frac{qK_3 V_d}{Mk\Theta}} - 1 \right]$$

$$= I_s' \left[e^{\lambda v_d} - 1 \right] \qquad (15)$$

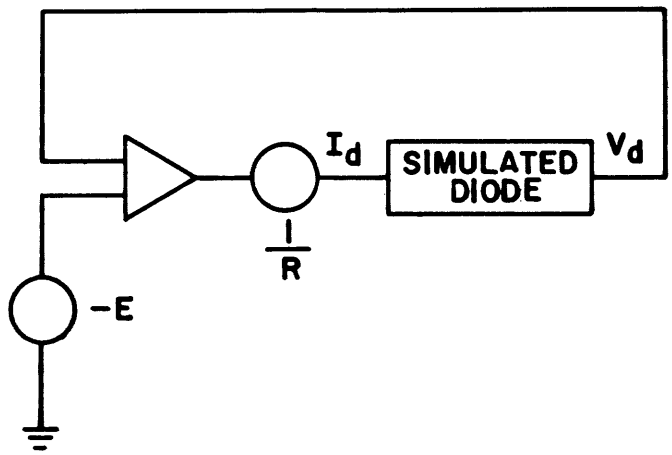
therefore any diode at any temperature can be simulated simply by changing the parameters I_s' and λ' which are easily adjustable by varying the potentiometers K_3 and K_4.

Experimental results (diode analog method)

A diode circuit as in Figure 4a was simulated by the analog method in Figure 4b. The transient parameters of the circuit were $C_t = 2pF$; $\tau_f = 6\times10^{-9}$ sec; $r = 1$ kΩ; $E = 10 \sin(2\pi \times 10^7 t)$. The chosen scale factors were $\alpha_t = 10^7$; $\alpha_v = 1$; $\alpha_I = 1000$; where α_t, α_v, α_I were the time, voltage and current scales respectively.



(a)



(b)

Figure 4 – Diode circuit and simulation

The waveforms V_d and I_d in response to a sine wave are shown in Figure 5. The negative spike "A" in the current waveform is caused by the stored charge in the diode and is controlled by variation of the recombination time τ_f. The slope "B" is influenced by the depletion layer capacity C_j.

The simulation provided very close agreement with experimental diode behavior.

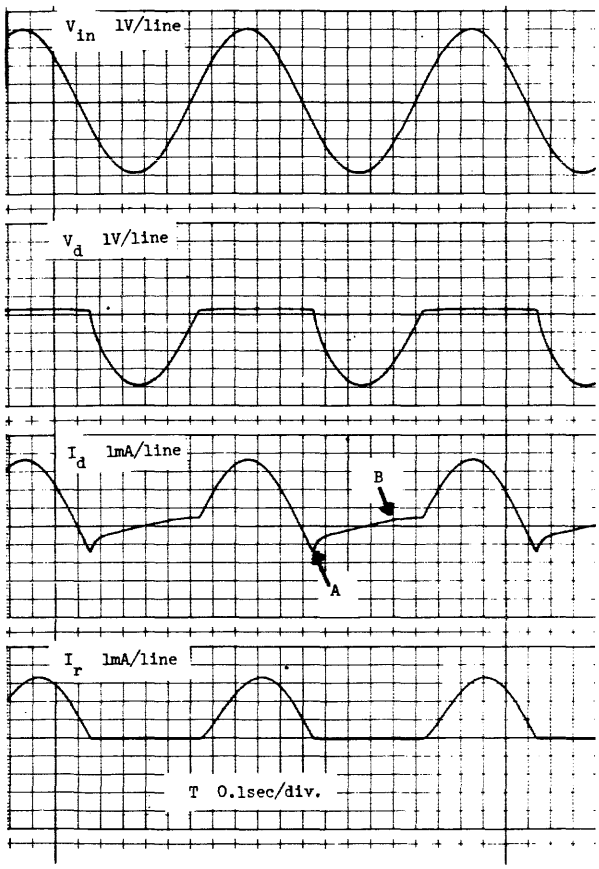


Figure 5 – Simulated diode response

The separation principle for transistors

Theory

The charge control equations for a transistor are[3]

Base Current,

$$i_b = -\left[\frac{q_f}{\tau_{bf}} + \frac{q_r}{\tau_{br}} + \frac{d}{dt}(q_f+q_r) + C_{je}\frac{dv_{ej}}{dt} + C_{jc}\frac{dv_{cj}}{dt}\right] \quad (16)$$

Emitter Current,

$$i_e = q_f\left(\frac{1}{\tau_f} + \frac{1}{\tau_{bf}}\right) - \frac{q_r}{\tau_r} + \frac{d}{dt}q_f + C_{je}\frac{dv_{ej}}{dt} \quad (17)$$

Collector Current,

$$i_c = -\frac{q_f}{\tau_f} + q_r\left(\frac{1}{\tau_r} + \frac{1}{\tau_{br}}\right) + \frac{d}{dt}q_r + C_{jc}\frac{dv_{cj}}{dt} \quad (18)$$

where

- q_f is the forward component of charge stored in the base.
- q_r is the reverse component of charge stored in the base.
- τ_f is the forward injection charge control parameter.
- τ_r is the reverse injection charge control parameter.
- τ_{bf} is the effective base recombination lifetime for reverse injection.
- τ_{br} is the effective base recombination lifetime for forward injection.
- v_{ej} is the emitter junction voltage.
- v_{cj} is the collector junction voltage.
- C_{je} is the emitter junction transition region capacitance.
- C_{jc} is th collector junction transition region capacitance.
- α_f is the forward current gain parameter.
- α_r is the reverse current gain parameter.

Consider the base current i_b and assume that

$$\tau_{bf} = \tau_{br} = \tau \quad (19)$$

Then the low frequency current is

$$i_j = -\frac{1}{\tau}(q_f+q_r) = \frac{q_b}{\tau} \quad (20)$$

where the total charge in the base is $q_b = q_f + q_r$.

The instantaneous base current becomes

$$i_b = i_j + \tau\frac{di_j}{dt} + C_{je}\frac{dv_{je}}{dt} + C_{jc}\frac{dv_{jc}}{dt} \quad (21)$$

which is of the same form as Equation (5). Again, if

$$T = \alpha_t t$$

Equation (20) becomes

$$i_b = i_j + \alpha_t\tau\frac{di_j}{dt} + \alpha_t C_{je}\frac{dv_{je}}{dt} + \alpha_t C_{jc}\frac{dv_{jc}}{dt} \quad (22)$$

where the effective lifetime is now $\alpha_t\tau$ and the transition region capacitances are $\alpha_t C_{je}$ and $\alpha_t C_{jc}$.

The breadboard method for transistors

Figure 6 shows the arrangement for time scaling transistors in a similar manner to that used for diodes. The operational amplifiers provide the effect of a differential amplifier and apply a voltage proportional to the low frequency current i_j across the capacitance C.

The current through capacitance C is

$$i_s = C\frac{d(v-v_b)}{dt} = CAr\frac{di_j}{dt} \quad (23)$$

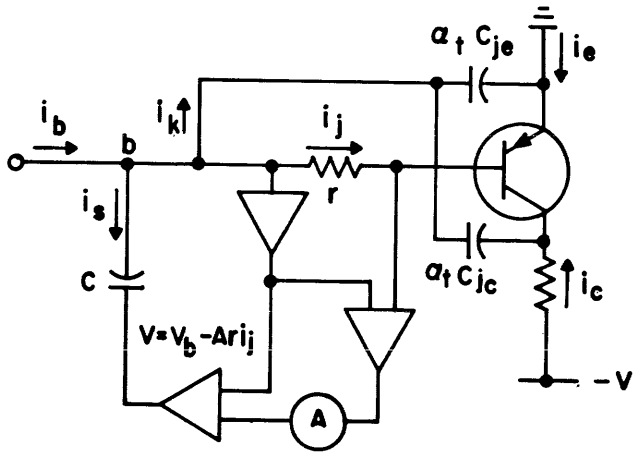


Figure 6 – Breadboard method of transistor simulation

assuming that the voltage drop across the sensing resistor r is small compared with v_{je} and v_{jc} the current into the two capacitors $\alpha_t C_{je}$ and $\alpha_t C_{jc}$ is

$$i_k = \alpha_t C_{je} \frac{dv_{je}}{dt} + \alpha_t C_{jc} \frac{dv_{jc}}{dt} \qquad (24)$$

Thus $i_b = i_j + i_s + i_k$

$$= i_j + CAr \frac{di_j}{dt} + \alpha_t C_{je} \frac{dv_{je}}{dt} + \alpha_t C_{jc} \frac{dv_{jc}}{dt} \qquad (25)$$

If $CAr = \alpha_t \tau$, Equation (24) is identical to Equation (21) and Figure 6 is the simulation for a transistor time scaled by the factor

$$\alpha_t = \frac{CAr}{\tau}$$

It is important to note the assumption that $\tau_{bf} = \tau_{br} = \tau$. This restriction becomes important only when the collector junction becomes forward biased and it appears that the best solution then is to use the analog method described next.

The capacitances in the simulated model are determined by the measured values[6] and the desired time scale. The magnitude of the sensing resistor r depends on the base current level to be detected. Generally 100 Ω resistor is adequate. In cases where this value affects the transistor behavior, feedback can be used to compensate completely for the voltage drop across the sensing resistor.

The advantage of the breadboard approach is that the real transistor is used as its own dc model. All dc nonlinearities and parameter interdependencies within the transistor are therefore available without the need for complicated function generators. The programming and debugging are similar to breadboard circuit building and checking. This generally results in much shorter setup times than in conventional analog programming.

The analog method for transistors

To simulate the transistor the extended Ebers-Moll model of Figure 7 was used. This model is equivalent to those used in digital analysis programs such as CIRCUS[4] or NET 1[5], etc.

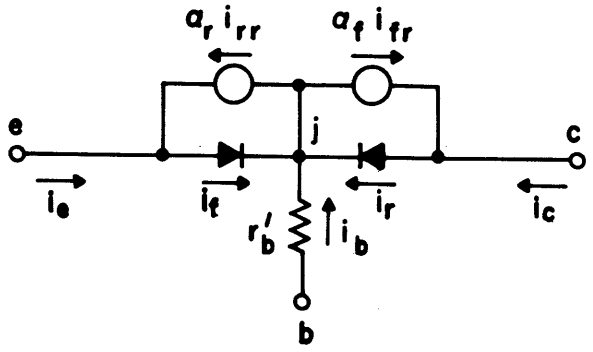


Figure 7 – Modified Ebers-Moll transistor model

The emitter and collector currents i_e and i_c are given in Equations (17) and (18).

Designating the values:

$\frac{q_f}{\alpha_f \tau_f} = i_{fr}$ which is the forward conduction current

$\frac{q_r}{\alpha_r \tau_r} = i_{rr}$ which is the reverse conduction current

and using the relations.[3]

$$\tau_{bf} = \tau_f \frac{\alpha_f}{1-\alpha_f} \; ; \; \tau_{br} = \tau_r \frac{\alpha_r}{1-\alpha_r} \qquad (26)$$

The Equations (17) and (18) can be rewritten as

$$i_e = i_f - \alpha_r i_{rr} \; ; \; i_c = i_r - \alpha_f i_{fr} \qquad (27)$$

where

$$i_f = i_{fr} + \alpha_f \tau_f \frac{di_{fr}}{dt} + C_{je} \frac{dv_{ej}}{dt} \qquad (28)$$

$$i_r = i_{rr} + \alpha_r \tau_r \frac{di_{rr}}{dt} + C_{jc} \frac{dv_{cj}}{dt} \qquad (29)$$

These equations can be implemented on the analog computer using the simulated diodes in Figure 3a or Figure 3b. Either of those diode configurations can be used at the discretion of the designer. One is probably more useful in a loop analysis solution and the other

in a nodal analysis. Mixed methods can also be used. As an example, the NPN transistor amplifier shown in Figure 8 was simulated using the layout of Figure 9. The simulation uses one diode of Figure 3a and one of Figure 3b.

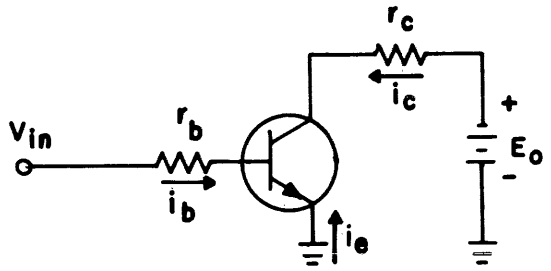


Figure 8 – Grounded emitter amplifier

One of the important advantages of this type of a simulation (compared with the breadboard method) is that the lifetime for the emitter diode, τ_{bf}, and the collector diode, τ_{br}, can be varied independently which avoids the assumptions in Equation (19).

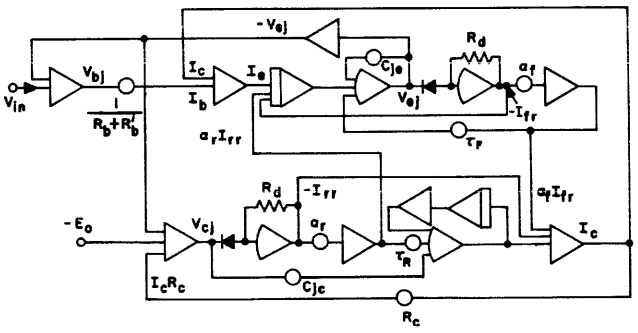


Figure 9 – Simulation of grounded emitter amplifier

Experimental results

To illustrate the analog method the simulated grounded emitter amplifier of Figure 8 was driven by a squarewave.

The transistor and circuit paramenters used in the simulation were: $r'_b = 10\Omega$; $\alpha_f = 0.985$; $\alpha_r = 0.2$; $\tau_f = 1.13 \times 10^{-9}$ sec; $\tau_r = 20 \times 10^{-9}$ sec; $C_{je} = 1$pF; $C_{jc} = 0.3$pF; $E_o = 6$V; $r_c = 500\Omega$; $v_{in} = 10^7$ Hz squarewave 5v peak.

The scale factors were chosen to be: $\alpha_t = 10^7$; $\alpha_v = 1$; $\alpha_i = 1000$, where α_t, α_v, α_i are the time, voltage and current scales respectively.

The resulting waveforms of the base, emitter, and collector currents are shown in Figure 10. Qualitative rather than quantitative tests were performed to observe the influence of the various transient terms on the waveforms. Those waveforms show very close correspondence to those observed in real transistors. The delay time (D), rise time (R), storage time (S) and fall time (F) are clearly seen in the I_c waveform of Figure 10. The spike (B) in the I_c waveform is determined by the charging of the base-collector capacity C_{jc}, the small pulse (C) in the I_e curve is controlled by the charging of the base-emitter capacity C_{je}. The spike (A) in the base current I_b is caused by the charging of C_{je} and C_{jc} and the large negative pulse SC is due to the removal of the stored charge from the junction.

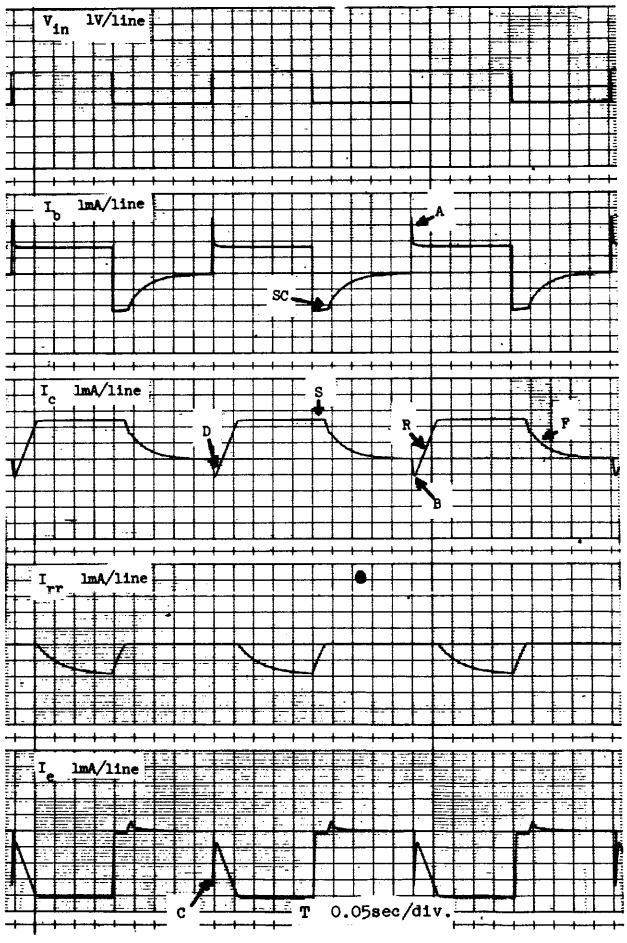


Figure 10 – Simulated amplifier response

Application of the breadboard method to logic circuit simulation has been described in the literature[6] where very close agreement is shown to exist between the simulated logic gate and the actual circuit.

CONCLUSION

A description has been given of methods for simulating diodes and transistors using semiconductor devices as computing elements to generate the nonlinear dc characteristics. Two distinct approaches exist: one, called the breadboard method, uses the actual device and requires sensing resistors for current detection; the other, known as the analog method, simulates the devices using traditional analog computing methods such that currents and voltages appear as analog variables. A comparison of the breadboard and analog methods is summarized below.

Validity of models

Both methods rely on the separation of the low-frequency behavior of the device from its high-frequency response. The analog method replaces voltage sensing resistors by operational amplifiers. This reduces the possibility of swamping any small bulk effect resistances with comparable sensing resistors. For this reason, the simulation of diodes tends to be more accurate in the analog method. In the simulation of transistors, however, the breadboard method appears to have an advantage. It accounts correctly for all effects simulated in the analog method except collector lifetime. In addition it also provides an accurate representation of low-frequency effects such as base widening and nonlinear current gain.

The model in the analog method corresponds exactly to charge control and Ebers-Moll models used in digital programs such as CIRCUS and NET-1. Nonlinear functions, such as current gains α_f and α_r, or transition region capacitances C_{je} and C_{jc} can be readily modeled in the digital programs. These features can also be incorporated in the analog method at the expense of additional function generators. The analog method by its modular nature does allow for a simple simulation of the collector lifetime—a parameter of significance in applications where the transistor saturates. Finally, it allows for the simulation of arbitrary base resistances and changes in the parameters of the dc model. This control enables a designer to do speculative design with devices that are not yet fabricated as well as characterize a device with arbitrary parameters.

Computing flexibility

The question of flexibility can be resolved into two parts. The first question deals with the ease of programming—a measure of the difficulty encountered in translating a circuit sketch to a working computer simulation.

For simulating diodes, the analog method is as simple to use as the breadboard method. A 10-diode logic gate was simulated and debugged in a few hours of analog computing time.

In transistor simulations, the breadboard method, which retains the topology of an experimental breadboard, has a distinct advantage in ease of programming over the analog method. The analog method involves traditional analog programing with little or no similarity to the experimental breadboard.

The second question of flexibility concerns the ability of adapting the circuit simulation to a form suitable for optimization procedures and parameter tolerance analysis. The analog method has a clear advantage for these purposes. There are no external capacitors or sensing resistors used in the method. The traditional analog computer components that comprise the simulation are easily controlled by digital computer software in a hybrid computer arrangement. Optimization procedures and sensitivity analysis become feasible and it becomes possible to envision the circuit simulation as a digital subroutine.

Typical solution times for both methods are 10 to 100 msec per solution. This means a favorable factor of approximately 10^4 in comparing solution times of digital programs such as NET-1,[5] CIRCUS,[4] etc. The implication of this speedup is the ability to do sensitivity studies and apply optimization procedures to the design of integrated circuits.

The separation method has most significant value in problems where time scaling the experimental circuits is useful. Such applications include the analysis and design of high-speed logic gates, microwave circuits, and high-frequency amplifiers.

ACKNOWLEDGMENT

The authors are indebted to C. F. Simone and J. Chernak for active encouragement and contributions in the course of this work; to H. Gummel, B. T. Murphy and E. J. Angelo, Jr. for invaluable discussions on theoretical aspects of the charge control representation; to K. W. Sussman and H. C. Rorden for assistance in developing the analog programs; to J. V. Wait of University of Arizona for many stimulating discussions.

REFERENCES

1 H K GUMMEL B T MURPHY
Circuit analysis by quasi-analog computation
Proc IEEE (letters) vol 55 p 1758 October 1967

2 A W LO
Introduction to digital electronics
Addison-Wesley 1967 p 43

3 P E GRAY et al.
Physical electronics and circuits models of transistors
SEEC vol 2 J Wiley and Sons 1964 p 206

4 L D MILLIMAN W A MASSENA R L DICKHAUT

CIRCUS, A digital computer program for transient analysis of electronic circuits
Harry Diamond Laboratories 346-1 January 1967

5 H F MALMBERG F L CORNWALL F N HOFER
NET-1 network analysis program
Los Alamos Scientific Laboratory Report LA-3119 September 1964

6 E J ANGELO JR J LOGAN K W SUSSMAN
The separation technique—a method for simulating transistors to aid integrated circuit design
IEEE Trans Electronic Computers vol EC-17 no 2 February 1968

A new stable computing method for the serial hybrid computer integration of partial differential equations

by ROBERT VICHNEVETSKY
Electronic Associates, Inc.
Princeton, New Jersey

INTRODUCTION

Partial differential equations involving one space dimension and time can be solved by hybrid computers using the serial (or continuous space-discrete time) method. In so doing, the continuous integration capability of the analog computer is used along the space axis while integration along the time axis is performed in a discrete fashion by making use of finite differences.

The continuous integration problem in the space direction is in many practical cases of a mixed boundary nature, and is furthermore often unstable from the error propagation standpoint.

The purpose of the present paper is to introduce a decomposition method which, under quite broad applicability conditions, allows the spatial integration problem to be separated in a finite number of subproblems, each of them computationally stable and free of the iteration requirement present in the mixed boundary original equations.

This decomposition method applied to the spatial integration problem is to be contrasted with the Green's functions method.[1,4,5] In the latter method, Green's functions of the spatial differential operator are pre-computed, and the solution to the differential problem is replaced by an integral relation to the non-homogeneous terms. This method however, cumbersome in its computer implementation in terms of computing time or hardware requirements. Furthermore, the integral expression of the solution presents unfavorable error propagation properties which are not present in the decomposition method presented here.

The method is first illustrated by the example of the heat diffusion equation and then generalized to more extended problems into a formal way.

Serial hybrid integration of the diffusion equation

We consider the problem of integration of the diffusion equation in one spatial dimension (the slab problem – Fig. 1).

$$\frac{\partial x}{\partial t} = k \frac{\partial^e x}{\partial u^2} \qquad (2.1)$$

$$\left.\begin{array}{l} u \in (0,1) \\ x(o,t) = X_0(t) \\ x(1,t) = X_1(t) \\ x(u,o) = X_u(o) \end{array}\right\} = \text{given boundary conditions} \qquad 2.2$$

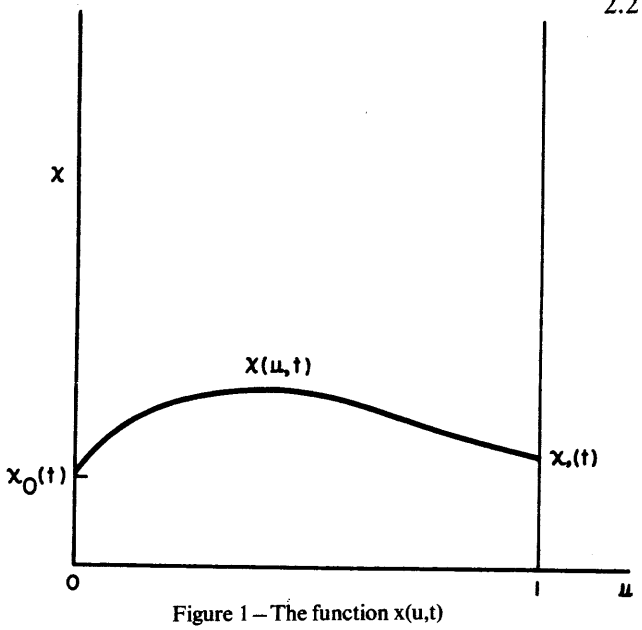


Figure 1 – The function x(u,t)

The serial method of integration by hybrid computation consists in discretizing time and integrating continuously in space. If we denote by $x^l(u)$ the value of x(u,t) at time $t^l = i\Delta t$, then (2.1) can be approximated by

$$\frac{x^{i+1}-x^i}{\Delta t}=k\left(\Theta\frac{d^2x^{i+1}}{du^2}+(1-\Theta)\frac{d^2x^i}{du^2}\right)$$

or

$$k\Theta\Delta t\frac{d^2x^{i+1}}{du^2}-x^{i+1}=-\left(x^i+(1-\Theta)k\Delta t\frac{d^2x}{du^2}\right)$$

which can be rewritten:

$$k\Theta\Delta t\frac{d^2x^{i+1}}{du^2}-x^{i+1}=-S^i \tag{2.3}$$

where $$S^i=x^i+(1-\Theta)k\Delta t\frac{d^2x^i}{du^2} \tag{2.4}$$

Equation (2.3) is an ordinary differential equation in the independent variable u; the expression (2.4) represents stored past information at time t^{i+1} when (2.3) is integrated to produce x^{i+1}. The new value S^{i+1} to be stored is computed by a recursive expression, which can be derived from (2.4)

$$S^{i+1}=x^{i+1}+(1-\Theta)k\Delta t\frac{d^2x^{i+1}}{du^2}$$

or, taking (2.3) into account.

$$S^{i+1}=x^{i+1}+\left(\frac{1-\Theta}{\Theta}\right)(x^{i+1}-S^i) \tag{2.5}$$

An alternative expression is:

$$S^{i+1}=S^i+\frac{1}{\Theta}(x^{i+1}-S^i) \tag{2.6}$$

Since

$$k\frac{d^2x_i}{du^2}=\frac{dx}{\partial t}\Bigg|_{t=t_i}$$

we can see that S^i, as defined by relation (2.4), is in fact the approximated value of x(u) at the time

$$t=t_i+(1-\Theta)\Delta t$$

Thus the serial method of integration described above consists in integrating at time $t=t^{i+1}$ the ordinary differential equation (2.3) which produces the solution $x^{i+1}(u)$ at that time, and storing S^{i+1} (given by (2.5)) which represents both the solution at time $t^{i+2-\Theta}$ and the right hand side of equation (2.3) at the next integration time (Fig. 2). A block diagram representing this sequence of computer operations is shown in Fig. 3.

The spatial integration

We are left with the problem of finding a computing algorithm permitting the integration of Equation

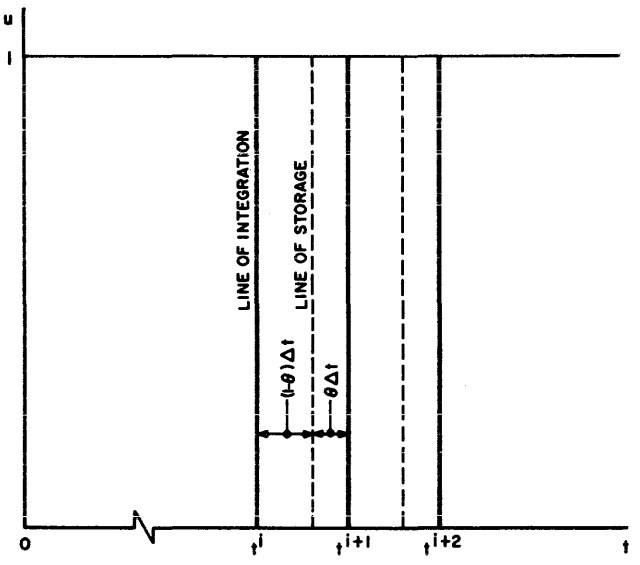


Figure 2 – Lines of integration and function storage in the space-time plane

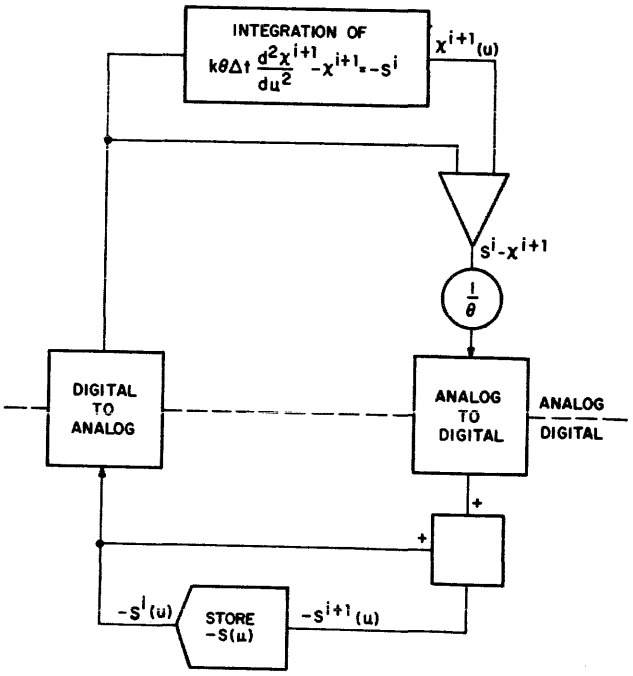


Figure 3 – Simplified block diagram for the serial-hybrid computer solution of the heat diffusion equation

(2.3) taking the appropriate boundary conditions into account.

Let us rewrite (2.3) in the form:

$$\frac{d^2x^{i+1}}{du^2}-\frac{x^{i+1}}{k\Theta\Delta t}=-\frac{S^i}{K\Theta\Delta t} \tag{2.7}$$

The boundary conditions (2.2) yield the boundary conditions of (2.7):

$$\left.\begin{aligned} x^{i+1}(0) &= X_0(t^{i+1}) \\ \text{and} \quad x^{i+1}(1) &= X_1(t^{i+1}) \end{aligned}\right\} \tag{2.8}$$

If we try to perform the integration of (2.7) in either the forward or the backward direction,

$$u \in (0 \rightarrow 1)$$
$$\text{or } u \in (1 \rightarrow 0)$$

we find that this equation is computationally unstable in both directions. It is true that, due to the fact that the integration interval (0,1) is finite, a solution with an acceptable accuracy may be expected; but, in addition, this problem is of a mixed-boundary values nature, requiring an iterative type solution. Since (2.7) is a differential equation of the second order, both x and $\frac{dx}{du}$ have to be known in u = 0 to allow the integration to be started. An iterative solution might consist in assuming a value for $\frac{dx}{du}\Big|_{u=0}$ performing the integration from u = 0 to u = 1, and using the final error

$$x^{i+1}(1) - X_1(t^{i+1})$$

to correct the initial assumption of $\frac{dx}{du}\Big|_{u=0}$ until convergence of the solution to the satisfaction of the final boundary value is achieved.

In view of the unstable nature of the integration, the convergence may prove to be somewhat slow, and the presence of random computing errors, as found in analog computers, may prevent absolute convergence to be practically reached. The method of solution by decomposition described in the following section alleviates these problems.

Integration by decomposition

The differential operator

$$L = \frac{d^2}{du^2} - \frac{1}{k\Theta\Delta t} \tag{3.1}$$

appearing in (2.7) is inherently unstable. Its characteristic equation

$$\lambda^2 - \frac{1}{k\Theta\Delta t}$$

has two real roots

$$\left.\begin{aligned} \lambda_1 &= -\frac{1}{(k\Theta\Delta t)^{1/2}} \\ \text{and} \quad \lambda_2 &= +\frac{1}{(k\Theta\Delta t)^{1/2}} \end{aligned}\right\} \tag{3.2}$$

As was previously remarked, changing u into −u by inverting the direction of integration would simply invert λ_1 and λ_2, leaving the same instability properties. However, we can write

$$L(x) = \left(\frac{d}{du} - \lambda_1\right)\left(\frac{d}{du} - \lambda_2\right) \cdot x \tag{3.3}$$

In view of this, any function $x_1(u)$ solution of

$$\frac{dx_1}{du} - \lambda_1 x = 0 \tag{3.4}$$

is also a solution of $L(x_1) = 0$, since we have

$$L(x_1) = \left(\frac{d}{du} - \lambda_2\right)\left(\frac{d}{du} - \lambda_1\right) \cdot x \tag{3.5}$$
$$= \left(\frac{d}{du} - \lambda_2\right) \cdot 0 = 0$$

The integration of (3.4) is stable in the forward direction, and requires one boundary value only (in u = 0)

Similarly, any solution of

$$\frac{dx_2}{du} - \lambda_2 x_2 = 0 \tag{3.6}$$

is also a solution of

$$L(x_2) = 0 \tag{3.7}$$

The integration of (3.6) is stable in the backward direction, and requires one boundary value only (in u = 1).

Finally, if $y_{(n)}^{i+1}$ is a solution of:

$$\frac{dy^{i+1}}{du} - \lambda_1 y^{i+1} = -\frac{S^i(u)}{k\Theta\Delta t} \tag{3.8}$$

and $x_3^{i+1}(u)$ is a solution of:

$$\frac{dx_3^{i+1}}{du} - \lambda_2 x_3^{i+1} = y \tag{3.9}$$

then we have

$$L(x_3^{i+1}) = \left(\frac{d}{du} - \lambda_1\right)\left(\frac{d}{du} - \lambda_2\right) x_3^{i+1}$$
$$= \left(\frac{d}{du} - \lambda_1\right) y^{i+1} = -\frac{S^i(u)}{k\Theta\Delta t}$$

Thus

$$\frac{d^2x^{i+1}}{du^3} - \frac{1}{k\Theta\Delta t}\, x_3^{i+1} = -\frac{S^i(u)}{k\Theta\Delta t} \tag{3.10}$$

(3.10) holds for *any* boundary conditions chosen to integrate (3.8) and (3.9). For simplicity, these can be chosen to be

$$\begin{aligned} & y^{i+1}(0) = 0 \\ \text{and} \quad & x_3^{i+1}(1) = 0 \end{aligned} \tag{3.10A}$$

Now we can observe that any combination

$$x_3^{i+1} = a_1^{i+1}x_1 + a_2^{i+1}x_2 + x_3^{i+1} \tag{3.11}$$

satisfies equation (2.7). (This results merely from equations (3.5), (3.7) and (3.10)).

If $x_1(u)$, $x_2(u)$ and $x_3^{i+1}(u)$ are known, then two constants a_1^{i+1} and a_2^{i+1} can always be found so that (3.11) satisfies the boundary conditions (2.8). These two constants are the solutions of the two simultaneous equations:

$$\begin{aligned} & a_1^{i+1}x_1(0) + a_2^{i+1}x_2(0) + x_3^{i+1}(0) = X_0(t^{i+1}) \\ & a_1^{i+1}x_1(1) + a_2^{i+1}x_2(1) + x_3^{i+1}(1) = X_1(t^{i+1}) \end{aligned} \tag{3.12}$$

Furthermore, $x_1(u)$ and $x_2(u)$ are time-independent. Therefore, equations (3.4) and (3.6) need being integrated only once, which can be done prior to the integration of (3.8) and (3.9).

Therefore only (3.8) and (3.9) need to be integrated at computation time for each step of the sequence $i = 1, 2, \ldots$. Both of these equations are of first order, and are computationally stable.

The application of (3.12) and (3.11) then produces the solution $x^{i+1}(u)$ by a simple addition process.

We can still simplify this procedure further by defining two new function $x_1^*(u)$ and $x_2^*(u)$ which satisfy the relations

$$x_1^*(u) = C_{11}x_1 + C_{12}x \tag{3.13}$$

$$x_2^*(u) = C_{21}x_1 + C_{22}x_2 \tag{3.14}$$

$$x_1^*(0) = 1 \quad ; \quad x_1^*(1) = 0 \tag{3.15}$$

$$x_2^*(0) = 0 \quad ; \quad x_1^*(1) = 1 \tag{3.16}$$

(The constants C_{ij} are computed by the conditions (3.15) and (3.16).)

It is obvious, that, in view of (3.5) and (3.7), $x_2^*(u)$ and $x_2^*(u)$ both satisfy the homogeneous equation

$$L(x_1^*) = L(x_2^*) = 0 \tag{3.17}$$

If we now seek a solution of the form

$$x^{i+1}(u) = b_1^{i+1}x_1^*(u) + b_2^{i+1}x_2^*(u) + x_3^{i+1} \tag{3.18}$$

which automatically satisfies the equation (2.7).

$$L(x^{i+1}) = -\frac{S^{i+1}}{k\Theta\Delta t}$$

the boundary condition equations equivalent to (3.12) become in this case

$$\begin{aligned} b_1^{i+1} + x_3^{i+0} &= X_0(t^{i+1}) \\ b_2^{i+1} &= X_1(t^{i+1}) \end{aligned}$$

or

$$b_1^{i+1} = X_0(t^{i+1}) - x_3^{i+1}(0)$$

$$b_2^{i+1} = X_1(t^{i+1}) \tag{3.20}$$

These equations are more simple than their equivalent (3.12).

The overall integration procedure can thus be summarized as follows (fig. 4).

1. Obtain, by integration of (3.4), (3.6) and by the application of (3.13) thru (3.16) the two elementary solutions $x_1^*(u)$ and $x_2^*(u)$. The solutions are stored for their use at computation time.
2. At computation time, integrate the two stable differential equations (3.8) and (3.9), with boundary conditions (3.10A).
3. Compute b_1^{i+1} and b_2^{i+1} by application of (3.20), and compute the solution $x^{i+1}(u)$ by application of (3.18).
4. Compute $S^{i+1}(u)$ by the application of (2.6).
5. Return to 2.

Starting procedure

At the first integration step ($t = t^1$), no value $S^0(u)$ is available. Starting the integration can be achieved by making use of the fact that $S^0(u)$ is the approximated value of $\chi(u)$ at time $t = (1-\Theta)\Delta t$

We can write, using an implicit formulation,

$$\frac{S^0(u) - x^0(u)}{(1-\Theta)\,\Delta t} = k\,\frac{d^2S^0}{du^2} \tag{3.21}$$

or

$$\frac{d^2S^0}{du^2} - \frac{S^0}{k(1-\Theta)\Delta t} = X_u(0) \tag{3.22}$$

with boundary conditions:

$$\left.\begin{aligned} & S^0(0) = X_0((1-\Theta)\Delta t) \\ \text{and} \quad & S^0(1) = X_1((1-\Theta)\Delta t) \end{aligned}\right\} \tag{3.23}$$

$S^0(u)$ can be computed by the same procedure as that defined in paragraph (3.1) by the integration of (3.22).

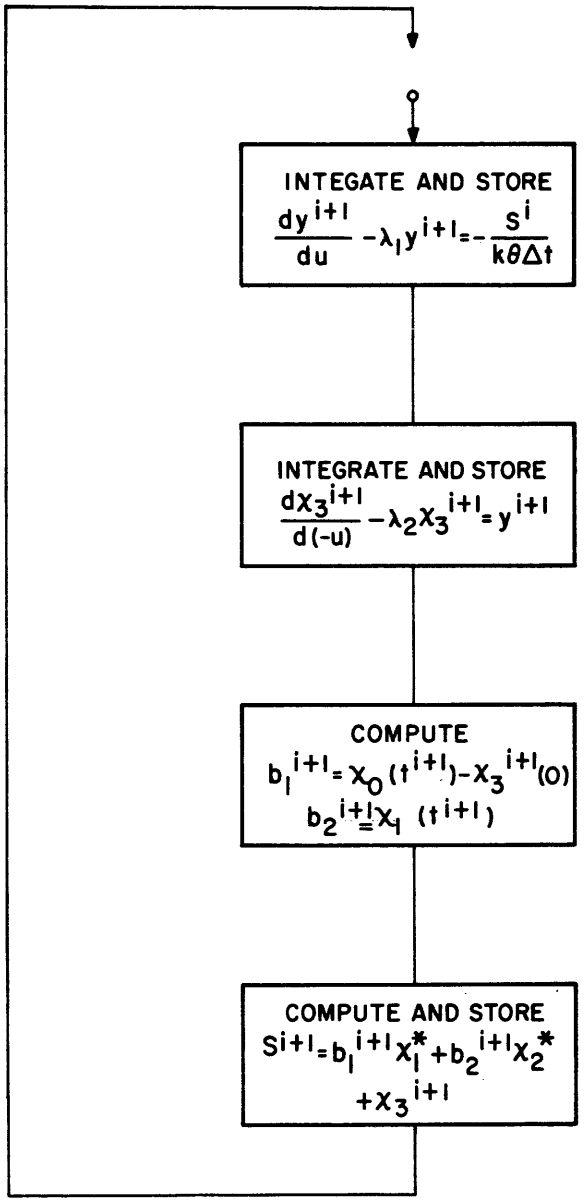


Figure 4—Computer block diagram describing the sequence of operations for the serial hybrid computer integration of the heat diffusion equation

Generalization

We now address ourselves to the more general initial values problem of solving a partial differential equation of the form:

$$U(x) = T(x) + G(u,t) \qquad (4.1)$$
$$u \in (0,1)\ ;\ t \geq 0$$

where

x(u,t) is the desired solution, function of one space dimension and time

U( ) is a linear partial differential operator with respect to the space independent variable u, which may contain time dependent coefficients.

T( ) is a linear partial differential operator with respect to time t

G(u,t) is a time-space dependent forcing function.

By application of the discrete-time-continuous-space (DTCS) or serial method of hybrid computer solution, (4.1) can be transformed into a sequential set of ordinary differential equations.

This is obtained by writing the approximation:

$$\left.\begin{aligned} T(x(u,t))_{t^{i-1+\theta}} &\simeq \sum_{j=0}^{m} d_j\, x^{i+j}(u) \\ t^i &= i\Delta t\ ;\ 0 < \theta \leq 1 \\ i &= 1, 2, \ldots \end{aligned}\right\} \qquad (4.2)$$

The d_j are fixed coefficients, which can, for instance, be obtained by equating the $(m+1)$ first coefficients of the Taylor series expansion of the left-hand side and right hand side of (4.2).

Substitution of (4.2) into (4.1) yields:

$$\sum_{j=0}^{m} d_j x^{i-j} = \theta[U(x^i) + G^i] + (1-\theta)\,[U(x^{i-1}) + G^{i-1}]$$

$$(G^i(u) = G(u,t^i)) \qquad (4.3)$$

This can be rewritten as:

$$(\theta U - d_0)x^i = \theta G^i + (1-\theta)G^{i-1} + (1-\theta)U(x^{i-1}) + \sum_{j=1}^{m} d_j x^{i-j} \qquad (4.4)$$

Let:

$$L = \theta U - d_0 \qquad (4.5)$$

and

$$H^i(u) = \theta G^i + (1-\theta)\,(G^{i-1} + U(x^{i-1})) + \sum_{j=1}^{m} d_j x^{i-j} \qquad (4.6)$$

We can now rewrite (4.4) as:

$$L(x^i) = H^i(u) \qquad (4.7)$$

At time $t^i = i\Delta t$, (4.7) appears as an ordinary differential equation in u, where $x^i(u)$ is the unknown. At that time, all the elements of $H^i(u)$ are known, and $H^i(u)$ is thus strictly a forcing function to the differential equation (4.7).

The central problem in applying the serial method of integration of (4.1) in hybrid computation hinges on

the capability to integrate (4.7) on the analog computer in a stable and practical fashion. When boundary values for x(u,t) are given along one space boundary only, and when (4.7) is computationally stable in the direction originating from that boundary, this is relatively easy. However, in most practical cases, space-boundary values are mixed, and furthermore, L( ) is often an unstable operator.

We shall devote Section 5 to the development of a general computational algorithm which generalizes the method described in Section 3. This algorithm permits in almost every case to obtain a stable, iteration free, solution.

Starting procedure

We note that for $i = 1, 2, \ldots m-1$, not enough past history points are available to allow an explicit expression of (4.2).

However, for the problem to be defined as an initial-value problem, boundary conditions are available along the line $t = 0$; $u \in (0,1)$. The boundary conditions are, in general, the values of

$$\dot{x}, \left.\frac{\partial x}{\partial t}\right|_{t=0}, \left.\frac{\partial^2 x}{\partial t^2}\right|_{t=0}, \ldots \left.\frac{\partial x^{r-1}}{\partial t^{r-1}}\right|_{t=0}$$

where r is the order ot T( )

The explicit use of these values, plus the development of special approximation forms to be used in the $(m - r + 1)$ first steps of the integration, permits the development of starting algorithms, very much in the same fashion as starting algorithms used in the numerical integration of ordinary differential equations.

The general problem of the spatial integration

We have seen in the above section that, after the suitable approximation process expressed by (4.2), the integration of the partial differential equation (4.1) resolves itself to the integration of ordinary differential equations of the form (we have deleted the superscript i):

$$L(x) = H(u) \tag{5.1}$$

where

x(u) is the unknown dependent variable
u is the independent variable
L( ) is a linear differential operator, of order n
H(u) is the forcing function.

In addition to satisfying (5.1), the solution must also satisfy given boundary values of the problem, s of which are given in $u = 0$ and $n - s$ are given in $u = 1$.

These boundary conditions are of the form

$$\begin{aligned}
\varphi_1(x(0)) &= B_1 \\
\varphi_2(x(0)) &= B_2 \\
\varphi_s(x(0)) &= B_s \\
\varphi_{s+1}(x(1)) &= B_{s+1} \\
&\vdots \\
\varphi_n(x(1)) &= B_n
\end{aligned} \tag{5.2}$$

In general, φ are linear differential operators in u.

Decomposition

Let us assume that the operator L(u) can be decomposed into two operators L_F and L_B of order k and $(n-k)$, respectively:

$$L(x) = L_B \cdot L_F(x) \tag{6.1}$$

where L_F is stable when integrated in the forward direction and L_B is stable when integrated in the backward direction. (It may be observed that, in many physical problems, k is equal to s, the number of boundary conditions given in $u = 0$.)

We now define y(u) as being a solution of

$$L_B(y) = H(u) \tag{6.2}$$

and we define $x_{n+1}(u)$ as being a solution of

$$L_F(x_{n+1}) = y(u) \tag{6.3}$$

(boundary values are unspecified).

By virtue of (6.1) and (6.2), we have obviously:

$$L(x_{n+1}) = L_B \cdot L_F(x_{n+1}) = L_B(y)$$

or

$$L(x_{n+1}) = H(u) \tag{6.4}$$

If we know, independently of x_{n+1}, n linearly independent elementary solution of the homogeneous equation

$$\begin{aligned} L(\chi_l^*) &= 0 \\ l &= 1, 2, \ldots n \end{aligned} \tag{6.5}$$

with boundary conditions:

$$\begin{aligned}
&\varphi_l\,(\chi_l^*)_0 = 1;\ l \leq s \\
&\varphi_j\,(x_l^*)_1 = 0;\ j \neq l \\
&\varphi_k\,(x_l^*)_1 = 0;\ k = 1, 2, \ldots n
\end{aligned}$$

and

$$\begin{cases}
\varphi_l\,(x_l^*)_1 = 1;\ l > s \\
\varphi_j\,(x^*)_1 = 0;\ j \neq l \\
\varphi_k\,(x_l^*)_0 = 0;\ k = 1, 2, \ldots n
\end{cases} \tag{6.6}$$

then, a linear combination of the form

$$x(u) = x^*(u) + \sum_{l=1}^{n} b_l x_l^*(u) \tag{6.7}$$

will satisfy both the differential equation (5.1), and the prescribed boundary conditions (5.2), provided the coefficients b_l satisfy the relations:

$$\left.\begin{aligned} &\varphi_1(x_{n+1})_0 + b_1 = B_1 \\ &\varphi_e(x_{n+1})_0 + b_2 = B_2 \\ &\vdots \\ &\varphi_s(x_{n+1})_0 + \lambda b_s = B_s \\ &\varphi_{s+1}(x_{n+1})_1 + b_{s+1} = B_{s+1} \\ &\vdots \\ &\varphi_n(x_{n+1})_1 + b_n = B_n \end{aligned}\right\} \tag{6.8}$$

or,

$$\left.\begin{aligned} b_l &= B_l - \varphi_l(x_{n+1})_0 \; ; \; l \leqslant s \\ b_l &= B_l - \varphi_l(x_{n+1})_1 \; ; \; l > s \end{aligned}\right\} \tag{6.9}$$

Thus, assuming that n linearly independent solutions of the homogeneous equation (6.5) are known, the problem of finding a solution to the unhomogeneous, mixed boundary problem expressed by (5.1) and (5.2) can be reduced to the integration of the two differential equations (6.2) with any set of convenient boundary conditions, plus the straightforward calculation of the n coefficients b_l by the application of (6.9). (Fig. 5).

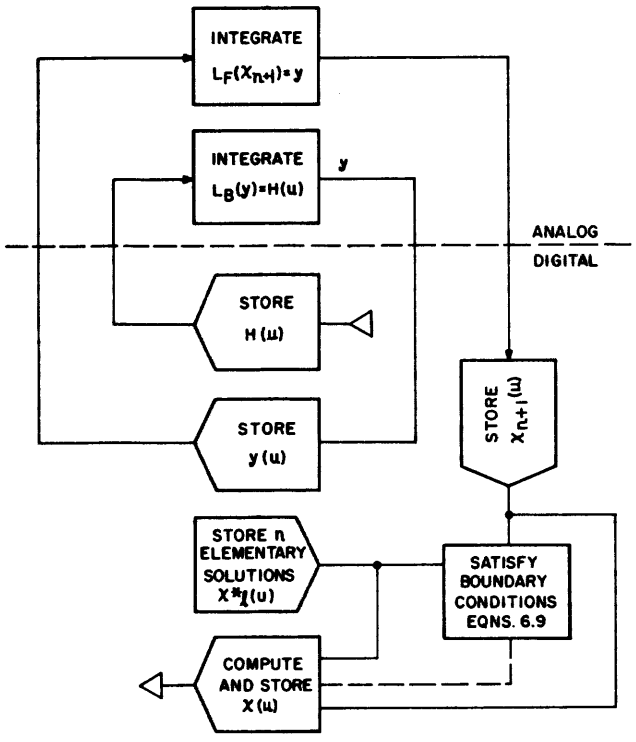


Figure 5 – Hybrid computer block diagram for the integration of L(x) = H(u) by the method of decomposition

It is not actually necessary to specify the n elemenoary solution of the homogeneous equation by boundary conditions as strictly defined as (6.6).

The method of decomposition can be applied on the basis of any set of n independent elementary solutions of (6.5).

Suppose we have such a set of n elementary solutions:

$$\begin{aligned} &L(x_l(u)) = 0 \\ &l = 1, 2, \ldots n \end{aligned} \tag{6.10}$$

with boundary conditions:

$$\begin{aligned} &\varphi_1(x_l)_0 = \beta_1, l \\ &\varphi_2(x_l)_0 = \beta_2, l \\ &\vdots \\ &\varphi_s(x_l)_0 = \beta_s, l \\ &\varphi_{s+1}(x_l)_0 = \beta_{s+1}, l \\ &\vdots \\ &\varphi_x(x_l) = \beta_n, l \\ &l = 1, 2, \ldots n \end{aligned} \tag{6.11}$$

A linear combination of the form

$$x(u) = x_{n+1}(u) + \sum_{l=1}^{n} a_l x_l(u) \tag{6.12}$$

will satisfy the boundary conditions (5.2) of the problem provided the a_l satisfy the relations:

$$\left.\begin{aligned} &\varphi_1(x_{n+1}) + \sum_{l=1}^{n} a_l \beta_1, l = B_1 \\ &\varphi_2(x_{n+1})_0 + \sum_{l=1}^{n} a_l \beta_2, l = B_2 \\ &\vdots \\ &\varphi_s(x_{n+1})_0 + \sum_{l=1}^{n} a_l \beta_s, l = B_s \\ &\varphi_{s+1}(x_{n+1}) \pm \sum_{l=1}^{n} a_l \beta_{s+1}, l = B_{s+1} \\ &\vdots \\ &\varphi_n(x_{n+1})_1 + \sum a_l \beta_n, l = B_n \end{aligned}\right\} \tag{6.13}$$

;(6.12) also satisfies (5.1) by virtue of (6.4) and (6.10).

However, the coefficients a_l will have to be computed by the solution of the linear set of algebraic equations (6.13) for each value of t^i.

The computation of n elementary solutions

The method by which n elementary solutions $x_l(u)$; $l = 1, 2, \ldots n$ are obtained is not necessarily of geat importance to the method described in the previous section; unless L( ) is time- (or i) dependent. We have seen in the case of the diffusion equation *Integration by Decomposition* that the elementary solutions could themselves be obtained previously to the space-time hybrid integration process by the use of the decomposed operators L_B and L_F themselves. This will still be true in the general case if:

1. L_B and L_F are permutable operators, i.e.,

$$L_B L_F = L_F L_B$$

2. k (the order of L_F) is equal to s (the number of specified boundary conditions in u = 0) and, thus, n − k (the order of L_B) is equal to n − s (the number of boundary conditions in u = 1)

3. L( ) is not dependent of time .

If condition #1 above is not fulfilled, i.e.,

$$L_B L_F \neq L_F L_B \tag{7.1}$$

then the k independent elementary solutions which can be obtained by the (stable) forward integration of

$$L_F(x_l) = 0 \tag{7.2}$$

are still independent elementary solutions of $L(\chi_l) = 0$, since

$$L_B L_F(\chi_l) = 0$$

But the remaining (n–k) needed elementary solutions of (6.10) cannot be obtained by the integration of L_B in the backward direction, since the relation

$$L_B(\chi_l) = 0 \tag{7.3}$$

does not imply that $L(x) = 0$. Indeed, this would imply that $L(x) = L_B L_F(x) = 0$, which is only true if x is an elementary solution of $L_F(x) = 0$. Thus, one may need to use some other means to obtain the additional (n–k) elementary solutions. This may be done by the (forward or backward) integration of $L(x_l) = 0$, with any set of boundary conditions, provided they are linearly independent of those already used.

There is a variety of ways in which these elementary solutions can be obtained, which actually depend on the peculiarities of the problem at hand. However, it is of importance to note that the algorithm described in Sections 6.1 and 6.2 permit this general method to be applied with any set of n elementary solutions, and that therefore iteration is never needed to obtain such a set.

The set of elementary solutions (6.6) can be derived from the set (6.10) by an algebraic process performed prior to the time-space integration similar to (3.13) - (3.16). This results in relations (6.9) which are somewhat simpler than (6.13), but can only be done if L( ) is not time dependent.

In problems where L( ) is time (or i) dependent, a complete set of elementary solutions will have to be obtained at each computation step.

REFERENCES

1 R M TERASAKI
Analog computation of Green's function for integrating two point boundary value problems
IRE Transactions EC-11 57 1962

2 R VICHNEVETSKY
Error analysis in the computer simulation of dynamic systems: variational aspects of the problem
IEEE Transactions on Electronic Computers vol EC-16 no 4 August 1967 pp 403-411

3 E E L MITCHELL
Hybrid techniques for solution of partial differential equations
Electronic Associates Inc SAG Report #19 October 1963

4 H S WITSENHAUSEN
On the hybrid solution of partial differential equations
Proceedings of IFIP Congress 1965 vol 2 pp 425-431 Spartan Books

5 T SCHRÖDER
Neue fehlerabschätzungen für verschiedene iterations - verfahren
2 Augev Math und Mech 36 168 1956

6 W T REID
Generalized green matrices for two point boundary value problems
SIAM J Appl Math vol 15 no 4 July 1967

BASP-A biomedical analog signal processor*

by WILIAM J. MUELLER, PAUL E. BUCKTAL, PHILIPPOS LAMBRINIDIS, KARL E. SCHULTZ and LEO F. WALSH

State University of New York, Upstate Medical Center
Syracuse, New York

INTRODUCTION

With the advent of low-cost digital logic modules, discussions on hardware-software trade-off have become popular.[1] Suggestions have been made to construct an analog computer of digital modules and the availability of digital differential analyzer modules is a step in this direction. Further it has been suggested that a re-evaluation of hardware and software organization be made,[2] considering that there has not been any great systems variation in the implementation of computers.

In the past several years, a number of special purpose digital devices have become available for the processing, generally in real time, of analog signals obtained from patients in a medical center or from animals in an investigator's laboratory. These devices have a limited accuracy and data rate capability and, although they may be faster than a general-purpose digital computer, are too slow for real time analysis of some of the more complex biological systems. If more than one full channel is required or if several types of analyses are necessary, the cost is prohibitive. Three years ago we wired a large array of printed circuit logic modules to programmable patch panels and a system similar to the more recent macromodules of Ornstein et al.[3] was connected to an analog computer (TR48) to carry out some real time analysis of biomedical data. The system operated properly but, because of the large number of connections, programming was horrendous. At the same time investigators at our Medical Center began asking for faster and more sophisticated data processing of biomedical analog signals. Several special-purpose high-speed real time processors were constructed using the newly available Motorola "MECL" microcircuit modules.

It became apparent that a medium priced but programmable processor of analog data at a sample rate of one megahertz was needed. General purpose computers currently available cannot perform the necessary data manipulations in real time and an "unconventional system"[2] was required. This paper concerns itself with the philosophy and investigation of such a system specifically designed for the real time processing of biomedical analog signals.

Processor requirements

Some of the slower biomedical signals such as from EEG, (brain waves) ECG, (heart waves) and breathing or blood pressure transducers are easily analyzed on a general purpose computer although the programming is sometimes extensive. We are more concerned with the analysis of heart sounds, (1000 Hz-range) pulse trains on a nerve, (2000 pulses per second) and responses, obtained from single cells in a body, that have a risetime of less than one hundred microseconds.

These analog signals require various types of statistical analyses. The most general procedures are histograms of pulse intervals or pulse height, averaging of signals buried in noise, where "noise" may be non time related phenomena, variance analysis, and various other time series analyses such as auto and cross-correlation, spectral density, convolution and others related through a Fourier transformation. Some signals can be sampled and stored for processing a few seconds later, but if the number of samples required is large, excessive high-speed memory is required. To reasonably define signals that have peculiar, and as yet, unexplained points of inflection in their rising phase, an analog sampling rate of one megahertz is required. This data rate would overload any ordinary memory size. Therefore, the data must be processed continuously with one sample coming into the processor each microsecond and one sample leaving the processor each microsecond. This requires developing an analog-to-digital converter, an adder-subtractor module, a multiplier module, a divider module and a memory module that has a fixed point

*The development of this processor was supported by the General Medical Sciences branch of the National Institutes of Health through Grant GM11413.

and floating point operating time of less than one microsecond.

Several calculations involve small ratios of relatively large analog signals and a resolution of at least 0.1% is necessary for the analog-to-digital converter, which must operate at a one megahertz sampling rate. For slower biomedical signals we would use a 14 bit converter to reduce quantization error. Fortunately in many cases where the highest digitization rate is required, such as for auto and cross correlation and related statistical analyses, the resolution of the Analog-Digital converter can be 10 bits and still produce 14 bit accuracy[4] because quantization errors tend to average out. However, this assumes that the calculations made on the quantized data do not introduce additional round off or truncation error during the multiplication, addition and normalization procedures. For correlation we would have fixed point numbers from 0 to ±999, in steps of ±1, multiplied against their lagged values producing a range from 0 to 998,001 as a product which when summed for 1000 correlation points produces a thirty bit fixed point number. Similarly taking the average of a thousand responses evoked from a brain, by, for example a flashing light or a tone, we would have a sum of one thousand 14 bit numbers for each sample point on the response and require a 24 bit fixed point word. At times we may require 10,000 samples for adequate signal to noise enhancement which improves as the square root of the number of samples. As we require a 32 bit floating point arithmetic unit and a 32 bit memory, outlined below, we could use a 30 bit fixed point system if desired. A higher operating speed would be possible in fixed point.

As about one third of our data processing is the generation of models of electrical transmission of information along nerves or conductive tissue such as heart muscle, we prefer using values that are real instead of having to scale the true values of resistance and capacitance used in the model. A solution of an equation, such as one described below for the membrance resistance of a cell, Rm, requires the representation of resistance over a range of 10^5 to 10^8 ohms while the values of membrane voltages V_1, V_2 may only vary, when digitized, from 0 to ±999 in fixed point format. However when fixed point arithmetic is used on a fraction such as $\frac{V2-V1}{V2}$, where V1 and V2 are nearly equal, the result would have to be expressed in decimal format, with at least two decimal places, or in floating point format to be meaningful.

Floating point operation is the only way to represent the decimal range of 10^{-12} to 10^{12}, which we require for our simulation studies. To obtain this range we would use a 24 bit word with a sign bit and, a six bit exponential to the base two with a sign bit. The memories when storing arithmetic results would function as 32 bit words but when storing digitized input data would operate as two sixteen bit words having 14 bits for data and two bits for sign and parity.

A floating point arithmetic sysem is being constructed and if we find that adequate speed can be obtained in floating point operation the fixed point hardware will probably not be constructed.

We recognize that in several particular instances, of processing biomedical analog signals, smaller word lengths and preprocessing by analog methods could be used. However, we have several problems that are not immediately apparent.

Requests for data processing vary widely in size and complexity and any system we develop must have sufficient capabilities that a minimum of programming effort is required and that an overall accuracy of 0.1% is maintained without paying strict attention to truncation or round off errors.

The pick up point of our biomedical analog signals, from an operating room or an investigator's laboratory, may be a mile away from the processor. Although we have developed carrier type systems with a 1000:1 signal to noise ratio we prefer sending the analog data in digitized form, especially when multichannel time shared operation is being used. A decision to modify the analysis may be made by the surgeon or investigator during the operation or experiment. As the program is not remotely controllable the operator of the processor would have to make the required changes. In instances such as these a digital integrator may be used at the processor instead of an analog inegrator, which would be difficult to control a mile away, at the site of the surgical operation or animal experiment. There are also some equations, relating to the breathing capacity of an individual, that require integration after some previous addition and division of digitized data. Because of this an adder-subtactor module will be internally programmed to perform integration and be under control of appropriate start and stop signals.

A programmable clock would be required to permit the sampling of data at a fast rate when the data is moving fast and at a slower rate when the data slows down. This is a common feature of cells in a heart where the activity changes at a rate of 1000 volts per second for 100 microseconds and then slows down to 1 volt per second for the next several hundred milliseconds. This curve would be sampled and stored for comparison with the one to follow which is obtained under slightly different conditions. By this procedure memory requirements could be reduced from about 35,000 words to about 1,200 words.

A number of registers would be used depending on

the data being processed. Various counters and comparators are also required. All these devices would operate up to a 50 megahertz rate.

Display systems will vary depending on the data rate and may be either in digital or analog form. The display would have to be available in an investigators laboratory as well as at the processing unit.

Even though the sampling rate may be high the actual soution rate and therefore display rate, may be slow. For example a single cell response has a 100 micro-second rise time with a crucial notch in the signal. This data must be sampled and stored with at least a one megahertz rate but the total response may only reoccur twice a second. Therefore the output point plot on a digitally driven oscilloscope would only have 2000 data points per second. Most displays will be photographed to record them but in some cases a storage oscilloscope will be used for immediate readout.

A character generator and large screen analog and digital display system has been developed. At this time we will photograph the screen for digital and analog data storage at a rate of 20,000 characters per second or 10,000 data points. Analog and digital data may be interleaved. An investigation is presently being made on a combination of a small high speed, 8 millisecond access time, disk to be used with a buffer memory to provide a static display of digital and analog data using a P7 phosphor screen because of the 5000 digits that are repeated four times a second.

On slower data the results will be plotted on a standard X-Y recorder for analog plots. Digital data that can be held in the disk-core memory system will be printed on a recently available strip printer at 20 characters a second.

Machine organization

The machine organization depends on the data rate requirement of a particular biomedical data processing problem. At the maximum data rate of 1 megahertz for operation on one analog signal (or two signals if for example a cross-correlation is desired) the input signal flows through the A-D converter and the various arithmetic and memory modules in the sequence of the required program steps. For example, if an average of many evoked responses from an animal is required and the signal requires a sampling rate of 1 megahertz, the signal would flow as in Figure 1. Assuming each unit takes less than one microsecond to complete its function, the procedure is as follows:

1. At T= O usec, sample and hold input signal 1- Start read of MEM 1
2. At T= 1 usec, dump A-D OUTPUT to A/S INPUT with MEM 1 data

2a. At T= 1 sample and hold input signal 2- Start read of MEM 2

3. At T= 2 usec, A/S dumps to MEM 1 and to divider with counter data, MEM 1 changes address

3a. At T= 2 usec, A-D OUTPUT goes to A/S INPUT with MEM 2 data

4. At T= 3 usec, 1st data point is displayed at location zero on the display unit

4a. At T=3 usec, sample and hold input signal 3- Start read of MEM 1- MEM 2 changes address

4b. At T= 3 usec, etc.

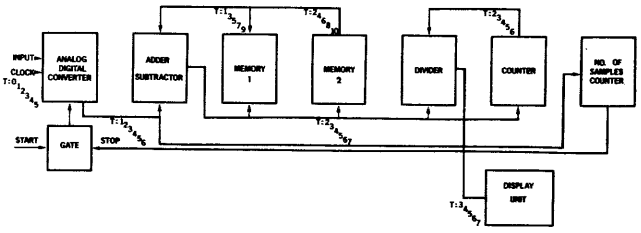


Figure 1 – Module signal flow for HIGH speed averager

Two completely independent memories are used. A single memory with a read-write cycle of 1 microsecond would, because the data is delayed by the A/S module, be asked to read and write simultaneously. Partitioning a memory would be of no value in this case.

Various control signals switch the memories and counter but the flow of data through the modules is obtained by the use of multiconductor cables from the A-D unit to the A/S unit to the MEMORIES and to the DIVIDER. The first result arrives at the display device in three microseconds. A longer delay would occur if the signal passed through more modules for further analysis. However, once the first signal appears at the display unit the next solution follows in one microsecond and there is a data point in and out of the processor each microsecond. If some data have to be retained for later calculations, it is delayed the appropriate amount by a shift register.

Under these circumstances a module, the divider for instance, could take five microseconds to divide but as long as a new piece of data enters the divider each microsecond and a piece of data leaves the divider each microsecond we only suffer a delay of five microseconds but maintain the one megahertz data rate. The number of arithmetic operations could be at a ten to twenty megahertz rate.

All modules, regardless of their delay time are designed to accept a new set of data each microsecond and deliver an output each microsecond.

However, if more than one particular operation is required, a second addition for instance, then a second adder-subtractor module is required. Fortunately, when the fastest possible operation is required, we usually do not require extremely complex arithmetic operation and two modules of each type would be sufficient. As the modules cost less than $1000 each, this is a feasible procedure. A trade-off between speed and the number of arithmetic operations is possible.

Time-shared operation

As only 20% of biomedical analog signals require the full 1 megahertz conversion rate, we decided to share the operation of the processor among various analog signals. To maintain accuracy and hold aliasing problems to a minimum, a system of five channels with a 200 kHz data rate per channel was decided upon. This permits filtering of the input signal by an analog filter operating at a cut-off frequency of 50 kHz, thereby keeping the time delay error of the analog signal at a minimum over a bandwidth of 5 kHz which is adequate for signals other than single-cell responses. Any of the 200 kHz channels could be subdivided into more, say 20, slower channels operating at 10 kHz.

If we were to continue on with the procedure of Figure 1 for connecting the modules to perform a certain function we would require five cables for every one we used previously so that the five pieces of input data could proceed through the five time-shared inputs and outputs of each module in a different sequence of modules for each piece. This procedure of plugging modules together by multiconductor cables and changing the cables when a different operation is to be performed has been advocated by several authors because, perhaps, of the similarity to an analog computer. However, we seemed to be going back to our original system with its unwieldy programming procedure. As this does not fit our more recent concepts and requirements for an easily programmable real time processor, an improved system has been designed.

System implementation

Our major concern was the removal of the patching from one module to another with a sixty-six wire connector each time we wanted to change the program. A large matrix with control line switching to tie the output of one module to the input of another was considered but a quick check of the permutations possible showed that a massive array of "and" gates would be required. A data bus system, similar to that used for peripheral control by a digital computer, was adopted.

All modules have their inputs and outputs connected to a common 32-bit data bus. When an adder-subtractor module has some data ready for a multiplier module, the A/S unit dumps its output on the common data bus and sends a gate signal through a patchable twisted pair line to the multiplier input causing the multiplier to connect to the data bus and accept the data from the A/S unit.

With this system we could operate similar to an analog computer and only need to patch a simple 2-pin connector to provide a data path from one module to another. However, a system was needed that guaranteed the integrity of each data transfer so that no two sets of modules were on the common data bus at the same time.

Each module is being constructed so that it can receive data on its input while delivering a result on its output. Further, a time position is assigned to each module so that only that particular module can place data on the common data bus at the assigned time. A partial chart, set up for five time-shared channels at a 200 kHz data rate, (one megahertz total data rate) is shown in Figure 2. All devices have outputs spaced 100 nanoseconds apart in time but can only place data on the common data bus for fifty nanoseconds thus providing a guard band of 50 nanoseconds. This of course limits the data bus transfer rate to 10 megahertz using 10 modules. To improve the rate at which data can be maneuvered, and also permit the simultaneous dumping of data to the x and y inputs of a module, two data buses will be used. Figure 3 is the block diagram of a system to average five analog input signals with a sampling rate of 200 kHz each. This is equivalent to having five separate on-line averagers such as those commercially available but in this case the sampling rate on each channel and the resolution per channel has been improved by an order of magnitude. The output display, a digitally driven oscilloscope, would not show a gradual build-up in size but rather, because the output is continuously normalized, the output signal would maintain a constant height while the noise just faded away. In some averaging procedures one may find a small average but a large variance. As variance and average could be found for the same set of data in real time, we may obtain more meaningful information from future studies of biomedical analog signals.

We still had to provide some simple means of connecting control signals from module to module to implement a program. Consideration was given to punching a program on an IBM card and then inserting the card in a simple reader to connect the appropriate control paths. However, we found that we have more combinations available for signal flow than

ALL TIME IN MICROSECONDS AFTER TIME ZERO

CHANNEL NUMBER	ANALOG / DIGITAL DATA IN	DATA OUT	ADD / SUBTRACT DATA IN	DATA OUT	DIVIDER DATA IN	DATA OUT	MULTIPLIER DATA IN	DATA OUT	MEMORIES DATA IN	DATA OUT
1	0	1	•1	1•1	•2	1•2	•3	1•3	ANY	•7
2	1	2	1•1	2•1	1•2	2•2	1•3	2•3	TIME	1•7
3	2	3	2•1	3•1	2•2	3•2	2•3	3•3		2•7
4	3	4	3•1	4•1	3•2	4•2	3•3	4•3		3•7
5	4	5	4•1	5•1	4•2	5•2	4•3	5•3		4•7
1	5	6	5•1	6•1	5•2	6•2	5•3	6•3		5•7
2	6	7	6•1	7•1	6•2	7•2	6•3	7•3		6•7
3	7	8	7•1	8•1	7•2	8•2	7•3	8•3		7•7
4	8	9	8•1	9•1	8•2	9•2	8•3	9•3		8•7
5	9	10	9•1	10•1	9•2	10•2	9•3	10•3		9•7
1	10	11	10•1	11•1	10•2	11•2	10•3	11•3		10•7

Figure 2 – Module timing chart for five channels

there are holes in a card. A 12″×12″ patch panel with a 100 × 100 matrix was decided upon. Diode pins placed at appropriate locations provided the required signal paths. Plastic overlay sheets, with holes punched for a given program, will be used to allow rapid changing of the desired data analysis.

Several other features to expand programming capabilities are incorporated into the system. Memories will be partitioned into two sixteen-bit words. Read-and-write operations are under external and internal control so that various options such as read-write or read-modify-write are available. The address registers can be modified by external signals or operated internally, up or down, on command. Sequential stepping of the memories allows 1 microsecond operation even though the memory cycle time is five microseconds. All modules will be able to input data from or send data to either data bus. The display unit, an alphanumeric generator and analog point plotter CRT system, will, by a timing system, rapidly display data occurring at any point in the system. Thus debugging, even under time-shared operation, can be very simple.

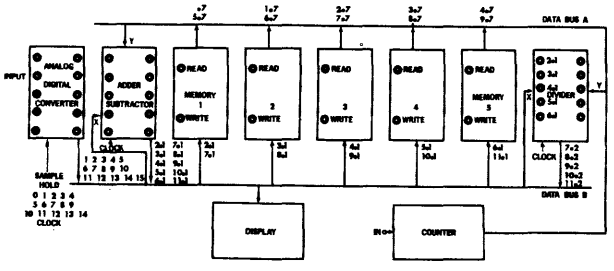


Figure 3 – Module signal flow for five channel averager

A drawing of a portion of the programming panel is shown in Figure 4. Diode pin locations to implement the multiple averager described above are indicated by small dots.

We find that medical people do not like binary, octal or other such representation of data. In particular when an investigator is interacting with the processor

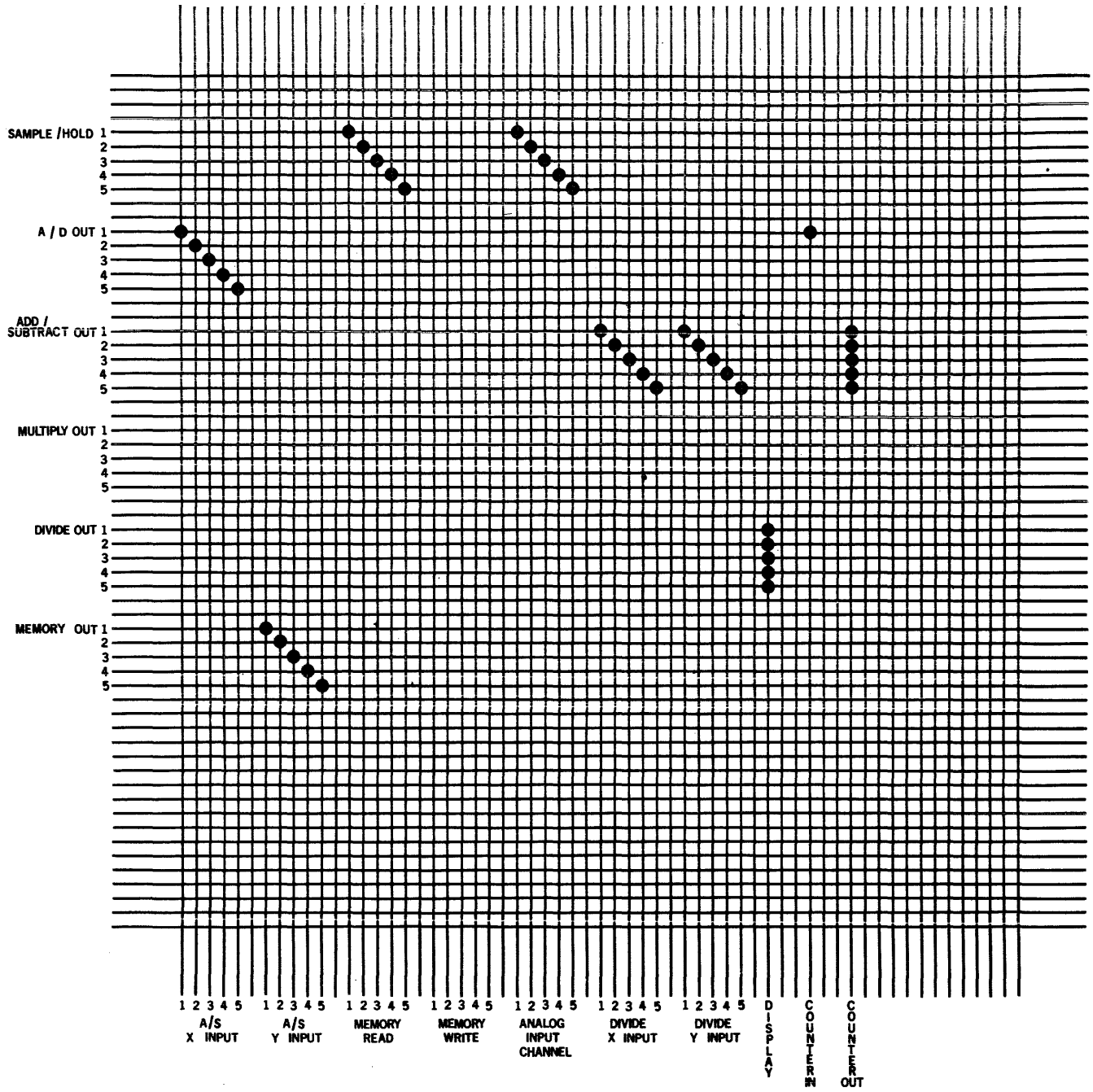


Figure 4 – Programming board pin positions for five channel averager

through a keyboard while watching an analog display of some model, for example the propagation of electrical activity in a heart, he will want to display the decimal values along with the analog plot. New constants of propagation would be entered and the change in the display noted. When the analog plot is as desired the data points would be displayed in decimal form for observation and photography.

All of the input-output display systems and keyboards would operate in ASCII or BCD code and therefore, because the processor is a binary system, the results of arithmetic operations and core storage will have to be code converted before being entered or displayed in decimal form. In many cases scientific notation will be used instead of decimal format.

Hardware implementation

The Biomedical Analog Signal Processor described here is not at full capability at this time. A twelve-bit version was implemented to gain experience with

the new high-speed logic modules we are using. A 32-bit version is being constructed at the time of writing this paper.

Modules are being constructed using 2 ns and 5 ns MECL II logic devices in a plastic dual inline case. Printed circuit boards, on which the logic devices are mounted, are bolted to a ground plane inside a narrow case which provides shielding and a front panel for indicator lights and test switches. These modules which are a complete unit such as a memory, adder-subtractor, multiplier and others, are being mounted in a cabinet and by twisted pair lines from the rear panel connected to the data bus system and the program panel.

Some of the units, because of the internal clock-speed of 50 megahertz, would not tolerate wire lengths of three inches. These boards were arranged in sections and placed back to back with a ground plane between them. Then connecting pins were inserted from the first board through the ground plane to the second board. The operation of shift registers and clock lines was improved by this procedure. We have our own printed circuit facility but cannot make multilayer boards.

Each module has a decoupling 5-volt zener diode operating from a partially regulated seven-volt line. As the logic modules require about an ampere for thirty devices each arithmetic unit uses two to three amperes of current. Clamping of the power supply to a maximum of eight volts protects the logic modules from destruction.

Control signals are run to the programming panel by 130 ohm twisted pair lines terminated at the panel. The control signal, a 30-nanosecond pulse which comes from the output of a module, is badly degraded in passing through the twelve = to twenty-four inch path length from input to output of the programming panel. At the output terminal of the panel (bottom in Figure 4) a one-shot multivibrator rejuvenates the control signal and sends it, by twisted pair lines, to the specified input of another module.

A master clock at 1 megahertz controls the timing of the modules. The outputs from the modules are dumped on the data bus at the correct (specified) time by delaying the clock the required amount internal to the module. Thus the clock is delayed 0.1 microsecond in the adder-substractor, 0.2 microsecond in the divider, 0.3 microsecond in the multiplier, etc. The memories because of their peaking time of 0.58 microsecond are dumped on the data bus at 0.7 microsecond after a read command which is synchronized with the clock. Write commands to the memory are started immediately unless a read command has not been completed. In this case the data is stored in the input register until it can be stored in core.

It has been difficult to obtain an analog to digital converter than can do a conversion of one million 10-bit words per second. The major difficulty was that ten trials had to be made with each trial having a certain settling time before the next digital approximation to the analog signal could be made. A decade system was used, instead of binary, and only three approximations, one per decade, had to be made. Of course, one now has to know between which numbers in a decade the true analog value lies. By using nine microcircuit comparators, Fairchild 710's, per decade the level in a decade can be found in about 50 nanoseconds. The major portion of the conversion time is now used waiting for the subtraction of the required amount from the input analog signal so a trial for the next decade can be made. This takes about 150 nanoseconds with the microcircuit operational amplifiers, Fairchild 702A, that we used. Thus a conversion, analog to BCD, can be done in, 800 nanoseconds. We have found that our resolution could be 12-bits but settling time of the analog amplifiers prohibited our taking advantage of it. We have obtained some amplifiers with faster slewing rate and settling time and will attempt to improve the converter to a 12-bit system. A good sample and hold circuit is yet to be developed.

Because we require that this processor appear in digital form to the operator we needed a fast code converter from BCD to binary and from binary to BCD. A hardware system with a conversion time of about a microsecond for 30 bits has been developed. A ten-bit code converter worked in 120 nanoseconds and thus provided the interface of the above A-D converter to the binary system of the processor.

Floating point add and subtract or multiply modules take a microsecond. We have had difficulty designing a divider to operate below five microseconds for a 24-bit dividend. We are investigating some of the classical iteration formulae for division to reduce the number of approximations.

A square root module using a Newton-Raphson iteration is being developed.

CONCLUSION

We require high speed arithmetic devices in this processor so that we may do on-line many time related analyses presently being done off-line. In some cases, a correlation for example, we would do a thousand points on a correlation curve in the time between two events, which may be on the order of one millisecond. This requires one thousand time shifts, one thousand multiplications, one thousand additions, for summation, and two thousand read-write operations per second. This is a rate of five million operations on data per second not including of course such opera-

tions as shifts which go on at a 50 megahertz rate. For the case of correlation of pulse intervals the "multiplication" becomes a Boolean "and" function.

Further we require that the processor be easily programmable and be able to do all the different types of analyses now being done on Biomedical data in an on-line manner by the several special purpose devices commercially available.

The processor described here could be operated by a stored program other than the patch panel we propose.

All that would be required is a memory that indexed to the next address each time a completion signal was obtained from an output of one of the modules. A compiled instruction signal obtained from memory would be decoded to turn on the next module that is to operate on the data.

The major problem is the obtaining of a memory with a cycle time of 20 nanoseconds so that the decoded statement would start the required operation. We do seriously consider using a shift register in lieu of a core memory to investigate this type of operation. One of the new LSI scratch pad memories may be useful in this respect. The advantage of a stored program is that the same arithmetic unit could be used over and over but the price to be paid is a slowing of data throughout.

It is probably obvious that the time-shared mode can be turned off and channel 1 used at full computational speed or the time-shared mode can be turned on and the processor divided into two channels of 500 kHz each or various other combinations. When an expanded timing chart is drawn, it can be seen that one of the five major time-shared channels can be subdivided, as noted before, into, say twenty subchannels. A set of working registers is necessary for each subchannel and these subchannels can be programmed by a shift register or, if operated slow enough, by a core memory.

A recent review of this processor indicated the probability that by the addition of an instruction set and a decoder, the processor could be turned into a standard general-purpose computer when desired. We, however, doubt the wisdom of doing that at this time.

We are fortunate in having several physiologists and M.D.s at our Medical Center who have requested assistance in applying data analysis to biological signals more complex than the usual EEG, ECG type. Based on the increasing number of requests for data processing of the type we envisage we anticiapte an almost continuous usage of this processor when completed.

Because most of the types of analyses listed so far have been time-related phenomena, one may assume that this is the limiting area of activity. We do, however, have flow charts drawn (to see if we missed something in the design of the processor) of several modeling problems. One problem we simulated ten years ago in a limited manner with analog devices was the spread of electrical activity from cell to cell in a heart under various conditions of drugs and injury. Work in this area has been carried out on several large-scale computer systems, but with difficulty because of the usual batch processing environment and the relatively slow speed of general-purpose computers. Sufficient repeats of large amounts of arithmetic calculations to permit display on a CRT, were unobtainable. It is important to have easy man-machine interface for varying parameters from a keyboard to immediately obtain the new pattern of wave propagation. We estimate that the cost of the processor we are constructing will be less than half the cost of the programming and computer time spent on just one simulation study such as described.

A different analysis to be done is the solution of an equation $R_m = \frac{V_2 - V_1}{V_o} R_1 - R_2$ where the parameters vary continuously with time. The author has a major interest in the dynamic solution of this equation as it is related to a physiological study, being carried out by him and others of our faculty, on the resistance changes in a heart cell during electrical excitation. The full capability of BASP will be required because of the 100-microsecond rise-time of the cell's electrical analog signal.

The processor as it has evolved thus far was conceived because of the need of such a device by the author and others of the faculty. Its present form has been molded by specific requests for data processing assistance and by an almost continuous series of discussions between the author and the co-authors. Mr. Buckthal was responsible for design innovations that permitted the speed of the code converters and the adder-subtractor modules. Mr. Schultz did much of the work on the character generator and display system as well as establishing our printed circuit facility without which we could not produce the modules. The development of the core memory system was done by Mr. Lambrinidis. Mr. Walsh was the major antagonist and did much to clarify the goal and verify the logical concept.

Thanks are due to Mrs. Kathryn Vellrath who made the drawings and typed the manuscript under the duress of getting several other reports completed by the deadline for submittal of this paper. We owe much to Al Buettgens who constructed most of what has been completed on this project.

REFERENCES

1 L C HOBBS
Effects of large arrays on machine organization and hardware/ software trade-offs
Proceedings of the 1966 Spring Joint Computer Conference
2 D L SLOTNICK
Unconventional systems
Proceedings of the 1967 Spring Joint Computer Conference
3 S M ORNSTEIN
A functional description of macromodules
Proceedings of the 1967 Spring Joint Computer Conference
4 G A KORN
Statistical measurements with quantized data
Random-Process Simulation and Measurements (McGraw-Hill 1966)

Electrically alterable digital differential analyzer

by GILBERT P. HYATT and GENE OHLBERG
Teledyne Systems Company
Northridge, California

INTRODUCTION

Computer simulations are performed for scientific problems using analog computer techniques for speed and functional similarity to the actual problem. The analog computers are severely limited in accuracy. This inherent accuracy limitation on high speed simulation requirements can be overcome by the application of the Teledyne Electrically Alterable Digital Differential Analyzer (TEADDA), which is a high speed completely parallel "digital analog" of an analog computer.

The TEADDA is a parallel word, parallel computation Digital Differential Analyzer (DDA) with a speed of one million iterations (complete incremental solutions) per second and a resolution of one part per million. The TEADDA is programmed by interconnecting the computational elements like an analog computer with the exception that the "patchboard" is electronic in nature; controlled from a general purpose digital computer, tape reader, or other peripheral equipment.

The baseline TEADDA resolution is one part per million (20 bits). This resolution is not an inherent limitation, but can be increased virtually without limit. Problem scaling is accomplished under program control, permitting increment scaling to vary over a range of approximately 65,000; which is almost five orders of magnitude.

The TEADDA is implemented with Teledyne's hybrid LSI* Micro-Electronic Modular Assembly (MEMA) technique to achieve the high speed operation. The MEMA is a hybrid multi-chip flatpack with up to 32 integrated circuits interconnected and sealed in a 1/20 cubic inch volumn. The high speed computer performance is achieved because of the small computer size and short interconnections resultant from the MEMA packaging.

The significant features of the TEADDA are:

a. One megahertz iteration rate
b. Twenty bit resolution (one part per million)
c. "Electronic Patchboard" for completely automatic programming
d. 100% interconnectivity between 64 integrators (256 programmable connections for each increment)
e. Extrapolative trapezoidal integration
f. Automatic register length selection to vary the incremental weighting over a range of 2^{16} (approximately 65,000).
g. Multiple incremental inputs to each integrator
h. Ternary incremental communication
i. Twos complement negative number representation.
j. MEMA packaging
k. High reliability
l. Low cost

The baseline system is mechanized to take advantage of what is considered to be the most important tradeoffs. These tradeoffs in order of significance are:

a. A high degree of versatility, permitting the programmer and operator to make maximum use of the computer.
b. Achieving a one megahertz iteration rate.
c. Minimizing the number of integrated circuits.

These considerations were weighted toward programming convenience, flexibility, and cost-of-ownership considerations. A lower cost computer could be achieved by reduction of the versatility and/or speed. It is considered that the system contains a near-optimized weighting of all three considerations in order of their significance.

Functional description[1,2]

The functional diagram of the programmable DDA system is illustrated in Figure 1. In the baseline system, a GP digital computer is used to set up the problem to be solved by the DDA integrators. The GP computer addresses and loads data into the program registers, which set up the electrically alterable program for the DDA. This program can also be entered by a tape reader or other types of peripheral equip-

*1967 International Electron Devices Meeting (IEEE).

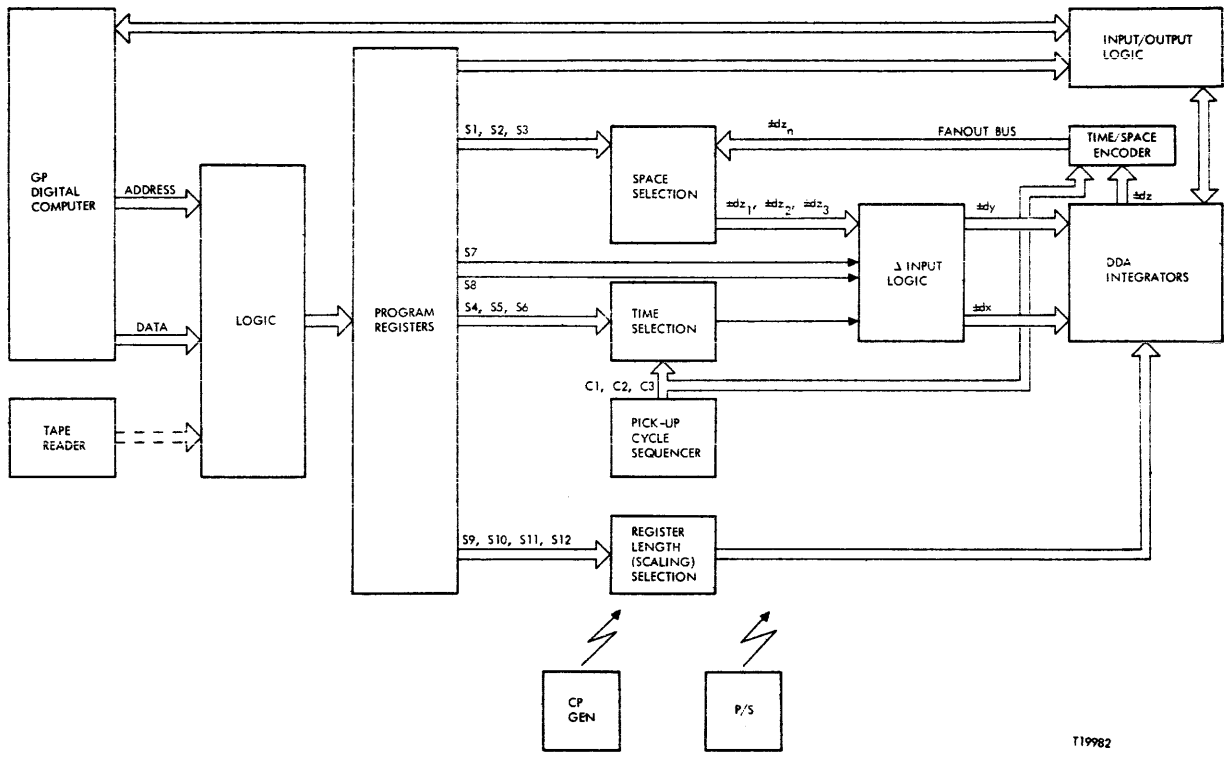


Figure 1—System functional diagram

ment. The program registers will be used to hold the interconnection and scaling program for a particular DDA function. In addition, these program registers are time shared to load initial conditions into the integrators and to transfer the readout data to the GP computer.

The interconnection control for the DDA is accomplished in a two dimensional array through a space domain and time domain selection. Programmable digital bits S1 thru S3 and S4 thru S6 set up the space domain selection and time domain selection, respectively. Each integrator contains four of these registers and associated selection logic to implement the four incremental inputs, defined as the three dy inputs and the dx input to each integrator. In addition, the S7 programmable bit reverses the polarity of the incremental input to implement a sign change function. The S8 programmable bit is used to disable the input, to prevent incremental signals to that input from affecting the integrator computation.

The pickup cycle sequencer is actually the bit timer, which is used both to encode and to decode the time domain multiplexed signals. The space domain selection logic will select the appropriate interconnection bus. The time selection logic will generate a pulse, the time position of which corresponds to the time multiplexed position of the programmed input increment. In addition, the Δ input logic will perform the polarity reversal and input disable functions under program control. The incremental inputs to the DDA integrators will be received from the Δ input logic, under program control, to update the integrators.

Programmable integrator scaling is accomplished by program register bits S9 through S12. These bits will select the register length for the particular integrator, thereby scaling the parameter over a range of approximately 65,000.

The four incremental inputs to each integrator are programmed with bits S1 thru S8, repeated for each of the four incremental inputs. The register length selection is accomplished with bits S9 thru S12 for each integrator. Therefore, a 36 bit program register is required to completely define the input and scaling for each DDA integrator. The incremental output of the integrator is multiplexed onto the dz bus under the control of the pickup cycle sequencer. This time multiplexed position, a pulse position identification, is characteristic of each integrator and is not programmable. The time and space multiplexed incremental outputs of the integrators are recirculated to the selec-

tion logic for fanout to four inputs for each of 64 integrators, yielding a total fanout of 256 incremental inputs.

The input/output logic controls the loading of initial conditions during the Initialize Mode and the transfer of computational parameters during the Readout Mode. There is provision for direct GP computer control for the initializing and readout of the DDA computer parameters.

Certain types of overhead equipment are required for computer operation, including the clock pulse generator and power supplies. This overhead equipment permits the DDA computer to be completely self-contained, thereby minimizing the interface requirements with the GP computer and peripheral equipment. This DDA computer can operate asynchronously relative to the GP computer and other control equipment. Interface synchronization is obtained with buffer logic and registers, thereby permitting the DDA computer to operate independent of limitations that are characteristic of the external equipment.

The dynamic operation of the computer is controlled by the modes, cycles, bit times, and clock pulses. The modes of operation are commanded by the GP computer or other peripheral control equipment. The cycles and timing pulses are generated within the DDA computer.

The modes of operation of the DDA computer are defined as the Compute Mode, Initialize Mode, and Readout Mode; described in Table I.

TABLE I—Operating Modes

MODE	CYCLE	BIT TIMES	TIME (μ SEC)	NOTES
COMPUTE	1 PICKUP	8	0.570 μs	REPETITIVE EACH 1 μs
	2 INCREMENT	3	0.214 μs	
	3 INTEGRATE	3	0.214 μs	
INITIALIZE	LOAD INITIAL CONDITIONS	64 WORDS 20 BITS/WORD 1280 BIT TIMES	ASSUME 1MC CLOCK 1280 μs	INITIATED AT THE START OF THE COMPUTATION
READOUT	RECIRCULATE	64 WORDS 20 BITS/WORD 1280 BIT TIMES	ASSUME 1MC CLOCK 1280 μs	(1) ON COMPUTER COMMAND (2) REPETITIVE AS REQUIRED

T19995

During the Compute Mode, the DDA integrators automatically sequence through the computation and generate solutions to the equations that they mechanize. This mode of operation is composed of repetitive DDA iterations. Each of these iterations is composed of three cycles that require a total of 14 clock pulses to complete. These 14 clock pulses constitute a full DDA compute iteration, illustrated in Figure 2. The duration of each cycle is determined primarily by the propagation time requirements through the longest logical chain to perform the required function. Cycles 2 and 3 are implemented as sequential computations to simplify the logic required for the integrator arithmetic operations.

Cycle 1 is defined as the Pickup Cycle, performing the encoding and decoding functions that interconnect the incremental outputs, obtained during the previous

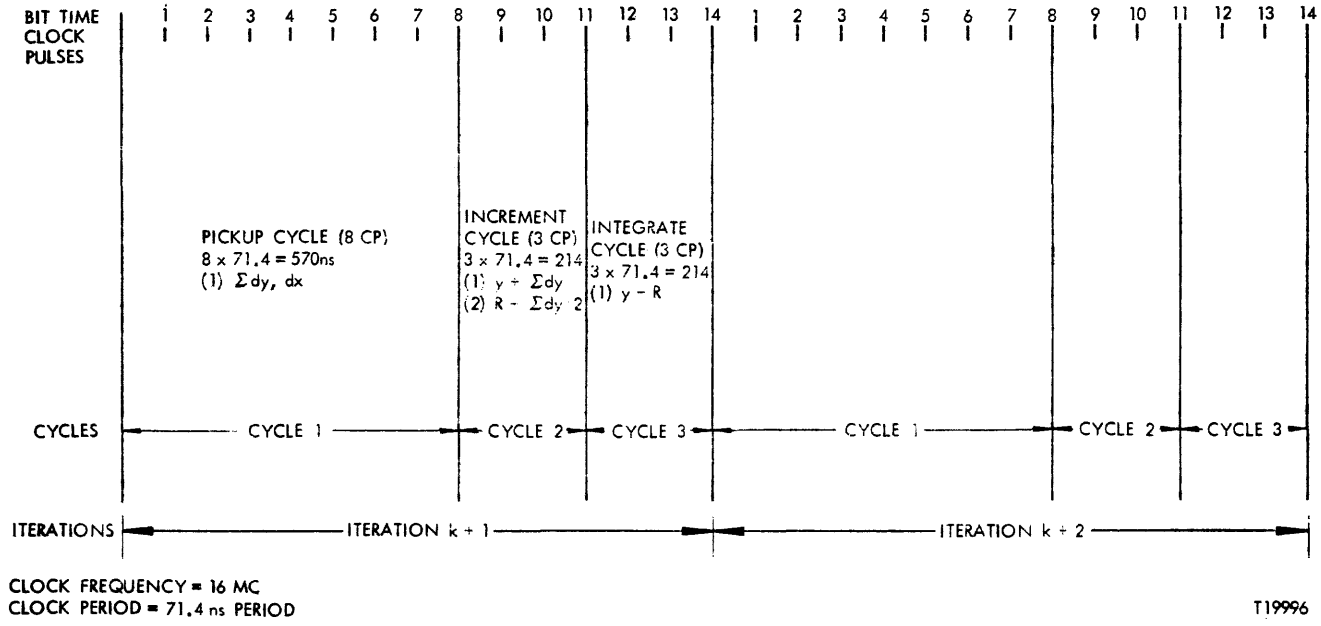


Figure 2—Compute mode sequencing

iteration, to the incremental inputs of the appropriate integrators for use during the next iteration. These incremental quantities are encoded both in space and in time. The space encoding merely selects the appropriate incremental bus line. The time encoding requires a pulse position type of incremental encoding. The Pickup Cycle is composed of 8 bit times that are coincident with the sequential multiplexing of 8 integrator outputs on each interconnection bus. A particular bit time in the Pickup Cycle will be selected, under program control; resulting in one of 8 sequential increments on each bus selected as the input to the integrator. In this manner, the Pickup Cycle will permit the decoding of the increment for use during Cycle 2, the Increment Cycle. Three dy and one dx inputs are permitted for each integrator, where the Pickup Cycle will permit four input increments to be received.

The Increment Cycle, Cycle 2, permits the integrator to accept the incremental dy inputs, received during the pickup cycle, and simultaneously add these increments to the Y and R registers. The algebraic sum of the input dy increments are accumulated in a two bit counter, to be algebraically added to the Y register to update the Y parameter. This dy increment sum is also added to the R register, where the relative weighting is one half of the dy increment weighting. This R register arithmetic operation implements a trapezoidal type integration algorithm. The incremental sum is added to the R and Y registers simultaneously in parallel arithmetic fashion. Therefore, at the completion of the Increment Cycle the value in the Y register is the updated Y value and the value in the R register has the new trapezoidal correction.

The Integrate Cycle, Cycle 3, permits the new Y parameter to be algebraically added to the R register under control of the dx incremental input. At the completion of the Integrate Cycle, the R register will contain the new remainder and the dz output will be available from the integrator to be picked up during Cycle 1, the Pickup Cycle, of the next iteration.

It should be noted that the Y and R registers will be updated with the dy increment input only for the increments accumulated for the Pickup Cycle pertaining to that iteration. In addition, the Y register will be added to the R register during the Integrate Cycle only if a dx increment had been received during the Pickup Cycle pertaining to that iteration. Therefore, the operation in the Increment Cycle depends upon the dy increments and the operation during the Integrate Cycle depends upon the dy and dx increments received during the Pickup Cycle of the corresponding iteration.

The sequencing logic will permit the entrance into the Compute Mode to be accomplished only at the start of the Pickup Cycle. Similarly, exiting from the Compute Mode can only be accomplished at the completion of the Integrate Cycle. This will assure the proper sequential operation of the DDA computer, independent of mode changes and interruptions. Therefore, the GP computer that interfaces with the DDA can command readouts at any time without the loss of DDA information or synchronism. A readout command will, effectively, stop time for the DDA; readout the required information; then permit the DDA to resume proper operation.

The Initialize Mode is intiated by the external GP computer or other equipment that will load initial conditions. This Mode will clear the R register and other selected flip-flops in the system. In addition, it will switch the Y register in each integrator into a serial shift mode of operation. This will permit initial conditions to be loaded into a selected register in serial form from the GP computer, the external tape reader, or various other types of peripheral equipment. The loading of the Y register is accomplished in serial to significantly reduce interconnections and logic. The time required to load 64 integrators with initial conditions will only take several milliseconds of time. This is not considered a significant duration of time to warrant the extra hardware to load the registers in parallel fashion.

The Readout Mode of operation will discontinue the computation, but preserve the parameters contained in the DDA computer. The readout will be accomplished by placing the Y register of each integrator into a shift register mode of operation with a recirculation loop. These registers will continually recirculate, thereby permitting the contents to be available in serial word form to the external equipment, while preserving the Y register information through the recirculation loop. A bit timer will keep track of the word, permitting the shifting operation to be concluded when the serial word is properly assembled in the shift register. All of the Y register outputs will be multiplexed into the input of the GP computer. The particular register can be selected with an address from the GP computer.

The program registers are used to hold the programmed interconnections and scaling information during the Compute Mode of operation. During the Initialize and Readout Modes, the program registers are time shared as interface buffer registers for the serial transfer of information.

A bit timer is used to sequence through the three cycles of the Compute Mode and synchronize the shifting of the Y register in the Initialize and Readout Modes. In addition, it will synchronize all basic timing operations in the computer. A clock pulse generator is used to generate the precision timing required for

synchronous operation of the complete DDA computer.

Software approach to DDA programming

The versatile concept of the TEADDA will permit the implementation of associated software, resulting in a significant increase in the utility of the system. With the advent of the Electrically Alterable DDA, it becomes feasible to describe a DDA compiler and associated software aids. The compiler is a software package that is run on a GP computer to generate a functional program for the DDA. This program is then organized into the machine language format with the DDA assembler program and printed out with the map printer program.

the DDA compiler accepts inputs from the programmer in a programmer oriented form. The inputs are the equations to be mechanized, the initial condition parameters, and a definition of the auxiliary functions such as the map printout. The compiler will automatically generate a detailed program for assembly into machine language.

The input format for the compiler is defined with distinction made between the various parameters and operators, as listed in Table II. The general symbol is defined as a capital letter, while the specific parameter or operator is defined with numbers and lower case letters associated with the general symbol. The programmer will generate an equation in analytic form, shown in equation (1), then convert it to the compiler input form, shown in equation (2), with reference to Table II.

TABLE II—Compiler Input Symbology

Input	Symbol	Example
Variable	V	Vex
Constant	K	Kc1
Differentials	D	D Vex
Integrals	S	S Kc1
Whole Numbers	H	H Vex

$$Z = \int k_1 xydt \tag{1}$$

$$HVz = S(K1*Vx*Vy*DVt) \tag{2}$$

Independent initial conditions would be entered by the programmer, but initial conditions that are implicit in the other parameters would be derived by the compiler. The required accuracy of the solution would be entered as a scaling aid and to define the accuracy-speed tradeoff for the solution.

The compiler output is a scaled program ready for assembly, a list of input programmer errors for diagnosis, and a DDA interconnection map program for documentation. The map program can be generated in punched tape form to be used on an off-line map printer.

The compiler will generate a DDA oriented program by using special algorithms that are presently used by DDA programmers to formulate the problems. The DDA programs are implemented with implicit servos, which are used to generate complex functions and solutions from basic operations. These basic operations include multiplication, addition, subtraction and sine/cosine generation. The DDA compiler will program relatively complex operations such as arctan, division, and exponential functions that are derived from the basic operations. These derivations are illustrated in equations (3), (4), and (5) respectively.

The arctan operation is derived as follows:

$$z = \arctan\frac{x}{y} \quad \text{(input form)} \tag{3a}$$

$$\tan z = \frac{x}{y} \tag{3b}$$

$$\frac{\sin z}{\cos z} = \frac{x}{y} \tag{3c}$$

$$x\cos z - y\sin z = 0 \tag{3d}$$

$$d(x\cos z) - d(y\sin z) = 0 \tag{3e}$$

$$x\,d\cos z + \cos z\,dx - y\,d\sin z - \sin z\,dy = 0 \quad \text{(implemented form)} \tag{3f}$$

The division operation is derived as follows:

$$z = \frac{x}{y} \quad \text{(input form)} \tag{4a}$$

$$yz - z = 0 \tag{4b}$$

$$d(yz) - d(x) = 0 \tag{4c}$$

$$ydz + zdy - dx = 0 \quad \text{(implemented form)} \tag{4d}$$

The exponential functions for most numbers are derived as follows:

$$z = x^b \quad \text{(input form)} \tag{5a}$$

$$\log z - b\log x = 0 \tag{5b}$$

$$\frac{dz}{z} - b\frac{dx}{x} = 0 \tag{5c}$$

$$xdz - bzdx = 0 \quad \text{(implemented form)} \tag{5d}$$

The development of subprograms for the DDA compiler is presently being conducted at Teledyne. A sample compiler flow chart is illustrated in Figure 3.

The DDA assembler program will accept inputs from a DDA compiler or programmer and assemble a detailed DDA program in machine language. The assembler inputs will consist of each computational element identification number, incremental inputs, word length, and element type. The assembler will assign the DDA elements to the computational functions. The coding will be defined for interconnections between elements to implement the computation and for word length to implement scaling.

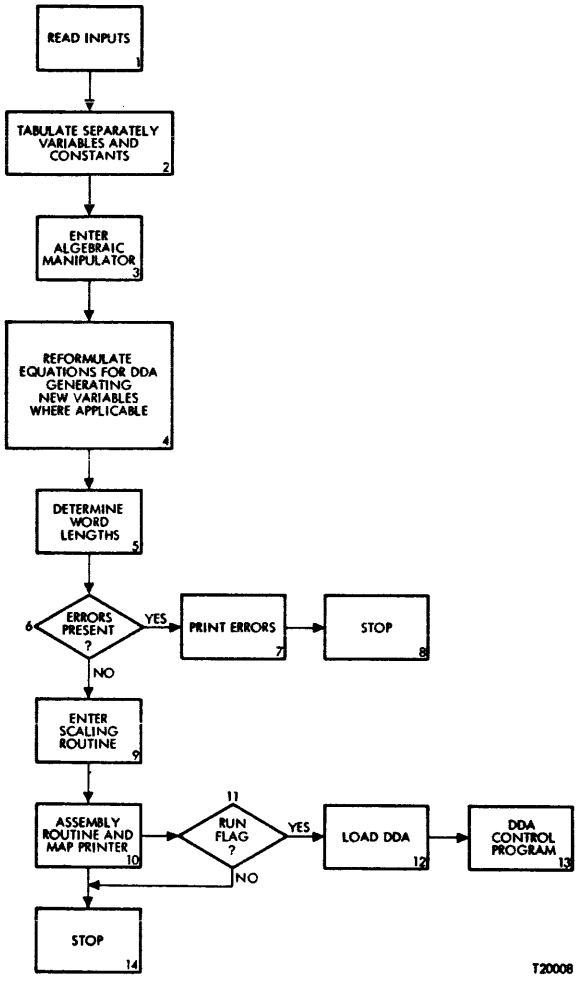


Figure 3 – DDA compiler flow chart

The output of the assembler for a hard wired DDA is a wire list from which the production personnel can wire the elements together. For a patch programmable DDA such as the TRICE; the output is a wire list for the patchboard, a market bit or other selection criterion of the word length, and initial condition parameters. For the TEADDA, the output of the assembler is the program register loading which defines and codes the routing, register length, and polarity for the computation. An ancillary output is the initial condition loading of the Y registers and the start/stop criteria of the program, along with intermediate printouts that may be required by the programmer.

The input format for the assembler is in symbolic form, illustrated in Table III. The DDA elements may be switches, integrators, servos or other types of computational elements. The first letter of the symbol in Table III indicates the element type, while the following three numbers form the identification of that type of element.

Table III. Assembler Input Format

ELEMENT NUMBER AND TYPE	$\pm dy_1$	$\pm dy_2$	$\pm dy_3$	$\pm dx$	$\pm Y$	$\pm Y$ Scale	Length
I125	-S012	S015	I035	-dt	+728.5	+3	14
S012	I120	I025	-S012		+0	+0	10
PRINT	-ITE	1500	Y125	S012			
STOP	-ITE	30000					

TABLE III – Assembler Input Format

The nomenclature for the first row in Table III defines that integrator I 125 shall accept dependent variable incremental inputs from servo S 012, servo S 015, and integrator I 035; with an independent variable incremental input (dt) from the computer clock. The initial conditions, scaling, and register length parameters are also defined.

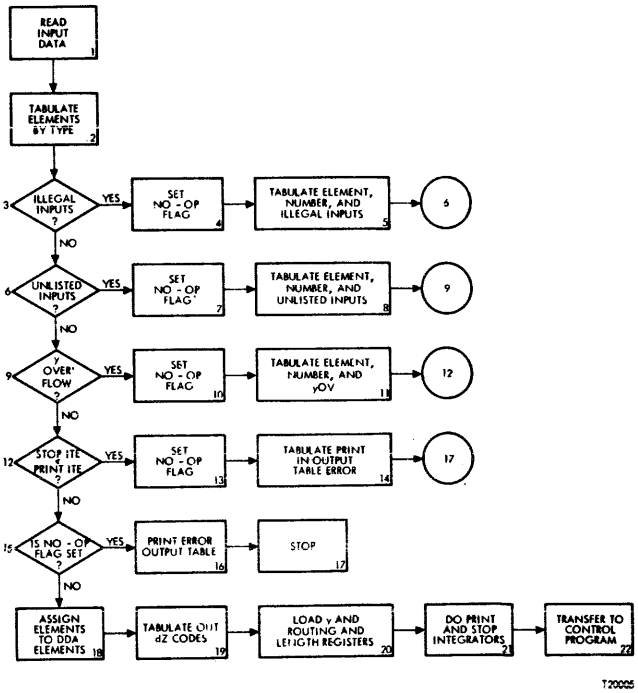


Figure 4 – Assembler flow chart

A typical assembler program flow chart is illustrated in Figure 4. This program is similar to DDA assembler that is presently used at Teledyne to program a hard wired avionic DDA that is in production.

The map printer program is primarily used as a checkout aid and for documentation of reports and programs. The inputs are identical to those generated for the assembler, with the added feature of assigning an acronym to name the variables. Because of lack of industrial standards for DDA symbology and the difficulty of interconnecting loops by a shortest lead concept, the map printer output nomenclature shall be as defined in Figure 5 and illustrated in Figure 6.

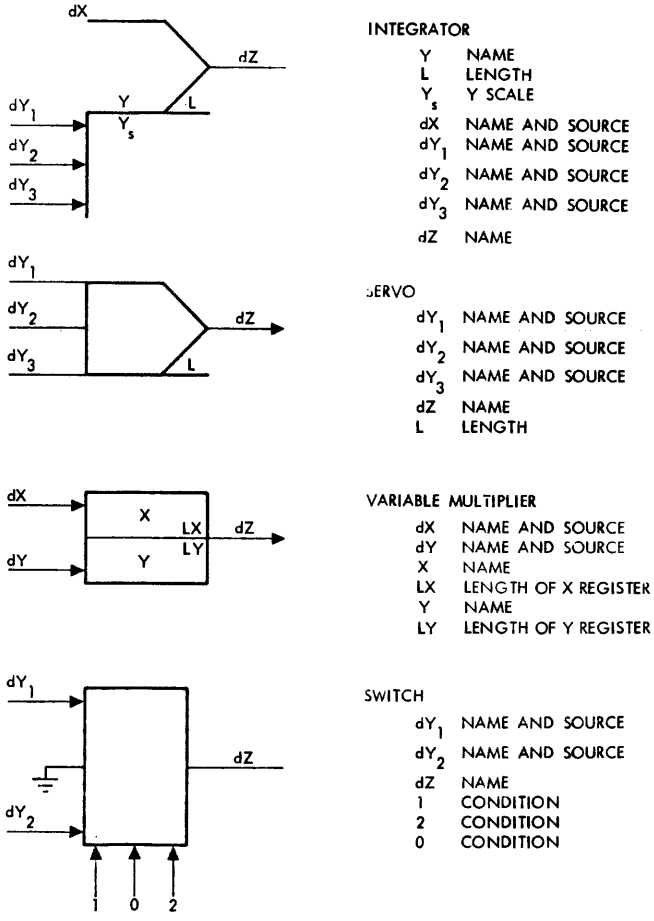


Figure 5 – Key to DDA map printer

Hardware description

The electronics industry has made tremendous progress in the development of advanced electronic components and techniques. In particular, the advent of integrated circuits has virtually revolutionized the electronics industry, especially the digital computer area. The trend has been toward components that are very small, fast, low in power consumption, and highly reliable. In order to take advantage of the advanced

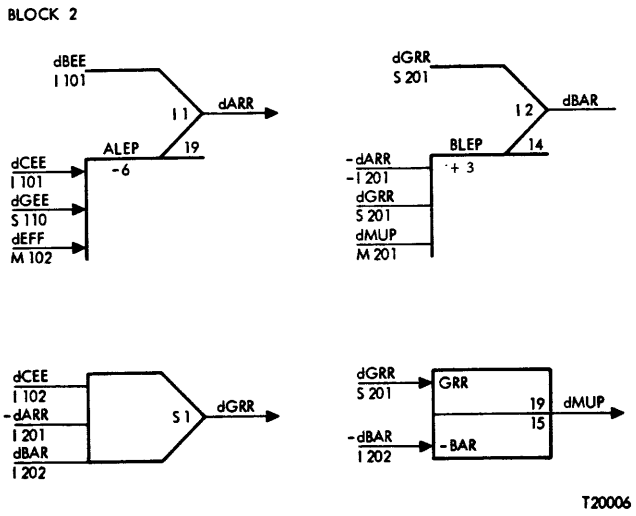


Figure 6 – Map printer output

hardware, new and sophisticated handling and packaging techniques have been required. In order to handle and package the miniature components, much of the advantage of the small size has been lost. Teledyne has developed packaging techniques that make maximum use of the characteristics of the advanced components. In particular, the Micro-Electronic Modular Assembly (MEMA) takes maximum advantage of the characteristics of integrated circuits. These are:

a. Preserving the small size characteristic of the integrated circuit chip by placing many chips in a single package.
b. Preserving the high speed characteristics of the integrated circuit chip by placing many chips in very close proximity with extremely short interconnections.
c. Preserving the inherent reliability of the integrated circuit chip by eliminating multiple packaging levels.

In addition, greatly improved manufacturing and maintenance concepts are achieved and manufacturing costs are significantly reduced. The basic MEMAs are hermetically sealed flat packs with 24 or 36 leads and dimensions of 1.0 × 0.75 × 0.06 inches. Each MEMA can contain up to 32 digital integrated circuit "bare chips" or 25 analog chips (integrated circuit, resistor, capacitor, etc.). The digital MEMA is illustrated in Figure 7 and the analog MEMA is illustrated in Figure 8. The "bare chips," 1 mil wire leads, and substrate interconnections are clearly visible in the photographs on the MEMAs with the covers removed. The interconnection pattern is photoetched onto the ceramic substrate, the chips are die-bonded to the substrate, then leads are bonded to connect the chip signal pads to the substrate interconnection pattern.

The leads are composed of 1 mil (1/1000 inch) aluminum wire that is ultrasonically bonded to the chip and substrate.

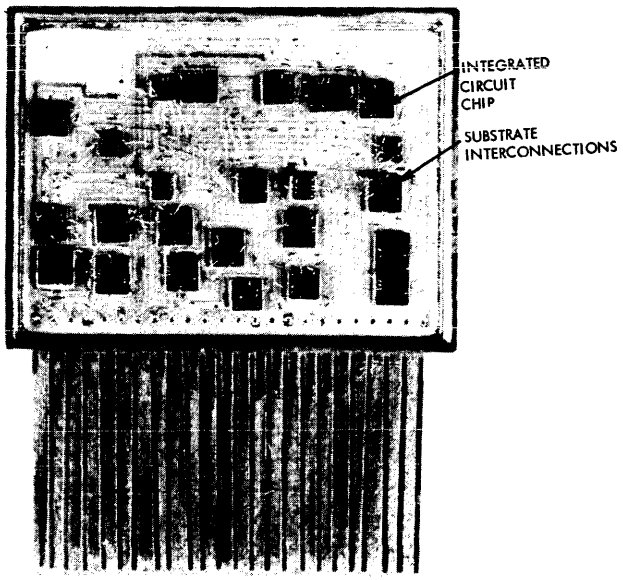


Figure 7—Digital circuit MEMA

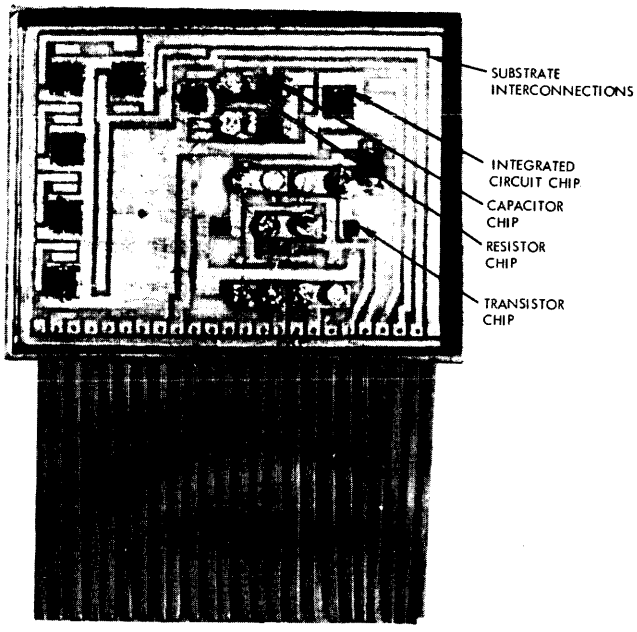


Figure 8—Analog circuit MEMA

The MEMA contains the functional complexity of a large printed circuit board, but in a highly miniaturized package; yielding enhanced performance, reliability, cost, size, and weight.

Several levels of modularity are provided in the computer design, as illustrated in Figure 9. Monolithic integrated circuits are assembled to form MEMAs, MEMAs are assembled to form functional

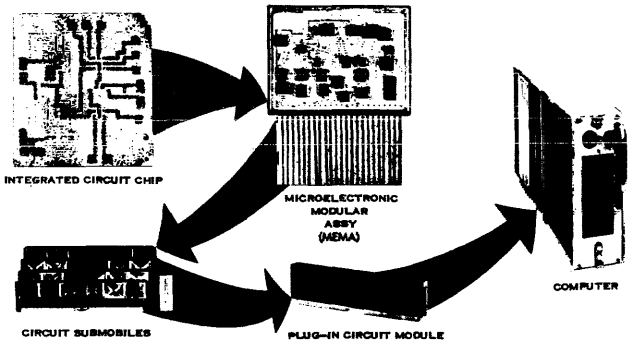


Figure 9—Computer build-up

submodules, submodules are assembled to form functional modules, and the modules are then assembled to form the computer unit.

Due to the high frequency of the clock pulse generator, considerable design effort has been expended to assure that propagation delays and the clock pulse skew effect are minimized.[3] Propagation delays through logic and interconnections are made nearly constant to all parts of the computer. The clock pulse generator is located at the physical center of the computer, with the pulses fanned out to all modules in a radial manner. It should be noted that the dimensions of the computers are in inches rather than feet, reducing the propagation delay problem. The small dimensions are primarily the result of the MEMA packaging technique.

The TEADDA packaging technique implements each integrator and the associated programmable interconnections and scaling in twelve MEMAs. Each integrator and its associated logic will have a Mean Time Between Failure (MTBF) of almost one million hours, based on the ten million hour MTBF of a MEMA. Therefore, the MTBF for the TEADDA containing 64 integrators is over 12 thousand hours or 500 days. Using a degradation factor of two for higher levels of packaging and interconnection, the MTBF would be greater than 250 days of continuous operation. The low failure rate and the ease of malfunction isolation and correction yields an extremely low down time for the TEADDA. This is a significant factor in the low "cost of ownership" that is an important part of the TEADDA concept.

CONCLUSION

The electrically alterable DDA should prove to be an extremely useful simulation and computing "tool"

to be used in conjunction with analog, digital, and hybrid computers. The TEADDA fills a technological gap associated with computer systems, where the TEADDA will complement the other computing "tools" to more efficiently cover the spectrum of computer applications.

REFERENCES

1 E L BRAUN
Digital computer design
Academic Press New York 1963 Chapter 8 P 448

2 G A KORN T M KORN
Electronic analog computers
McGraw Hill New York 1956

3 G P HYATT
High speed digital computer implementation techniques
1967 International Electron Devices Meeting

DATA FILE TWO–A data storage and retrieval system

by REUBEN S. JONES*
General Electric Company
Phoenix, Arizona

INTRODUCTION

The Data File Two* was conceived to meet the needs of a corporate headquarters in the $500 million sales category. The file would contain monthly sales data for the past two years and the customer master file, current month orders and sales, railcar locations, current year profit plans, personnel records, ledgers, one-year profit plan data, and five-year profit plan data. This file is assumed to be about 400,000 items.

The stimulus for trying to put "all" corporate data in one file came from dealing with these data handling problems:

- Coding and coding structure is difficult to discipline and change. Yet, the acquisition of new companies and the launching of new products require fast change and severe discipline.
- Relating profit planning data for the next one- and five-year periods to current performance, corporate goals, and the capital plan is done manually. The planning process is severely limited by the slow turn-around on analyses of these relationships.
- Inquiry into sales history, personnel files and planning files, and current order and sales files is available only through large periodic reports.
- New data structures (such as proposed organization changes) cannot be easily proposed and analyzed without creating new files and programs or doing the analysis manually.
- Data external to the corporate operations cannot be easily put into the current corporate data structure for comparisons and analysis.

The reader should find the following ideas acceptable after reading this paper:

1. A data file with 100% indexing is feasible.
2. A file like DF-2 can be used for large data banks (400,000 items).
3. Synonyms and dictionary word hierarchies can be had with little extra use of disc space or processing time.
4. List processing (as used here) is the best technique for indexing large data files.
5. The notational system used in the Integrated Date Store[1] literature is powerful and makes file descriptions easily understood.
6. The IDS/COBOL verbs are useful in describing data file procedures.

File description

A. The directory-dictionary approach to communication with a data storage file

A directory-dictionary is used to classify data in the file, for storing that data, and for retrieving the data from the file.

The directory contains a list of all of the attributes under which a data item may be classified. The directory might also be considered a list of fieldnames. In DF-2, the directory would contain the following kinds of entries:

PRODUCT EMPLOYEE NAME
CUSTOMER SOLD TO RAILROAD CAR NUMBER
DATE SHIPPED

There is a dictionary for each directory entry. This dictionary contains the specific attribute values for the directory item under which the dictionary is found. A dictionary under the directory entry PRODUCT might contain:

DIAMMONIUM PHOSPHATE FELDSPAR
MONOSODIUM GLUTAMATE BENTONITE

When storing and retrieving data, the directory item and the dictionary item are used in pairs:

PRODUCT DIAMMONIUM PHOSPHATE
CUSTOMER SOLD-TO WESTINGHOUSE
DATE INVOICED 01/02/67
CAR NUMBER ACL 40781

---

* This paper was written while Mr. Jones was employed at International Minerals and Chemical Corporation where he held positions as Operations Manager - Accent International Division, Manager - Systems Engineering, and Operations Research Engineer.

* A name only. No relation to other file design names.

Synonyms may be declared for both directory entries and dictionary words so that short spellings, common abbreviations and presently used coding may be used to store and retrieve data in the file.
Data retrieval is accomplished by issuing to the system lines of data description. Each line of data description consists first of the directory entry and, second, the dictionary value. Any number of descriptive lines can be used to describe a data item.

DF-2 is to be implemented using Integrated Data Store.[1] Integrated Data Store notation and terminology will be used throughout this file description. The reader may wish to familiarize himself with IDS notation and language by referring to the publication titled "Integrated Data Store," published by the Information Systems Division of the General Electric Company. Some IDS diagram conventions and other notation conventions are given in the Appendix.

B. The elementary file structure

The records and description which follow pertain to the elementary file. "Elementary" here means the minimum file necessary to implement the directory and dictionary and the storage of data items. In the actual implementation, the data storage file will be somewhat more elaborate in order that additional features such as synonyms, storage of character strings, cartesian intersections such as month, day and year, and hierarchical relationships such as Belgium and Luxembourg being part of the Benelux group. These elaborations, however, can, at this point in the description, serve only to confuse the basic file structure description.

The elementary file contains the following records:

File Entry - one record only in the file. The point at which the file is entered.

Directory Records - contain the names of the classes into which the dictionary is divided. These records may contain words such as PRODUCT, CUSTOMER, SHIP FROM, SHIP TO, NAME, DATE, or EMPLOYEE NAME. They are spaced out across the file.

Dictionary Records - contain the particular descriptors such as DIAMMONIUM PHOSPHATE, WESTINGHOUSE, NAHX 94872, a Duns Number, or JANUARY. They are placed near the directory records.

Value Records - These records contain only numeric quantities in the elementary file. They are placed at regular intervals across the file.

Coupler Records - contain nothing but chain links. They are placed near the Value records.

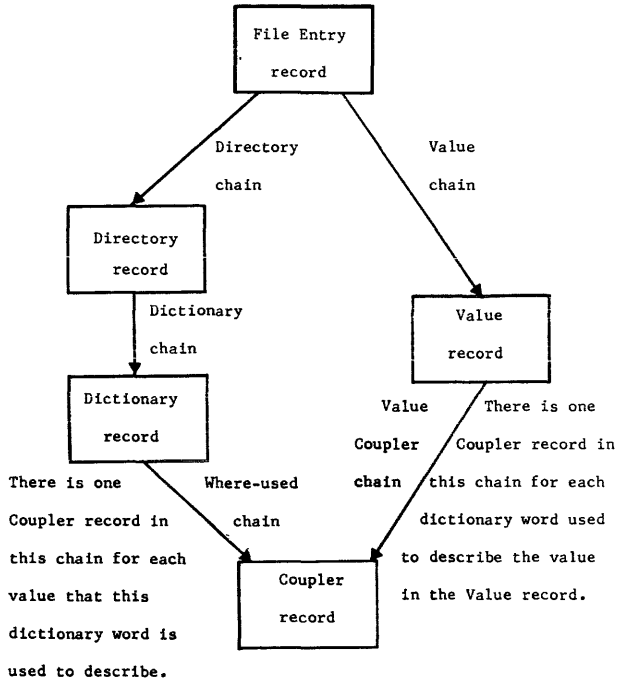


Figure 1 – The elementary file structure diagram

The Elementary File Structure Diagram is shown in Figure 1. There is one Value record for each data item. The Value record contains the one numeric piece of data. There is one Coupler record in the Value Coupler chain for each dictionary word needed to classify the "value." The Directory chain connects all of the directory entry records. The Dictionary chain connects all of the dictionary words to a directory entry. The Value chain connects all of the value records. The Where-used chain connects, via the Coupler records, all of the places that a particular dictionary word is used to describe a data item.

An illustration of how a data item would be stored in the Elementary File is shown in Figure 2. The data item is the numeric value *407* classified under PRODUCT POTASH, and UNITS TONS. The blank rectangle at (2) is the Coupler record linking the number *407* to the word TONS under the directory entry UNITS. The blank rectangle at (3) is a Coupler record linking the number *407* to the dictionary word POTASH under the directory entry PRODUCT.

The dictionary is to contain all the words necessary to describe the data items. Computational numeric values are stored in the value record. In the elementary file, the value record contains only a numeric value and two file chain links.

In the full-blown file, the value records are allowed

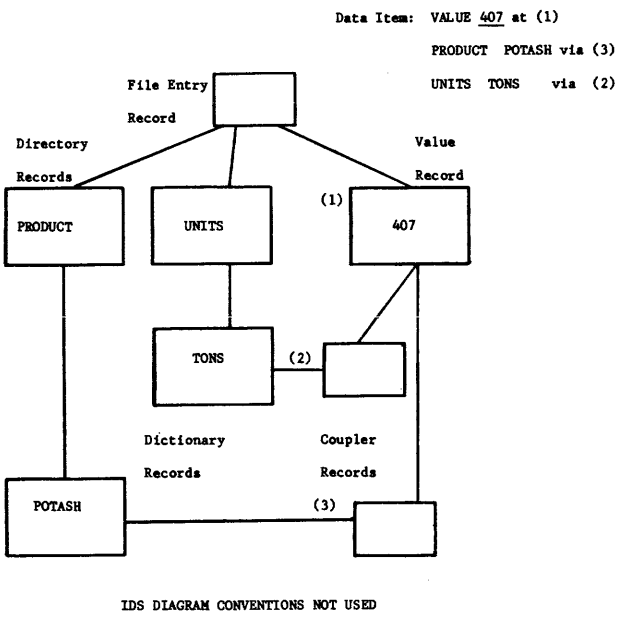


Figure 2 – An illustration of how a data item would be stored in the elementary file

to contain character strings such as address, terms phases, or shipping instructions.

C. A more elaborate file structure

1. Synonyms

A means of declaring synonyms in the dictionary is provided. Synonyms are used to name coding or short words which could be used to store data and to make inquiries. The synonym for AC'CENT 24-4(1/2) OZ might be: 51307, 24-4 1/2 OZ, 4 1/2 OZ AC'CENT, and AC'-CENT 4 1/2 OZ. The principal descriptor used as file output is called the prime dictionary word, and the other synonyms are called dictionary synonyms.

Synonym declaration is also provided for the directory entries. Synonyms for the directory entry PRODUCT could be PROD., PROD, PRO and PROD CODE.

Figure 3 shows the elementary file diagram with synonym records added. The discussion on synonyms applies to directory synonyms as well as dictionary synonyms.

The synonyms use up file space only to the extent of adding one dictionary record for each synonym.

The dictionary chain is in alphanumeric sequence. The synonym records are in the same dictionary chain so that prime words and synonyms are intermingled in the list. Each synonym record is also a detail record of a chain of which the prime word is master. The synonym records have a pointer to master so that when a search hits on a synonym, the prime word may be retrieved directly.

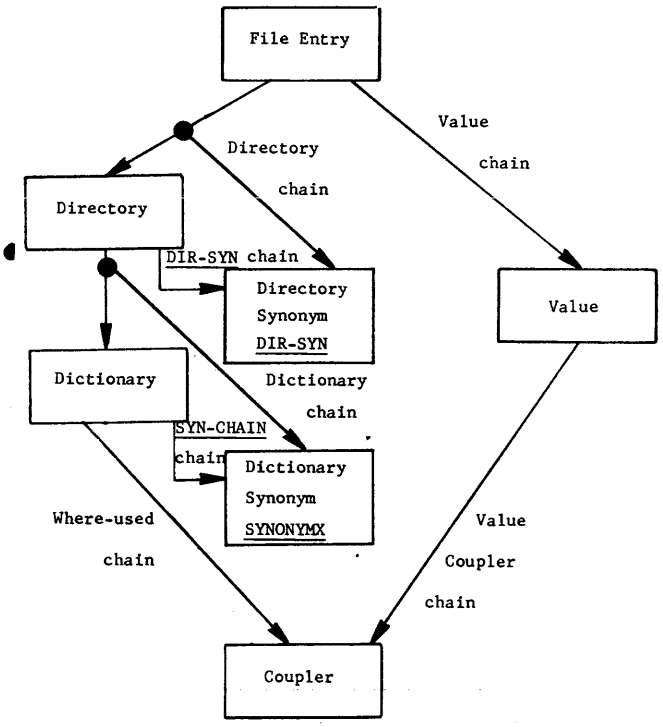


Figure 3 – The elementary file diagram with synonym records

2. Character String

Long credit terms, phrases, shipping instructions spelled out company names and street addresses would probably not be dictionary words. It is, therefore, necessary to provide a means of storing longer character strings.

To provide storage and retrieval of character strings, three new record types are provided. The character string record is comparable to the value record in the elementary file. The first 40 characters of a string are stored in the character string record, CHAR-STRING. The remaining characters in the string are stored in records named STRING-LINES. These records hold 40 characters each and are chained to the *CHAR-STRING* record as master thus enabling the file to hold character strings of any length.

Another type of Coupler record is needed which is named STRING-COUPLER. The elementary file with character string records added is shown in Figure 4.

The Character-String records could be thought of as a second file which uses the same dictionary as the first file.

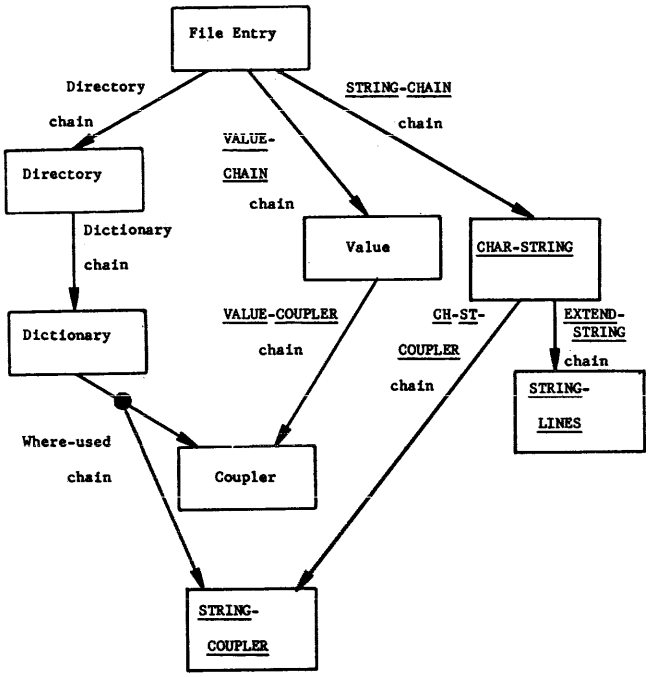


Figure 4 – The elementary file with character string records

3. The Cartesian Directory Record

When storing data, the question arises as to whether to put the date, "01/02/67", in the dictionary or to store the date as three parts: Month 1, day 2, and year 1967. Usually, the decision would be to store month, day, and year separately in order to make it easier to summarize data into monthly and yearly periods.

The casual system user who is retrieving data on, for example, shipments for May 2, 1968, is not likely to remember that date is stored as month, day, and year. A solution would be to store the date under DATE 01/02/67, and MONTH 1, DAY 2, and YEAR 67. This would, however, increase the number of Coupler records needed in the file and thus would increase the use of physical file space.

The splitting of date into month, day and year is provided for in the directory without redundant data classification. This is done by using the *CARTESIAN* directory record. The Elementary file with the *CARTESIAN* record is shown in Figure 5.

The *CARTESIAN* record is a detail record in the Directory chain just as are other Directory records.

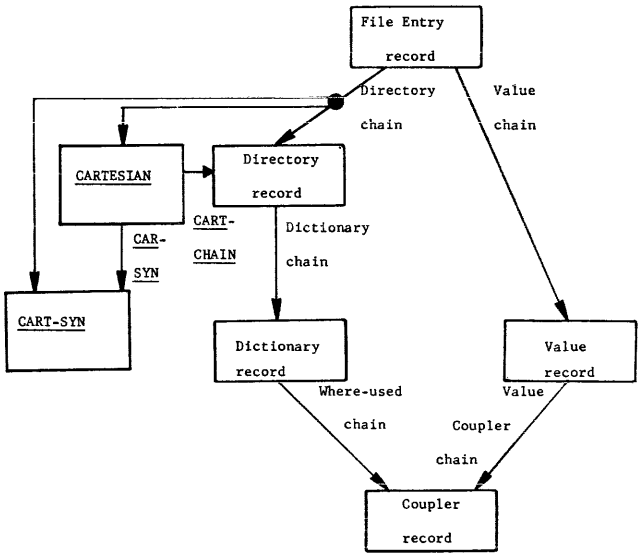


Figure 5 – The elementary file with the *Cartesian* record

The *CARTESIAN* record is master of the chain *CART-CHAIN*. The detail records in the *CART-CHAIN* chain are the Directory records which make up the *CARTESIAN* directory entry. Using the date example, the directory word DATE would be stored in the *CARTESIAN* record and the directory entries DAY, MONTH and YEAR would be stored in Directory records. The DAY, MONTH and YEAR records would be detail records of the *CART-CHAIN* chain which had the DATE record as master. The *CARTESIAN* record has no dictionary since the elements of date are stored in the MONTH, DAY and YEAR dictionaries. Synonyms for the *CARTESIAN* directory record are contained in the *CART-SYN* record.

When retrieving data using DATE 1/2/67, the retrieval program would, upon finding that date was in a *CARTESIAN* record, fracture the inquiry into 1, 2, and 67, assigning "1" to the first directory item in the *CART-CHAIN* chain, which would be MONTH, assigning "2" to the next *CART-CHAIN* chain record, DAY, and assigning "67" to the next record item YEAR. An inquiry which used

DATE 1/2/67

for a retrieval criteria would be translated into:

MONTH 1
DAY 2
YEAR 67

The same would occur when storing a data item.

4. The Benelux Hierarchy

Dictionary words have hierarchical relationships to other words that need to be defined in the data file. An example might be sales to Belgium, the Netherlands, and Luxembourg which make up the Benelux group. It is desirable to have data for these three countries summed when a call for sales to Benelux is given. All data stored under Belgium, Netherlands, and Luxembourg could also be connected to the word Benelux. As in the consideration of synonyms, this would require an additional Coupler record for each data item. By use of the *BENELUX* record and the *LUXEMBOURG* record in the Where-used chain, the number of extra records to define the Benelux problem described above is reduced to four regardless of the number of data items. The diagram of the Elementary file with *BENELUX* and *LUXEMBOURG* records added is shown in Figure 6.

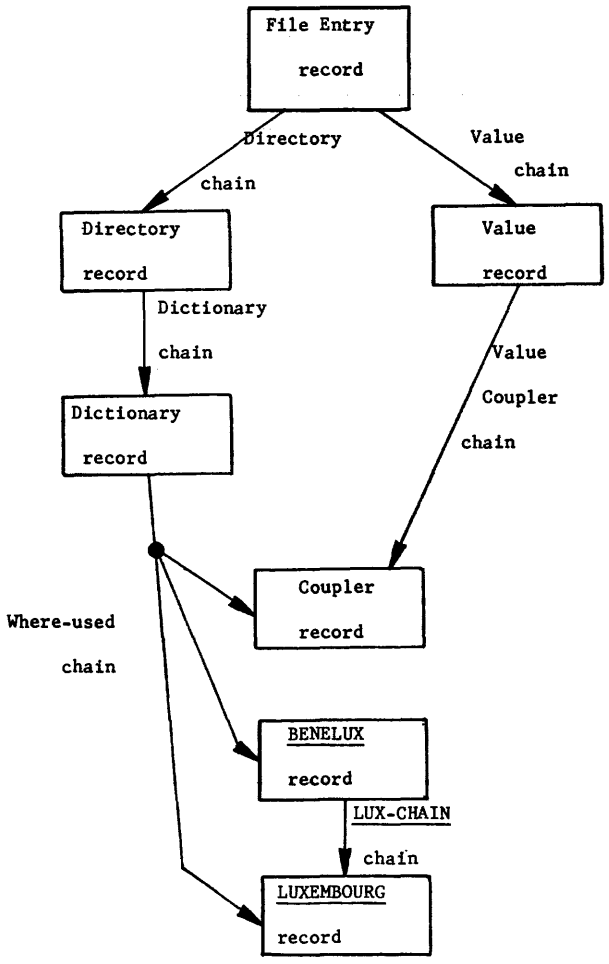


Figure 6 – The elementary file with *Benelux* and *Luxembourg* records

The *BENELUX* and *LUXEMBOURG* records are detail records in the Where-used chain, and the *LUXEMBOURG* record is also a detail record in the chain *LUX-CHAIN*.

The *BENELUX* record is master of the *LUX-CHAIN*. The *BENELUX* record is found in the Where-used chain of the dictionary word higher in the hierarchy (the word *BENELUX* in our example) and the *LUXEMBOURG* record is found in the Where-used chain of the dictionary word lower in the hierarchy (Belgium, Luxembourg, and the Netherlands, in our example). The depth of the hierarchy is unlimited, and the number of different hierarchies in which a word may be defined is unlimited since there is no limit to the number of BENELUX and LUXEMBOURG records which may be placed in the Where-used chain. The linking together of these records for the Scandinavian countries is illustrated in Figure 7.

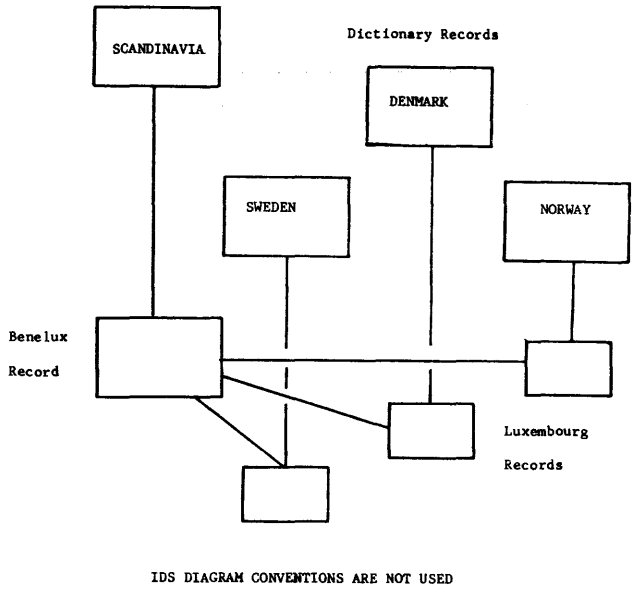


Figure 7 – How Benelux records are used to tie the Scandinavian countries together

Data extraction scheme

A. Data extraction scheme for the elementary file

In order to explain the data extraction scheme, the elementary file will be used along with a simplified retrieval routine and inquiry. In the next section on retrieval timing, other IDS features and record fields not mentioned before will be added to improve the retrieval times.

The user wishes to see a total of all potash sales. The inquiry would be:

```
FOR
PRODUCT        POTASH
TRANSACTION    SALE
DISPLAY        TOTAL
```

Figure 8 shows a data extraction procedure for the elementary file which would serve the above inquiry.

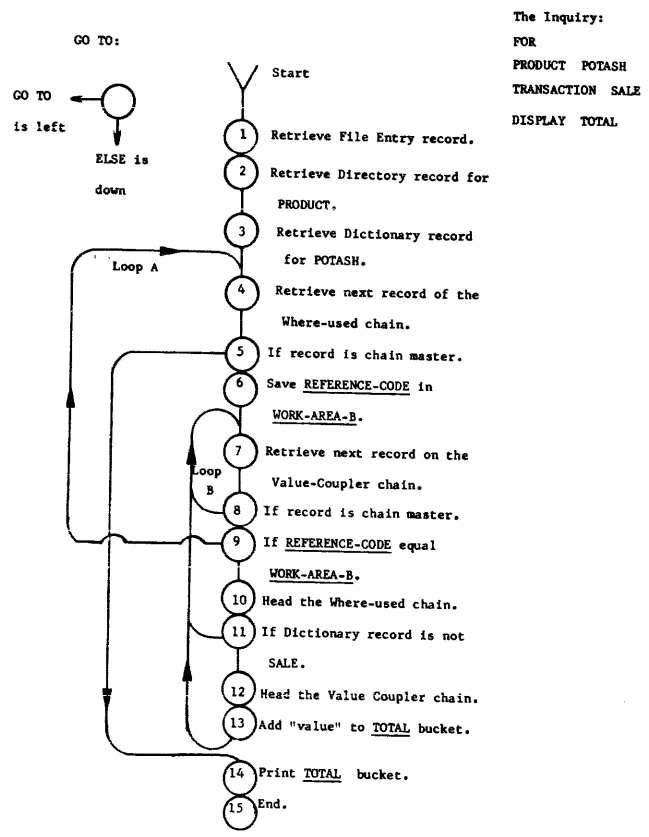


Figure 8 – A data extraction procedure for the elementary file

B. Modifications to the elementary data extraction scheme for faster operations and a more elaborate inquiry.

The following discussion concerns data extraction speed and the file and procedure modifications necessary for speed-up.

The following timing assumptions and definitions will be made so as to simplify this discussion.

- Magnetic disc with movable arms will be the file device
- A large number of IDS pages will remain in core memory (10 or more).
- The file device will have one data channel and will allow no simultaneous arm seeks.
- In-core processing time will be either overlapped or otherwise insignificant.
- For retrievals of records along a chain which uses the "Place-Near" IDS statement, there will be one disc read for each five (5) retrieve commands and one arm seek for each ten (10) retrieve commands.
- For retrievals of records not along a "Place-Near" chain, there will be one seek and one read for each retrieve command.
- An average of two synonyms will be assumed for each prime dictionary word and directory entry.
- V = the number of value records (data items).
- C = the average number of dictionary words needed to describe one value. Also, the average number of Coupler records in the Value-coupler chains.
- D = the number of dictionary prime words.
- E = the number of directory prime entries.
- $\frac{VC}{D}$ = the average number of Coupler records in the Where-used chains.
- t = average disc rotational delay.
- T = average arm seek time.
- N = number of search (item selection) conditions given per inquiry.

The chart in Table I gives a brief statement on how the file or procedure was changed for speed-up, the resulting timing formula for each block in Figure 8, and the file search time computed for a specific example.

The example data base used in the timing and file size illustrations is as follows:

Prime dictionary words	100,000
Synonyms per word	2
Prime directory entries	100
Synonyms per entry	2
Data items	400,000
Classifications per data item	20
Number of item selection criteria per inquiry	2

The largest time loss in the Elementary procedure is in Step 10. This can be reduced to in-core time by defining a link to master of the Where-used chain in the Coupler-record and by using the reference code* of SALE to compare to this link (reference code). This change would cause the repitition of the processing in Steps 1 through 3 "N" times since the reference code of SALE must be found for comparison.

*Reference code is an IDS term for the records relative address on the file.

The next largest time loss is in Step 7. This will stay except for a small reduction. The number of executions of Loop B will be reduced by putting a count of Coupler-records on the Where-used chain in the prime Dictionary record. This will allow us to select the search path (in the search example, the word POTASH or the word SALE) which is the shortest. This would reduce the number of Loop B executions by a factor of about $\frac{\sqrt{N}}{N}$.

Steps 2 and 3 will be reduced by breaking the Dictionary and Directory chains with range master* records. These range master records will be spaced across the file allowing room for Dictionary records to be clustered around them.

The conditional branch at Step 11 would be elaborated to include a check against all search conditions with each condition being marked for hit or no-hit. The "add Value to bucket" step at 13 would be replaced by whatever report or computational requirements are called for.

Step 12 time will probably be zero since the Value coupler chain is a "Place-Near" chain, and the record is likely to be in core.

Table I gives the arithmetic expressions for the search time consumed by each procedure step.

The average search time into the example data base after making the modifications noted above is:

$$1425\ \frac{T+t}{10}\ \frac{+64\,(T+t)}{5}$$

For disc units with the seek and rotational delay times of

T	and	t	The average search time is:
200		25	50 seconds
90		10	22 seconds
0		25	9 seconds

File size for full blown file with all features discussed

The full blown file structure is shown in Figure 9.

The size of an Integrated Data Store record

where: R = Record size in characters
L = Number of chain links, counting links to master
is: R = 5 + 4L + (Number of Data Char.)

The sizes of the DF-2 records are shown in Table II. Also in Table II, the expressions for computing the disc space used by each record type is shown. A file size is computed for the example data base.

*Range masters break up a long detail chain. Where S = number of records in the chain, search speed improvement is about $\frac{2\sqrt{S}}{S}$.

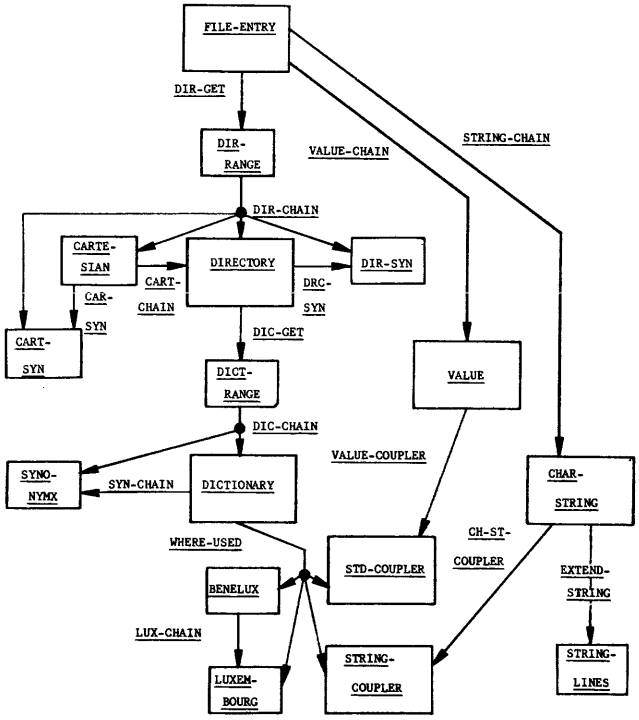


Figure 9 – The full blown file structure diagram

The data storage and retrieval language

A. Function

The function of the Data Storage and Retrieval Language is to provide a means of storing data in the file from common sources and making changes to that data, to make changes to the descriptive words used in the directory and dictionary, and to retrieve and report data from the file.

More specifically, it is assumed that the following operations with the file will be desirable:

1. To define dictionary words and their synonyms.
2. To define directory entries and their synonyms.
3. To load data into DF-2 from magnetic tape and card files used as part of existing systems.
4. To make changes to dictionary words, directory entries and their synonyms already stored in the file.
5. To change data previously loaded into the file.
6. To select and display data from the file "on-line".
7. To extract and print data in more elaborate report forms "off-line".

TABLE I

AVERAGE SEARCH TIME EXPRESSIONS

Search Procedure Step No. in Figure 8	Explanation of File Changes For Speed-Up	Expressions For Average Search Time	Average Search Time For the Example Given in the Text
1	File master is usually core resident.	0	0
2	Direction chain is a "place near" chain subdivided by range master records. Two synonyms per prime directory entry.	$N\sqrt{3E}\left(\frac{T}{10}+\frac{t}{5}\right)$	$35\left(\frac{T}{10}+\frac{t}{5}\right)$
3	Dictionary chain is a "place near" chain subdivided by range master records. Two synonyms per prime dictionary word.	$N\sqrt{\frac{3D}{E}}\left(\frac{T}{10}+\frac{t}{5}\right)$	$110\left(\frac{T}{10}+\frac{t}{5}\right)$
4	Always a random retrieval $\frac{VC}{D}$ times.	$\frac{VC}{D}(T+t)\sqrt{\frac{N}{N}}$	$64\ (T+t)$
5 & 6	In core operations.	0	0
7	Value Coupler chain is a "place near" chain. Step is executed C times for each Loop A execution.	$C\left(\frac{VC}{D}\right)\left(\frac{T}{10}+\frac{t}{5}\right)\frac{\sqrt{N}}{N}$	$1280\left(\frac{T}{10}+\frac{t}{5}\right)$
8 & 9	In core operations.	0	0
10 & 11	A link to master is placed in the Coupler record for the Where-used chain. The reference code in this link is used to compare to the reference code of the dictionary word SALE. In this way, the retrieval of the dictionary word is avoided at this procedure step.	0	0
12	Since Step 7 accessed all couplers for the Value record, and since the Value Coupler chain is a "place near" chain, the page containing the Value record is almost certain to be in core.	0	0
13	In core operation.	0	0

TABLE II

DISC SPACE USAGE

Record Type	Links (L)	Characters Per Record	Expression For Number Of Records	Number Of Records	Estimated File Size For Example In Millions of Characters
DIR-RANGE	2	13	$\sqrt{3E}$	18	-
DIRECTORY	4	39	E	100	-
CARTESIAN	3	35	-	-	-
DIR-SYN	2	31	2E	200	-
CART-SYN	2	31	-	-	-
DICT-RANGE	2	31	$\sqrt{3D/E}$	55	-
DICTIONARY	4*	44	D	100,000	4.40
SYNONYMX	2	31	2D	200,000	6.20
LUXEMBOURG	3*	25	1,000**	1,000	.03
BENELUX	3*	35	300**	300	.01
STD-COUPLER	3*	17	CV	8,000,000	136.00
STRING-COUPLER	3*	17			
VALUE	2	21	V	400,000	8.40
CHAR-STRING	3	57	V/10**	40,000	2.28
STRING-LINES	1	49	V/10**	40,000	1.96
TOTAL					159.28

* Links to master included

** An estimating basis assumption.

All of these functions are to be defined by simple statements which can be typed in at remote terminals and which will be executed by interpretive programs written in IDS/COBOL.

The data retrieval capability is to be used by persons with little data processing experience and probably no formal typing training. Except for logic of two or more levels and "greater than" "less than" conditionals, no special characters will be needed. Descriptions with imbedded single blanks may be used since two or more blanks are used to separate words instead of single blanks or commas.

B. Examples of language uses

A formal language description is not given because it has not been completed. Some illustrations of the use of the DF-2 language for retrieval of railcar tracing data, planning data, and personnel data are shown below.

Example 1

Railcar locations might be stored and updated in Data File Two. The first example is a request for all railcars passing St. Louis carrying Triple Super Phosphate which are loaded but unassigned (Roller) or destined for the company's own plant (Plant Food). The system responded, as shown, with four hits.

```
FOR
STATUS          ROLLER
OR
SOLD TO         PLANT FOOD
CAR LOCATION    ST.LOUIS
PRODUCT         TRIPLE SUPER
SIGHTING        NOT     DESTINATION
UNITS           TONS

DISPLAY                 REPORTING RR
CAR NUMBER              VALUE
PRODUCT TYPE            END
```

CAR NUMBER	PRODUCT TYPE	REPORTING RR	VALUE
ACL 086541	ROP	MONON	76.3
SAL 030655	GRANULAR	MOPAC	54.4
ACL 761453	COARSE	KCS	74.6
SAL 088304	ROP	MONON	79.3

END

Example 2

Company controlled leased cars would be maintained on the file whether loaded or returning. Another type of inquiry would be to count the hopper cars returning to three shipping locations (Noralyn, Bonnie, Port Sutton).

```
FOR
STATUS          RETURNING
CAR TYPE        HOPPER

COUNT RETURN TO     PORT SUTTON
NORALYN             END
BONNIE

END
```

RETURN TO	NORALYN	5
RETURN TO	BONNIE	10
RETURN TO	PORT SUTTON	16

END

Example 3

The profit planning data for the company might also be maintained on the file. The next example of the language use shows the TOTAL verb and the system response.

```
END
TRANSACTION         SHIPMENT
UNITS       TONS
FISCAL YEAR         68/69
GENERATION          FORECAST 1
PRODUCT GROUP       POTASH

TOTAL       MONTH       ALL

END
```

JULY	45260
AUGUST	43565
SEPTEMBER	42536
OCTOBER	32564
NOVEMBER	50632
DECEMBER	86734
JANUARY	361271
FEBRUARY	201000
MARCH	107271
APRIL	50000
MAY	501761
JUNE	250302

END

Example 4

The next example is an internal personnel search for an engineer with business systems experience.

```
FOR
DEGREE          MASTER
MAJOR           ENGINEERING
```

```
FIELD       MECHANICAL      CHEMI-
            CAL     MINING
AGE         LESS THAN 35
EXPERIENCE      BUSINESS SYSTEMS
DISPLAY     NAME        LOCATION

END
```

(System response not shown)

APPENDIX

CONVENTIONS USED IN THE TEXT AND IN EXHIBITS IN THIS PAPER WITH REGARD TO RECORD NAMES, CHAIN NAMES, WORDS APPEARING LITERALLY IN A RECORD, AND SCHEMATIC DIAGRAMS OF LIST STRUCTURES

Record, field and chain names are in italics and are in all capital letters.

Examples:

FILE-ENTRY	a record name
WHERE-USED	a chain name
WORK-AREA-B	a field name

Words which are contents of record fields such as dictionary words in the data file are in all capital letters. Numeric values which are contents of Value records are in italics.

Examples:

MONTH	JANUARY
YEAR	67
VALUE	*245.678*

Discriptive names of records are used in describing the elementary file to avoid having the reader memorize shortened COBOL names. These names are in beginning capitals only.

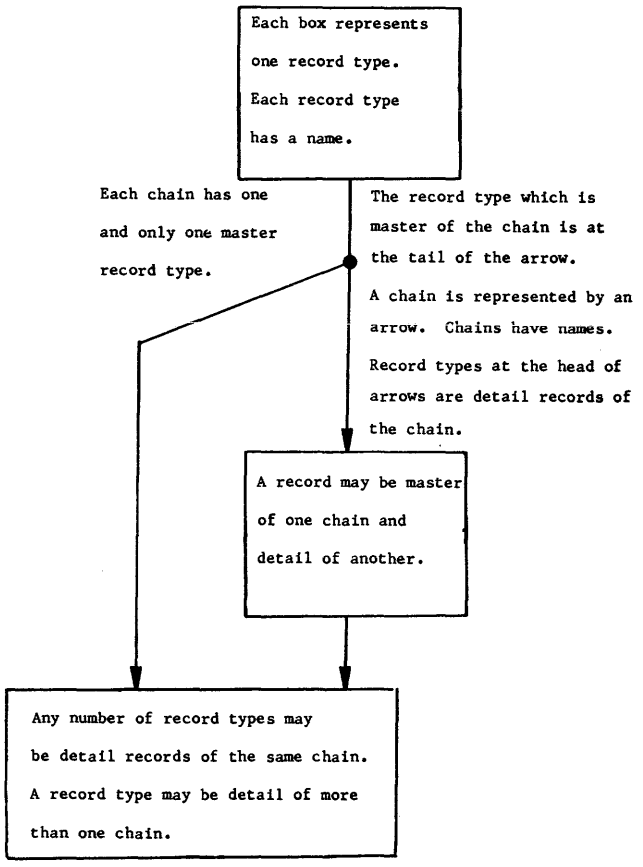


Figure 10 – IDS diagram conventions

Integrated Data Store schematic diagram conventions are used in figures except where noted. A brief review of IDS Diagram Conventions is given in Figure 10.

REFERENCES

1 INFORMATION SYSTEMS DIVISION GENERAL ELECTRIC COMPANY
Integrated Data Store AS-CPB-483A Rev 7-67

GIPSY–A generalized information processing system

by GIAMPAOLO DEL BIGIO
International Atomic Energy Agency
Vienna, Austria

INTRODUCTION

The problem of mechanized documentation at the International Atomic Energy Agency was first approached by using the IBM KWIC system, which proved, however, to be insufficient for most of the applications which were envisaged.

Consequently, it was decided that a new system, GIPSY, should be developed which had the following basic characteristics:

- –the input material must be entered in such a way that it can be unambiguously recognized, i.e., retrieved, by the computer;
- –the system must be able to store and retrieve the information, and display it in a form which is not fixed a priori, but which can vary according to the user's needs;
- –the system must provide for such additional functions as sorting, duplication check and file maintenance.

From the consideration of these points it became clear that the system had to comprehend two subsystems:

1) the input subsystem which permits consistent cataloging and encoding of the material to be processed; and
2) the machine subsystem which processes the material by means of the computer.

The input subsystem has been discussed in detail elsewhere,[1] while the description of the machine subsystem is the purpose of this paper. However, since the two subsystems cannot be described completely independently, a short description of the first is given in the next section.

Input organization

The input to the system consists of the bibliographic descriptions of documents, or input "units." An input "unit" consists of a set of bibliographical data, or "data elements," e.g., author, title, publisher, etc.

Some data elements are composed of several subelements, e.g. multiple authors. These are identified by special marks in the body of the data element. Where needed a fixed structure, according to normal documentation practice, has been defined, using key-words such as VOL., NO., P., etc. and/or punctuation signs.

This permits the program to check the formal correctness of input and give specific error messages where the defined input rules are not followed. It also permits the enforcement of consistency in input preparation, which ultimately means consistency in the output products.

Each "data element" is given a 4-position code, the B-code, as follows:

First position:	Class and subclass code
Second position:	Data type code
Third position:	Data subtype code
Fourth position:	Language code.

The concepts of class, subclass, type and subtype are described below. We have identified, in GIPSY, three types or "classes" of documents:

1) Original: the publication whose bibliographic description is being entered into the system;
2) Source: the publication in which the "original" document was documented, e.g. an abstract journal;
3) Citation: a document cited by the "original" document.

In addition, a document of any of the three classes above may be independent, e.g., a technical report, or it may be part of some larger work, e.g., a journal article, a book in a series, etc. We have therefore introduced subclasses to indicate a dependency relationship: the first subclass pertains to the main document, the others to the related document.

Consequently a bibliographic description of a document is the description of the document itself *and* the description of any other related document which may exist.

For example, the bibliographic description of a journal article found in an abstract journal requires

the description of the following three documents:

1) the article itself (class original, subclass main document);
2) the journal (class original, subclass related document); and
3) the abstract journal (class source, subclass main document).

A document of any class is described by giving the pertinent bibliographical data in a predetermined sequence.

The code for a data element is a two-level code, the first level representing the general type of data (e.g., author, title, etc.), the second level a specific type (e.g., personal author, corporate author, main title, subtitle, etc.).

For some data elements an additional identifier may be needed: the language in which the information is entered.

A typical example of a B-code is the following:

A	2	0	E
↓	↓	↓	↓
class: original	type:	subtype:	language:
subclass: main	title	main title	English
document			

Since data type and subtype apply to any class of document, assuming the above code represents a title of a journal article, the title of the parent journal could have the following B-code:

C	2	0	F
↓	↓	↓	↓
class: original	type:	subtype:	language:
subclass: related	title	main title	French
document (journal)			

The actual B-codes used in the GIPSY system are given in Appendix 1.

Overall system description

System organization

The machine subsystem consists of a set of nine special purpose programs, a monitor, a generalized sort program, and a system maintenance program.

The approach taken in the design was to construct a modular system in such a way that those features which were important but might not be needed to produce a given output may vary according to the particular job.

The execution time of individual programs may also vary greatly according to the features selected.

Special purpose programs

The nine special purpose programs may be subdivided into four groups as described below. (The information flow throughout the system is shown in Figures 1, 2, 3, 4, and 5.)

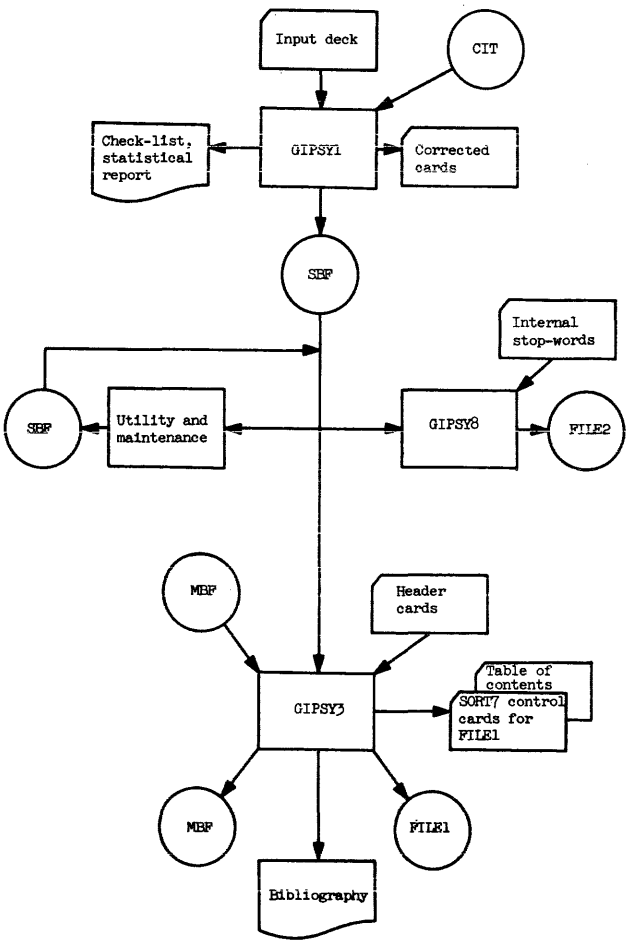


Figure 1 – Major programs: input checking, printing of bibliography, generation of index files

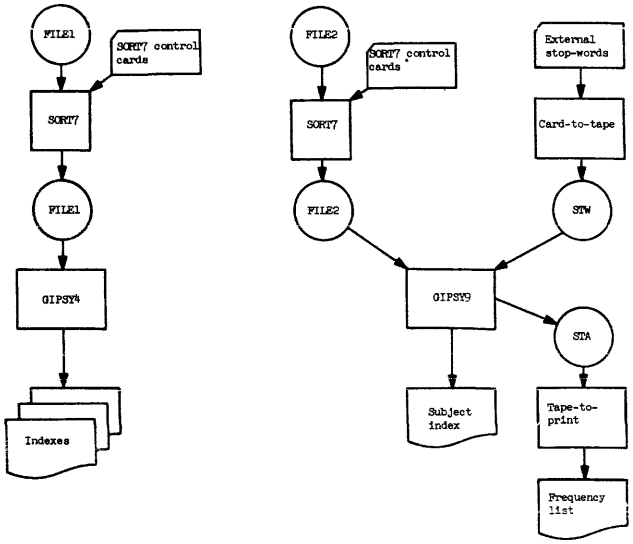


Figure 2 – Major programs: printing of indexes

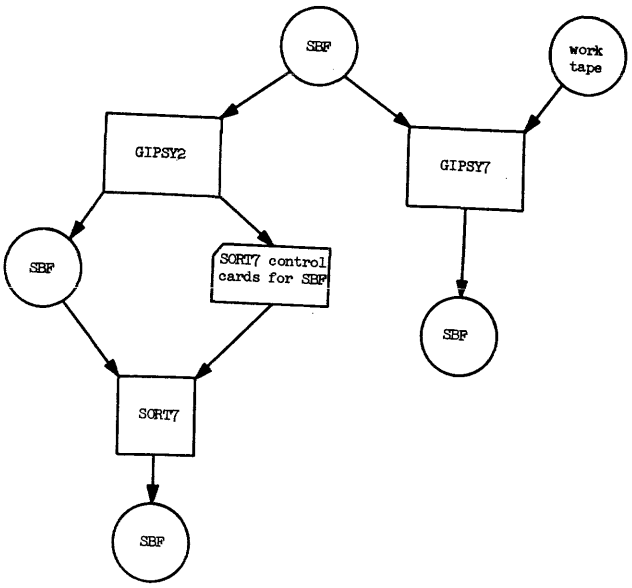


Figure 3 – Utility programs

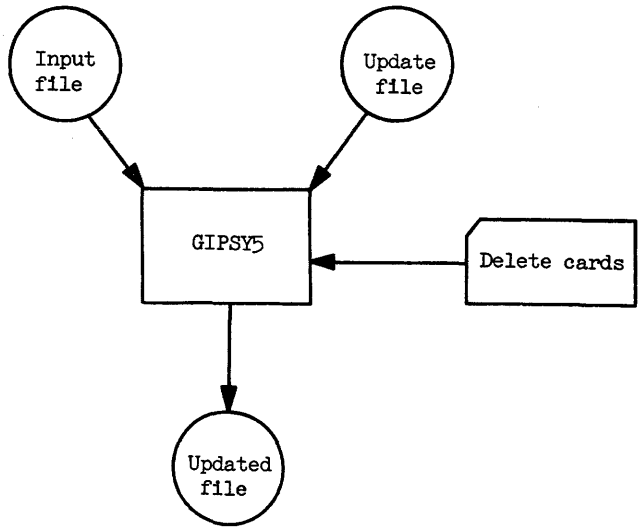


Figure 4 – File maintenance programs

1) Index 1: composed of the index entry and the document numbers of the relevant citation in the printed bibliography.
2) Index 2: composed of the index entry, a portion of the input unit and the document number of the relevant citation in the printed bibliography. A special case of Index 2 is a KWIC (*K*ey-*W*ord-*I*n-*C*ontext) index.

d) Catalog cards.

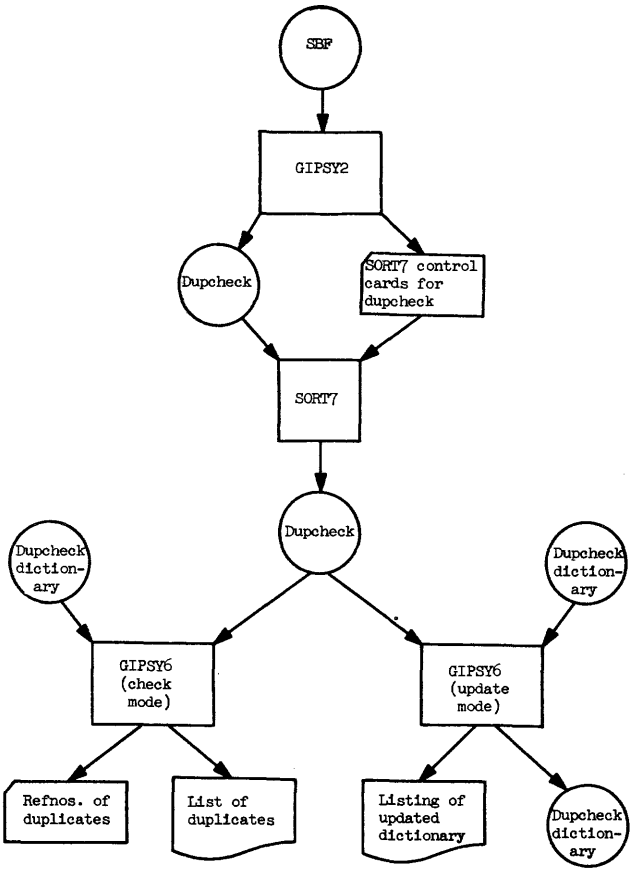


Figure 5 – Duplication check programs

Major programs

These permit the production of:

a) A check-list of the input material for proof-reading, which includes program-generated error messages, concerning formal input errors such as sequence, punctuation, coding, missing data elements, etc.
b) A printed bibliography, which may be classified according to a user-supplied subject classification scheme. Provision for cross references is also made.
c) Two types of indexes:

Utility programs

These allow the user to change the order of the units within a file, and/or of data elements within a unit.

Since, for checking purposes, data elements within an input unit must be entered in ascending order by the respective B-code, a utility program is provided to change the printing sequence of these data elements. This feature is particularly useful when producing catalog cards, where the first data element is normally the one by which the cards are filed (e.g. title, author, etc.).

Duplication check

These programs permit the automatic generation of a duplication check code, for each input document, and the construction and updating of a dictionary file containing duplication check codes of all documents processed by the system.

They provide for checking the duplication check codes of the new material against the duplication check dictionary to ensure that there are no duplicated documents.

File maintenance

A generalized file maintenance program permits the user to change the contents of a GIPSY file either by replacement, deletion or insertion of items. The possibility of extracting selective items from any one file is also provided.

Monitor and system maintenance

The GIPSY system is tape-resident and therefore a monitor is provided to call the desired program for execution. Since each program returns control to the monitor after completion of its functions it is possible to "stack" several jobs in a single run. To reduce operator intervention to a minimum the system permits the user to allocate tape units dynamically by means of control cards.

The system maintenance program provides for updating the GIPSY system tape, by adding new programs, replacing entire programs or parts, or deleting obsolete programs.

Generalized sort

The sort program utilized by GIPSY is the standard IBM SORT 7 for the 1401. Whenever needed the relevant GIPSY program automatically generates SORT 7 control cards.

System control

Because of the large number of features provided by the system, the operation of every GIPSY program is controlled by one or more control cards, as explained below.

System control cards

These permit the selection of the desired program from the system tape, allocate tape units and print messages for the machine operator.

Standard GIPSY control card

This card is present for every program, has a fixed format, and indicates items such as page size, record length, presence or absence of an optional file, spacing, etc.

Control statements

These cards specify options applicable to individual data elements, e.g., indentation, sorting, etc. Control statements have a free format.

Description of individual programs

Major programs

GIPSY1

GIPSY1 is the input and editing program. It accepts as input either a punched card or a tape file (CIT). B-codes within a bibliographical unit are checked for ascending sequence and validity. The ascending sequence of document numbers can also be checked. For those data types for which a structure is defined, this is also checked.

Outputs from GIPSY1 are a check-list of the input data with diagnostic messages, a statistical report and a tape file called Standard Bibliographic File (SBF), containing the material to be processed by the system. GIPSY1 also provides for on-line error recovery.

GIPSY3

The functions performed by GIPSY3 are the following:

a) Produce a ready-to-publish listing of the information stored on the SBF;
b) Add this information to a Master Bibliographic File (MBF);
c) Generate an index file (FILE1)

The powerful report generator included in GIPSY3 provides for several features, the most important of which are:

– variable page size;
– variable spacing between units;
– suppression of the printing of selected data elements (by using this feature one can print, for example, a title list);
– indentation of selected data elements;
– overprint (bold-face) of selected data elements;
– spacing before printing selected data elements.

In addition, provided that the bibliographic units have been assigned a subject identification code, subject headings, subheadings and cross references can be printed by using a special card file (header cards).

During or independently of the printing of the SBF it is possible to generate another file containing selected data elements (FILE1). The purpose of FILE1 is to obtain printed indexes such as author, corporate author and report number indexes.

The ability to generate several indexes at one time permits considerable saving of machine time and set-up time; only one sort and one printing pass are necessary.

When a report number index is requested, this is generated in such a way that the report numbers will be sorted in the correct alphanumeric sequence, e.g., TID-47 before TID-100. This is done automatically by the program and does not require manual editing of the report number when punching.

The information processed can be stored on a master file (MBF) which can be subsequently used to produce, for example, cumulative indexes.

GIPSY4

GIPSY4 prints the contents of a sorted index file (FILE1) generated by GIPSY3. The main features are the ability to print directly on two columns and to print several indexes in different formats. The report generator includes, where applicable, the same features as in GIPSY3. By using header cards a fixed title can be printed at the top of every page.

Since for every index a new set of control cards is required, this makes it possible to change the printing format for every index.

GIPSY8

GIPSY8 accepts as input an SBF and generates an index file (FILE2). Any data elements may be selected as the index entry (author, corporate author, keywords, etc.), and any data element may be selected as additional information (e.g., an author index showing the title as additional information). GIPSY8 can also generate a subject index by extracting keywords from selected data elements, e.g., title, abstract. In this case either KWIC (*K*ey *W*ord *I*n *C*ontext) or KWOC (*K*ey*W*ord *O*ut of *C*ontext) indexes can be generated. When generating such a subject index there are certain words which should *not* be selected, e.g., prepositions or articles. These are referred to as stopwords. There are 15 stopwords which, according to our own and others' experience constitute about 25% of the total number of words in a given file. These are the following:

a, of, in, on, by, to, as, at, an, the, and, for, with, from, some. There are certainly other words which appear quite frequently and which could be included in the list. It was felt, however, that the user should decide on these words rather than the system designer, since in fact most of them depend on the specific application. The approach taken in GIPSY8 is to provide a built-in stopword table containing the words given above, but let the user be free to provide his own list, which can contain about 300 words maximum. This also permits the use of stopwords in different languages. The stopword lists currently used at the International Atomic Energy Agency show that about 50% of the words in a given file can be eliminated.

It should be noted, at this point, that the stopword list of GIPSY8 does not attempt to produce the final version of the index (this function is accomplished by the external stopwords in GIPSY9) but rather to reduce the sorting time of the index file as much as possible. Another important feature of GIPSY8 is to let the user decide which characters are to be taken as part of the selected word.

GIPSY9

GIPSY9 prints the contents of a sorted FILE2 generated by GIPSY8. The report generator in GIPSY9 provides for a large number of printing features.

In addition to the index file itself, GIPSY9 accepts a stopword file, referred to as the external stopword file. This is the counterpart of the stopword list of GIPSY8.

GIPSY9 can be conditioned to print only those words which are on the external stopword file; in this case the file acts as a dictionary of meaningful words.

To improve the consistency and usefulness of the index, the external stopword file provides for the printing of cross references and for the possibility of printing certain words only for selected documents. This feature is particularly useful for handling those words which have the same spelling but different meanings.

Utility programs

GIPSY2

The main feature of GIPSY2 is the automatic generation of sort fields. The SBF is a variable-length record file, and the length of different data elements is variable within the record itself. The existing sort programs, on the other hand, require that the sorting keys are always in a fixed position of the record. The automatic generation of sort fields permits sorting of the SBF by any desired data elements: this is done by extracting information from the selected data elements and inserting it in a fixed position of the output SBF.

When the contents of the SBF are written on the master file (MBF) by GIPSY3 the generated sort fields are deleted.

In the case of periodic announcement lists, the bibliographic citations are normally assigned a sequential ascending document number, which, if the bibliography is sorted by the computer, is not known at the time of input preparation. A feature of GIPSY2 allows the user to assign such a document number after the file has been sorted. (Other features of GIPSY2 are described in a later section.

GIPSY7

This program permits the arrangement of data elements within a printed bibliographical unit in an order other than the ascending sequence of the B-codes, as required by GIPSY1. GIPSY7 is normally used before GIPSY3.

Duplication check

GIPSY2

Besides the features described earlier, GIPSY2 may also be used to generate a 13-character duplication check code as follows:

NNNNIIYYTTTTT

where: NNNN are the first 4 characters of the name of the first personal author (or editor);
II are the initials of the above; if neither an author nor an editor was given then NNNNII are the first 6 characters of the corporate author;
YY is the year of publication;
TTTTT are the first characters of the first 5 words of the title.

If any item, or part of it, is missing the corresponding positions of the dupcheck code contain full-stops (.). The duplication check code, the corresponding document number, and some additional information are written on the dupcheck file. The additional information is selected automatically by the program according to the particular type of document, e.g., report number(s) for technical reports, subtitles for books, journal citation for journal articles, etc. This need was felt after some experience with a previous version of the system, in which the additional information was not present.

In fact, in some cases (in particular for progress reports), the dupcheck code generated by the program was the same for different documents. This required visual scanning of the full citation in order to be sure that two documents with the same duplication check code *were* actually the same document. Since the additional information was put on the duplication check file this scanning is practically no longer required.

GIPSY6

GIPSY6 accepts as input a dupcheck file generated by GIPSY2 and a dupcheck dictionary which contains the dupcheck codes of all documents previously processed by the system.

GIPSY6 may be used in either of two modes: the check mode, in which the codes on the dupcheck file are compared with the codes in the dictionary and each time they match a message is printed showing the information on the file for those documents; or the update mode, in which the codes on the dupcheck file are added to the dictionary file. When operating in the update mode the program does not check for duplication, thus assuming that possible duplicate codes represent different documents.

A print-out of the updated dictionary can also be requested by the user.

Implementation

The GIPSY system has been implemented for an IBM-1401 with 12000 storage positions, high-low-equal compare and advanced programming features (the space suppression feature, 4000 additional core positions and the 1407 console typewriter may be used if installed), 5 magnetic tape units, card reader and punch unit.

It has been used since 1965 for the preparation of several nonperiodical and two periodical publications of the IAEA, the "List of References on Nuclear Energy," an announcement list with annual personal, corporate author and report number indexes, and "Nuclear Medicine, A Guide to Recent Literature," also an announcement list, each issue of which includes author index, an isotope index and a KWIC index. The total number of documents processed and sorted to date is about 50,000.

In addition to the above, GIPSY has been used with success to produce other types of publications which were not strictly documentation-oriented.

SUMMARY

GIPSY is a generalized system to process bibliographic information.

The input to the system describes the material which is being documented. The bibliographic components are identified by means of a code (B-code), permitting access to any of those components which may have an independent documentary value (e.g., authors for author index, report numbers for report number index, etc.).

The information can then be processed to obtain printed reports: bibliographies, classified if desired, with or without abstracts; and various indexes, including subject indexes (KWIC). A duplication check procedure is also included in the system.

The aim of the system is to produce ready-to-publish copy, and therefore various editing and format features are provided. The information processed can be stored for subsequent use, e.g. production of cumulative indexes or retrieval.

CONCLUSION

After two years of practical experience with the GIPSY system it can be said that the objectives out-

lined in the introduction of this paper have been fully met and that in fact the system has proved to be general enough to handle applications which were not envisaged at the time it was designed.

Another important factor has been that through the use of the system we have been forced to be much more consistent in the application of descriptive cataloguing rules than we were during manual operation.

We also feel that the approach taken is a valid contribution to the creation of a generalized information processing system in which the system adapts itself to the user's needs, rather than the user to the system. The need for this characteristic becomes more and more evident with the information explosion of recent years and with the increasing number of organizations faced with the problem of mechanized documentation.

ACKNOWLEDGMENT

The author is indebted to the staff of the INIS Section of the IAEA and in particular to Mrs. J. Robinson and Mr. T. W. Scott for their suggestions during the development of the system. He is also grateful to Dr. F. Lang, IBM Austria, for the assistance given during the inception of the GIPSY system.

REFERENCES

1 T W SCOTT F LANG
Coding and structuring input data for the GIPSY system
Proceedings of the American Documentation Institute 1967
2 G DEL BIGIO
GIPSY program manual
International Atomic Energy Agency 1967

APPENDIX 1

GIPSY Coding Matrix

The matrix given in Figure 6 shows data types and subtypes, with their appropriate codes, running from top to bottom along the left side of the figure. All of the respective subclass codes form a sequential row running from left to right across the top of the figure.

The sequence of data elements is that normally followed in documenting an entry for bibliographic use. In the GIPSY system, this sequence must be adhered to in entering data, but may be subsequently changed if desired. Similarly, the sequencing of classes and subclasses follows normal bibliographic practice in describing an entry.

Within the matrix, an x is shown where a given data element is normally entered for the particular subclass column. An (x) is shown for data elements which may be present or which are optional. In the fourth position of the B-code column, language code, the word "yes" appears for those data elements which logically can have and normally will have a distinction according to the language in which they appear.

APPENDIX 2

An example of some features of GIPSY

Figures 7-15 show some printed outputs of the system. We have taken a sample bibliographic description and followed it throughout most of the GIPSY programs.

GIPSY1: Figure 7 shows how the input unit is printed on the check list.
Spacing is inserted by the program to improve legibility.
The sample unit contains the following data elements:

A00:	document identification. This card gives information such as subject category code, type of document code (R = report)
A10:	personal authors
A16:	corporate author
A20E:	English title
A30:	report number
A41:	collation
960E:	English keywords

Since the > sign, which is used in GIPSY to separate subelements within a data element and to terminate a data element, is a non-print character on the 1403 printer, it is replaced by the program by a record mark (‡) to simplify proofreading.

The input unit in Figure 7 contains some mistakes indicated by GIPSY1 as follows:

PUNCTN:	a full-stop, not a comma, must separate initials of names.
LAST UNFLAGGED SUBTYPE IS INCOMPLETE:	the title (A20E) does not end with a > sign.
MISSNG:	the report number (A30) is missing. Since the document is a report, the report number is a required data element.
DATA:	B is not a valid element of the collation (A41). This should have been P.

Figure 8 shows the same input unit after the indicated errors have been corrected.

GIPSY7: Figure 9 is an example of a catalog card. The filing entry in this case is the keyword (960E), which appears first. The order of data elements has been changed by GIPSY 7. The card was printed by GIPSY3.

GIPSY CODING MATRIX — DEFINED ITEM	B - CODE: SUB CLASS	B - CODE: TYPE	B - CODE: SUB TYPE	B - CODE: LANG	CLASS = ORIGINAL: DOCUMENT OR PART ENTERED	"IN" AN IND. PUBL'N	"IN" A JOURNAL	"IN A SERIES	PROC.	CLASS = SOURCE: INDEPENDENT PUBLICATION OR PART	"IN" AN IND. PUBL'N	"IN" A JOURNAL	"IN" A SERIES	PROC.	CLASS = CITATION: INDEPENDENT PUBLICATION OR PART	"IN" AN IND. PUBL'N	"IN" A JOURNAL	"IN" A SERIES	PROC.
					A	B	C	D	I	J	K	L	M	R	1	2	3	4	9
0 IDENTIFICATION/SYSTEM UTILITY		0																	
Identification (categories, species, etc.)			0	YES	X					X					X				
Utility (sortfield)			1-9		(X)														
1 AUTHORS		1																	
Personal author(s)			0		(X)					(X)					(X)				
Editor(s)			1		(X)	(X)	(X)	(X)		(X)	(X)	(X)	(X)						
Translator(s)			2		(X)					(X)									
Common affiliation			3		(X)	(X)	(X)	(X)		(X)									
Corporate author(s)			6		(X)					(X)					(X)				
Notes			7		(X)	(X)	(X)	(X)		(X)	(X)	(X)	(X)						
Secondary corporate author(s)			8		(X)					(X)									
2 TITLES		2																	
Journal or document title			0	YES	X	X	X	X		X	X	X	X		(X)	(X)	(X)	(X)	
Subtitle			1	YES	(X)	(X)	(X)	(X)											
Addenda to title			2	YES	(X)	(X)	(X)	(X)											
3 IDENTIFYING NUMBERS		3																	
Report number(s), primary			0		(X)	(X)				(X)					(X)	(X)			
Patent number(s)			1		(X)										(X)				
Specification number(s)			2		(X)	(X)									(X)				
Report number(s), secondary			9		(X)	(X)													
4 DESCRIPTION		4																	
Place of publ'n, publisher, date			0		(X)	X		X		X	X		X		(X)	(X)		(X)	
Physical description			1		(X)	X				X	X				(X)	(X)			
Journal description			2				X					X					(X)		
"Inclusive" pagination (book)			3			X					X					(X)			
Notes			9		(X)	(X)	(X)	(X)		(X)	(X)	(X)	(X)						
5 NOTES OR AVAILABILITY		5	0						(X)					(X)					(X)
6 KEYWORDS OR KEYTERMS		6	0						(X)					(X)					(X)
7 ABSTRACT/ABSTRACTOR(s)																			
Abstract		7	0	YES					(X)					(X)					(X)
Abstractor(s)		9							(X)					(X)					(X)

Figure 6—GIPSY coding matrix

```
51542        A 00 E  001                    0205  R
51542        A 10    001   PETRUKHIN V.J.,♦PONOMAREV L.I.,♦PROKOSHKIN YU.D. ♦        PUNCTN
               16    001   JOINT INST. FOR NUCLEAR RESEARCH, DUBNA (USSR). LAB. OF HIGH
                     002     ENERGY. ♦
             20 E    001   ON A POSSIBLE APPLICATION OF A NUCLEAR REACTION IN CHEMICAL
LAST UNFLAGGED SUBTYPE IS INCOMPLETE
51542        A 30
               40    001   1966.♦                                                   MISSNG
51542        A 41    001   5 P. ♦
             9 60 E  001   NUCLEAR REACTIONS,♦USES,♦CHEMICAL ANALYSIS,♦HYDROGEN,♦PIONS,   DATA
                E    002   ♦PROTONS,♦RESEARCH.♦
```

Figure 7—GIPSY1 printout with error messages

```
51542        A 00    001                    0205  R
               10    001   PETRUKHIN V.J.,♦PONOMAREV L.I.,♦PROKOSHKIN YU.D. ♦
51542        A 16                                                                   MISSNG
             20 E    001   ON A POSSIBLE APPLICATION OF A NUCLEAR REACTION IN CHEMICAL
                E    002   RESEARCH. ♦
               30    001   JINR-P-2558.♦
               40    001   1966.♦
               41    001   5 P. ♦
             9 60 E  001   NUCLEAR REACTIONS,♦USES,♦CHEMICAL ANALYSIS,♦HYDROGEN,♦PIONS,
                E    002   ♦PROTONS,♦RESEARCH.♦
```

Figure 8—GIPSY1 printout, corrected

```
CHEMICAL ANALYSIS, HYDROGEN, PIONS, PROTONS,
   RESEARCH, NUCLEAR REACTIONS, USES.

PETRUKHIN V.J., PONOMAREV L.I., PROKOSHKIN
   YU.D.

   ON A POSSIBLE APPLICATION OF A NUCLEAR
REACTION IN CHEMICAL RESEARCH.

JINR-P-2558. 1966. 5 P.                                    (51542)
```

Figure 9—GIPSY3: catalog cards

GIPSY2: Figure 10 shows the document number reassignment and the dupcheck code generation features of GIPSY2. The input unit 51542 is assigned the document number 00044 and the dupcheck code PETRVJ-660APAO.

```
00041        51512        OGIEVI66EFISA    OGIEVI66EFISA    02    05
00042        51760        OSHEVI64BSAHE    OSHEVI64BSAHE    02    05
00043        51447        PETKI.65OTAOI    PETKI.65OTAOI    02    05
00044        51542        PETRVJ66OAPAO    PETRVJ66OAPAO    02    05
00045        51514        PONOLI66OTTOT    PONOLI66OTTOT    02    05
00046        51445        ROTTI.65FPCFA    ROTTI.65FPCFA    02    05
00047        51543        SHIRDV64PIUT1    SHIRDV64PIUT1    02    05
```

Figure 10—GIPSY2: assignment of sequential reference numbers

GIPSY3: Figure 11 shows the unit as it could be printed by GIPSY3 in a bibliography. Spacing, indentation and overprinting are controlled by the use of control cards. Note that the A00 data element has been prevented from printing by means of a control card.

```
43  PETKOV I.
    JOINT INST. FOR NUCLEAR RESEARCH, DUBNA (USSR). LAB. OF HIGH
    ENERGY.
      ON THE AMPLITUDE OF INELASTIC SCATTERING OF FAST PARTICLES ON
    NUCLEI.
    JINR-P-2037. 1965. 8 P.

          SCATTERING, PARTICLES, NUCLEI, ENERGY, EXCITATION,
          ELECTRONS, CROSS SECTIONS.


44  PETRUKHIN V.J., PONOMAREV L.I., PROKOSHKIN YU.D.
    JOINT INST. FOR NUCLEAR RESEARCH, DUBNA (USSR). LAB. OF HIGH
    ENERGY.
      ON A POSSIBLE APPLICATION OF A NUCLEAR REACTION IN CHEMICAL
    RESEARCH.
    JINR-P-2558. 1966. 5 P.

          NUCLEAR REACTIONS, USES, CHEMICAL ANALYSIS, HYDROGEN, PIONS,
          PROTONS, RESEARCH.


45  PONOMAREV L.I.
    JOINT INST. FOR NUCLEAR RESEARCH, DUBNA (USSR). LAB. OF HIGH
    ENERGY.
      ON THE THEORY OF THE ASYMPTOTIC REPRESENTATION OF SPHEROIDAL
    FUNCTIONS.
    JINR-P-2564. 1966. 6 P.

          EQUATIONS, MATHEMATICS, METHODS.
```

Figure 11—GIPSY3: printing a reference, with indentation and overprinting

GIPSY4: Figures 12 and 13 show the author and the keyword indexes, generated by GIPSY3 and printed by GIPSY4. Both indexes are generated and printed in one pass.

```
BARASHENKOV V.S.   8
BELYAEV V.B.   6
BILEN'KII S.M.   9
BILENIKAYA S.I.   10
BIRYUKOV V.A.   11
BLANK I.   12

DANILOV V.I.   13
DAO WONG DUC   14
DESIMIROV G.   15
DEUTSCH M.   16

PERELYGIN V.P.   32
PETKOV I.   43
PETRUKHIN V.J.   44
PISAREV A.F.   11
PODGORETSKII M.I.   36
POLUBARINOV I.V.   40,41
PONOMAREV L.I.   44,45
POPOVA A.B.   25
PROKOSHKIN YU.D.   44
PYATOV N.I.   19
```

Figure 12—GIPSY4: author index

```
CHANNELS
   13
CHARGED PARTICLES
   28
CHEMICAL ANALYSIS
   44
CLOUD CHAMBERS
   1
COINCIDENCE METHOD
   21
ELEMENTARY PARTICLES
   8,56
EMISSION
   6
ENERGY
   3,21,43,47,50,55
ENERGY LEVELS
   3,4,5,7,8,10,16,18,21,25,26,35,
   42,47,51,56,57
EQUATIONS
```

Figure 13—GIPSY4: keyword index

GIPSY9: Figure 14 shows the same index as figure 13 but with titles. Figure 15 is an example of a KWIC index from titles. Both these indexes were generated by GIPSY8 and printed by GIPSY9.

```
CHARGED PARTICLES
   THE ISOTOPIC DISCHARGE CHAMBER WITH HYDROGEN AND HELIUM FILLING.      28

CHEMICAL ANALYSIS
   ON A POSSIBLE APPLICATION OF A NUCLEAR REACTION IN CHEMICAL
   RESEARCH.                                                             44

CLOUD CHAMBERS
   SEARCH FOR NEW DECAY MODES OF THE K-ZERO-TWO MESONS.                   1
```

Figure 14—GIPSY9: KWOC index

```
ING.=  THE ISOTOPIC DISCHARGE CHAMBER WITH HYDROGEN AND HELIUM FILL      28
 TRACK COORDINATES IN A SPARK CHAMBER.=     +FOR EVALUATING PARTICLE     11
OTRON BEAM.=          MAGNETIC CHANNEL FOR FOCUSING A DEFLECTED CYCL     13
TION OF A NUCLEAR REACTION IN CHEMICAL RESEARCH.= +POSSIBLE APPLICA      44
  SHELL MODEL EXCITATIONS AND CLUSTER EXCITATIONS IN LIGHT NUCLEI.=       7
       FRACTIONAL PARENTAGE COEFFICIENTS FROM ALPHA-PARTICLES.=          46
ED+      WAVE FUNCTIONS OF THE COLLECTIVE STATES OF EVEN-EVEN DEFORM     57
```

Figure 15—GIPSY9: KWIC index

Figure 16 is a listing of the control cards used to produce the sample cases. This listing has been included so that the reader can get a "feeling" of how a job is described to GIPSY.

```
*CALL    GIPSY1
*1*      102 JINR                                    130367 JINR DEMO RERUN

*CALL    GIPSY7
*7*1000
*MOVE 960E TO A1

*CALL    GIPSY3
*3* PJINR 21040                  1  1       014001048P12001
*EDIT =(= (5,9) =)=
*DELETE A0, A16
*INDENT L 960E, A10. T A20E
*SKIP 1 A10, A20, A30

*CALL    GIPSY2
*ASSIGN  SBFINP TO 3
*ASSIGN  SBFOUT TO 4
*2* S00001       05000011060  1 1
*SORT-FIELDS A10-V (30), A40-V (6)

*CALL    SORT7
45 62 010559001001P111 11 5020210062015                  100040094
0077006
L

*CALL    GIPSY3
*ASSIGN  SBFINP TO 4
*3* PJINR 2 001                3A3001037065008067210610504     132
*EDIT (5,9,5)
*DELETE A0
*OVERPRINT A20E
*INDENT 0 A10, A16. 5 A20. 0 A30. 6 960E
*SKIP 1 960E
*FILE1 A10,A30,960E

*CALL    MONITOR
OPERATOR PLEASE REPLACE TAPES ON UNITS 2, 3 AND 4 WITH THE FOLLOWING TAPES ==
FILE1 ON UNIT3
FILE2 - KWIC ON UNIT4
FILE2 - KWOC ON UNIT5
SCRATCH ON UNIT6
*PAUSE

*CALL    GIPSY4
*4*A10 2000110650350371100l 03531  3  01
H3 A10                                   AUTHOR INDEX
*EDIT (5,9,P)
*4*A30 2    1065035037110O7 0352       012
H3 A30                                   REPORT NO. INDEX
H  A30   REPORT NO.                 REF. NO.    REPORT NO.                  REF.1
H  A30     NO.
H2 A30   ------ ---                 ---- ---   ------ ---                  ----1
H2 A30     ---
*4*960E2    1065035037100O0 03521 13  01
H3 960E                                  KEYWORD INDEX

*CALL    GIPSY8
*8*1 080       01960EA20E
*8*5 11. +$ -,=   03*(-03*)-
*CALL    SORT7
52 34 010500001001P111 11 5020340080021                  100040170
0017013
L

*CALL    GIPSY9
*ASSIGN  FILE2  TO 4
*9* 1                        A3001038065075070001
*EDIT (5,9,P)

*CALL    GIPSY8
*8*1108003106? 01A20E
*8*5 11. +$ -,=   03*(-03*)-

*CALL    SORT7
52 34 0102380010O1P111 11 5021170080080
0201037
L

*CALL    GIPSY9
*ASSIGN  FILE2  TO 5
*9*  1001                   A3001034065075066004?
*EDIT (5,9,P)
```

Figure 16—GIPSY control cards used to prepare Figures 7 through 15

The ISCOR real-time industrial data processing system

by W. M. LAMBERT
Control Data Corporation
Vanderbijlpark, South Africa
and
W. R. RUFFELS
South African Iron and Steel Corporation (ISCOR)
Vanderbijlpark, South Africa

A real-time industrial data processing system collects information, processes it, and responds to the user in sufficient time to influence or control the production processes, material distribution and accounting records.

At ISCOR (South African Iron and Steel Corportation) a comprehensive real-time industrial data processing system is being installed. This system will utilize 'in-plant' communications terminals and automatic data logging units to gather and display data. It will respond to remote terminals and the Mills process control computers to initiate and control production flow through the manufacturing process. Batch processing jobs will also be initiated through the real-time system to provide production reports and accounting records.

In view of the number of production centres involved it was decided to divide the system into two major application areas, namely customer order control and production control. It was also decided that the customer order control application would be installed on a corporate wide basis, while the production control application would be further subdivided into works and mills areas.

This paper will discuss the approach to the development of the system and its installation at the ISCOR Works, Vanderbijlpark, South Africa.

The computer system to be employed consists of two large Control Data 3300 Central Processors located at the Vanderbijlpark Works. Each processor will have its own operating system. One system will be primarily dedicated to the real-time programs while the other handles batch processing programs and also serves as a standby system for real-time programs in the event of failure of the real-time system. Furthermore process control computers will also be installed to control the actual processing of material on the mills.

The real-time system will capture and process data at the actual time the steel is passing through the different phases of production while the batch processing will be carried out according to a preset schedule. All commercial applications and reports emanating from the data processed by the real-time system will be processed into reports periodically in the batch processing system. The real-time processor is also capable of processing batch jobs whenever its real time load permits.

Since the real-time operations require a high degree of processor, peripheral, and terminal reliability, each system will monitor the condition of the other. Should the real-time processor not be functioning properly the batch processor will abort all its jobs and assume the real-time load. In order to accomplish this interchangeability of work load, the system is configurated so that each processor may access all peripherals and terminals via alternate data input/output routes.

The failure recovery procedures to be employed, in the event of the 'in-plant' terminals or real-time system peripherals not being in operation, have been carefully laid out in order that the highest degree of back up possible is obtained with a minimum of manual operation or recording. For example, when a message is transmitted to the central processor from a mill process control computer or data logger, the

central processor must acknowledge the receipt of the message within five seconds. If the acknowledgment is not received by the sending unit, the message is transmitted again. Should the second message not be acknowledged, the transmitting unit assumes there has been a failure within the communications network and will automatically dump the data into punched paper tape for later entry via a tape reader in the computer centre.

Remote 'in-plant' terminals are being designed and located in such a way that a terminal situated in close proximity, can be used should a unit be out of operation. Maintenance of the 'in-plant' terminals units will generally follow the policy of replacing a unit with a spare rather than to repair it on site.

In order to create a smooth running system, essential 'on-line' files, which cannot be re-established quickly and efficiently, are maintained in duplicate on separate disc units each of which is accessed through different channels and controllers. 'On-line' files are available to the real-time system at all times, usually without requiring computer operator action.

Overall the dual system approach is the one followed, with at least two routes provided from the processor for each type of input or output.

The order control system provides for the direct entry of orders into the computer system from the various sales offices via papertape transmission and teleprinter units. The system receives the order and accepts it commercially and industrially and confirms the order status with the initiating sales office.

Acceptance of the order includes validating it from the standpoint that all of the data required to process the order are present and correct. Such items as customer data and product specifications are also vetted by the system. The order is then priced and the customer credit arrangements are checked. Assuming that the order is correct and complete and that the product specifications are acceptable, the order is forward loaded on a mathematical model of the works and a forecast delivery period for the material is computed. If the delivery period arrived at is not within the limits specified by the customer, the sales organisation is notified and may adjust priorities if it so desires. Should the sales organisation change the priority, the order will be reloaded on the mathematical model. In the event of the order failing a specific test, a query is generated and the appropriate section is notified automatically. For example, if order data is missing or invalid, the originating sales office is advised. When product specifications are new or unusual, the works metallurgist is notified. Should credit arrangements be inadequate, the credit department is called. In order that all computer-generated queries receive prompt attention, and that no information is lost, each query is followed up with a reminder if the response has not been received within a set time limit.

Once the order has been accepted, and forward loaded on the plant, it remains in the system until its entire life cycle is complete. An order's life begins with its entry by sales, matures when it has been dispatched by the works, and dies when it has been paid by the customer. After that, it becomes another statistic for use in the operations research or sales forecast system.

Periodically, blocks of forward loaded orders are passed from the order control system to the production control system. The production control system then schedules these forward loaded orders on the works production units, and reports their progress through the units until the order is complete and ready for dispatch.

Blocks of orders are transferred from the order control system at intervals which will maintain a sufficient pool of orders to provide an adequate product mix for scheduling. The system schedules the orders based on existing conditions within the mill, and issues recommended schedules in advance for approval by the production planner.

Preliminary schedules are prepared several days in advance of their actual production data, but the final scheduling is 'on-line' in the case where the mill is controlled by a process control computer. For example, an ingot to be rolled at the slab mill is on a preliminary schedule several days in advance. The actual detailed rolling instructions, however, are issued only when the ingot is charged into a soaking pit to be heated to rolling temperature, approximately six hours in advance of the actual rolling. The rolling instruction is in the form of a punched card which is read by the mill control computer when the mill operator is ready to roll the ingot. The card is produced by the real-time system on a remotely located card punch at the mill production office.

The major real-time function of the computer system is the collection and display of data from a large number of 'in-plant' terminals. This enables the system to do accurate production and management reporting, and the scheduling of production.

To illustrate how the production control system functions, let us consider the steel melting plant and slab mill complexes. The steel melting plant produces casts of steel. A cast is a ladle of about two hundred tons of molten steel tapped from a

furnace. The molten steel is teemed from the ladle into molds which form ingots. The molds are stripped from the ingots after they have hardened, and the ingots are rolled and sheared into slabs at the slab mill. An ingot is a block of steel weighing between ten and twenty five tons, depending upon the mold size. A slab is a smaller flat block of steel, usually weighing around five tons. Its size and weight depend upon the requirements of the rolling mill which will use it.

Starting with the forward loaded orders taken from the order control system, the production control system computes the number and specifications of the slabs required to satisfy the orders over a production cycle. Once it has determined the slab requirement, it computes the number and specifications of the ingots from which the slabs will be rolled. It then computes the number and specifications of the cast of steel that must be made at the steel melting plant to produce the required ingots. As each computation is done an 'on-line' file of slab, ingot and cast requirements is built up. When these computations are complete, the computer prints preliminary schedules for the slab mill and a steel order for the steel melting plant. Once approved, the schedules and steel order are issued to the production units by the production planner.

The steel order specifies the cast and ingots to be produced, but does not stipulate the sequence in which they are to be manufactured. Thus, the actual slab mill schedule and rolling instruction cannot be prepared until the ingots have actually arrived at the mill.

The progress and specifications of the cast and ingots are reported to the computer system by personnel using 'in-plant' terminals at the steel melting plant, the ingot weigh bridges and the soaking pits. The terminals are also used to display information and instructions regarding the steel flowing through the production facility.

Once the ingots reach the soaking pit, where they are heated to proper rolling terperature, their arrival is reported through an 'in-plant' terminal. The system then refers to its 'on-line' files to provide the necessary rolling instructions. The specifications of each ingot are checked, and insturctions are generated which will use the ingot for the highest priority order having the same specifications.

When the ingot is drawn from the soaking pit and enters the mill, the rolling instruction card, which was prepared on an 'in-plant' terminal card punch, is read into the mill control computer by the mill operator. The mill control computer then controls the process of rolling the ingot into a slab, and the shearing of the rolled slab into smaller slabs. At the same time, the mill control computer logs the particulars about each slab rolled, and automatically transmits the data to the real-time computer system. After each slab has been rolled, sheared, and moved to the slab stocking yard; information about its condition, treatment, and location is reported to the real-time computer through 'in-plant' terminals. The system updates'on-line' disc files so that up-to-date records of material stocks, production status, and order progress are available at all times.

These 'on-line' disc files are used to satisfy inquires for remote terminals, and as the basis for scheduling and production reporting.

Periodically, the 'on-line' disc files in the real-time computer are dumped onto magnetic tape. The magnetic tape is transferred to the batch processor, and production and management reports and accounting records are prepared daily, weekly, monthly, or as required. The periodic dumping of the data from disc to tape also provides an added measure of back-up, should it become necessary to reconstruct an 'on-line' file.

The design of the data collection and display network is an area which involves a tremendous amount of preplanning. Since the average 'in-plant' terminal operator will not be highly trained or educated, it was necessary to develop techniques and methods which would enable the system to capture the data with a minimum of operator action.

To do this, communication methods are employed that involve basically three types of remote terminals: standard keyboard send-receive teleprinters; numerical data entry devices, which are standard keyboard send-receive teleprinters with the addition of a modified push button keyboard; and cathode ray tube display units.

To use the standard keyboard send-recieve teleprinters, the operator contacts the computer by transmitting a call code. This action steps-up the communications and program linkages, and starts a 'computer to operator' conversation that is controlled by the computer. The computer asks a series of questions, and the operator answers each question sequentially. The first question must be answered before the second is asked. Each response to a computer generated question is validated by the system to make certain the data received are reasonable before the conversation is continued. Normally, an 'in-plant' terminal will be utilizing only one communications program. However, the ability to conduct several simultaneous conversations with a terminal has been incorporated into the system.

When using the numerical data entry devices, the terminal operator is provided with a preset message

format. He sets up his entire transmission on the push button keyboard. The terminal operator then sends the message to the computer by depressing a 'transmit' button. This type of unit reduces the number of messages required to collect the data, and makes it possible for the terminal operator to check the data before he sends them.

The cathode ray tube display units are used for both data collection and as display terminals. To collect data, the technique of displaying a form on the unit for the operator to fill in, is most commonly used. To obtain a display of data, the operator calls for a specific display by sending a call code. The computer automatically presents the latest data on the files regarding the request.

Normally, all conversations between the computer and the 'in-plant' terminals are on an immediate response basis. On the other hand, provision has been made for terminal operators to request information which they will collect at a later time on the same terminal, or on a different 'in-plant' terminal. Provision has also been made for the computer to advise the terminal operator that there will be a delay in displaying the data requested. At all of the 'in-plant' terminals, the programs have been designed so that the operator can converse with the computer in either Afrikaans or English.

When an order has been produced and is ready to be dispatched, the real-time processor is advised through the data collection terminals, and the order control system is brought back into action.

The order control system will produce, on remotely located 'in-plant' terminals at the dispatch area, the necessary documents for shipping the product to the customer. Such items as consignment notes, gate release note, loading particulars, and special handling instructions are printed on remote units.

The order control system also prepares the invoices and customer accounts on a batch processing basis.

In order to provide an adequate pool of data so that queries about orders can be answered promptly, all of the data relative to an order are retained in the 'on-line' files for thirty days after the order has been dispatched. Accounting and statistical data are retained in off-line files indefinitely.

The system outlined has been installed and tested. All of the necessary software has been developed. The order control system and the first portions of production control system are now operational.

Martin Orlando reporting environment

by MICHAEL J. McLAURIN and WALTER A. TRAISTER
Martin Marietta Corporation
Orlando, Florida

Design objectives

The system was designed to alleviate the problem of "one time" or special request reports. A method was needed to quickly produce reports on request without having to write, compile and debug programs in order to produce the reports. The original design objectives were to permit any individual who understood how to prepare the input parameters the capability of producing any report within thirty minutes preparation time. MORE eliminates the necessity of having to write and maintain great numbers of special purpose report programs. It eliminates the necessity for special passes of master, sort parameter and activity files in order to produce the desired report. This is possible since the MORE system is a series of external subroutines which may be called by any existing program which is already accessing or passing a specific file. This system affords the user the advantage of using the built-in general print program or providing his own format program to the system. It has been determined that this system satisfied 85 to 90 percent of Martin Orlando business report requirements.

System modules

The following system modules can be followed on Figure 1. The system consists of the following programs. A detailed explanation of the function of each can be found under MECHANICS.

1. *Input Parameter Audit Program*—This program's primary function is to sort the input parameter cards and audit every field of the parameter cards.
2. *Communication Program*—This may be any program which calls the data selection program, hereafter referred to as the "PICKER." The communication program may be any existing file maintenance, audit, or any program which passes files containing potential report data.
3. *Data Selection or Picker Program*—This is the program which does the actual data selection of the information to compose the report. Data is selected, based upon satisfying the requirements of the input parameter cards.
4. *Executive Monitor Program*—This program sorts the selected or picked report data into report sequence and controls the programs required to format the report data.
5. *General Print Program*—This program formats the picked data records based on the print parameter cards which were input to the system. This program may be substituted for by any user written program if desired.
6. *Systems Interface Program*—This program will take any COBOL source program and add to it all linkages and coding necessary to utilize the MORE system. The output from this program is an updated COBOL program. The user also has the option of simultaneously compiling the program and placing the executable load module on program library.

Mechanics

All references to programs in this section may be followed on Figure 1.

Parameter card audit program

This program is run before the communications program which "calls" the Picker program. The audit program sorts the parameter cards on request number (report number) and sequence number within the report. All fields of the parameter cards are edited and all appropriate error messages are issued (see Exhibit E for detailed error messages). The exception report is produced on line so that the parameter cards may be corrected and the job can be rerun immediately. The edited parameter cards are written onto a temporary disk file along with a good or bad indicator. During the pick phase, only those requests which are error-free will be honored by the Picker. The Audit program currently restricts the system to a maximum

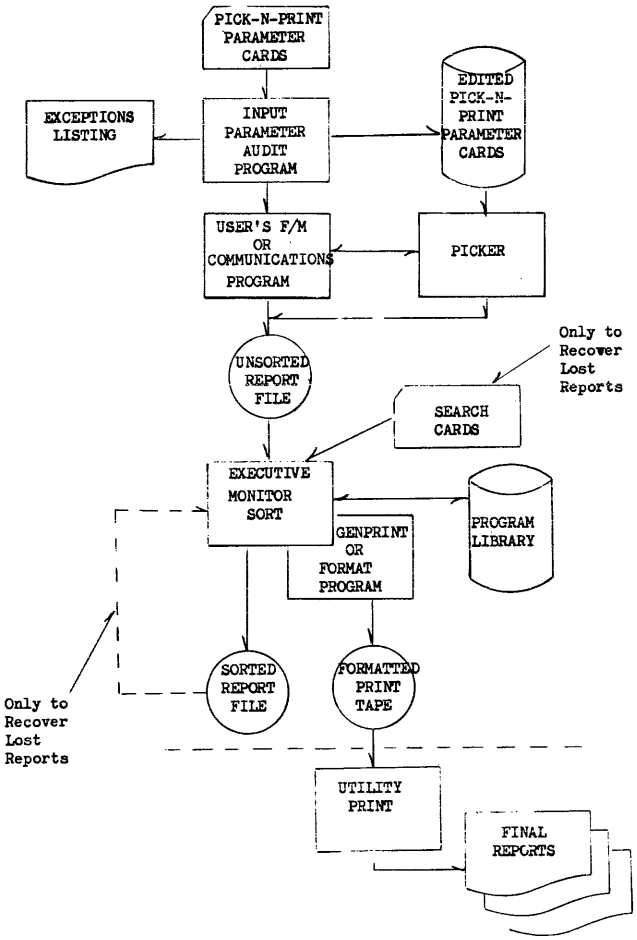


Figure 1 – Systems flow

of thirty-six reports and 600 pick and print parameter cards.

Picker program

When the picker is called the first time, it reads the temporary disk file created by the audit program and builds entries into the picker table. (See Figure 2 to follow processing). Only those requests which were error-free will be built into the table. Only pick parameter information is built into the table. If print parameter cards are encountered, the picker immediately builds a format record and releases it to the unsorted stacked report tape. Once the table is built, the picker examines the first updated master record (if the calling program is a file maintenance program) sitting in the user's output area. The picker compares every record passed through the user's output area against each report request in the pick table. Any records which get a hit against a report request is built according to build field specifications on the pick parameter cards. The following Figure 2 shows how the communication program and the picker interface.

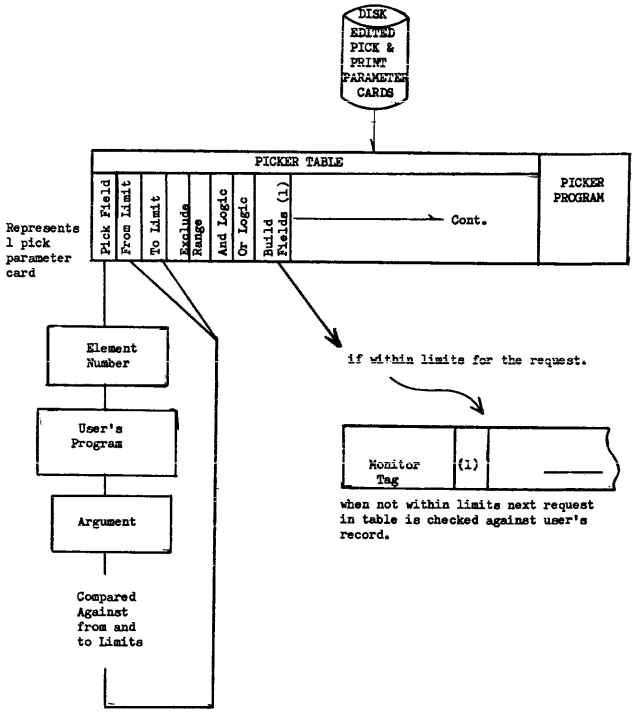


Figure 2 – Communication program and picker interface

The following coding is generated in the user's program by the SYSTEM's Interface Program. This coding accomplishes all linkage between the user's program and the picker.

READ-MASTER

READ or MOVE master to be picked into PICK-AREA and when updating is completed GØ TØ CALL-1.

CALL-1.

MØVE 0223 TØ ENTRY-CØUNT ØF PICKER-RECØRD.
ENTER LINKAGE.
CALL "PICKERPK" USING REQUEST-CØNTRØL,REQUEST-NØ-STØRE,ELEMENT,NUMBER,ARGUMENT,ENTER-N-WAY,PICK-AREA-LENGTH,PICK-AREA, FØRMAT-RECØRD,PICKER-RECØRD, FØRMAT-HF-RECØRD,FØRMAT-SW.
ENTER CØBØL.
IF ENTER-N-WAY IS EQUAL TØ 4 WRITE PICKER-RECØRD THEN GØ TØ CALL-1.
IF ENTER-N-WAY IS EQUAL TØ 5 AND FØRMAT-SW IS EQUAL TØ 1 WRITE FØRMAT-RECØRD THEN GØ TØ CALL-1.
IF ENTER-N-WAY IS EQUAL TØ 5 WRITE FØRMAT-HF-RECØRD THEN GØ TØ CALL-1.

```
IF ENTER-N-WAY IS GREATER THAN
ZERØ PERFØRM FIND-ELEMENT THRU
ANY-EXIT THEN GØ TØ CALL-1, ELSE
GØ TØ READ-MASTER.
```

FIND-ELEMENT

```
GØ TØ A1,A2,A3, etc.
DEPENDING ØN ELEMENT-NUMBER,
    ELSE GØ TØ ANY-EXIT.
A1.MØVE F1 TØ ARGUMENT THEN GØ
    TØ ANY-EXIT.
A2.MØVE F2 TØ ARGUMENT THEN GØ
    TØ ANY-EXIT.
A3.MØVE F3 TØ ARGUMENT THEN GØ
    TØ ANY-EXIT.
ANY-EXIT.
    EXIT.
```

WORKING-STORAGE SECTION

```
** 77 FØRMAT-SW              PICTURE 9.
   77 REQUEST-CØNTRØL        PICTURE X(2).
   77 REQUEST-NØ-STØRE       PICTURE X(2).

   77 ELEMENT-NUMBER         PICTURE 9(3).
   77 ARGUMENT               PICTURE X(15).
** 77 ENTER-N-WAY            PICTURE 9.
** 77 PICK AREA-LENGTH       PICTURE S9999.
   01 PICK-AREA.
      02 USERS--RECØRD.
      02 PICK-RØW REDEFINES USERS--
         RECØRD.
         03 CHAR-PØS         PICTURE X
                            ØCCURS
                            (user's record
                            size) TIMES.
```

REDEFINITION OF USER'S MASTER FILE

```
FD USER-MASTER-FILE-EXAMPLE
    BLOCK CONTAINS XXX CHARACTERS
    RECORD CONTAINS XX CHARACTERS
    RECORDING MODE IS F
    LABEL RECORDS ARE STANDARD
    DATA RECORD IS MASTER-RECORD.
01 MASTER-RECORD
   02     GREGORIAN-DATE.
          03 MONTH          PICTURE 99.
          03 DAY            PICTURE 99.
          03 YEAR           PICTURE 99.
   02     F1 REDEFINES GREGØRIAN-
          DATE.
          03 F2             PICTURE 99.
          03 F3             PICTURE 99.
          03 F4             PICTURE 99.
```

USER'S EØJ

```
When the user reaches END-ØF-JØB, GØ TØ
CLØSE-ALL.
CLØSE-ALL.
CLØSE USER'S FILES.
STØP RUN.
END PRØGRAM.
```

Every entry in the picker table is compared against each record on the User's Master File. When the table has been traversed for a given master record, the communication program is signaled to read the next record. If the communication program is a File Maintenance program the new record should be updated and written on the new master file before the PICKER is called again. This cycle is continued until an end-of-file condition is encountered.

The following is a list of PICKER capabilities, all of which are explained in detail in Exhibit C:

1. PICK RANGE LIMITS
2. EXCLUSIVE FIELD RANGE LIMITS
3. AND LOGIC
4. OR LOGIC
5. RANGE
6. PACKED BUILT RECORDS

Executive monitor program

This program sorts all records on the stacked report file into monitor tag and sort tag sequence. It also creates the report header and report trailer records on the formatted print file. The Executive Monitor dynamically "calls-in" the respective format program from the program library when the monitor tag changes. Each previous program is deleted from core when the change occurs. The program which is "called-in" is based on the program number specified in the monitor tag of the stacked report records. After loading the appropriate format program, the Executive Monitor passes to the format program each record to be formatted. The format program creates the print line image and releases it to the formatted print tape. The format program has the responsibility of writing the formatted records due to the possible necessity of condensing or exploding records.

Each format program is contained on the production library and the control cards necessary to

** In HSEKPG of user program move zero to FORMAT-SW, ENTER-N-WAY, and move 0223 to ENTRY-COUNT of picker record, move (user's Pick-Area size) to PICK-AREA-LENGTH.

execute the Executive Monitor and all Format programs reside on the Procedure Library. It is essential that a program exist on the production library for every data set passed to the Executive Monitor or else an abnormal termination will occur and the entire "job stream" will be interrupted.

In order to recover lost or damaged reports more efficiently, the Executive Monitor contains a search capability. Any or all reports can be recovered from the Sorted Report File by supplying the appropriate MONITOR TAG CARD used when the report was picked, or by punching a MONITOR TAG CARD if the source of the report was other than the Picker. A user's format program may require other data in addition to the data contained on the stacked report file. In this event, additional peripherals may be assigned and controlled by the format program.

The purpose of the HEADER RECORDS which precede each report is that they initiate a pause on the printer after having been printed. This record provides information to the operator (i.e., form number to be mounted, paper ply, etc.). The purpose of the trailer records is that they provide a record count from which machine utilization accounting and charge reports are prepared.

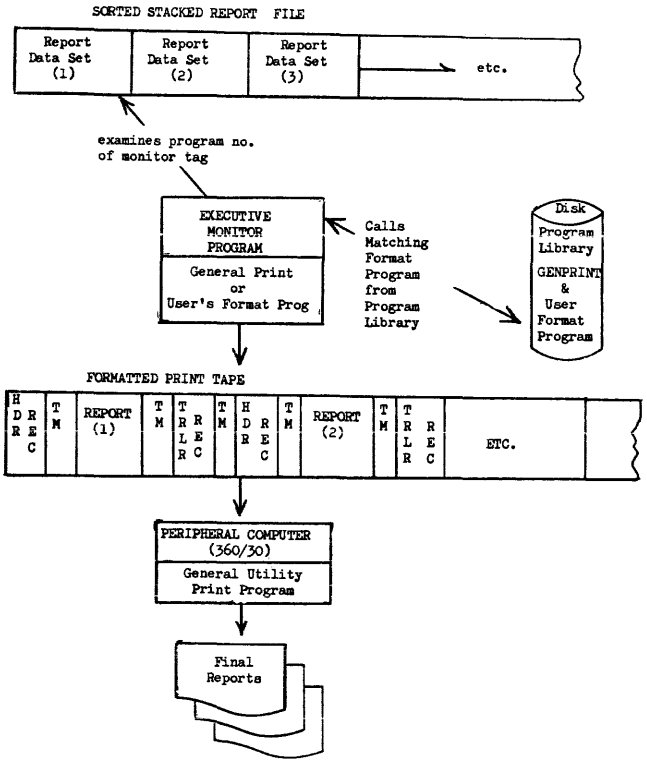


Figure 3 – Executive monitor data flow

General print program

The user has two modes under which he may use the MORE system. He can effect a pick only, in which case he must provide his own format program to print the picked records, or he can effect a pick and print, in which case the GENERAL PRINT PROGRAM (GENPRINT) will be used to do the formatting of the picked records. The mode is determined by the presence or absence of print parameter cards. (A detailed description of all parameter cards is contained in Exhibit D).

When print parameter cards are submitted, they are intercepted by the picker and reformatted with appropriate monitor and sort tags and released to the unsorted report file, while the pick parameter cards are being tabularized by the PICKER. The print parameter records are built such that they will sort immediately preceding the picked records for a given report request.

When GENPRINT receives these print parameter format records, it builds a series of meaningful tables against which all picked records are page image formatted.

GENPRINT capabilities

All of the following capabilities and how to initiate them are explained in detail in Exhibit D.

CONTROL BREAKS: GENPRINT has the capability of recognizing six levels of control breaks from minor to major as specified in the print parameter cards.

CONTROL HEADINGS & FOOTINGS: When a control break occurs it is possible to get from one to six control footings and/or control headings, depending upon the level of the break. The user also is provided the capability of specifying any vertical line spacing before, between and after the printing of the control headings and footings.

OVERFLOW HEADINGS & FOOTINGS: When an overflow condition occurs (page limits exceeded) the user has the capability of specifying from one to six overflow footings and/or headings. The same vertical line spacing capabilities are available as stated for control headings and footings.

MASK MANIPULATION: GENPRINT has all the mask manipulation afforded by the PICTURE CLAUSE of most current COBOL COMPILERS. This mask manipulation uses the same symbols and follows the same rules as are applied to the PICTURE CLAUSE.

PAGE IMAGE FORMATTING: This program will handle any vertical line spacing between DETAIL lines and any of the above mentioned HEADING and FOOTING lines. It will also permit any

horizontal field or character prositioning up to and including a maximum of 132 characters.

GROUP INDICATE: This feature was implemented the same as in most current CØBØL compilers.

VARIABLE DATA AND HEADINGS AND FOOTINGS: GENPRINT has the capability of "floating" into any heading or footing line any information which was built into the picked data record or the detail print line. Thus, the user is afforded the flexibility of creating headings and footings which consist of both literal information and "live" data.

TOTALING: This system provides any totaling capability required by most normal business reports. This includes control break, level totaling.

SYSTEMS INTERFACE PROGRAM: This program was written to relieve the problem programmer from having to include any additional coding in his program in order to use the MØRE system. It also permits easy incorporation into already existing programs. The PICKER requires all fields of the record to be picked to be redefined by "F" numbers, (i.e., F01-----F99, G01--etc.). The interface program redefines all the user specified fields with the proper "F" numbers. This program also provides all linkage working-storage fields required by the picker. It also inserts the procedure division code at the proper point in the Communication program.

The input to the interface program is the user's CØBØL source program. The interface program produces a new source deck with all the necessary coding required by the MØRE system. The user has the option of simultaneously compiling the new CØBØL program and placing it on the program library.

EXHIBIT A

PICKED RECORD LAYOUT

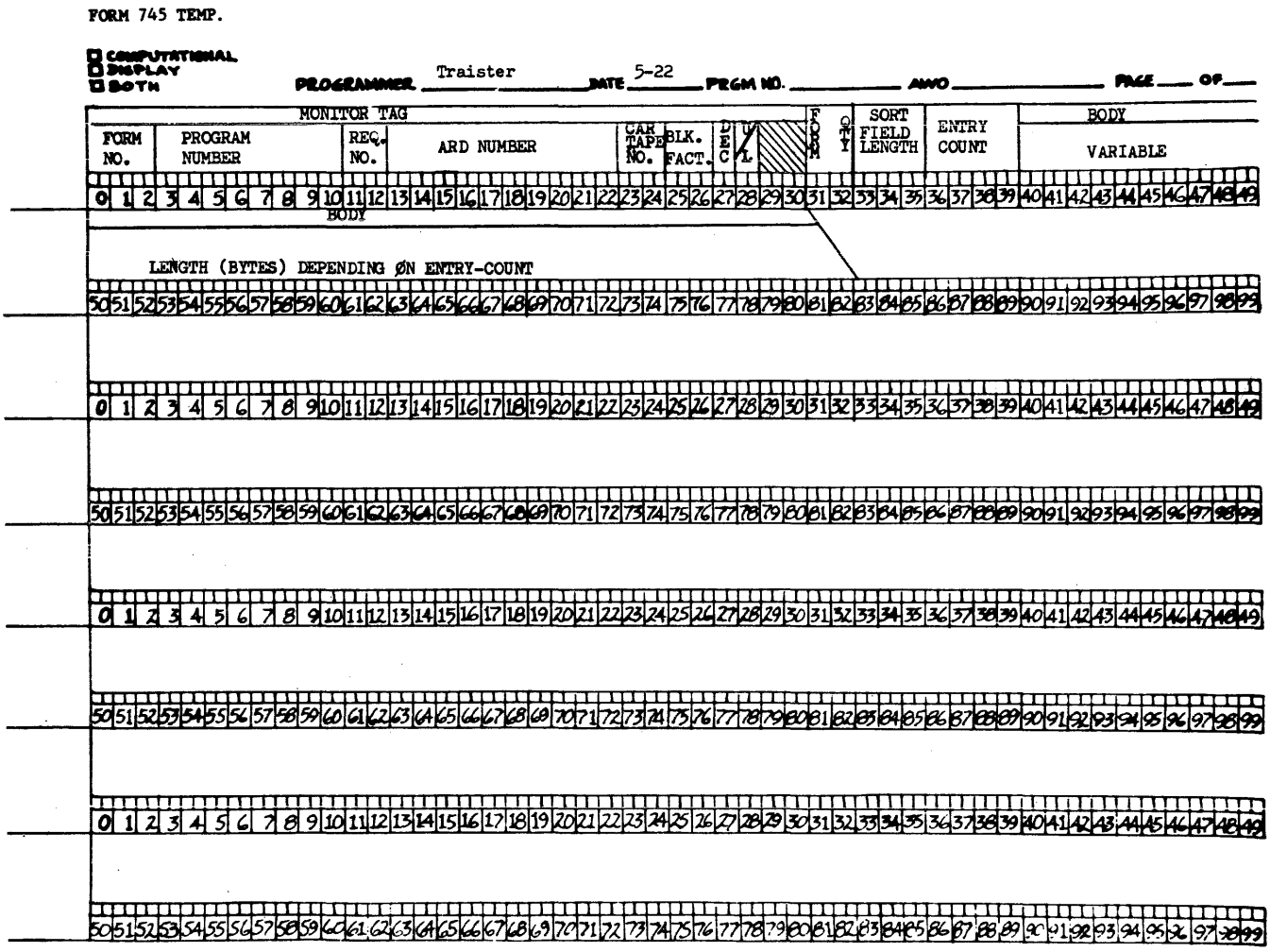


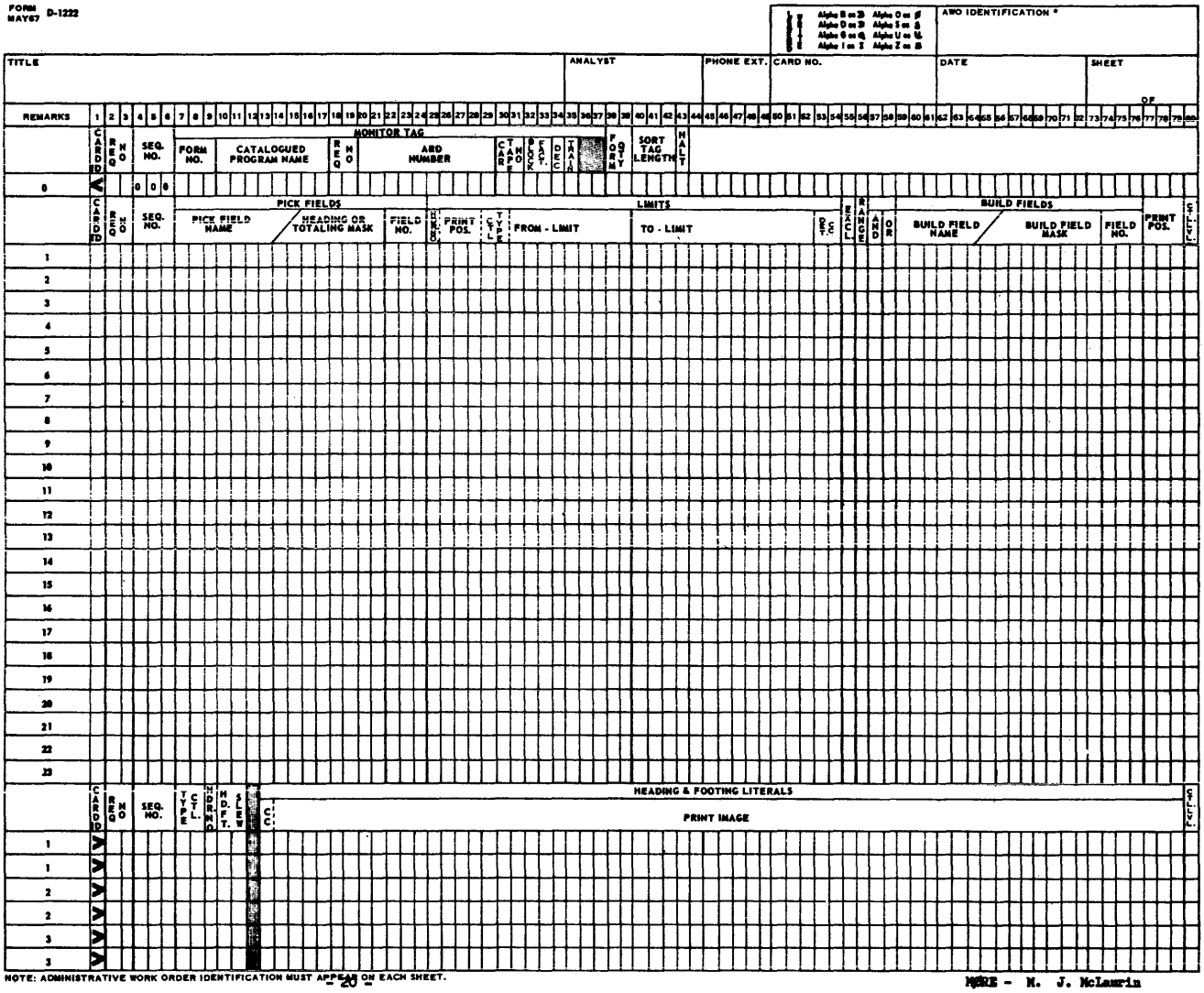
EXHIBIT B

STACKMASTER FD ENTRY PRODUCED BY PICKER

```
FD ØUTPUT-FILE
   BLØCK CØNTAINS 1 RECØRDS
   RECØRDING MØDE IS V
   LABEL RECØRDS ARE STANDARD
   DATA  RECØRDS  ARE  STKMSTR-
   RECØRD,PICKER-RECØRD,FØRMAT- RECØRD,FØRMAT-HF-RECØRD.
01 STKMSTR-RECØRD
   02 MØNITØR-TAG
      03 FØRM-NUMBER                PICTURE 999.
      03 PRØGRAM-NUMBER             PICTURE X(8).
      03 REQUEST-NUMBER             PICTURE 99.
      03 ARD-NUMBER                 PICTURE X(10).
      03 CARRIAGE-TAPE-NØ           PICTURE 99.
      03 BLØCK-FACTØR               PICTURE 99.
      03 DECØLLATE-CØDE             PICTURE 9.
      03 UPPER-LØWER                PICTURE 9.
      03 FILLER                     PICTURE XX.
   02 FØRM-QUANTITY                 PICTURE 99.
   02 SØRT-TAG-LENGTH               PICTURE 9(3).
   02 ENTRY-CØUNT                   PICTURE 9(4).
   02 BØDY.
   02 BØDY-SERIAL-MØVE PICTURE X ØCCURS 960 TIMES DEPENDING
      ØN ENTRY-CØUNT ØF STKMSTR-RECØRD.
01 PICKER-RECØRD.
   02 FILLER                        PICTURE X(36).
   02 ENTRY-CØUNT                   PICTURE S9999.
   02 BØDY                          PICTURE X OCCURS 960 TIMES DEPENDING
                                    ON ENTRY-COUNT OF PICKER-RECORD.
   01 FØRMAT-RECØRD
      02 FILLER                     PICTURE X(403).
   01 FØRMAT-HF-RECØRD
      02 FILLER                     PICTURE X(493).
```

EXHIBIT C

360/50 PICK-N-PRINT PARAMETERS



NOTE - N. J. McLaurin

EXHIBIT D

TRANSMITTAL DEFINITIONS

A. *GENERAL*

1. cc1. CARD-IDENTIFICATION. < Denotes monitor-tag-card; * denotes pick parameter card, $ denotes print parameter card, and > denotes heading or footing print parameter card.

2. cc2-3. REQUEST NUMBER. A two digit sequential number for each request submitted.

3. cc4-6. SEQUENCE NUMBER. A three digit sequential number which is continuous throughout all requests.

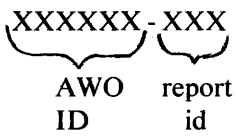
B. *MONITOR TAG CARD*

1. cc7-9. FORM-NUMBER. A three digit number assigned to the form for printing purposes.

2. cc10-17. PROGRAM NUMBER. An eight digit PROGRAM-ID that will process or format this data after sorting.

3. cc18-19. REQUEST NUMBER. A two digit sequential number for each request submitted.

4. cc20-29. ARD NUMBER. A six digit number which identifies the AWO identification and a three digit number which identifies the report. The ARD field has the following format:



5. cc30-31. CARRIAGE TAPE NO. A two digit number which identifies the carriage tape to be used to print the formatted data which was output by the format program.

6. cc32-33. BLOCKING FACTOR. A two digit number indicating the number of logical records contained within each physical tape record written out on the formatted tape.

7. cc34. DECOLLATING CODE. A one digit code which indicates the decollating procedure and bursting procedure.

8. cc35. PRINT TRAIN DESIGNATOR. A one digit code which indicates the type of print train to be used.

9. cc38-39. FORM QUANTITY. A two digit number which indicates the number of boxes of cards or paper required by the punch or print program. (May be blank.)

10. cc40-42. SORT TAG LENGTH. The number of characters in the body of the output record which will constitute the sort field. This field must be 223 which means that the minimum record length of all records passed to the executive monitor sort will be 263 bytes.

11. cc43. HALT CODE. A non-blank character in this field indicates to the pick program that if all requests of a pick were bad, the picker should 'ABEND' the user's program. It is only necessary to include this code in the first Monitor Tag card of a pick. This code must be included for non-file maintenance programs.

1. ccl. CARD IDENTIFICATION. < Denotes that this is a monitor-tag card; * denotes that this is a pick parameter card; $ denotes that this is a print parameter card; > denotes that this is a heading or footing print parameter card.

2. cc2-3. REQUEST NUMBER. A two digit sequential number for each request submitted.

3. cc4-6. SEQUENCE-NUMBER. A three digit sequential number for each parameter card within a request.

4. cc7-21. PICK FIELD NAME. A fifteen digit field which is used by the customer to fill in the name of the field upon which he wants to pick.

5. cc22-24. PICK FIELD NO. A three digit field which is used by Management Information Systems to fill in the 'F' number (corresponding field number in the program) which corresponds to the field name supplied by the customer.

6. cc25-39. FROM LIMIT. Indicates the lower limit against which the argument will be compared (left justified).

7. cc40-54. TO LIMIT. Indicates the upper limit against which argument will be compared (left justified).

8. cc55. EXCLUSIVE PICK CODE. If cc55 contains a non-blank character everything will be picked except the range represented in the FROM and TO LIMITS fields.
9. cc56. RANGE CONTROL. If the pick field number of the parameter card represents a range, the first parameter card of the range describes the upper limit and cc56 contains an X. The second parameter card of the range describes the lower limit and cc56 contains an X.
10. cc57-58. LOGIC CODE. If "AND" logic is to be applied, column 57 will contain an X or any non-blank character for this specific parameter card.

    If "OR" logic is to be applied, column 58 will contain an "X" or any non-blank character for this specific parameter card.

    Card columns 57 and 58 are mutually exclusive. For any specific pick parameter card only one of these columns may contain a non-blank character.
11. cc59-73. BUILD FIELD NAME. The name of the field which the customer specifies indicating that this field is to be represented in the output record.
12. cc74-76. BUILD FIELD NUMBER. A three digit number representing the relative location of an element in the pick-area from which the body of the output record is to be built. The absence of any build fields indicates that the entire record is to be picked.
13. cc77-80. Not used.

D. *PRINT PARAMETER FORMAT CARD*

a. DETAIL PRINT DATA:

The following columns will be used to describe the individual fields of a detail print line:

1. cc53-54. DETAIL SLEW VALUE. A two digit numeric field which indicates the number of lines to be slewed before printing. This field must be specified for each field of the line. If group indicating - $-'CH' type control must show positive slew and $-'CF' type control must show 00 slew.
2. cc59-73. DETAIL FIELD EDIT MASK. A maximum of a fifteen position edit mask which will describe the print format for the preceding detailed field to be picked. This mask is right justified.
3. cc77-79. DETAIL FIELD PRINT POSITION. A three digit field which reflects the right most horizontal print position where the data are to be printed. If group indicating - $-'CF' type control lowest level must contain '999' and higher levels must contain '000'.
4. cc80. LEVEL INDICATOR. A one position field which indicates that the build field number is to be used for control purposes. Levels 1 thru 5 indicate minor thru major respectively with FINAL level being the highest level specified. The maximum number of levels is six. This column will be blank if not applicable.

b. HEADING AND FOOTING LITERALS:

The following columns will be used to describe literal information of the user's heading and footing lines. These fields are shown at the bottom of the input transmittal (Exhibit B).

1. cc7-8. TYPE CONTROL FIELD. A two digit field which may contain any one of the following types:
   CH Control Heading
   OH Overflow Heading
   OF Overflow Footing
   CF Control Footing
   (a) Control Heading and Control Footing literals are associated with their respective control level indicator (cc80) of the detail format card.
   (b) Overflow Heading and Overflow Footing literals are associated with their respective 'HDR-NO' (cc9) indicating at overflow time the relative sequence in which they are to be printed. There is a maximum of six Overflow Headings Headings and Footings for each report.
2. cc9. HEADER NUMBER INDICATOR. A one digit numeric field

which is used to associate fields to to be 'floated' into the Overflow Heading and Footing lines from the detail format cards. This field is used only with types OH and OF. A maximum of six Overflow Headings and Overflow Footings may be specified. A 1 represents the first and a 6 represents the last.

3. cc10-11. HEADING OR FOOTING SLEW. A two digit numeric field which indicates the number of vertical lines to be slewed before this heading or footing line is printed. This value should be specified only in the first of each set of two cards.
4. cc12. Not used.
5. cc13. CARRIAGE CONTROL. A one digit field which indicates whether or not this heading or footing line is to be slewed to the top of the next page. The only acceptable values for this field are 1 or A 1 indicates slew to top of page before printing. This field should be used only in the first of each set of two cards.
6. cc14-79. HEADING OR FOOTING PRINT IMAGE. A total of 132 characters which contain the heading or footing literal information to be printed. If group indicating and final totals are required, a literal line with 00 in the slew field must be specified.
7. cc80. CONTROL LEVEL INDICATOR. A one position field which associates a specific control Heading or Footing with the respective control level. A maximum of six Control Headings and six Control Footings may be specified. A 1 represents the first and a 6 represents the last.

c. FLOATED DATA FOR HEADINGS AND FOOTINGS:

1. cc7-21. HEADING OR FOOTING MASK. A maximum of 15 digits which represents the edit mask for the data to be floated into the heading or footing literal line. This mask must be right justified. justified.
2. cc22-24. FIELD NUMBER. A three digit field which serves one of the following functions:
   (a) Specifies the field number to be totaled for Control Footings or inserted for Control Headings.
   (b) *** indicates that this is the mask for the page counter.
   (c) NN$ indicates that this is the mask for the date. NN is the number of the date to be obtained from the GETDATE subroutine. Any one of the eleven dates of GETDATE may be specified.
   (d) 'N' specifies the literal to be floated into a heading or the increment to be added to a counter for every record to be printed.
3. cc25. HEADER NUMBER. A one digit field which associates fields to be floated into heading and footing with their respective Overflow Heading or Overflow Footing line. This field may contain a number from 1 to 6. A 1 represents the highest level OH or OF and a 6 represents the lowest.
4. cc26-28. HEADING OR FOOTING PRINT POSITIONS. A three digit field which indicates the right justified print positions of the heading or footing line where the floated data are to be printed.
5. cc29-30. TYPE CONTROL. A two digit field which indicates to which Heading or Footing type (i.e., OH or OF, CH or CF) the floated information applies.

EXHIBIT E

PARAMETER CARD DIAGNOSTICS

A. *GENERAL*

MORE THAN 600 PARAMETER CARDS REMAINDER DELETED

PARAMETER CARDS ARE OUT OF SEQUENCE

MORE THAN 36 REQUEST – REMAINDER DELETED

B. *MONITOR TAG CARD*

MONITOR TAG CARD MUST BE 1st OF REQUEST

FORM NUMBER IS BLANK
PROGRAM NUMBER IS BLANK
INVALID REQUEST NUMBER
ARD NUMBER IS BLANK
BLOCKING FACTOR IS INVALID
SORT LENGTH FIELD IS BLANK
WARNING HALT CODE IS BLANK
DECOLLATING – BURST CODE IS BLANK
CARRIAGE TAPE NUMBER IS BLANK
UPPER-LOWER CASE PRINT CODE IS BLANK

C. *PICK PARAMETER CARD*

FROM LIMIT IS BLANK
TO LIMIT IS BLANK
LOGIC CODES ARE IN ERROR

FROM LIMIT OF FIRST RANGE CARD IS BLANK

TO LIMIT OF SECOND RANGE CARD IS BLANK

MUST HAVE 2 RANGE CARDS FOR EACH RANGE

FROM LIMIT IS GREATER THAN TO LIMIT

D. *DETAIL PRINT PARAMETER CARD*

WHEN PRINTING ALTERNATE PICK AND PRINT PARAMETER CARDS

WARNING-BOTH PRINT POS OF FORMAT CARD ARE BLANK

WARNING-BOTH MASKS OF FORMAT CARD ARE BLANK

MUST SPECIFY DE CC IN 1st FORMAT CARD OF REQUEST

SUPPRESSION ILLEGAL AFTER P
PARENS ILLEGAL FOLLOWING V

ILLEGAL SUPPRESSION
ZERO SUPPRESSION CANNOT BE SPLIT
ZERO SUPPRESSION ILLEGAL FOLLOWING DECIMAL POINT

V, g, S, $, *, Z,), (ILLEGAL FOLLOWING DECIMAL POINT

B FOLLOWING P IS ILLEGAL

CAN HAVE ONLY 1 DECIMAL POINT INDICATOR PER PICTURE

P's CANNOT BE SPLIT

ONLY ONE SIGN INDICATOR LEGAL PER PICTURE

(MAY NOT FOLLOW A SIGN
$ CANNOT FOLLOW DECIMAL POINT
$ CANNOT BE SPLIT
P CANNOT IMMEDIATELY FOLLOW A $
* ILLEGAL FOLLOWING DECIMAL POINT
*(s CANNOT BE SPLIT
(ILLEGAL AFTER X OR A

SUPPRESSION ILLEGAL FOLLOWING X OR A

ILLEGAL CHARACTER IN PICTURE
BLANKS WITHIN PARENS IS ILLEGAL
ONLY NUMERIC IS LEGAL WITHIN PARENS

NOTHING VALID SPECIFIED FOR PICTURE

E. HEADING AND FOOTING PRINT PARAMETER CARD

BOTH HEADER NUMBER AND CTL LVL CANNOT BE BLANK

NO SLEW VALUE SPECIFIED FOR THIS OH, OF, CH, CF

NO TYPE CONTROL SPECIFIED FOR THIS OH, OF, CH, CF

OH OR OF MUST HAVE HEADER NUMBER

WARNING PRINT IMAGE IS BLANK

Simulation applications in computer center management

by THOMAS F. McHUGH JR.
International Business Machines Corporation
Waltham, Massachusetts

and

DR. ELLIS L. SCOTT
University of Georgia
Athens, Georgia

Significant developments occurred during the 1960's in the use of computer-based simulation models for management analysis. Data processing personnel contributed extensively to these developments. However, the role characteristically attributed to them emphasized programming and machine operations. Relatively few models have been concerned with management aspects of computer center operations.*

A simulation study recently completed at the University of Georgia (UGa) Computer Center suggests that models for computer center management purposes are feasible and will have extensive applications. The UGa model was constructed in the interest of simplifying the complex decision making tasks involved in managing the operations of the computer center. The major processing configuration includes four IBM CPU's—a 7094 operated in a mag tape I/O mode, two 1401's used for both I/O support to the 7094 and for central processing, and a recently installed 360/65. The diversity of input and the elaborate operating system necessary to support the hardware complement combine to present the management with many examples of hardware interface, routing, scheduling, and queue management problems typical of most large- and medium-scale facilities. This suggests that the approach taken in the UGa study may have value for the many DP managers faced with similar decision making and evaluation tasks.

The objective of the UGa study was to develop a simulation model which could be applied on an ongoing basis by computer center management in decisions involving machine allocation, machine capability projections, system optimization and similar classes of management problems. In brief, the methodology involved the performance of a system analysis, and the construction of a simulation model to reflect the essential behavioral characteristics of the system. Once the model was completed and its validity determined, the model could then be utilized to compare alternative configurations and operating procedures without direct experimentation.

The UGa operating system

At present, input is batched for all three types of machines.* The user may have access to the computer he desires by simply submitting his input through a formalized data collection procedure.

7094 mode

7094 input in punch card form must be routed to one of the 1401's for card-to-tape processing. Input tapes are then introduced into the 7094 job queue to be processed on the basis of a priority system. Priorities in this and all other sub-systems are based on two criteria—user convenience and production efficiency. That is, jobs are rated according to the need of the user and the effect of job execution on system performance at that point in time. Output tapes generated by the 7094 are routed back to one of the 1401's for tape-to-list processing.

1401 CPU mode

In addition to serving as I/O support devices for the 7094, the 1401's are used to a limited extent for

*For a recent example see "SCERT: A Computer Evaluation Tool," by Donald J. Herman in *Datamation*, February, 1967.

*The 360/65 is scheduled to be upgraded to a model 67 to provide time-sharing capability.

central processing and utility jobs. One of the machines has a 16k character memory with four tape drives; the other has an 8k memory with two tape drives. Most central processing tasks are performed by the 16k machine, with the residual (i.e., tape-to-card, data-tape building, etc.) going to the 8k.

360/65 mode

All processing functions, including I/O, are performed on-line in the 360 system. Therefore, all input is routed directly to the computer upon entry into the system.

Development of the simulation model

Development of the simulation model occurred in three steps: collection of the data on the operations of the UGa system; construction of the model; and, execution of the model using data from step one as input. A sample period of one week was selected from operations during the first six months of 1967 and designated for both data collection and simulation purposes. Utilization records indicated that the week of May 7-13 met the selection criteria of freedom from excessive amounts of unusual kinds of inputs, and that input figures for the sample period be comparable with current operations. Data collection involved identification of the major variables of interest in the study and compilation of data on these variables.

Major variables

For the purposes of the study, average turnaround time was determined to be the most suitable measure of system performance. Average turnaround time was measured in minutes between entry of input (job-entry time) into the machine room and exit of the completed job from the machine room. The secondary variable, processing time, was defined in minutes for the execution of the job on the CPU, including I/O processing for the 7094 sub-system.

Data acquisition

The accounting routine records which are maintained on-line on both 7094 and 360 CPU's were the source of process time data for those two machines. 1401 process time was computed from information contained in a utilization log which is maintained for accounting purposes. Data on other operations, such as card-to-tape and tape-to-list, were compiled by stop-watch observations over a sub-sample during the sample period.

Model Construction

The model was conceptualized as the set of mathematical, functional and logical relationships which express the essential behavioral characteristics of the UGa computer system. Following the definition of these parameters the model was constructed using General Purpose Simulation System/360 as the programming vehicle. GPSS is well suited for simulation applications of this type.* Its block coding format is akin to the standard flow-charting techniques familiar to most all system analysts and data processing personnel. Hence, the utilization of GPSS does not necessitate extensive machine programming knowledge. In addition, GPSS also features several approximation devices, such as the random number generator, and user defined mathematical functions, which simplify simulation of complex systems.

In the UGa model, a mathematical function was employed to enter the simulated jobs (transactions) into the model at the same rate and time as the real jobs entered the machine room during the sample period. The dependent variable in this function was the number of jobs which entered the system during each of 168 hours in the sample week. The independent variable was the number of simulated hours which had elapsed since the start of the simulation run. Evaluation of the function at the close of each simulated hour caused the same number of transactions to be introduced into the model as were jobs into the real system at that hour during the sample period.

Process time for each of the major system operations was also defined by means of a mathematical function. The independent variable in this function was based on a graph showing the cumulative percentage of jobs requiring various amounts of processing time. A cumulative time distribution of processing time was constructed for each of the CPU's. Dependent variable values were determined by the random number generator. Values thus generated were interpreted to be percentages. The distribution of transactions among the simulated CPU's followed the same pattern as the distribution of the inputs in the real system.

Model validation

The simulation model was executed ten times (8330 simulated jobs) and the results of each execution were averaged. This procedure was considered necessary to reduce the probability of random effects biasing the results. Results from simulation runs were then compared with empirically derived data for the sample period of operations. Two methods of comparison were employed. The first method was comparison

*See "General Purpose Simulation System/360: Introductory Users Manual." White Plains: International Business Machines Corporation, 1967.

of overall average turnaround time data generated by simulation runs with real system data obtained from accounting program outputs and from utilization records maintained by the operations staff in the machine room. Mean turnaround time for the sample period was 283 minutes; for the model it was 275 minutes, a difference of 8 minutes or 3%. The second method of comparison was a Chi-squared test of the individual process-time observations for both the model CPU's and the real system processors. No significant difference was established.

Model applications

Following validation, the model was then applied to three classes of management problems: (1) machine capability projections; (2) machine allocation; and, (3) operating system optimization. Questions were formulated with the immediate interests of the UGa Computer Center management in mind.

Question One:

At what level of 7094 utilization do the 1401 I/O facilities become saturated and incapable of meeting the support demands of the 7094?

Transaction (simulated job) inputs into the 1401's were increased until the 7094 throughput stabilized and additional 1401 entries went into queue. Management had heretofore assumed that the 1401's would saturate before the 7094, requiring upgrading of support equipment. In the simulation runs, however, the 7094 showed an average utilization of 96%, while the average utilization for the 1401-16k was only 79%, and 77% for the 8k machine. An additional 150 7094 transactions were entered, and although 1401 utilization rose to 81% and 79%, the 7094 utilization average remained stable. In effect, this information eliminated the necessity for short term planning for the upgrade of 7094 support facilities.

Question Two:

At the saturation level of operations, how much improvement in turnaround time for the average user could be expected if all job types currently classified high priority were run on the 360 rather than the 7094?

Management assumed that as 7094 utilization increased to the saturation point it would be forced to reprogram and reallocate certain kinds of input. In the interest of establishing a priority hierarchy for reprogramming and obtaining maximum 7094 operating efficiency both during and after conversion, management wished to ascertain the effect of transferring 7094 jobs currently rated high priority to the 360 system. Given the saturation data in the previous run, the model was modified to re-route specially classified jobs. With the simulation of the 7094 subsystem in saturation, the re-routing to the 360 of that 7% of jobs classified as high priority resulted in a reduction of the mean turnaround time of the 7094 of 20%. Further manipulation of the job generation data showed that approximately 30% more transactions could be added before the 7094 returned to a state of saturation. In the new mode with special jobs excluded, turnaround time under saturation for the 7094 was 12% below that of the original model. Therefore, it was concluded that the proposed change in machine allocation would increase throughput on the 7094 and reduce turnaround time.

Question Three:

What would be the effect of eliminating third shift operations at the current level of job input?

Although anticipating increased utilization in the future, management was nevertheless interested in determining the effect on turnaround time of eliminating the third shift at the current level of utilization. The model was modified to simulate an inactive third shift. Turnaround time on the basis of a 16-hr. processing day increased from 275 to 523 minutes. This was not surprising since the elimination of the third shift at the current level pushed utilization to near saturation. Furthermore, in the three shift operation the third shift has a low input rate and provided slack time for backlog processing. On the basis of both service time and production efficiency criteria, elimination of the third shift would be detrimental to system performance. Turnaround time would increase by nearly 100% and relatively minor system perturbations, such as unscheduled downtime, would almost certainly result in serious backlogging.

CONCLUSIONS

The simulation approach represented by the UGa study manifests the general applicability of the concept of computer simulation in DP installations. Each facility has its own singular operating characteristics and hardware configuration, and must tailor the idea to its own requirements. Nevertheless, DP installations generally are in a favorable position to profit from simulation applications. The technology to facilitate these applications is at hand. Further, recent advances in simulation programming systems, such as GPSS, make the employment of simulation techniques available without elaborate machine programming efforts. In this evolving technological environment the expanded use of simulation for computer center management seems worthwhile.

Multiprogramming system performance measurement and analysis

by H. N. CANTRELL and A. L. ELLISON
General Electric Company
Phoenix, Arizona

Why

"... design without evaluation usually is inadequate."[1]

"Simulation... is applicable wherever we have a certain degree of understanding of the process to be simulated."[1]

"The key to performance evaluation as well as to systems design is an understanding of what systems are and how they work."[1]

"The purpose of measurment is insight, not numbers."[3]

Why should we spend time and money analyzing the performance of computer systems or computer programs? These systems or programs have been debugged. They work. They were designed for optimum performance by competent people who are just as convinced that their performance is optimum as they are that the program or system *is* logically correct. Why then should we analyze performance? There are three main reasons:

1. There may be performance bugs in a program. Performance bugs are the result of errors in evaluation or judgment on performance optimization. We have no reason to suspect that performance bugs are any less frequent or less serious than logical bugs. Thus, if the performance of a program or a system is important then it should be performance debugged by measurement and analysis.
2. If a new or better system or program is to be designed, then a good, quantitative understanding of the performance of previous systems is necessary to avoid performance bugs in the new design.
3. If an important program or system is intolerably slow, then the real reasons for its poor performance must be found by measurement and analysis. Otherwise time and money may be spent correcting many obvious but minor inefficiencies with no great effect on overall performance. Worse yet, the whole thing may be reimplemented with all key bugs preserved!

In all three of these reasons the objective of performance analysis is to understand the unknown. We're looking for performance bugs. If we knew what these bugs were and what they cost in performance, then we wouldn't have to look for them. But because we are looking for *unknown* performance limiters, we don't know in advance what performance gains can be made by finding and fixing these bugs. We don't even know how hard or how easy it will be to fix these bugs after we find them.

Thus, there is no way of predicting the performance payoff from the time and money spent in performance analysis. This time and money must be spent at risk, essentially on the basis of faith that the payoffs from performance analysis will exceed the value of any alternative way of spending this time and money. This is nothing new. This "faith" concept is well understood and accepted in the scientific and engineering communities. One of the purposes of this paper is to demonstrate that analysis pays off in programming to at least the same degree as in engineering and science.

How

All good analysis consists of a combination of theoretical analysis and empirical measurement. This is the classical "scientific method." The theoretical analysis may be done through a mathematical or simulation model (2) or it may simply consist of an understanding of how the system or program is supposed to work. The empirical measurement is designed to test the theory.

Neither theory nor measurement alone is sufficient. Theoretical analysis alone may solve a nonexistent or unimportant problem. Measurement alone often misses a few critical parameters needed to test the theory. Since neither can stand alone, analysis usually consists of successive applications of the clas-

sical theory/measurement/revised theory/revised measurement/etc., cycle.

It is important to recognize the iterative nature of analysis, the theory/measurement cycle. Successive cycles revise or obsolete previous concepts. Therefore, time spent in polishing the first theory or the first measurement method will almost certainly be wasted.

In analyzing a computer system or program the quantity to be measured is time – the time required to perform different functions or the time spent waiting to perform those functions. Performance analysis consists simply of trying to answer the question, "Where does the time go?" and given an answer, applying a subjective judgment as to whether the amount of time spent on a given function is reasonable.

The application of these concepts to an analysis of the performance of a multiprogrammed computer system and to the programs which operate in this system will now be considered with examples and illustrations taken from such an analysis of the General Electric-625/635 GECOS II operating system (5) and the system software and user programs which operate in that system. Only software measurement techniques are discussed. Schulman,[4] for example, discusses hardware measurement techniques.

A. System analysis

For accounting purposes a multiprogramming operating system usually has to keep track of how much processor time is applied to each program and how much I/O channel time, by channel and device, is used by each program during its execution. Thus, the GECOS II system keeps a running total by program of all processor, channel and device time used. These totals are updated for each period of processor use and for each I/O transaction. When a program terminates, all of its accumulated times are transmitted to an accounting file and the totals are zeroed for the next program.

All of the major functions of the GECOS II operating system itself are executed in 64 different, functional, operating system "programs." The time spent on each of these programs is accounted for in substantially the same way as for user programs. Operating system times are not used or reported. They are accumulated by the system because it is faster to do it than to decide not to do it.

Thus, in its normal operating mode the GECOS II system is very highly and very accurately instrumented. In addition, the system has a TRACE mode of operation, usually used to debug changes to the operating system. In TRACE mode a trace entry is made in a circular list for every major operating system event, such as dispatching the processor to a program or servicing an I/O channel terminate interrupt.

Given that a great deal of data is available, how do we go about analyzing the performance of this system? A few chronological steps in the actual analysis of this system will illustrate the theory/measurement cycle.

1. User Program Accounting Analysis
   Given an understanding of how the multiprogrammed system was supposed to operate (the theory), normal user program accounting data was analyzed. This data showed the starting and terminating time-of-day for each program and the processor and I/O channel time used by that program. The question, "Where does the time go?" was asked but could not be answered from this data. Our understanding of how the system was supposed to operate could not fully explain how it was operating. A tentative theory, "The extra time goes into overhead," was advanced.
2. Overhead Analysis
   A complete and accurate breakdown of all GECOS overhead processor time was available in core and could have been printed in a report. However, following the rule that initial polished reports are a waste of time, several octal memory dumps of this core area were analyzed. These dumps showed that overhead processor time was significant but not excessive. They also showed a significantly high percentage of processor idle time in a system that was supposed to be heavily loaded. The theory, "The extra time goes into overhead," was dropped and replaced by the question, "Why is the processor idle time percentage so high?" A polished overhead summary report was never implemented. The octal dump analysis had demonstrated that overhead was not the problem and that, whatever the problem was, it could not be exposed by analysis of overhead time summaries.
3. Trace Analysis
   Since summary data were not adequate to test the theory of system operation, a more detailed approach was taken. The internal logical trace entries generated by the operating system in TRACE mode were captured by a trace collector program and written on magnetic tape with a high resolution time-of-day value for each trace entry. In the first successful 5 minute run of this program 350,000 trace entries were captured on tape. This represented a complete timed record of literally everything significant that had occurred in 5 minutes of full speed operation.

(The trace collector program itself was one of the user programs in core and accounted for about 1% of the lead on the system.)

350,000 trace entries, if printed at one entry per line, would produce an essentially indigestible pile of paper 2½ feet high. So only the first 10,000 trace entries were printed to provide an understanding of the data so that a data reduction method could be devised. A succession of data reduction and reporting methods was then tried. At each stage the results were analyzed, comparing the then current theory of operation to the actual measurements. At each step both the theory of operation and the method of reporting the data were usually revised and the cycle repeated. This was very hard, detailed analysis work but it produced a steadily improving understanding of how the system really worked and steadily improving methods for measuring and displaying critical performance phenomena. Ultimately two complementary measurement methods and two reports evolved. These are described in a later section of this paper.

At the same time this analysis also yielded a succession of well defined, understood, and evaluated system performance bugs. Like logic bugs, some of these performance bugs were very obscure, involving many levels of complicated inter-relationships, but many were very obvious once they had been found and pinpointed in the light of the improved understanding of system operation. Again, like logic bugs, some of the performance bugs were hard to fix but many were corrected with only a few minutes work (after the required change had been defined). As performance bugs were found they were corrected in the prototype version of the next system software distribution. Thus, the theory/measurement/improved theory/improved measurement/improved theory/improved measurement/*improved system.* A typical time to go through this cycle was one week although for each of the two examples described below, a significant performance bug was found, evaluated, corrected in the system, and the improved system performance measured confirming the correction, all in one eight hour day.

During the operating system improvement cycle, fourteen performance bugs were found and fixed, resulting in an average throughput improvement on customer sites of 30% with a range of from 10% to 50% depending on the load mix. Two examples of the performance bugs found are:

1. Operating System Core Space

The GECOS II Operating System was designed to operate within 16K of core but would use more core if it was available. This is quite adequate for small systems but for large systems with several system printers running the operating system actually used an average of about 30K of core. But only 16K was reserved for the operating system. Measurements of large systems in operation showed that when too many slave programs were packed into core, the operating system was squeezed down and performance was degraded by about 25% due to the operating system fighting to get needed overlays into core. Over a day's operation this degradation averaged about 5%. To correct this problem the core space reserved for the operating system was made dependent upon system size (about 25% of the total core space). The previously measured performance degradation immediately disappeared.

2. Dispatcher Interval

The GECOS II dispatcher assigned the processor to user programs on a round-robin basis, allowing 62.5 milliseconds of processing for each program. Many programs would become roadblocked, waiting for I/O, in much less than this 62.5 milliseconds. The next program would then be dispatched. During processing on one user program, I/O terminate interrupts would be serviced and any I/O activities waiting in the queue would be started, but only the program processing at that time could initiate new I/O activities. The I/O time to read or write a system standard block to drum or high speed tape is about 25 milliseconds. The dispatcher overhead time to switch from one program to another was measured at about 0.5 milliseconds.

This appears to be a fairly reasonable strategy but actually it is very bad. If I/O bound programs are multiprogrammed with a compute bound program, the ideal situation, then the compute bound program will retain control of the processor for 62.5 milliseconds every time it is dispatched. The I/O activities started by the other programs would have terminated within the first 25 milliseconds and those programs and their I/O resources would remain idle for the remaining 37.5 milliseconds. Thus, with one compute bound program in core *all* operating system and user program drum and tape I/O operated at about 40% of normal speed.

To correct this problem the dispatcher interval was changed from 62.5 milliseconds to 15 milliseconds. The results were immediately apparent. In one benchmark of two tape sort programs multiprogrammed with a compute bound program, the sorts ran twice as fast.

B. Program analysis

An operating system and an installation's operating procedures provide the environment in which users' programs and manufacturer-supplied compilers, assemblers, sort programs, etc., operate. The overall performance of a computer system obviously depends upon both the efficiency of the environment and on the efficiency of the programs which operate in that environment, or at least on the subset (usually small) of all programs which account for most of the load on the system. In an earlier section, System analysis, the performance analysis of the environment was discussed. This section is concerned with analyzing the performance of individual programs. Again the starting point is the question, "Where does the time go?"

1. Input/Output and Compute Time Profiles

   I/O transactions and their degree of concurrency with each other and with computation are important characteristics of system performance. Thus, they must be accurately measured as a part of system performance analysis. By measuring system performance, as described above, with only one program operating in core a complete I/O and compute time profile of that program is obtained.

   From this profile the programmer can apply value judgments and tradeoffs as to whether the time spent is appropriate to the function being performed and whether the optimum degree of concurrency has been attained. Such an analysis is normally a part of the initial design of any program, but surprises (performance bugs) are not unusual. Optimal blocking, buffering, and other I/O strategies are not always achieved. An example is a COBOL program which repeatedly wrote a very small transient file on a tape and then read it back again a few moments later. The total amount of data written and read was very small so this I/O time hardly entered the initial performance calculations. But the measured I/O compute profile for this program showed that *most* of its *total* time in the machine was spent in repeatedly opening, closing, and label checking this small transient tape file. The time spent on providing transient storage was outrageous compared to any of several alternative ways of performing the same function. This is an example of functional value analysis.

2. Compute Time Analysis

   System analysis methods and measurements provide adequate measures of the I/O-compute prorile for an individual program. But system oriented measures do not cast any light on where the compute time goes within a program. This is a classical problem whose solution is an essential part of performance analysis. It had been under attack throughout the GE-625/635 performance study described above but its solution was accidental. Actually, the data measuring the internal compute profile of a program had been taken and existed on a magnetic tape before it was recognized that this data represented a complete, accurate, and general solution to the problem. This is an example of one of the extremely valuable but completely unexpected discoveries which occasionally occur in the course of a serious effort made to understand the unknown.

   The solution is quite simple and can be applied on any computer which has a program interrupt capability. It is particularly easy to do in a multiprogrammed system.

   The method used is a high density sampling method. If an executing program is frequently interrupted according to some random or periodic time schedule which is known to be statistically independent of any natural execution pattern in the program, then the frequency with which the interrupt location falls within a particular instruction sequence is proportional to the total *time* spent by the program in executing that instruction sequence.

   To obtain a compute time profile of a program, that program is loaded and run in its normal fashion without any modifications and without any need for prior knowledge of any of its characteristics. By using the typical "timer run-out" feature of the multiprogramming operating system, the program is interrupted frequently during execution. (A one millisecond execution increment gives about 25 sample points per typical I/O transaction on a GE-635.) The address of the next instruction to be executed in the program at the time of the interrupt is recorded for each interrupt and written on magnetic tape. At the completion of the run this tape is sorted by interrupt location address and the resulting frequency distribution is printed. The relative amounts of compute time spent in various parts of the program are determined by comparing the distribution of interrupt locations to the program listing.

   An analysis run with this method takes about one-third more GE-635 computer time than a

normal run of the program being analyzed. At 1,000 sample points per second of compute time, the distribution of compute time within a program is resolved down to the individual instruction execution frequency for all but very infrequently executed routines, which are by nature uninteresting for this kind of analysis. All compute time concentrations within the program are very clearly defined and measured without any need for prior knowledge of where they are.

The "accidental" application of this method occurred because GECOS II normally set the interval timer to interrupt a program after 62.5 milliseconds of compute time. The location of the interrupt was a part of the trace entry which was collected in the Trace Analysis method described in A-3 above. Thus, this data had already been taken in crude, 62.5 millisecond interrupt period, form before the value of the data was recognized.

To be statistically rigorous the fixed, one millisecond sampling interval should be replaced with a random sampling interval with a one millisecond mean, but a fixed interval appears to be satisfactory. Some programs might conceivably have an execution pattern which repeats in synchronism with the one millisecond interval. Such synchronization has never happened in practice but if it did it would immediately be evident from indications of very high execution frequencies at the synchronization points and very low frequencies at nearby points in the same instruction sequences. The sampling interval could then be changed to some non-synchronous value for that program.

This method has been applied to a wide variety of programs and has been found to be a very valuable tool for tuning long running or frequently used programs. The method *finds* many types of compute time performance bugs if they are present and pinpoints the areas in which tuning will be of greatest value. For example, the first application of this method to the FORTRAN compiler led to fixes which increased the speed of the compiler by 27%.

What

In the current implementation both the system and the individual program performance measurements are taken by a small (4K or 6K) program, called MAPPER, loaded as one of the user programs in the GE-625/635 multiprogrammed system. This program originally ties itself into the standard operating system using a special privileged interface and later unties itself upon termination, leaving the operating system in its original form. (Only one instruction in the operating system is changed while MAPPER is operating.) The program is tied into the trace facilities of the operating system and scans all trace entries. Otherwise it uses no processor time unless it is called upon to print a line or write a block of data on tape.

A. MAP

Every two seconds MAPPER samples most of the various accounting time cells in the operating system, subtracts the value from two seconds ago, and prints out a single line showing the percentage of processor and channel time that has been applied to various programs and operating system functions over that two second period. Percentages are rounded to the nearest percent. Values less than .5% are printed as a period. If a user program is present in core but has not used any processor time over a two second interval, then an asterisk is printed for that program. An example of MAPPER output is shown in Figure 1.

Whenever a user program (slave program) is started or terminated the identification of that program is printed and a core map showing the locations of all user programs is printed. Each print position corresponds to 1K of core.

Page averages and run averages are shown at the bottom of the page. The report can be printed in real time on an on-line printer, or sent to the standard system output facility (SYSOUT), or stored on tape for later printing. Normally it is printed on an on-line printer.

This is an analysis report. It is not intended to be a display of all of the interesting data that are available and could be printed. Its objective is to display only that data which are essential to an understanding of what is going on in the system and to display the data on one line so that the vitally important interactions between parallel functions can be seen and understood.

Briefly, a line on the report shows the percentage of processor or channel time applied to various functions over a two second period. Idle processor time is included in dispatcher (DISP) time. The actual dispatcher time ranges from 2% to 5%. Tape channel time can exceed 100% because there are several tape channels available. The only columns which do not show time percentages are:

1. Time-of-day
   The left most column shows time-of-day in hours.

```
PG  97    <PROC><--DRUM->   F <--DISC-> TAPE F  W<----SLAVE PROGRAMS---->        1 1 2 2 2 2 2 3 4 4 4 4 4 5 5 5 5 6 6 6 6 7 M
                            R                 R T                               2 6 0 2 3 4 5 6 4 2 3 4 5 7 0 1 2 3 0 2 4 5 7 0 I
062767     S   M  T  S  M    E   T  S  M   T  E G                                                                            S
10(UP)     L   A  O  L  A    E   O  L  A   O  E      S  S  S  S  S  S  S  S  D  I  G G M G L A M D C B T R C C C C C D T G M G S G C
9341 1     A   S  T  A  S       T  A  S   T     Q  L  L  L  L  L  L  L  L  I  O  G B O E A L T U M O E E A M M M S Y A O E B P
MAP(D)     V   T  A  V  T  L    A  V  T   A  M  U  V  V  V  V  V  V  V  V  S  T  E T R I D O I M N R R L L P P P P Y P L N P E O M
           E   R  L  E  R  K    L  E  R   L  T  E  0  1  2  3  4  5  6  7  P  M  R S E N D C O P 1 T M S L 4 3 2 1 T E L N R R P S

14.0692    75  25 43 24 19       17   17       1  7          21 24 28     1  4  5              2                 1 5 5       . . 3
14.0697    80  20 26 25  1       24 . 24       1  7          28 24 27     1  4  4                                 5 5     1 . .
14.0703    81  19 24 21  3       54  9 46  1  1  7          30 30 20     .  3  4                                 4 5     1  1
14.0713                                                      ---                          FIN    C9292,1-3 FORTRA    14413,R CORLEY
    ----=---+------+------+-------+-------#---4444444444444444444444445555555555555555555555555555555555----#------+-------+-----------+----7777
14.0713    40  60 47  7 41   26 100  4 95       1  8          27 12    .  26  5          1 .       1 1 . 2 5 5     3   .11       . CMP5
14.0714                                                                  +++                SRT    S3473,1-6 RLHS      14422,H.M.SCHUC
    ----=---+------+------+-------+-------#---4444444444444444444444445555555555555555555555555555-666666666666666666666666666667777
14.0719    45  55 40 19 21   25  73     73  24  1  7          35  9  .  1  6  9          18 1 3          .   1 5 5     3 . . 4
14.0724    53  47 33 26  7   25  69     69 130  1  7          40 12  *  .  4  9          21                  1 5 6     1   .
14.0730    36  64 52 22 30   25  22     22 125  1  7          35  *  .  .  7 10          21 . 3 1            1 4 8       . 1 9
14.0735    36  64 63 23 40   24  17     17  98     8          36  .  *  .  12 9          14 1 5 .          . 1 5 4       .11
14.0741    47  53 61 34 27   13   9      9 115     8          44  1  2  .  11 9          12 1 3              1 5 5     . 1 4
14.0746    45  55 48 22 26   13  16     16 128     8          26 11  7  .  6 11          15   5 .            2 4 5     . . 7
14.0752    42  58 55 16 40   12  31     31  39     8          17  9 16  .  9 10          14 . 5 .            1 5 4     .   1 7
14.0758    58  42 35 24 11   11  15     15  42     8          22 15 20  .  5  9          16 .                  5 6     1   .
14.0763    59  40 25 16  9   11  64     64  31     8          25  9 24  .  4  8          17                    5 4     2   .
14.0769    57  44 39 17 23   10  31     31  29     8          17 16 23  .  5  8          13 . 2              1 5 4     1   . 3
14.0774    57  43 26  4 22   10  49     49  31     8          18 13 25  .  5  8          16   2              1 5 5     1     2
14.0780    60  40 36 19 17    9  43     43  27     8          23 13 24  .  5  8          14 . 1                5 4     1   1 1
14.0785                                                      ---                          EJB    A3781,1-5 RLHS      15221,WMSCHUCK
    ----=---+------+------+-------+-------#---4444444444444444444444----#------+-------+-----------+-----6666666666666666666666666666667777
14.0785    42  58 55 12 43   13  58     58  23  2  8          15    25  1  6  9          12 1 1 1      1 2 1   5 5     2 .  15       . CMP5
14.0786                                                                  +++                SRT    C3286,1-5 RLHS      15253,D WORKMAN
    ----=---+------+------+-------+-------#---4444444444444444444444-555555555555555555555555555555566666666666666666666666666667777
14.0791    43  58 40  3 37   12  96     96  18  2  7          4  . 38  1  11 7          12 1 3              1 4 3     4 .  12
14.0796    53  46 55 10 45   12  25 10 15  63     8          12  * 41  .  5  6          4 . 6 .           . . 5 4     .   .14       1 ENBM
14.0802    81  19 32 28  4   12  40  6 34 138     8          39  . 42  .  3  5                .             1 5 2     1 . 1 1
14.0808    72  28 31 20 11   12  50 13 37  91     8          31  1 40  .  4  5                3             1 5 5     1   . 4
14.0813    77  23 36 27  9   12  34  5 26        8          50  * 27  .  5  5                1               5 4     1   1 2
14.0819    65  35 46 41  5   12  53 11 43 200     8          65  .  *  .  14 6                .             1 5 6     1 . 1
14.0824    72  28 35 18 17   12  56  6 50  20     8          30 32  9  .  5  6                2             3 4 4     1   . 3
14.0830    78  22  5     5   12  34  5 29  87     8          18 60  *  .  3  4                      1       . 5 6     1   .   2
14.0834                                                      ---                          FIN    C9291,1-4 FORTRA    14426,GARRETSON
    ----=---+------+------+-------+-------#-------+-------+-------+-----555555555555555555555555666666666666666666666666666667777
14.0835                                                      +++                SRT    C5292,2-4 GELOAD    14413,R CORLEY
    ----=---+------+------+-------+-------#-------+-------+-------+-----444444444444444444444455555555555555555555555555566666666666666666666666666667777
14.0835    60  40 41  4 37   39  62     62  54     8                     40 19  1  5          1 2       . 2 . . 5 5     1   14       . CMP5
14.0841    68  32 40  6 34   39  69     69         8                      2 42 24  1  7  5          4               1 4 5     2 .   5
14.0847    76  24 23 15  8   39  28 17 10         8                     11 27 38  .  4  5          2 1               5 5     1     2
14.0852    81  19 28 28      39  53     53         8                     16 24 40  .  3  5                            5 6     1 .
14.0858    70  30 37 17 20   39  37     37         8                     14 28 28  .  4  5          . 4           . . 6 5     1     4
14.0863    46  54  6     6   39  86     86         8                      4 12 29  .  33 4                      1       5 5     3 .   2
14.0873                                                      ---                          EJB    C9292,2-4 GELOAD    14413,R CORLEY
    ----=---+------+------+-------+-------#-------+-------+-------+-----555555555555555555555555555555566666666666666666666666666667777

PG TOTL    59  41 38 17 20   18  47  3 44  48  .  8                                    8  7          7 . 2 .       . . . 1 5 5     1 . . 5       . ENBM
RUN TOT    45  55  5  2  3   19  39  9 30  20  8  2                                   40  3        . 1 . . . . . . . . . 1 3     2 2 . 1 .   . . SAVE
```

Figure 1 – Internal performance MAP of about one minute of GE-635 operation during a customer's normal work day. RUN TOTALS apply to about 2.3 hours of operation

2. FREE LK
   This is the number of free links of space available on the drum.

3. FREE MT
   This is the number of magnetic tapes which available and have not been allocated to user programs.

4. WTG QUE
   This is the number of user jobs waiting in the external queue.

The column headed SLV 7 shows the time spent on the MAPPER program itself. The column headed IOTM shows the time spent in handling I/O terminate interrupts. This value is inflated by about two-to-one because of extra overhead from MAPPER. The columns headed CMPl-4 are the operating system functions driving on-line printers while DSYT is the system output collector function. GEIN is the on-line card reader input function, which also builds the job queue.

The overall operation of a computer system is as dependent upon what goes on outside the machine in the machine room as it is upon what goes on inside the machine. Thus, the MAP report is normally printed out on-line in real time with the system analyst present and observing the operation of both the internal and external systems.

MAP runs are usually made over periods of several hours of normal system operation at a cost of one on-line printer, 4K of core, and about a 10% reduction in machine performance due to the increased overhead of operation in trace mode. From observation during such runs and later studying the data, the following types of system performance factors can be found and evaluated.

1. Hardware configuration bottlenecks-I/O channels, tapes, drum, disc, core, etc.

2. Indications of inefficient I/O strategy—blocking, buffering, etc., in individual user programs.

3. The effectiveness or ineffectiveness of the actual multiprogramming mix.

4. The effects of machine room procedures upon system performance.

5. The relative efficiencies of various optional strategies of machine operation with enough information to define why this way is better than that one.

6. Possible operating system performance bugs.

B. Major event report

In addition to producing the MAP as described earlier, the MAPPER program will optionally collect all operating system trace entries and write these on a tape. This is a very detailed mode of measurement and is normally done only over short periods, 10 to 15 minutes, of operation. The tape of trace data collected by such a run is then processed to produce a major event report. This report displays events such as dispatching the processor to a program or taking the processor from a program. It also displays all I/O initiate and terminate events and a few other trace events of special significance.

The major event report provides a microscopic view of what is going on in the system. One line on the MAP report expands to about 300 lines on the major event report. This microscopic view is used to measure and understand the detailed operation of the operating system.

C. Slave profile analysis

Another option in the MAPPER program is used to measure the distribution of compute time within user programs. With this option the MAPPER produces the MAP report and also sets the dispatcher interval to one millisecond and collects timer run-out trace events for the grogram being analyzed. These are subsequently processed and used. The MAP for this run provides the I/O-compute profile data for the program being analyzed.

Results

The results of system performance analysis work are both tangible and intangible. The tangible results are those which lead to significant improvements in system and program performance. These have been and are being achieved. The intangible results are the better understanding of the technology which is attained. Ultimately, these may be of the greatest value. This section is, therefore, devoted to some of these significant intangible results.

1. The critical path in a multiprogrammed system

The second by second performance of a multiprogrammed system is *always* limited by the speed of the processor or an I/O channel or by a path through several of these resources used in series, as with an unbuffered program which reads a tape, then computes, and then writes a tape. Thus, for every line on a MAP report, if some single limiting resource is not saturated, there must be a performance limiting critical path through some series of resources whose total utilization adds up to 100%. Such a critical path is the only mechanism by which each of all the resources in the system can be partially idle over some short period of time during which there is a load on the system.

2. Resource Utilization Strategy

For all practical purposes it is *never* advantageous to arbitrarily defer processor or I/O channel use to some future time if that resource can be used now. The idle resource time generated by such an action can never be recovered. An example might be a program which could overlap processor and I/O time with double buffering but does not, assuming that some other program will use the free processor and channel time. This is like betting a dollar that you already have, on a chance that you *may* win only that dollar back. You can only lose or at best break even.

3. Resource-Seconds Analysis

In a multiprogrammed system all programs share the same set of resources. Each program imposes a load of some number of resource-seconds upon each resource. Critical resource tradeoff decisions can be made on the basis of which approach uses the least critical resource-seconds. Two tradeoff examples follow.

(a) How many tapes should be used for a tape sort given that the sort time is dependent upon the number of tapes used? The critical resource is the number of tape drive-seconds available on the system. A plot of number of tapes used times the sort time for that number of tapes should be made. The sort that uses the fewest tape drive-seconds is the optimum in a shared resource system.

(b) How much core should be used for a program given that the total running time for that program is dependent upon the amount of core used? The critical resource is the total number of core

word-seconds available. The running time for each of various core size alternatives should be determined. The alternative which uses the least total core word-seconds is the optimum in a shared resource system.

Extreme cases, such as the program whose optimum core size is all of the available core, should be studied a little more carefully. In extreme cases several critical resources, core, tapes, etc., may be involved in the problem. When this occurs a resource-second weighting factor, such as the cost of the resource, should be assigned to each resource. Then the optimum is the combination which uses the least total weighted resource-seconds.

For even more rigorous analysis, *unused* resources may be weighted with the probability that they will be used by some other program. Usually the tradeoff values are sufficiently well separated that this degree of accuracy is not needed.

4. Core utilization

In the GE-625/635 system each program must be allocated to a block of contiguous core. Core is compacted as required by moving programs in core to make space for new programs. The significant point here is that this is *not* a problem. Measured core utilization is quite high and core compaction (program moving) time is insignificant, well under 0.5%.

5. Operating system overhead

A multiprogramming operating system must perform many functions which are not required in a uniprogramming operating system. But as measurements of the GE-625/635 GECOS II operating system demonstrate, all of these functions can be achieved in a *low overhead* operating system. Total overhead averages about 15% of processor time.

6. Measurements by customers

Using normal accounting or billing statistics, the customer can measure the gross performance of a multiprogrammed computer system in substantially the same way as he would measure the performance of a uniprogrammed system. In a uniprogrammed system these individual job statistics measure the performance of the total system and the efficiency of each program as each individual job is executed. The gross measures are simply the sum of a number of independently valid individual job measures.

But in a multiprogrammed system several programs are operating concurrently in core. The total time spent by any one program in core is dependent upon not only the work done by that program but also upon the degree of its competition for resources with the other programs in core. Thus, the individual job measurements usually measure neither system performance at a given time nor the efficiency of individual programs while they were in core. The customer can measure gross performance but he cannot measure detailed system or program performance with these statistics. For example, the amounts of compute and I/O channel time used by a program may be measured but there is no indication as to whether or not these resources were used concurrently. As another example, set-up time usually disappears as a measure but not as a problem.

Overall we find that when conventional, uniprogrammed type measures are applied to a multiprogrammed system the customer has lost many of his previous measures of system, operator, and program performance. But he is actually operating a considerably more complex system and needs better, not fewer measures. Thus, system and program performance measurement methods of the type discussed in this paper appear to be essential to the customer's effective operation of his multiprogrammed computer system.

7. Load mix – the top ten

If the total time per month used by each program in an installation were computed and the results sorted by the amount of time used, how much of the total load would be accounted for by the top ten programs? Typically, the top ten programs appear to account for from 50% to 80% of the load. (Actually the "top ten" may turn out to be the top five or the top twenty at any given installation, but the same principle applies.)

The "top ten" concept implies that the efficiency or throughput of an installation is from 50% to 80% determined by the efficiency of a very few programs. These programs are worth tuning, particularly when it is recognized that performance tuning can and has reduced program running times by two or three to one.

The top ten programs are worth the application of some expert talent and the use of program analysis methods such as those described in this paper. This is the easiest, fastest, and least expensive way to increase the capacity of a computer installation.

8. Turn around time

When a program arrives in the machine room it is almost sure to find that one of the top ten is there ahead of it. After all, one or another of the top ten is on the machine most of the time. In a uniprogrammed system almost everybody has to wait, almost every time, for one or more of the top ten programs to be finished.

But in a multiprogrammed system, many short jobs can be loaded and finished in parallel with the execution of one of the top ten. They no longer have to wait, as long as all the available core is not filled with

several of the top ten, all competing for the same resources.

This suggests a good operating procedure for a multiprogrammed system. The long jobs should *not* be saved for the second or third shifts. The top ten jobs should be identified. All processor saturating combinations of *one* or *two* of these top ten programs should be determined. Then the operators should load the machine so that *one* of these combinations, no more, no less, is in the machine at all times. The short jobs will still receive excellent turn around time and there will normally be a new top ten combination waiting to replace the one that has just finished.

CONCLUSION

Methods have been developed for analyzing the performance of a multiprogrammed computer system and the programs which operate within that system. Some of these methods, and many of the specific measures and reports developed do not apply directly to other systems but the general approach used does apply. The tangible and intangible results produced from the study reported here demonstrate the value of performance analysis. The methods and philosophy may be helpful as an example of how to do it.

REFERENCES

1 P CALINGAERT
System performance evaluation: survey and appraisal
Comm ACM 10 1 Jan 1967 pgs 12-18

2 N R NIELSEN
The simulation of time sharing systems
COMM ACM 10 7 July 1967

3 R W HAMMING
(Paraphrased by the author)

4 F D SCHULMAN
Hardware measurement device for IBM system/360 time sharing evaluation
Proc 22nd ACM Conference 1967 pgs 103-109

5 GE-625/635 COMPREHENSIVE OPERATING SUPERVISOR (GECOS) REFERENCE MANUAL
General Electric Information Systems Division CPB-1195

Multiprogramming, swapping and program residence priority in the FACOM 230-60

by MAKOTO TSUJIGADO
Fujitsu Limited
Tokyo, Japan

INTRODUCTION

FACOM 230-60 is a large size electronic digital computer developed by Fujitsu Limited. The system consists of

1) up to 2 processing units,
2) a 256k word (maximum) high speed core memory that operates at a 0.92 μ sec. cycle time, or at an effective cycle time of 0.15 μ sec. with 16 memory banks and
3) a 768k word (maximum) low speed core memory that operates at a 6.0 μ sec. cycle time, or at an effective cycle time of 1.0 μ sec. with 6 memory banks.

The average duration for execution of one instruction by the Gibson-Mix method is 1.6μ sec.[1] for each processing unit. Because each processing unit has 6 base-registers, it is easy to write a location-independent program which makes dynamic relocation of a program possible. The most commonly used large random-access storage devices are: magnetic drum with 544k words, a mean access time of 17 m sec., and transmission rate of 135k bytes/sec.; magnetic disk with 20,000k words, a mean access time of 130 m sec., transmission rate of 125 kilo bytes/sec.; and disk pack with 1,800k words, a mean access time of 87 m sec., and transmission rate of 156k bytes/sec. For the FACOM 230-60 Operating System, system requirements include real-time job processing and conversational job processing, in addition to the conventional batch job processing. To control these three processing modes together under a single control program, two concepts have been adopted in the system design. They are the multi-tasking operation as well as three priorities, i.e., job priority, execution priority and program residence priority.

This paper introduces the program residence priority concept.

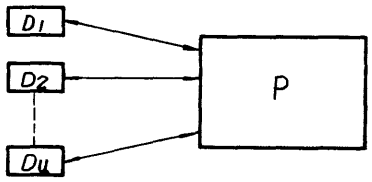
FACOM 230-60 multiprogramming requirement

To show why the system requires the multiprogramming property, consider the abstract computer indicated in Fig. 1. Assume that this model is required to handle u programs simultaneously, and that each program is processing quantity Di input and output data The following symbols are used:

- u: total number of jobs being executed in a system
- D_i: quantity of input and output data to be processed by each job (unit is 1 character)
- D: average value of Di above, i.e.,

$$D = \frac{1}{u}(D_1 + D_2 + \ldots + D_u)$$

- P: data quantity to be processed by the processing unit (characters)
- H: system overhead
- g: average time for execution of one instruction by the Gibson-Mix method (seconds)
  For one processing unit, g = 1.6 μ sec.
  For two processing units, g = 0.9 μ sec.
- s: number of instruction steps needed to process one character of input or output data.



An abstract model of a system simultaneously executing u jobs, each processing Di (i = 1, 2,...,u) quantity of input and output data.

Figure 1 – System model

The relation between D_i and P with overhead H is given in the following expression.

$$\sum_{i=1}^{u} D_i = (1 - H) \cdot P$$

When $\sum_{i=1}^{u} D_i > (1-H) \cdot P$, the processing unit causes the input and output devices to slow down, or loses a part of the input data, since the processor can not complete the processing of all the input and output data given.

When $\sum_{i=1}^{u} D_i < (1-H) \cdot P$, idling occurs in the processing unit.

Since $uD = \sum_{i=1}^{u} D_i$, differentiating the equation with respect to t,

$$u \frac{dD}{dt} = (1-H) \frac{dP}{dt} \tag{1}$$

where $\frac{dD}{dt}$ is the average quantity of the input and output data processed by one job in a unit time, and $\frac{dP}{dt}$ is the data quantity processed by processing unit P in a unit time.

On the other hand, from the dimension equation,

P [ch] · s [step/ch] · g [sec/step] = t [sec]

The following equation can be obtained by differentiating with respect to t.

$$\frac{dP}{dt} = \frac{1}{s \cdot g} \tag{2}$$

From Equations (1) and (2)

$$u = \frac{1-H}{\frac{dD}{dt} \cdot s \cdot g} \tag{3}$$

In Equation (3), u, the number of jobs simultaneously executed by the system, is given as a function of $\frac{dD}{dt}$ (average quantity of input and output data processed by each job in a unit time), s (number of steps needed to process one character of input or output) and H (overhead).

Further consequences of Equation (3):

1) In batch mode jobs

1.1) When card reader and line printer are used,

Card reader speed = 800 [cards/min]

$$= \frac{800 \text{ [cards/min]} \times 80 \text{ [bytes/card]}}{60 \text{ [sec/min]}}$$

$$= 1064 \text{ [bytes/sec]}$$

Line printer speed = 1500 [lines/min]

$$= \frac{1500 \text{ [lines/min]} \times 136 \text{ [bytes/line]}}{60 \text{ [sec/min]}}$$

$$= 3450 \text{ [bytes/sec]}$$

$$\frac{dD}{dt} = 1064 + 3450 \fallingdotseq 5000 \text{ [bytes/sec]}$$

therefore

$$u = (1-H) \frac{1.25 \times 10^2}{s}$$

1.2) When a magnetic tape handler with an effective speed of 45k (bytes/sec) is added to the above case,

$$\frac{dD}{dt} = 45{,}000 + 5{,}000 = 5 \times 10^4 \text{ [bytes/sec]}$$

$$u = (1-H) \frac{12.5}{s}$$

1.3) When ten magnetic tape handlers with an effective speed each of 62.5k (bytes/sec) are used in each job,

$$\frac{dD}{dt} = 62.5 \times 10^3 \times 10$$

$$= 6.25 \times 10^5 \text{ [bytes/sec]}$$

$$u = \frac{1-H}{s}$$

2) In real time processing

When 1,000 transmission lines with a speed of 1,200 baud are processed by one job,

$$\frac{dD}{dt} = 120 \text{ (bytes/sec)} \times 1000$$

$$= 1.2 \times 10^5 \text{ [bytes/sec]}$$

$$u = \frac{5.2(1-H)}{s}$$

3) In conversational mode job

It is safe to estimate that $\frac{dD}{dt}$ would be 2(bytes/sec) since it would take about one second per character to type-in the data and also to read the printed data.

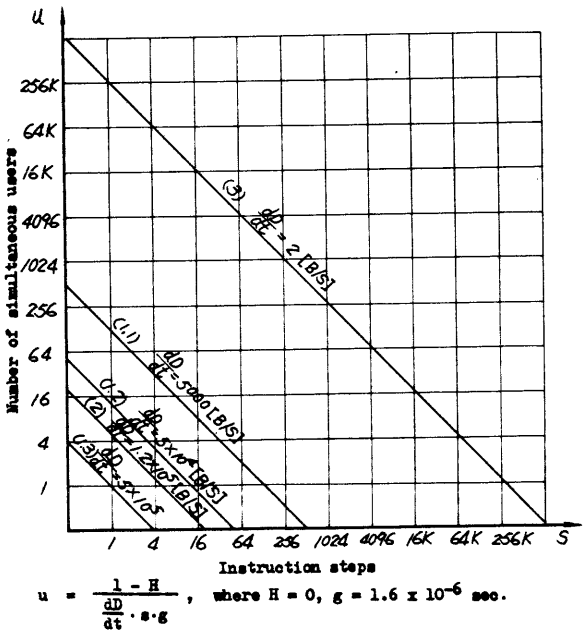
$$u = \frac{3.1 \times 10^5 \times (1-H)}{s}$$

Note: Program swapping affects H and is independent of

$$\frac{dD}{dt}$$

The above relations are shown in Fig. 2, where H is assumed to be zero.

On the other hand, from the experience with the FORTRAN compiler of FACOM 230-50, which is smaller than FACOM 230-60, the value of s is presumed in the range from 32 to 128. So, in batch processing, even when only a few jobs are simultaneously executed while read-to-random access storage and print-from-them multiple operations are concurrently performed, the idle time will no longer be available to the processing unit. In the conversational mode, however, more than 256 jobs can simultaneously be executed, even if each job needs such a large quantity of information processing that, with the overhead, s = 512.



The relation between u (number of simultaneous users) and s (number of instruction steps needed to process one character of input or output data) at each fixed value of the parameter $\frac{dD}{dt}$ (input and output quantity processed in the unit time)

Figure 2 – Job multiplicity

Analysis of swapping time

From the preceding section, it is concluded that the FACOM 230-60 System could simultaneously process 256 jobs in the conversational mode. In this case, simply assuming that each job requires an area of 32k words, the core memory of 8192k words would be necessary to execute 256 jobs. It is impractical to make such a large core memory, so the situation naturally calls for program swapping techniques. In this section, we will discuss the idle time of the processing unit (and of the system) caused by swapping. The symbols below are used in the following discussion.

- u: number of on-line users who utilize the processing unit once in each response cycle.
- t_r: response cycle, a period of time wherein each user utilizes the computer at least once (seconds)
- f: average period of time assigned to a user's program in each response cycle, including the overhead.
- e: average period of time excluding the overhead assigned to a user's program in each response cycle, that is

$$e = f \cdot (1 - H)$$

- t_s: average period of time for a swapping (one way)
- t_i: average period of time for a roll-in ($= t_s$)
- t_o: average period of time for a roll-out ($= t_s$)
- H: overhead; the ratio of the number of program steps, in a control program, either for services or for idling, to the total number of program steps.
- d: number of data channels available for the program swap.
- d_a: number of data channels actually provided for the program swap.
- n: number of jobs in the main memory at a certain time.
- L: average length of users' programs which correspond to jobs (K words)
- V: transmission rate of one random access-storage (K Bytes/Sec)
- A: average access time of one random access storage (seconds)

As pointed out by J. I. Schwartz,[2]

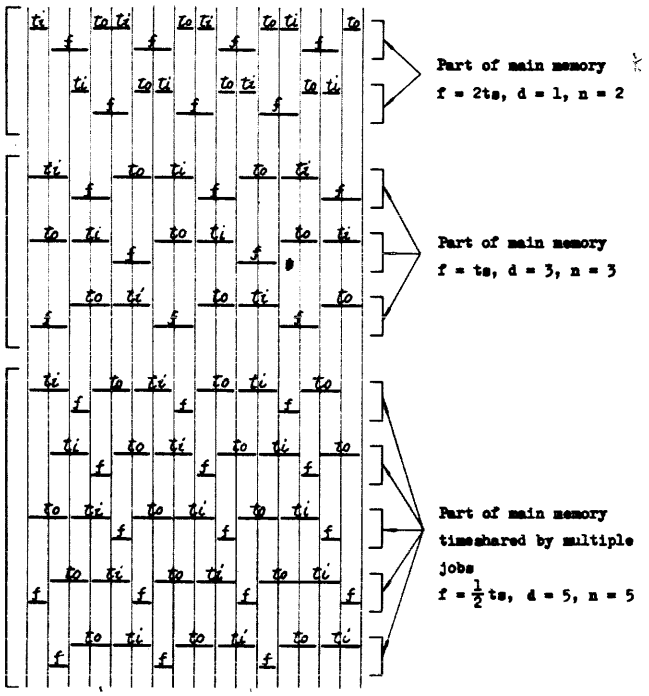
$$u = \frac{t_r \cdot (1 - H)}{2t_s + e} \text{ in worst case}$$

If, however, a sufficiently large number of data channels and enough random access storages to swap programs smoothly are provided, and the system can handle the dynamic relocation of a program, u can be written as

$$u = \frac{t_r}{f} = \frac{t_r}{e}(1 - H) \qquad (4)$$

In order to obtain the relation between the swapping time and the execution time of the processing unit, the time chart in Fig. 3 is used, since the channels transfer data one way at a time. From the Fig. 3,

$$t_i + f + t_o = n \cdot f \tag{5}$$



The relation between program swapping time, execution time and multiplicity of jobs in the main memory.

ti: period of time for roll-in (= ts)

to: period of time for roll-out (= to)

f: period of time for execution, including overhead

Figure 3 – Main memory operating diagrams

Equation (5) is valid when the capacity or power of the data channels is equal to that of the processing unit, and, if the former is greater than the latter,

$$t_i + f + t_o < n \cdot f$$

and, no idle time will be available in the processing unit. When the capacity of the data channels is less than that of the processing unit,

$$t_i + f + t_o > n \cdot f$$

and, idling will occur in the processing unit. In order to use the processing unit without idling, conditions $d \geqq 2$ and $n \geqq 2$ are necessary when $f < 2t_s$. In the FACOM 230-60 Operating System operating in the conversational mode, the condition $f < 2t_s$ will always hold even if a high speed drum (V = 1000 (KB/S), A = 0.017 (S) is attached. Therefore we assume $d \geqq 2$ and $n \geqq 2$ in the following.

From Equation (5)

$$2t_s = (n-1) \cdot f = \frac{(n-1) \cdot e}{1 - H} \tag{6}$$

Further, the next relation is provided between the average size of programs each constituting one job, the data transfer rate and the average access time of a random access storage, and the program swapping time

$$L\,[KW] \cdot 4[Bytes/Word] = V[KB/s]\,(t_s[s] - A[s])$$

or

$$t_s = A + \frac{4 \cdot L}{V} \tag{7}$$

From Equations (4), (6) and (7) and since n = d

$$n = d = 1 + \frac{2 \cdot A \cdot u}{t_r} + \frac{8 \cdot u}{V \cdot t_r} \cdot L \tag{8}$$

If A = 0.017 [S], V = 135 [KB/S], t_r = 16 [S] (since an on-line user carrying out a FORTRAN compilation with a typewriter normally takes 16 seconds on the average to type-in one statement; the value 16 or so is sufficient for t_r), the following expressions are obtained according to u, or the number of on-line users.

1) In the case of 64 on-line users,
   $n = d = 1.14 + 0.24 \times L$
2) In the case of 128 on-line users,
   $n = d = 1.3 + 0.5 \times L$
3) In the case of 256 on-line users,
   $n = d = 1.5 + 0.9 \times L$

The three cases are illustrated in Fig. 4. In addition, when A, V and u are assigned 0.017, 1000 and 256, respectively

$$n = d = 1.5 + 0.128 \times L$$

The relation is illustrated in Fig. 4.

It was concluded from the above analysis that providing a sufficiently large number of data channels and of random access storages for the program swap should be abandoned and that the remaining time should be applied to batch mode jobs to eliminate idle time caused by reduction in the number of data channels used.

The idle time rate increases in a processing unit as the number of data channels actually used decreases. From Fig. 3,

$$\text{the rate of idle time} = \frac{d - da}{d} \times 100\ [\%]$$

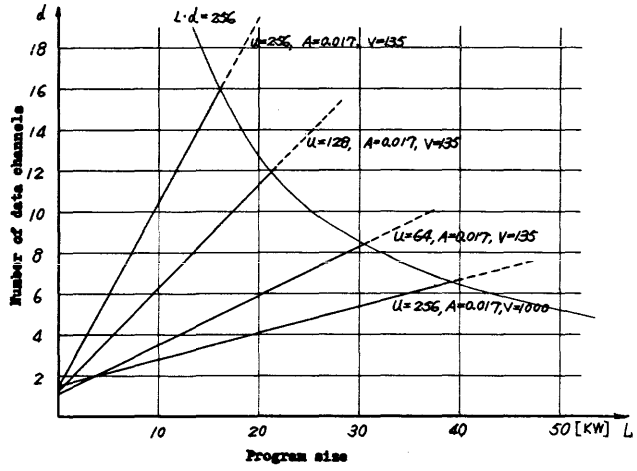


Figure 4 – Relation between data channels and program size

From the relation in Equation (8)

$$L = \frac{V}{4}\left(\frac{n-1}{2} \cdot \frac{t_r}{u} - A\right)$$

For $t_r = 16$ [S], $u = 128$, $A = 0.017$ [S],
$V = 135$ [KB/S]

$$L = 34\left(\frac{n-1}{16} - 0.017\right)$$

Figure 5 shows the rate of idle time considering $L \cdot d_a = 256$[KW]. In addition, the number of on-line users simultaneously accessing the system will be obtained from Equation (8). (d must always be greater than 1 as discussed between Equations (5) and (6), and, from the Fig. 3, d = 3, 5, 7, . . .)

$$u = \frac{d-1}{2} \cdot \frac{t_r}{A + \frac{4L}{V}}$$

Therefore, for any number of data channels actually provided,

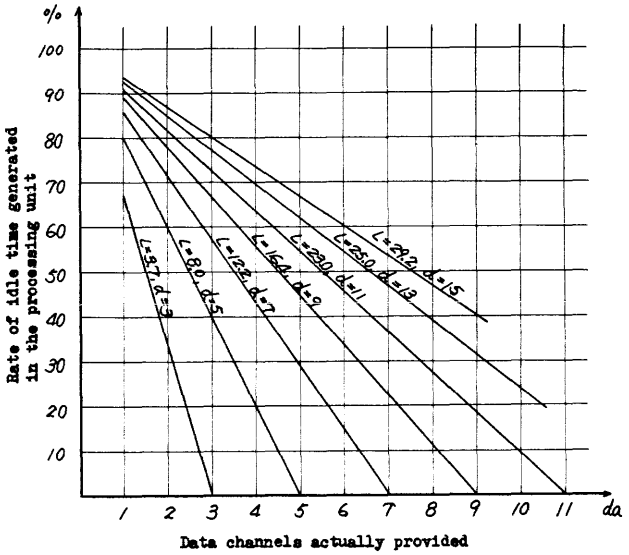
$$u = \frac{da_t}{d} \cdot \frac{d-1}{2} \cdot \frac{t_r}{A + \frac{4L}{V}} \qquad (9)$$

where d must be 2m + 1 when $d_a = 2m$ or 2m +1 for m = 1, 2, 3, . . .

In case of A = 0.017, V = 135

$$u = \frac{d_a}{d} \cdot \frac{d-1}{2} \cdot \frac{t_r}{0.017 + 0.03L}$$

and in case of A = 0.017, V = 1000



The relation between the number of data channels actually provided for program swapping (da) and the rate of the idle time generated in the processing unit.

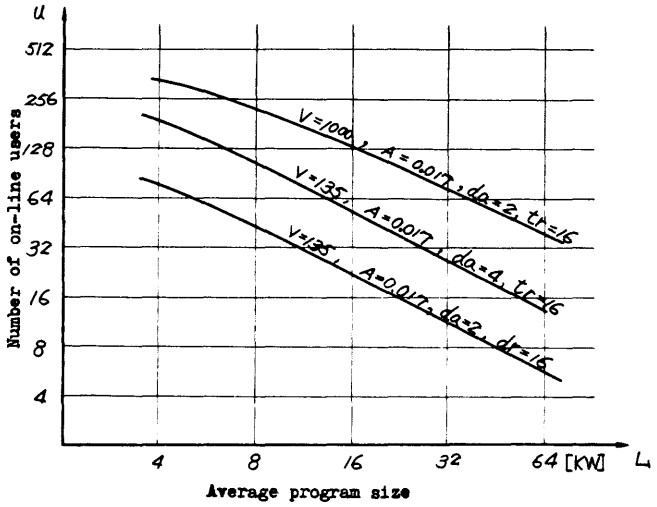
L: average length of programs

d: desired number of data channels for program swapping

Figure 5 – Idle time ratio

$$u = \frac{d_a}{d} \cdot \frac{d-1}{2} \cdot \frac{t_r}{0.017 + 0.004L}$$

The relation is shown in Fig. 6.



V: data transfer rate

A: average access time

da: number of data channels actually provided for program swapping

tr: response cycle

Figure 6 – Relation between number of on-line users and program size

The division of main memory space between batch jobs and conversational jobs is presented as follows. The size of main memory space for conversational jobs is determined by $d_a \cdot L$, and that for batch jobs is determined by (the total main memory size $-d_a \cdot L-$ the size of the area for the control programs). The batch job control program requires the job-step initiator to allocate jobs until the main memory area reserved for batch jobs fills up, while the conversational job control program requires the job-step initiator to allocate jobs until the number of active users reaches u, which is calculated by Equations (9).

Introduction of program residence priority

Figure 5 indicates that over 60 percent idle time is generated in the processing unit with 4 data channels for program swap when the average size of programs in the conversational mode is 20k words. Thus, the system should not be operated in the conversational mode alone, but together with the batch mode. Furthermore, program swapping, i.e., swapping of jobs, should not be used for batch mode processing. These are the reasons why program residence priority has been considered in the design of the FACOM 230-60 operating system.

In the system, one load module consists of up to 127 program blocks. This is the unit of swapping, to which the program residence priority is attached. The program residence priority determines the priority for staying in the main storage between program blocks when competition occurs. For example, in conversational mode jobs, a higher execution priority and a lower program residence priority than for the batch mode jobs are to be set respectively. With this priority system, the conversational mode task is executed before execution of the batch mode task, and the former jobs are rolled out before the latter jobs are rolled out. Generally, the priorities will be determined at each installation, as in Table I.

Before introducing the concept of program residence priority into the system design, the following method was applied. The total main memory capacity for conversational jobs is prescribed by the administrator, or at the system generation time, and the rest of the memory is allocated to batch jobs. Conversational jobs are always rolled out immediately after they are put into the waiting status. However, for control program logic simplicity and in order to execute batch jobs and conversational jobs under a single control program, the program residence priority concept was chosen. Arbitrarily, about 10 percent of the control program would be a special control program for batch jobs and about 15 percent would be for conversational jobs.

	Execution Priority	Program Residence Priority
Real Time Job	High	High
Conversational Job	Medium	Low
Batch Job	Low	Medium

TABLE I – Normal Usage of Priorities

ACKNOWLEDGMENT

Dr. Toshio Ikeda of Fujitsu Limited pointed out that resources should first be allocated to both real time mode jobs and batch mode jobs, with the remaining resources allocated to the conversational mode jobs. Mr. Takeshi Maruyama of Fujitsu Limited introduced the program residence priority concept. Thanks are due to them and to Mr. Isao Saoda, Mr. Takuma Yamamoto, Mr. Akira Okamoto and Mr. Haruo Kunizawa for their encouragement and assistance.

REFERENCES

1 N UKAI
Operation analysis of FACOM 230-60 by system simulator SOL
Proceedings of the 1967 Conference of the Four Electrical Associations in Japan

2 J I SCHWARTZ E G COFFMAN C WEISSMAN
A general-purpose time-sharing system
Proceedings of the 1964 Afips Spring Joint Computer Conference

A storage-hierarchy system for batch processing

by DAVID N. FREEMAN
Triangle Universities Computation Center
Research Triangle Park, North Carolina

Fundamental performance problems

Operating System/360 was designed to meet a severe core-memory constraint: a 14K-bytes resident supervisor plus a repertoire of compilers, utility programs, sort programs, and application packages fitting into 18K bytes (approximately 4500 data words and executable instructions). Many supervisory functions included in the nucleus of pre-360 systems were repackaged into 1000-byte overlays for OS/360 (e.g. logic to OPEN and CLOSE files—hereafter called *data sets*, following OS/360 nomenclature).[5] Specification of device type, buffering technique, and data set identification—which was assembled, compiled, or link-edited into many pre-360 application programs—is deferrable in OS/360 until the data set is actually opened for processing, essentially "latest-possible binding of data-set attributes and processing mode" (cf. Part 3 of Reference 5 for a complete discussion).

Likewise, the assemblers, compilers, and utility programs offer a wide range of language facilities for relatively small machines. In particular, the *E-level* compilers require approximately 18K bytes to match a 14K resident supervisor in a 32K machine; the *F-level* compilers require approximately 44K bytes, to match a 20K supervisor in a 64K machine; the *G-level* Fortran compiler requires approximately 80K bytes in a 128K machine; and the *H-level* Fortran compiler requires approximately 210K bytes in a 256K machine.

The gross effects of this packaging of the supervisor, compilers, and other programs has produced three major performance problems in large-memory, fast-CPU systems: *reliability, operator-intervention losses,* and *system I/O inefficiencies.*

Reliability

Many core-resident subroutines of pre-360 systems were divided into multi-overlay structures in OS/360, initially riddled with logical errors and interface inconsistencies. Although these errors are continually being identified and corrected by IBM—with the co-operation of its customers—the aggregate reliability is barely satisfactory today. Our experience is similar to that of many OS/360 users: the system frequently encounters an uncorrectable software error and stops dead, whence it must be reloaded.

On small systems, job losses from these dead stops are typically of less consequence; the operator reloads the system and restarts the job that failed.

On a large communications-oriented system, dead stops are intolerable; the TUCC system is continually sending/receiving jobs from 5-25 satellites simultaneously, and re-transmission of jobs is rarely completely successful, e.g., jobs are lost, partially processed, processed twice, etc. In an early version of the TUCC communications package—comprising both IBM- and locally-written subroutines—we found that 30% of all jobs were reruns, incurred by some failure of the collection/processing/distribution network (shown in Figure 1, "The TUCC Computer Net-

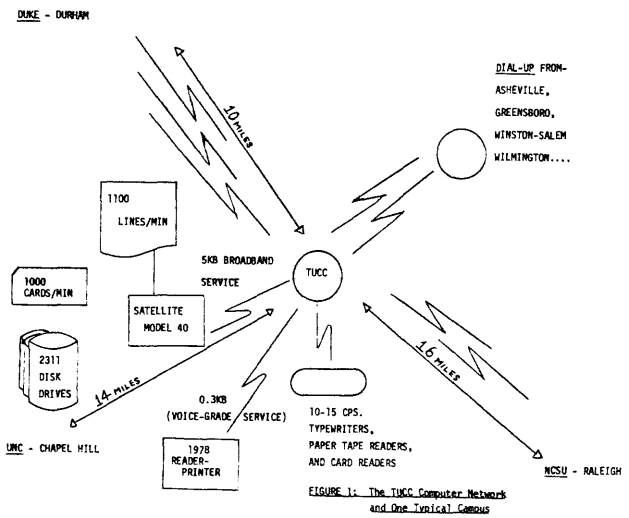


Figure 1—The TUCC computer network and one typical campus

work and One Typical Campus" and Figure 2, "Disk Queuing for Job Flow between TUCC and a Typical Satellite Computer").

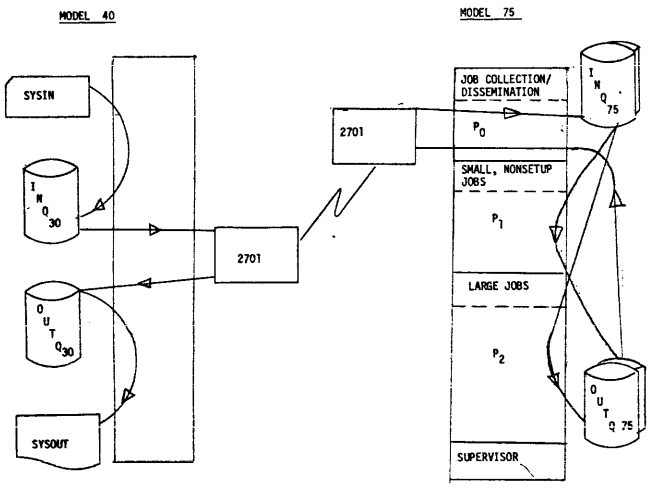


Figure 2 – Disk queuing for job flow between TUCC and a typical satellite computer

Although this loss rate has dropped considerably during 1966-67, further improvements in system reliability are mandatory for better user throughput, i.e., as experienced by the remote customer rather than as measured only by gross efficiency of the central computer.

In many cases, job losses are attributable to unusual hardware-related events, e.g., unforeseen status signals from communications lines.

For our collection/processing/distribution programs, we have therefore developed three guidelines:

- Make minimal changes to IBM programs, relying on IBM's gradual rehabilitation of their programming support into error-free status;
- Restrict modifications to those subroutines whose alteration can produce important performance improvements;
- Evolve from each current system into the next production system (rather than install sweeping changes), to minimize dead stops and lost jobs in the network.

Operator-intervention losses

OS/360 requires significant operator intervention for the following situations:

1. The system must be re-loaded after a dead stop.
2. A tape reel or disk pack must be mounted, possibly displacing another reel or pack.
3. A job has reached some logical dilemma which can only be resolved by the operator (e.g. no tape drives remain for a reel-mounting need).
4. A job requests information/acknowledgment from the operator, (e.g., authorization to overwrite an *unexpired* data set – one normally retained until a pre-specified date).

For situation (1), TUCC has added significantly to IBM-supplied software, to furnish a faster, more reliable checkpoint/restart facility based on a *checkpoint hierarchy*. Multiple checkpoints is a well-established concept, offered by several operating systems to permit flexible roll-back and restart operations. The TUCC hierarchy offers the following facility: large capacity core storage (LCS)* contains at all times the essential job-status data. Restart procedures incur no worse than the following inefficiencies:

- Jobs currently being transmitted to TUCC must be re-transmitted, but only the last job from each terminal;
- Job output being transmitted from TUCC must be re-transmitted, but only the last block of print images per terminal;
- Jobs currently in process at TUCC can be restarted or skipped.

The advantage of LCS for checkpoints is obvious; they can be taken more frequently than with disks, drums, or tapes with far less processing overhead. Furthermore, LSC is "safe." During a three-month evaluation of dead stops (10 to 15 per day), LCS checkpoints were always intact. This peculiar "safety" is attributable to the TUCC modifications to OS/360; most of LCS is "concealed" from the OS/360 main-storage supervisor, which furnishes core blocks to the supervisor, compilers, and application programs. Only certain TUCC-written routines can access the upper 1700K-bytes of LCS, wherein are stored checkpoint data (and other functions described below). Altogether, reloading from dead stops has been made faster and less cumbersome for human operators.

Time losses from intervention situations (2) - (4) above can be alleviated only by ample advance notice; mounting/dismounting tapes or disks is necessary, but painfully slow on a high-performance system. No job tickets accompany source decks submitted from remote terminals – tickets usually essential for rapid set up of private reels and disk packs in a non-satellite job-shop facility.[6]

Recognizing this problem, TUCC has written "job-hold" logic into its job-manager subroutine, such that volume-setup messages are issued 100-200 seconds before each setup job is due for processing. Only

* The IBM LCS is available in one- or two-million byte modules, either uninterleaved or two-way interleaved. The TUCC system has two million bytes uninterleaved. Access time is 3μs, full cycle time is 8μs, the rental cost is 0.6¢/byte/month. By contrast, the fast core storage on a Model 75 is normally four-way interleaved, with an access time of 0.4μs, full cycle time of 0.75μs, and a rental cost of 3.7¢/byte/month.

when the operator has performed this setup to the system's satisfaction—correct volumes in correct status (e.g., a private data reel with ring out) on correct drives—will the job be dispatched.

No core storage or other resources are committed to "held" jobs other than their source-program image area on disk. The TUCC system allocates 28 million bytes of disk space to queue jobs awaiting processing; with the text-compression algorithm of Reference 7, this becomes effectively 80 million bytes, i.e., 1,000,000 card images. Based on our measurements of average source-deck size—approximately 300 cards—this suffices for 2500 jobs at point INQ75 in Figure 2, assuming 25% aggregate track wastage. (Each job is allocated an integral number of tracks to improve scheduling flexibility and reduce job losses.) Approximately two days' work can be queued on a single disk pack, which has proved to be a more-than-adequate reservoir.

System I/O inefficiencies

We noted above that OS/360 performance suffers from heavy I/O activity supporting:

a. Supervisory services. OPEN, CLOSE, and various interruption handlers have been divided into 1000-byte overlays which flow through a small number of core buffers. Typically 70K bytes of overlays flow through 5K-10K bytes of core, each subroutine being overlaid when its core is reclaimed by another subroutine.

b. System data sets, such as macros for the job-control function (PROCLIB) and disk workspace for the control program (SYSJOBQE);

c. Multiple overlays of the job scheduler, linkage editor, and compilers. The demand rate for these overlays is extremely high in a multiple job-stream environment, and typical OS/360 systems spend a significant fraction of clock time awaiting their retrieval from secondary storage.

In References 8-11, TUCC described this system-I/O problem and presented its three-element strategy for alleviating the problem:

(i) Pseudo-readers, pseudo-punches, and pseudo-printers, which simulate real I/O devices using a combination of program logic, LCS, and disk queue space:

(ii) Pseudo-disk, which simulates a single-drive 2314 using a different combination of logic, LCS, and real disk storage;

(iii) A monitor for small nonsetup jobs, which represent an especially significant load on a multi-university system.

Two fundamental guidelines—based on the performance of our system and other large 360s—are as follows:

1. Only LCS is a sufficiently fast source and sink for system I/O on a 360/75 system. The fastest conventional rotating device (for S/360, the 2301 drum) interlocks processing too long for satisfactory CPU utilization, under normal TUCC operating conditions.

2. After re-assigning most system I/O from disk/drum to pseudo-devices (in LCS), sufficient I/O activity has remained to permit additional job processing using conventional multiprogramming. At TUCC, multiple job streams use distinct pseudo-devices concurrently, each yielding CPU control when it must await an I/O completion (for a real device, of course). Multiprogramming concepts are required in most third-generation operating systems, where a large CPU performs concurrent peripheral operations as well as multiple, unrelated jobs. TUCC modifications to OS/360 focus first on improving the *speed* of system I/O to improve aggregate throughout; by contrast, IBM's Multiprogramming with Variable number of Tasks (MVT) option in OS/360 focuses first on *multiprogramming*.

Our justification may be peculiar to our job mix and operating characteristics, although we suspect they resemble those of other universities and scientific establishments. Unmodified, our disk-oriented OS/360 system spent 85%-90% of clock time in *CPU-Wait* state (i.e., awaiting the completion of one or more I/O events. This figure was obtained by numerous readings of a home-made CPU meter, which integrates the CPU-waiting signal over a one-minute interval). Clearly, running two job streams of similar characteristics could not reduce CPU-Wait below 70%.* Considering channel contention and unit contention for one-of-a-kind data sets like the processor library (LINKLIB), it seems improbable that CPU-Wait could be reduced below 80%. Although our job mix and equipment are significantly different from TSS running on the 360/67, CPU-Wait under OS/360 resembles the "page-wait" problem analyzed by Neilson,[12] Lauer,[13] and Smith.[14] In particular, Neilson discovered by simulation that a zero-latency disk—essentially a specification of TUCC hyperdisk—would significantly raise the aggregate throughput of TSS on the 360/67. Lauer used LCS as a paging device after demonstrating that a conventional drum could

* Twice the inefficiency (15%) of a single job stream.

not deliver pages at a rate to keep the CPU satisfactorily utilized.

If CPU-Wait could be reduced to 60% *and* if channel and device contention could be averted, two job streams should reduce CPU-Wait nearly to 20%. Three job streams could reduce CPU-Wait still further, although IBM claims to have demonstrated—by analyses and simulations not yet publicly available—that more than 3-4 job streams cannot raise the gross throughput of an average 360/75-class system running a conventional job-shop mix. We are currently measuring CPU-Wait against core requirements for each job stream. If CPU-Wait remains high, we will consider adding fast core storage so that additional job streams can utilize otherwise-idle CPU time. In this respect, the storage-hierarchy system and MVT concur on how to convert CPU-Wait to productive operation.

Early-1968 readings of CPU-Wait range between 30-35% with a single job stream, less than half the figure with unmodified OS/360.* More important is the complementary figure: CPU activity has been raised fivefold. Some of this activity supports locally-written code and adds to the CPU overhead of the total system. Indeed, 95% of the I/O operations in the system are now simulated by locally-written code: the pseudo-devices cited above and described in the following section. Nonetheless, the gross number of jobs per hour has been raised significantly: this is the true measure of system throughput. The addition of minor CPU overhead has acted as a catalyst to relieve CPU-Wait due to system I/O; this overhead is evaluated in Appendix C.

Elements of the TUCC storage-hierarchy system

Environment

As enumerated in above section, the principal elements of the TUCC storage-hierarchy are the checkpoint system, pseudo-devices, and small-job monitor. The remainder of this paper addresses the last two elements. The discussion will focus on system I/O, then discuss the relative suitability of fast core, LCS, disk, and drum storage for (a) supervisory code, (b) supervisory tables, (c) I/O buffers, (d) application-program code, and (e) application-program work-space.

The 360/75 CPU has a mean instruction time of 1μs; the TUCC machine has a fast core of 524K bytes and a large, slower core of 2097K bytes

The two 2860 selector channels each are rated at 1300KB. Each is attached to a 2314 disk system, containing eight drives mounted with removable disk packs. The packs contain over 28M bytes apiece, aggregating a total on-line disk storage of 466M bytes. Each disk system can perform one 312KB data-transfer operation from one drive at a given instant.

Five magnetic tapes, the card/printer system, and communications control units all interface to a third (multiplexor) channel. Since they can all operate simultaneously with less that 5% utilization of fast memory, they will not be further discussed here (cf. References 8-11 for details).

This study is therefore restricted to core and disk storage, since effective CPU utilization under OS/360 depends principally on judicious allocation of functions to these media. Basic postulates of this paper are as follows:

- IBM-supplied compilers, sorts, utilities, and application packages for OS/360 will not soon reduce/modify their usage of system I/O.
- For reasons of compatibility, documentation and maintenance, the TUCC community insists on using IBM-supplied compilers and other software—excepting only the WATFOR compiler[15] and TSAR*—and makes little use of other customer-written, high-performance, reduced-function software. Thus, facilities like PUFFT[16] or CORC/CUPL[17] would have to offer significant functional advantages to displace IBM-supplied compilers in the TUCC community even if their compile-execute performance were markedly superior. WATFOR meets this test: it is functionally equivalent to FORTRAN G and offers approximately a 30:1 throughput advantage (mean job times of 0.5 seconds and 15 seconds respectively, for small student jobs).
- IBM will not soon change the relative price or relative performance of its fast-core, LCS, drum, or disk products.

Pseudo-readers, pseudo-punches, and pseudo-printers

Application programs, compilers, and the control program itself are each allocated one pseudo-card reader, one pseudo-punch, and one pseudo-printer, furnishing the standard SYSIN, SYSPUNCH, and SYSOUT functions. These are simulated by the TUCC system using a combination of locally-written subroutines in the OS/360 nucleus, two disk packs (one accepting input from several dozen remote

*The double-job-stream system shown in Figure 2 keeps CPU-Wait under 10%.

*TSAR is a statistical data-retrieval package developed at Duke University. It furnishes simplicities not currently available in IBM's Scientific Subroutine Package, as well as significant performance superiority.

job entry (RJE) sources, one accumulating output for the terminals), and buffer storage in LCS. Pseudo-readers furnish images to the two TUCC job streams at over 1200 card images per *second*, and pseudo-printers at over 1000 print images per second. SYSIN/SYSPUNCH/SYSOUT are entirely CPU-limited, even for program loops which "drive" the pseudo-devices as fast as possible. Details of the TUCC implementation can be found in Appendix B.

Hyperdisk

The I/O supervisor CALLs the disk simulator for all service to a single-drive disk addressed over (non-existent) channel 4 of the TUCC system. This hyperdisk is "transparent" to its users; it services all legal channel programs using a combination of LCS and real disk storage. The flow of track images between LCS and disk is totally controlled by the disk simulator and is decoupled from processing activities of partitions using the hyperdisk. In other words, many partitions can simultaneously have I/O requests queued on hyperdisk, although the simulator is actively servicing only one at a time.

In addition to completing legal channel programs, the disk simulator appropriately terminates illegal channel programs, e.g., for an invalid SEEK address, illegal sequence, incorrect length, etc. The only statuses which it never returns are, of course, "data check" or "channel check," indicating hardware failure.

The current allocation of LCS to the hyperdisk is 1650K, equivalent to 220 tracks. An arbitrary number of real tracks—currently 2400—comprise its other storage area. The hyperdisk is thus represented to the user as a 40%-unavailable pack; in OS/360 nomenclature, the aggregate number of allocated and unallocated tracks in the "volume table of contents" is exactly 2400, divided into two major areas by the simulator, as shown in Figures 3 ("Core Map of a Hyperdisk Simulation Using Only LCS"), 4 ("Hyperdisk Structure"), and 5 ("Sample Track Control Table and Chained Lists"):

1. The *read-only area* comprises frequently-used compiler and subroutine libraries. The simulator knows exactly where the read-only area ends and the write-read area begins. As each track is addressed by the user, the simulator consults its *track control table:* if the track is in LCS, the simulation is performed at once; if the track is not in LCS, the simulator must first retrieve the track from its (fixed) position within the 2400-track extent on real disk. The retrieved track has the same information content as the original track

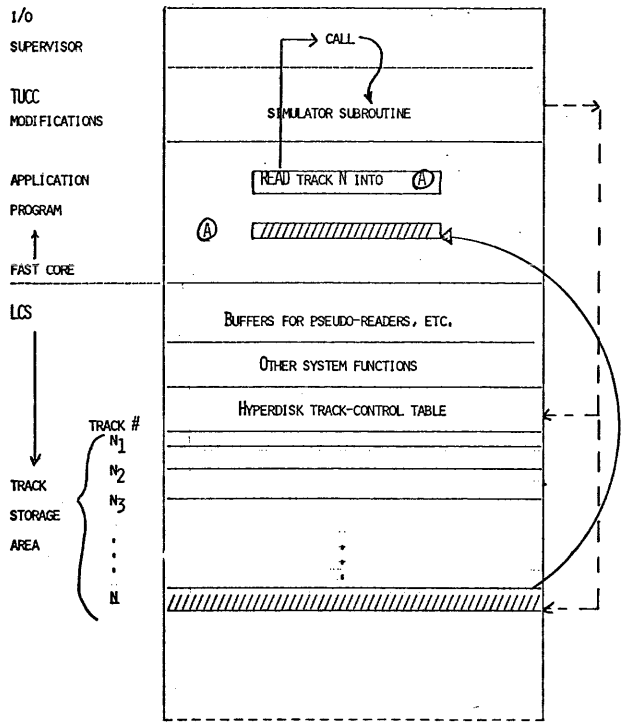


Figure 3 – Core map of a hyperdisk simulation using only LCS

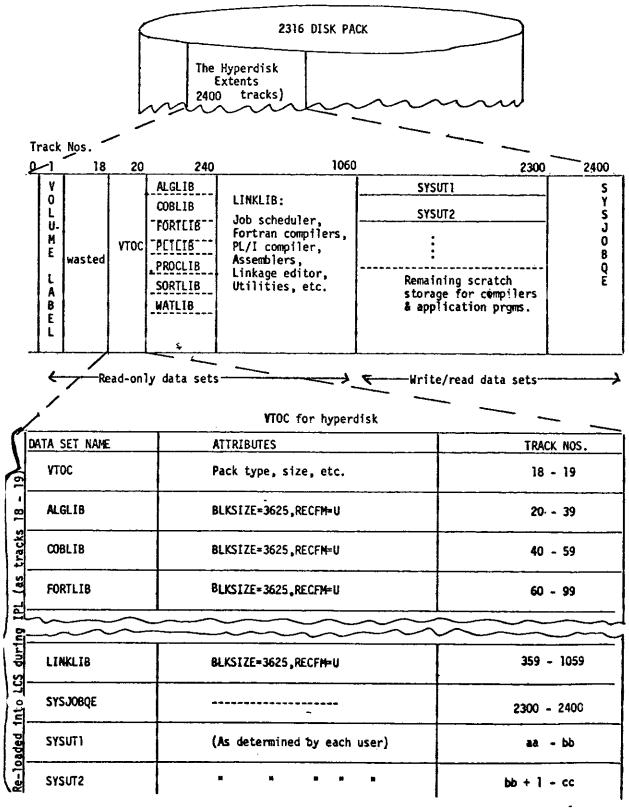


Figure 4 – Hyperdisk structure

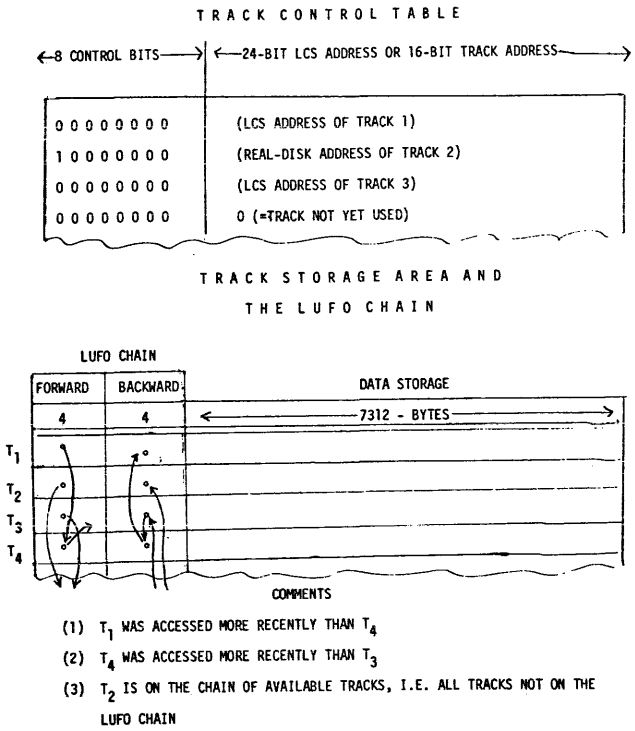


Figure 5 – Sample track control table and chained lists

of the disk dataset; however, it has been re-formatted – during a once-only run of the simulator – to match the internal representation used by the simulator, i.e., inter-record gaps of real tracks are represented by control fields in LCS.

A longest-unused-first-out algorithm *(LUFO)* controls retention/overlaying of track areas in LCS. When all areas have been allocated and a new track must be created (or retrieved from real disk), an existing track must be spilled or overlaid. For read-only data sets, the choice is obviously the latter. The LUFO algorithm is as follows: as each in-LCS track image is referenced, it is promoted to the "head" of the LUFO chain, as shown in Figure 5. Inactive tracks logically "drift" to the bottom of the LUFO chain. When a new track area is required, the bottom track is released, i.e., overlaid if read-only.

2. *Write-read* tracks are those above track-address 1060 in Figure 4; they are always written at least once before they are read. Except for the disk workspace of the control program (SYSJOBQE), they are released at the end of each job. If a write-read track must be spilled – because it has drifted to the bottom of the LUFO chain – it is written to an arbitrary track within the write-read area of the real disk. The relative address of the real track is retained in the track control table (Figure 5, "Sample Track Control Table and Chained Lists"). Just as for pseudo-readers and pseudo-printers, a seek-minimizing algorithm selects the nearest available track in the write-read area on real disk.

Expart, the small-job monitor

In OS/360, core memory may be divided into two or more contiguous blocks (*partitions*) which process independent streams of work. To reduce the overhead for initiating/terminating small jobs – typically FORTRAN and PL/I debugging jobs – TUCC has written a monitor which:

- Operates in a 100K "express" partition independent of the full-function batch-processing partition:
- Processes nonsetup jobs with small-memory needs; and
- Receives control only when the batch-processing partition is in CPU-Wait.

The express partition absorbs CPU-Wait from all higher-priority partitions, since its input/output is exclusively to pseudo-devices.

Experimental results and conclusions

Benchmark Jobs

Since June, 1966, TUCC has run benchmark jobs against each major new system (i.e., when new hardware or a new release of OS/360 or a major TUCC project is installed). These benchmark jobs, the "Golden Deck" described in Appendix A, fairly represent our job mix. We evaluate our instantaneous throughput rate based on this deck, although the substantial improvements seen in Appendix A are not directly reflected in jobs/day statistics. We process only 1200-1500 jobs/day for the following reasons:

- A large daily investment in preventive maintenance and systems development is necessary for a communications-oriented system with $\sim 10^5$ transistors;
- The system runs out of work during the midnight shift;
- An increasing number of jobs have entered "production" status, whereas most jobs in 1966-67 were debug runs.

Evaluation of the performance improvements may be found in Section III of Appendix A.

The core map

Early in this paper, OS/360 was characterized – for the TUCC environment – as having unsatisfactory design points for the supervisor, compilers, etc. By adding our storage-hierarchy system to OS/360,

CPU-Wait due to system I/O has been substantially reduced.

TABLE I – Core Map of TUCC System

DECIMAL ADDRESS	ELEMENT
0 - 80K	I/O supervisor, other necessarily-resident interrupt handlers, and device simulators.
80-330K	Batch-processing partition.
330-440K	EXPART partition.
440-524K	Executable code for job collection/dissemination partition.
(Beginning of LCS)	----------
524-830K	Buffers and control blocks for job collection/dissemination partition.
830-865K	Supervisory tables accessed by binary-lookup subroutines.
865-935K	Low-usage supervisor subroutines: OPEN, ABEND, WTO, etc.
935-2585K	Hyperdisk
2585-2621K	Checkpoint and accounting data

The system elements in fast core seem justifiable, on the whole; they include all executable code except low-usage supervisor subroutines. Executing instructions out of LCS is possible but generally undesirable; I-cycle timings are inflated from an average of 0.4μs to 4.5μs.

Low-usage supervisor subroutines are an exception to this general guideline; by their nature, they have few instruction loops (which are particularly undesirable in LCS). Each 1000-byte subroutine executes a few instructions (probably no more than 200), then yields control to the next subroutine or returns control to the caller. We contend that such subroutines are appropriately executed out of LCS, that I-cycle overhead is less than the overhead to retrieve them from LCS (or disk or drum) into fast core. To retrieve each supervisor subroutine from hyperdisk would require 3000μs (cf. Appendix C), which equals the overhead of executing 750 instructions from LCS instead of fast core. If each subroutine call indeed requires only 200 instruction executions, the advantage is clearly to execution-from-LCS.

Of course, these supervisor subroutines could be included in fast core as part of the system nucleus. This strategy would obviate either LCS-execution overhead or retrieval-from-LCS overhead. However, the 70K core requirement would have to either (a) be subtracted from the 250K batch partition or (b) displace the EXPART partition. In case (a), many user programs would require re-programming (e.g., FORTRAN programs with large matrices) into overlay structures. Possibly, an optimizing option of FORTRAN H would have to be constrained (or deck sizes constrained, equivalently). In case (b), TUCC would lose the important price/performance advantage of multiprogramming in a 524K-byte system. We note that IBM prices are approximately in the following ratio (lease price per byte per month):

2314:2301:LCS:fast-core=1:40:250:1600

Most executable code in the job-collection partition has higher aggregate activity than the low-usage supervisor subroutines. Thus, the TUCC system actually takes advantage of the multiple-overlay structure of the supervisor to segregate high-usage from low-usage code.

Using the LUFO algorithm described in an earlier section, the hyperdisk retains high-usage tracks for most non-supervisory code in LCS, whereas low-usage tracks drift back to disk storage – and remain there until they again become active. OS/360 compilers constrained to small design points have already segregated high-usage code and tables into resident segments of their overlays, whereas lightly-used compilation elements – both instructions and data – are diverted to disk. The TUCC system capitalizes on this *a priori* structuring.

Preliminary comparisons with standard, drum-oriented versions of OS/360 have confirmed the performance superiority – and price-performance superiority – of the TUCC system for a remote-job-entry university environment with a broad mix of student jobs, major compute-limited jobs, and I/O-limited file processing.

ACKNOWLEDGMENTS

The storage-hierarchy system is the work of 15 systems programmers at TUCC, of whom a number are employed by the Raleigh branch-office of IBM. Helpful suggestions have been offered by numerous IBM development groups and OS/360 customers. The HASP project of IBM-Houston includes approximately the same pseudo-readers and pseudo-printers as the TUCC system; it was developed at the same time as ours and implemented several months earlier.

REFERENCES

1 T KILBURN D J HOWARTH R B PAYNE F H SUMNER

The Manchester University atlas operating system parts I-II
The Computer Journal October 1961

2 T KILBURN D B G EDWARDS M J LANIGAN F H SUMNER
One-level storage system
IRE Transactions on Electronic Computers Vol II No 2 April 1962

3 F H SUMNER E C Y CHEN G HALEY
The central control unit of the atlas computer
Proceedings IFIP Congress 1962

4 D MORRIS F H SUMNER M T WYLD
An appraisal of the atlas supervisor
Proceedings of the 22nd Annual ACM National Conference 1967

5 G H MEALY B I WITT W A CLARK
The functional structure of OS/360
IBM System Journal Vol 5 No 1 1966

6 W C LYNCH
Description of a high capacity, fast turnaround, university computing center
Communications of the Association for Computing Machinery Vol 9 No 2 February 1966

7 G BENDER D N FREEMAN J D SMITH
Function and design of DOS/360 and TOS/360
IBM Systems Journal Vol 6 No 1 Page 19 1967

8 D N FREEMAN (editor)
Systems development at TUCC: 1966-1967
Paper distributed at SHARE XXVIII San Francisco February 1967

9 D N FREEMAN
Structure of TUCC linkage to three campuses: Hardware and software
Reprints of the 6th Annual Southeastern Regional Meeting of ACM, June, 1967

10 J F WALKER
OS/360 throughput problems
Reprints of the 6th Annual Southeastern Regional Meeting of ACM June 1967

11 J W STEPHENSON JR
Use of bulk core to improve system performance
Reprints of the 6th Annual Southeastern Regional Meeting of ACM June 1967

12 N J NEILSON
The simulation of time sharing systems
Communications of the Association for Computing Machinery Vol 10 No 7 July 1967

13 H F LAUER
Bulk core in a 360/67 time-sharing system
FJCC Proceedings 1967

14 J L SMITH
Multiprogramming under a page on demand strategy
Communications of the Association for Computing Machinery Vol 10 No 10 October 1967

15 P W SCHANTZ et al
Watfor—The University of Waterloo Fortran IV compiler
Communications of the Association for Computing Machinery Vol 10 No 1 January 1967

16 S ROSEN R A SPURGEON J K DONNALLY
PUFFT—The Purdue University fast Fortran translator
Communications of the Association for Computing Machinery Vol 8 No 11 November 1965

17 R W CONWAY W L MAXWELL
CORC—The Cornell computing languages
Communications of the Association for Computing Machinery Vol 6 No 6 June 1963

18 G A BLAAUW F P BROOKS JR
The structure of System/360 Part I: Outline of the logical structure
IBM Systems Journal Vol 3 No 2 1964

APPENDIX A
GOLDEN DECK TIMINGS

I. DESCRIPTION OF RUNS

A. *RUNCO* (130 source cards)

Procedure:	COBOL (COBOL-E: compile, link, go)
Compile:	source listing data division map procedure division map diagnostics
Link-Edit:	cross reference map
Go:	500 lines of print output

B. *CTEST* (2600 source cards)

Procedure:	COBOLLNK (COBOL-E: compile, link)
Compile:	source listing data division map hex code listing diagnostics
Link-Edit:	cross reference map

C. *TIME* (500 source cards)

Procedure:	ASMLNKGO (ASSEMBLER-F: assemble, link, go)
Compile:	source listing relocation dictionary cross reference table
Link-Edit:	cross reference table
Go:	50 lines output 120 cards input

D. *PL1T1* (200 source cards)

Procedure:	PL1 (PL/I-F: compile, link, go)
Compile:	source listing diagnostics
Link-Edit:	diagnostics only
Go:	no output 8 cards input

E. *PL1CP* (1100 source cards)

Procedure:	PL1CMP (PL/I-F: compile)
Compile:	diagnostics

F. *AFORT* (1200 source cards)

Procedure:	FORTCMP (FORTRAN-E: compile)
Compile:	diagnostics only

G. *BFORT* (775 source cards)

Procedure:	FTLNKGO (FORTRAN-E: compile, link, go)
Compile:	source listing storage map diagnostics
Link-Edit:	cross reference table
Go:	220 lines of print output 130 cards of input

H. *TSAR* (no source)

Procedure:	PGM = EXEC using JOBLIB (Statistical Prod. program)
Compile:	none
Link-Edit:	none
Go:	175 cards input 525 lines output

I. *SPLI* (null PL/I, 4 source statements)

Procedure:	PL1 (PL/I-F: compile, link, go)
Compile:	source listing diagnostics
Link-Edit:	diagnostics only
Go:	no input

J. *SAMB* (null assembly, 3 source statements)

Procedure:	ASMLNKGO (ASSEMBLER-F: assemble, link, go)
Compile:	source listing relocation dictionary cross reference table
Link-Edit:	cross reference table
Go:	no input

K. *SFORT* (null FORTRAN, 2 source statements)

Procedure:	FORTRAN (FORTRAN-E: compile, link, go)
Compile:	source listing storage map external references diagnostics
Link-Edit:	diagnostics
Go:	no input

II. *TIMINGS*

The test environment is indicated below and coded as follows:

1st position - S/360model number
2nd position - OS/360 release number
3rd position - SYSIN device
4th position - SYSOUT device
5th position - Locally written systems software

The following abbreviations are used:

C - card reader (2540)
P - 1430 printer (Model N1, universal character set)
T - 60KB tape
D14 - 2314 disk
D14M - 2314 disk with SYSIN disk-to-disk data movement
H - HYPERDISK (pseudo disk in LCS)
D - Directories in LCS
F - LCS FETCH

Times are given in seconds.

	COMPILE	LINK	L	
	COMPILE	*LINK*	*GO*	**JOB TIME*
RUNCO				
40,2,C,P				
40,3,C,P	102	50	84	236
75,6,C,P	73	29		
75,6,T,P	74	29	60	163
75,6,T,P (print train)	70	25		
75,6,T,T	60	24	12	96
75,6,T,T,D	83	21	11	115
75,6,T,T,DF	30	14	08	52
75,9,T,T,DF	19	11	08	38
75,11,D14M,D14,DH (2303 LINKLIB)	10	04	05	19
75,11,D14M,D14,DH (HDSK LINKLIB)	09	04	05	18
CTEST				
40,2,C,P	694	157		851
40,3,C,P	697	152		849
74,6,C,P	556	74		630
75,6,T,P	558	74		632
75,6,T,P (print train)	463	66		529
75,6,T,T	166			
75,6,T,T,D	140	61		201
75,6,T,T,DF	126	52		178
75,9,T,T,DF				
75,11,D14M,D14,DH (2303 LINKLIB)	97	07		104
75,11,D14M,D14,DH (CHDSK LINKLIB)	96	06		102
TIME				
40,2,C,P	1080	40	567	1680
40,3,C,P	403	43	690	1136
75,6,C,P	256	26	479	761
75,6,T,P	240	27	471	739
75,6,T,P (print train)	211	23	383	617
75,6,T,T				
75,6,T,T,D	121	19	325	465
75,6,T,T,DF				
75,9,T,T,DF				
75,11,D14M,D14,DH (2303 LINKLIB)	39	04	--	--
75,11,D14M,D14,DH (HDSK LINKLIB)	33	03	152	188
PLITI				
40,2,C,P	170	117	1540	1827
40,3,C,P	168	96	1549	1813
75,6,C,P	79	45	92	216
75,6,T,P	79	45	92	216
75,6,T,P (print train)	69	41	90	200
75,6,T,T	49	41		
75,6,T,T,D	43	36	89	168
75,6,T,T,DF	32	30		
75,9,T,T,DF	24	24	88	136

75,11,D14M,D14,DH (2303 LINKLIB)	16	08	89	115
75,11,D14M,D14,DH (HDSK LINKLIB)	15	08	89	114
PL1CP				
40,2,C,P	933			933
40,3,C,P	353			353
75,6,C,P	71			71
75,6,T,P	59			59
75,6,T,P (print train)	53			53
75,6,T,T	45			45
75,6,T,T,D	40			40
75,6,T,T,DF	25			25
75,9,T,T,DF	27			27
75,11,D14M,D14,DH (2303 LINKLIB)	16			16
75,11,D14M,D14,DH (HDSK LINKLIB)	13			13
AFORT				
40,2,C,P	303			303
40,3,C,P	185			185
75,6,C,P	172			172
75,6,T,P	72			72
75,6,T,P (print train)	69			69
75,6,T,T	78			78
75,6,T,T,D	65			65
75,6,T,T,DF	34			34
75,9,T,T,DF	28			28
75,11,D14M,D14,DH (2303 LINKLIB)	12			12
75,11,D14M,D14,DH (HDSK LINKLIB)	09			09
	COMPILE	LINK	GO	*JOB TIME
BFORT				
40,2,C,P	355	64	539	958
40,3,C,P	181	75	542	798
75,6,C,P	116	37	52	205
75,6,T,P	116	37	52	205
75,6,T,P (print train)	109	31	48	188
75,6,T,T	61			
75,6,T,T,D	50	27	26	108
75,6,T,T,DF	26	20	22	68
75,9,T,T,DF	21	18	22	61
75,11,D14M,D14,DH (2303 LINKLIB)	10	05	19	34
75,11,D14M,D14,DH (HDSK LINKLIB)	08	05	19	32
TSAR				
75,6,C,P			86	86
75,6,T,P			78	78
75,6,T,P (print train)			86	86
75,6,T,T			29	29
75,6,T,T,D			27	27
75,6,T,T,DF			21	21
75,9,T,T,DF			16	16
75,11,D14M,D14,DH (2303 LINKLIB)			11	11
75,11,D14M,D14,DH (HDSK LINKLIB)			11	11

SPLI				
75,6,C,P	35	28	07	70
75,6,T,P	35	28	07	70
75,6,T,P (print train)	31	24	06	61
75,6,T,T	29	24	05	58
75,6,T,T,D	24	21	05	50
75,6,T,T,DF	13	14	02	29
75,9,T,T,DF	08	12	02	22
75,11,D14M,D14,DH (2303 LINKLIB)	05	05	01	11
75,11,D14M,D14,DH (HDSK LINKLIB)	03	05	01	09
SAMB				
75,6,C,P	35	25	04	64
75,6,T,P	35	25	04	64
75,6,T,P (print train)	32	21	03	56
75,6,T,T				
75,6,T,T,D	26	17	03	46
75,6,T,T,DF				
75,9,T,T,DF	11	08	01	20
75,11,D14M,D14,DH (2303 LINKLIB)	07	04	01	12
75,11,D14M,D14,DH (HDSK LINKLIB)	06	03	01	10

	COMPILE	LINK	GO	*JOB TIME
SFORT				
75,6,C,P	26	26	07	59
75,6,T,P	26	26	07	59
75,6,T,P	23	22	06	51
75,6,T,T (print train)	22	22	06	50
75,6,T,T,D	18	19	05	42
75,6,T,T,DF	12	11	05	28
75,9,T,T,DF	07	10	03	20
75,11,D14M,D14,DH (2303 LINKLIB)	03	04	01	08
75,11,D14M,D14,DH (HDSK LINKLIB)	03	04	01	08

*Does not include termination time for each job. Prior to the installation of HYPERDISK this time was approximately 02 seconds/job. Termination time with HYPERDISK has been reduced to approximately 01 to 01.5 seconds/job.

III. DISCUSSION OF THE TIMINGS

RUNCO initially compiled only slightly faster on 360/75 than on our original 360/40; assigning SYSIN/SYSOUT to tape reduced compilation from 73 to 60 seconds. Moving SVCLIB and LINKLIB into LCS halved this time; this move was our first attempt at a hierarchical system and has been documented elsewhere.[11] The improved Release-9 compiler and control program halved the time again, and the pseudo-printer/hyperdisk additions reduced compile time to 8.4 seconds.

Similarly, link-edit time has been reduced 5:1 from the tape-in-tape-out (*TITO*) time of 24 seconds for the (non-hierarchical) Release-6 system. Since the linkage editor requires no card/tape input and writes few SYSOUT lines, the gains are attributable primarily to the hyperdisk (and its precursor, which serviced only SVCLIB and LINKLIB).

Aggregate job time has been reduced by 6:1 from a TITO system.

CTEST—This COBOL job requires voluminous SYSIN/SYSOUT during the compilation step; the link-edit step has been reduced by approximately 10:1 over the TITO system.

TIME—This job evaluates supervisory services offered by OS/360 such as OPEN, CLOSE, GET,

PUT, etc. Assembly time was reduced 4:1 over a TITO system. (TIME was not operable for several months due to a time-dependency in OS/360. This error was first exposed by our early storage-hierarchy system and was corrected by IBM when their improvement program also exposed it.)

PL1TI—This job has a prolonged, no-I/O execution phase—hence the dramatic reduction when TUCC replaced the 360/40 with the 360/75. The compilation time has been reduced 3:1 over the TITO system.

PL1CP—This non-trivial PL/I compilation has been reduced 3:1 from the TITO system; the compilation rate is over 5000 statements per minute.

AFORT—This non-trivial, multiple-compilation FORTRAN-E job has been reduced over 8:1 from the TITO system; the compilation rate is 8000 statements per minute.

BFORT, TSAR, etc.—The remaining benchmark jobs show comparable improvements.

The Release-11 runs were performed during concurrent job-collection/dissemination, whereas all previous timing runs were performed with communications equipment (and other I/O irrelevant to the experiments) turned off. Thus, the Release-11 figures are somewhat inflated due to interrupt-servicing and buffer manipulations unrelated to the timing tests.

APPENDIX B
IMPLEMENTATION DETAILS FOR THE PSEUDO-READERS, PSEUDO-PUNCHES, AND PSEUDO-PRINTERS

Within the I/O supervisor of OS/360 (in resident nucleus), there are approximately ten incidences of the following privileged-instructions:[18] Start I/O, Test I/O, Halt I/O, and Test Channel. TUCC has replaced each instruction with a CALL to an *I/O filter* subroutine, which determines if the operation is to be attempted on a real device or a pseudo-device. In the former case, the I/O instruction is issued by the subroutine, which then returns control to the I/O supervisor. In the latter case, the channel command words (CCWs) are interpreted by a card-reader simulator, print simulator, or punch simulator, as distinguished by the device address (a parameter of the I/O filter subroutine). Each card-reader interpretation comprises the following events:

1. An input buffer pool in LCS is tested for availability of the next card image from this pseudo-reader.
2. If the next image is available, it is decompressed and moved from LCS to the target area in fast core specified by the READ CCW. "Decompression" is "restoration of blanks" as in Reference 7; when the card image was originally captured by the TUCC computer network—possibly at the central computer, possibly at an "intelligent" satellite—it was scanned for strings of at least 2 blanks, each of which being replaced by a one-byte control field. The card image is retained in this compressed representation until the instant it is "read" by the reader simulator. This technique conserves LCS and disk storage throughout the queuing network depicted in Figure 2.
3. After the next card image has been moved out of LCS, control information for the buffer pool is appropriately updated. If the current buffer* is emptied, the track-manager subroutine is notified to refill the buffer.
4. The reader-simulator returns control to the I/O supervisor, indicating that the READ was instantly and perfectly performed.
5. At step (2), if no card images are available—because the track manager has fallen behind the pseudo-reader or because the system is out of work—the job-processing partition requesting the card image is idled until a fresh buffer is retrieved.

Anticipatory buffering keeps two or three tracks of card images in LCS, so that the pseudo-reader rarely goes idle; statistics are not yet available on this phenomenon. Pseudo-printers and pseudo-punches operate in an analogous fashion:

1. As each job-processing partition requests printing/punching of an image, the I/O supervisor issues a CALL to the I/O filter subroutine.
2. The image is compressed and inserted into an LCS buffer.
3. Buffer-pool control information is updated; if an output buffer is filled, the track manager is notified to write the track image to disk.

On the current TUCC system, one pack is used to queue SYSIN and one pack to queue SYSOUT. The track manager totally controls the status of these 8,000 tracks, using occupancy tables in LCS to record the tracks for each job and the queue of jobs arriving from each satellite. To conserve access-mechanism motion, a simple seek-minimizing algorithm is used to allocate tracks, viz., WRITE into that available track nearest the current mechanism position. Visual observation of the SYSIN/SYSOUT packs shows low mechanism activity, even when several

*One disk track can hold 7294 bytes.

5KB communications lines and two job-processing partitions are contending for a single mechanism. This activity is kept low primarily (a) by the high information density per track, averaging 240 card images or 180 print lines, and (b) by the seek-minimizing algorithm.

After processing, jobs can wait indefinitely in the output queue for each satellite; sizeable tables are required to service terminals which submit, say, 100 jobs on Friday and do not again establish contact with TUCC until Monday. However, this resulting convenience of operation is much esteemed by the satellite installations, and TUCC will remain the principal queue point of the complex for the foreseeable future.

APPENDIX C
DETAILED TIMINGS OF THE HYPERDISK

A. SIMULATOR OVERHEAD

To measure the overhead due to the I/O-filter and disk-simulator subroutines, the following experiments were performed:

1. 1000 "no-operation" *(NOP)* CCWs were issued to a real disk to determine EXCP/WAIT/interrupt overhead. Repeated measurements furnished a low-variance average of *1180μs* for an EXCP/WAIT sequence on a 360/75. This is used hereafter as a base figure.
2. 1000 NOP CCWs to the hyperdisk averaged 1550μs, i.e., a simulator overhead of *370μs* per EXCP/WAIT sequence.
3. Three 1000-event experiments were performed to time READ and SEARCH overheads for the disk simulator.
   a. To read the first one-byte block on each track required an additional *210μs*. To read a block of N bytes required an additional $N\mu s$, since LCS operates at 1MB for block transfers. (If the disk-simulator were used with two-way-interleaved LCS—an extra-cost option—this incremental time would be reduced to $0.5N\mu s$.) Note that the average time to read a block of N bytes from real disk is (12500 + 3.12N)μs; the average time from a 2301 drum is (8600 + 0.8N)μs. Representative values are as follows:*
   b. To read the 10th one-byte record on each track required *280μs* more than to read the first record. Thus, the time to search the identification of one record unsuccessfully—in OS/360 nomenclature, "SRCHID=" followed by "TIC"—is approximately 30μs. This is a negligible effect for hyperdisk performance, since there are rarely more than 40 blocks per disk track.

The hyperdisk (arbitrarily) begins each search with the first record on a track. The figures cited for the disk and drum in the above table assume half-track latency, on the average. The hyperdisk figures should therefore be somewhat increased to reflect simulated half-track searches:

$$\left\{\begin{matrix}2440\\2850\\4790\end{matrix}\right\} \text{ instead of } \left\{\begin{matrix}1840\\2760\\4760\end{matrix}\right\}$$

TABLE II—Average EXCP/WAIT Time (μx) on a 360/75 with 2314 Disk, 2301 Drum, and Hyperdisk

BLOCK SIZE	2314 DISK	2301 DRUM	HYPERDISK
80	13,930	9,344	1,840
1000	16,800	10,080	2,760
3000	23,040	11,680	4,760

B. REAL DISK I/O WITHIN THE HYPERDISK

To determine the average level of LCS activity vs. real-disk activity within the hyperdisk, we inserted counters at over 50 points within the simulator. The gross overhead due to counter activity is less than 0.01% of clock time.

Four recent readings of the counters are essentially in agreement: with 220 track images in LCS, less than 0.8% of the channel programs directed to the hyperdisk necessitated reading of a real track. For example, during one 50-minute period, 162,000 SIOs were issued to the hyperdisk (of which half were for data-transfer operations); this produced only 612 full-track READs and 38 WRITEs within the hyperdisk. These WRITEs were, in fact, for the 40-track SYSJOBQE data set,* which is formatted in LCS when the system is initially loaded. The READs were, of course, directed to the read-only data sets depicted in Figure 4. Included in the jobs were assemblies, compilations in all languages, and link-edits.

C. CPU OVERHEAD DUE TO THE HYPERDISK

During the 50-minute experiment cited just above approximately 81,000 non-trivial channel pro-

*The EXCP/WAIT overhead for servicing a drum in OS/360 is 180μs less (on a 360/75) than for disk, since the "stand-alone SEEK" is unnecessary. The hyperdisk behaves like a drum in this respect.

*Since the original measurements were made, SYSJOBQE has been increased from 40 to 100 tracks.

During the 50-minute experiment cited in Section III.C, approximately 81,000 non-trivial channel programs were directed to the hyperdisk. The aggregate number of interpreted CCW's was 940,000, and 69,000,000 bytes were moved by the simulator to/from LCS.

Assuming 1760μs per non-trivial channel program and 50μs per interpreted CCW, plus 1μs per byte moved from/to LCS, the aggregate overhead was as follows:

143 seconds for EXCP + WAIT + I/O filter
47 seconds for CCW interpretation
69 seconds for data movement
259 seconds of 3000 seconds clock time

Thus, 8.5% of clock time was spent servicing the hyperdisk, including all I/O-supervisor overhead (cf. Neilson's results[12]).

Burroughs' B6500/B7500 stack mechanism

by E. A. HAUCK and B. A. DENT
Burroughs Corporation
Pasadena, California

INTRODUCTION

Burroughs' B6500/B7500 system structure and philosophy are an extension of the concepts employed in the development of the B5500 system. The unique features, common to both hardware systems, are that they have been designed to operate under the control of an executive program (MCP) and are to be programmed in only higher level languages (e.g., ALGOL, COBOL, and FORTRAN). Through a close integration of the software and hardware disciplines, a machine organization has been developed which permits the compilation of efficient machine code and which is addressed to the solution of problems associated with multiprogramming, multiprocessing and time sharing.

Some of the important features provided by the B6500/B7500 system are dynamic storage allocation, re-entrant programming, recursive procedure facilities, a tree structured stack organization, memory protection and an efficient interrupt system. A comprehensive stack mechanism is the basic ingredient of the B6500/B7500 system for providing these features.

B6500/B7500 processor

The command structure of the B6500/B7500 Processor is Polish string, which allows for the separation of program code and data addresses. The basic machine instruction is called an operator syllable. This operator syllable is variable in length, from a minimum of 8 bits to a maximum of 96 bits. In the interest of code compactness, more frequently used operator syllables are encoded in the 8 bit form.

The Processor is provided with a hardware implemented stack in which to manipulate data and store dynamic program history. Also, data may be located in arrays outside the stack and may be brought to the stack temporarily for processing. Program parameters, local variables, references to program procedures and data arrays are normally stored within the stack.

The data word of the B6500/B7500 Processor is 51 bits long. Data are transferred between memory and within the Processor in 51 bit words. The first 3 bits of the word are used as tag bits, which serve to identify the various word types as illustrated in Fig. 1. The remaining 48 bits are data. Tag bits, in addition to identifying word type, provide the B6500/B7500 Processor with two unique features: (1) data may be referenced as an operand, with the processor worrying about whether the operand consists of one or two words, and (2) system integrity and memory protection are extended to the level of the basic machine data words. If a job attempts to execute data as program code, or to modify program code, the system is interrupted.

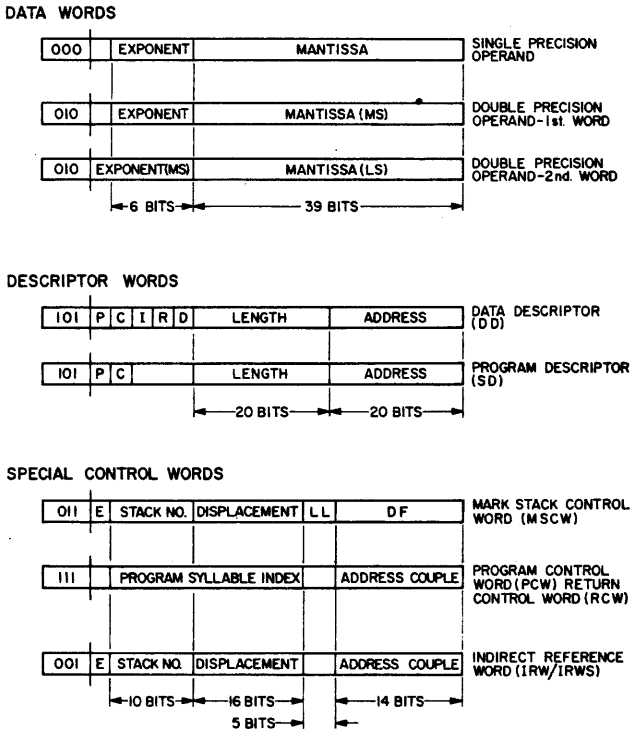


Figure 1 – B6500/B7500 word formats

The stack

The stack consists of an area of memory assigned to a job. This stack area serves to provide storage for basic program and data references associated with the job. In addition, it provides a facility for the temporary storage of data and job history. When the job is activated, four high speed registers (A, X, B and Y) are linked to the job's stack area (Fig. 2). This linkage is established by the stack pointer register (S), which contains the memory address of the last word placed in the stack memory area. The four top-of-stack registers (A, X, B and Y) function to extend the job's stack into a quick access environment for data manipulation.

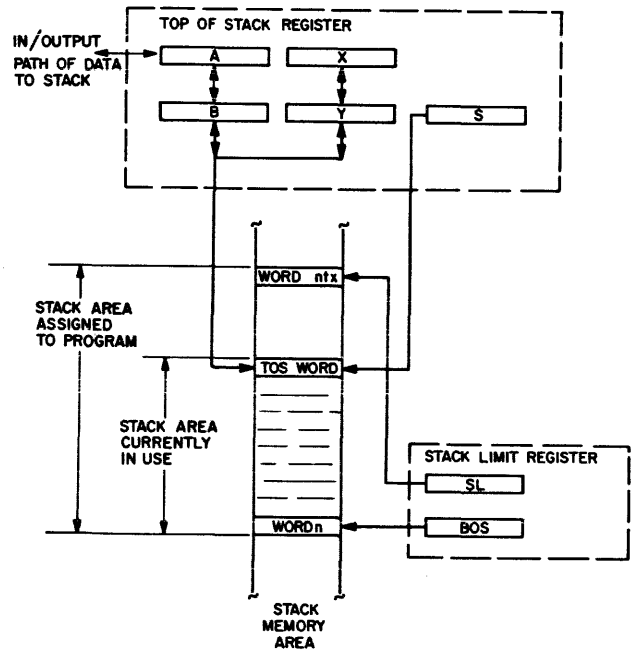


Figure 2—Top of stack and stack bounds registers

Data are brought into the stack through the top-of-stack registers. The stack's operating characteristic is such that the last operand placed into the stack is the first to be extracted. The top-of-stack registers become saturated after having been filled with two operands. Loading a third operand into the top-of-stack registers causes an operand to be pushed from the top-of-stack registers into the stack memory area. The stack pointer register (S) is incremented by one as each additional word is placed into the stack memory area; and is, of course, decremented by one as a word is withdrawn from the stack memory area and placed in the top-of-stack registers. As a result, the S register continually points to the last word placed into the job's stack memory area.

A job's stack memory area is bound, for memory protection, by two registers, the Base of Stack (BOS) register, and the Stack Limit (SL) register. The contents of the BOS register defines the base of the stack area, and the SL register defines the upper limit of the stack area. The job is interrupted if the S register is set to the value contained in either SL or BOS.

The contents of the top-of-stack registers are maintained automatically by the processor hardware in accordance with the environmental demands of the current operator syllable. If the current operator syllable demands that data be brought into the stack, then the top-of-stack registers are adjusted to accommodate the incoming data, and the surplus contents of the top-of-stack registers, if any, are pushed into the job's stack memory area. Words are brought out of the job's stack memory area and pushed into the top-of-stack register for operator syllables which require the presence of data in the top-of-stack registers, but do not explicitly move data into the stack.

Top-of-stack registers operate in an operand oriented fashion as opposed to being word oriented. Calling a double precision operand into the top-of-stack registers implies the loading of two memory words into the top-of-stack registers. The first word is always loaded into the A register where its tag bits are checked. If the word has a double precision tag, a second word is loaded into X. The A and X registers are then concatenated to form a double precision operand image. The B and Y registers concatenate when a double precision operand is moved to the B register. The double precision operand splits back to single words as it is pushed from the B and Y registers into the stack memory area. The reverse process is repeated when the double precision operand is eventually popped up from the stack memory area back into the top-of-stack registers.

Data addressing

Three mechanisms exist within the B6500/B7500 Processor for addressing data or program code: (1) Data Descriptor (DD)/Segment Descriptor (SD), (2) Indirect Reference Word (IRW), and (3) Stuffed Indirect Reference Word (IRWS). The Data Descriptor (DD) and Segment Descriptor (SD) are B5500 carryovers and provide the basic mechanism for addressing data or program segments which are located outside of the job's stack area. The basic addressing component of the descriptor is an absolute machine address. The Indirect Reference Word (IRW) and the Stuffed Indirect Reference Word (IRWS) are B6500/B7500 mechanisms for addressing data located within the job's stack memory area. The addressing component of both the IRW and IRWS is a relative address. The IRW is used to address within the im-

mediate environment of the job's stack, and addresses relative to a display register (described later in Non-local Addressing). The IRWS is used to address beyond the immediate environment of the current procedure, and addresses relative to the base of the job's stack. Addressing across stacks is accomplished with an IRWS.

The descriptor

In general, the descriptor functions to describe and locate data or program code associated with a given job. The Data Descriptor (DD) is used to fetch data to the stack or store data from the stack into an array which resides outside the job's stack area. The format of Data and Segment Descriptors are illustrated in Fig. 1. The ADDRESS field of both descriptors is 20 bits in length and contains the absolute address of an array in either main system memory or in the back-up disk store. The Presence bit (P) indicates whether the referenced data are present in main system memory or in the back-up disk store, and is set equal to ONE when the referenced data are present in main system memory.

A Presence Bit Interrupt is incurred when the job makes reference to data via a descriptor which has a P bit equal to ZERO. The Presence Bit Interrupt stimulates the operating system (called the Master Control Program, or MCP) to move the data from disk to main memory. The data location on disk is contained in the ADDRESS field of the DD when the P bit is equal to ZERO. After transferring the data array into the main memory, the operating system (MCP) marks the descriptor present by setting the P bit equal to ONE, and places the current memory address into the ADDRESS field of the descriptor. The interrupted job is then reactivated.

A Data Descriptor may describe either an entire array of data words, or a particular element within an array of data words. If the descriptor describes an entire array, the Indexed bit (I-bit) in the descriptor is ZERO, indicating that the descriptor has not yet been indexed. The LENGTH field of the descriptor defines the length of the data array.

A particular element of an array may be described by indexing an array descriptor. Memory protection is insured during indexing operations by performing a comparison between the LENGTH field of the descriptor and the index being applied to it. An Invalid Index Interrupt is incurred if the index value exceeds the length of the memory area defined by the descriptor.

If the value being used to index the descriptor is valid, the LENGTH field of the descriptor is replaced by the index value. At this time the I-bit in the descriptor is set to ONE to indicate that indexing has taken place. The ADDRESS and LENGTH fields are added together to generate an absolute machine address whenever a present, indexed Data Descriptor is used to fetch or store data.

The Double Precision bit (D) is used to identify the referenced data as being either single or double precision and, as a result, is also associated with the indexing operation. The D bit being equal to ONE signifies double precision and implies that the index value be multiplied by two before indexing.

The Read-Only bit (R) specifies that the memory area described by the Data Descriptor is a read-only area. An interrupt is incurred upon referencing an area through a descriptor with the intention to write if the R bit is equal to ONE.

The Copy bit (C) identifies a descriptor as being a copy of a master descriptor and is related to the present bit action. The intent of the copy action is to keep multiple copies of an absent descriptor linked back to one master descriptor. Copy action is incurred when a job attempts to pass by name an absent Data Descriptor. When this occurs, the hardware manufactures a copy of the master descriptor, forces the C bit equal to ONE and inserts into the ADDRESS field the address of the master descriptor. Thus, multiple copies of absent descriptors are all linked back to the master descriptor.

Non-local addressing

The most important single aspect of the B6500/B7500 stack is its facility for storing the dynamic history of a program under execution. Two lists of program information are saved in the B6500/B7500 stack, the stack history list and the addressing environment list. The stack history list is dynamic in nature, varying as the job is driven through different program paths with changing sets of data. Both lists are generated and maintained by the B6500/B7500 hardware system.

The stack history list is formed from a list of Mark Stack Control Words (MSCW) which are linked together by their DF fields (Fig. 3). A MSCW is inserted into the stack as a procedure is entered, and is extracted as that procedure is exited. Therefore, the stack history list grows and contracts in accordance with the procedural depth of the program. Mark Stack Control Words serve to identify the portion of the stack related to each procedure. When the procedure is entered, its parameters and local variables are entered in the stack following the MSCW. When executing the procedure, its parameters and local variables are referenced by addressing relative to the location of the related MSCW.

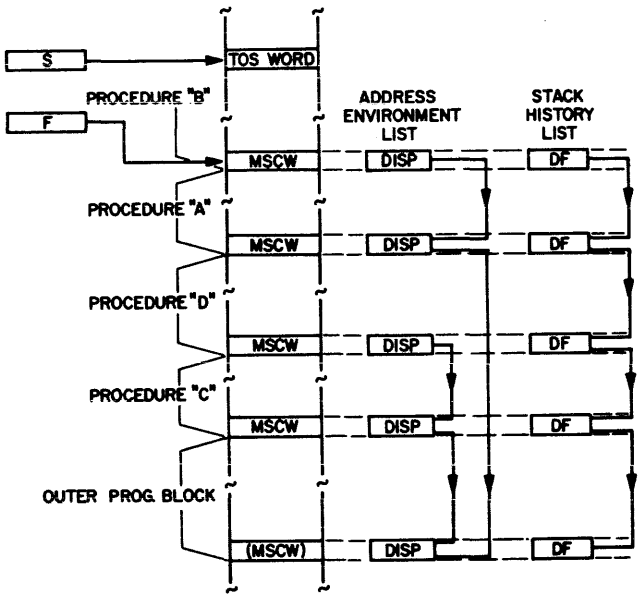


Figure 3 – Stack history and addressing environment list

Each MSCW is linked back to the prior MSCW through the contents of its DF field to identify the point in the stack where the prior procedure began. When a procedure is exited, its related portion of the stack is discarded. This action is achieved by setting the stack pointer register (S) to point to the memory cell preceding the most recent MSCW (Fig. 4). This top-most MSCW, pointed to by another register (F), is in effect deleted from the stack history list by causing F to point back at the prior MSCW, thereby placing it at the head of the stack history list.

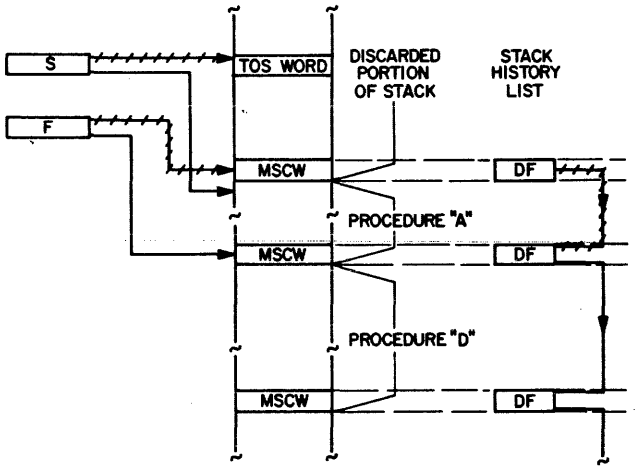


Figure 4 – Stack cut-back operation on procedure exit

This concept is implemented in the Burroughs' B5500 system, and it provides a convenient means to handle subroutine entry and exit. But this mechanism alone also gives rise to one of the most serious limitations of the ALGOL implementation on the B5500. In the B5500 stack, local variables are addressed relative to the first Mark Stack Control Word (which corresponds to the outer-most block), or relative to the most recent Mark Stack Control Word (which corresponds to the current procedure). All intervening Mark Stack Control Words, however, are invisible to the current procedure. This means that the variables declared global to the current procedure, but local to some other procedure, cannot be addressed at all! This inability to reference variables declared non-local to the current procedure but local to some other procedure is termed the non-local addressing problem.

The manner in which these variables are addressed in the B6500/B7500 stack can best be understood by analyzing the structure of an ALGOL program. The addressing environment of an ALGOL procedure is established when the program is structured by the programmer, and is referred to as the lexicographical ordering of the procedural blocks (Fig. 6A). At compile time, this lexicographical ordering is used to form address couples. An address couple consists of two items: 1) the lexicographical level (ℓ) of the variable, and 2) an index value (δ) used to locate the specific variable within a given lexicographical level. The lexicographical ordering of the program remains static as the program is executed, thereby allowing variables to be referenced via address couples as the program is executed.

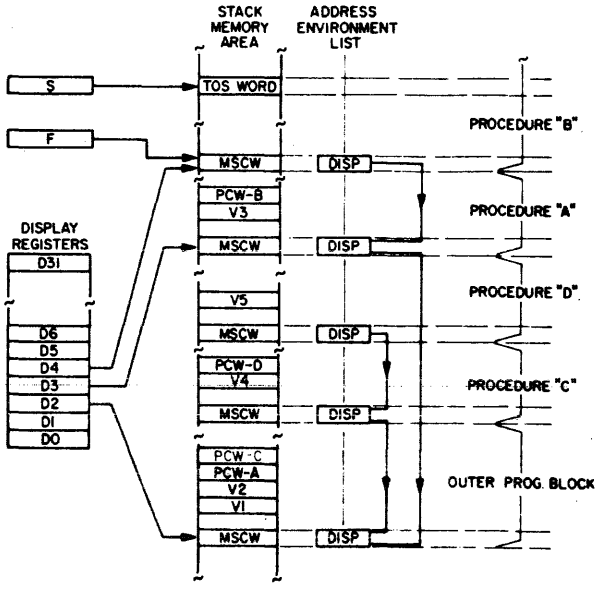


Figure 5 – Display registers indicating current addressing environment

```
BEGIN                                        Lexicographical Level "2"
   REAL V1;                                  ll=2, δ=2
   REAL V2;                                  ll=2, δ=3
   PROCEDURE A;                              ll=2, δ=4
      BEGIN                                  Lexicographical Level "3"
         REAL V3;                            ll=3, δ=2
         PROCEDURE B;                        ll=3, δ=3
            BEGIN                            Lexicographical Level "4"
               V3←3;
               V1←V3;
            END;
         B;
      END;
   PROCEDURE C;                              ll=2, δ=5
      BEGIN                                  Lexicographical Level "3"
         REAL V4;                            ll=3, δ=2
         PROCEDURE D;                        ll=3, δ=3
            BEGIN                            Lexicographical Level "4"
               REAL V5;                      ll=4, δ=2
               V4←4;
               V5←5;
               A;
               V2←V4;
            END;
         D;
      END;
   C;
END;
```

Figure 6a—ALGOL program with lexicographical structure indicated

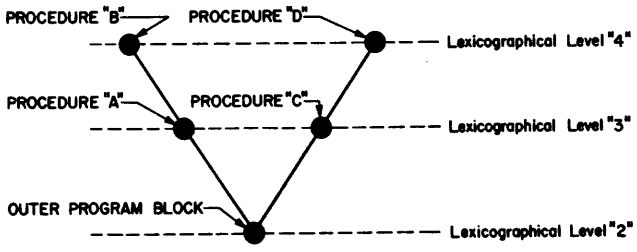


Figure 6b—Addressing environment tree of ALGOL program in Figure 6a

The B6500/B7500 contains a network of Display Registers (D0 through D31) which are caused to point at the appropriate MSCW (Fig. 5). The local variables of all procedures global to the current procedure are addressed in the B6500/B7500 relative to the Display Registers.

The address couple is converted into an absolute memory address when the variable is referenced. The lexicographical level portion of the address couple functions to select the Display Register which contains an absolute memory address pointing at the MSCW related to the procedural block (environment) where the referenced variable is located. The index value of the address couple is then added to the contents of the Display Register to generate an absolute memory address to locate the variable.

It should be recognized that the address couples assigned to the variables in a program are not unique. This is true because of the ALGOL scope of definition rules, which imply that two variables may have identical address couples only if there is no procedure within which both of the variables can be addressed. So this addressing scheme works because, whereas two variables may have the same address couples, there is never any doubt as to which variable is being referenced within any particular procedure.

What this does imply, however, is that there is a unique place (a MSCW) to which each Display Register must point during the execution of any particular procedure, and that the settings of the Display Registers might have to be changed, upon procedure entry or exit, to point to the correct MSCW. This list of MSCWs to which the Display Registers must point is called the addressing environment of the procedure.

The addressing environment of the program is maintained by the hardware. It is formed by linking the MSCW's together in accordance with the lexicographical structure of the program. This linkage information is contained with the Stack Number (Stack No.) and Displacement (DISP) fields of the MSCW, and is inserted into the MSCW whenever a procedure is entered. The contents of the DISP field indicate the environment in which the entered procedure was declared. Thus the addressing environment list is formed by linking each procedure entry Mark Stack Control Word back to the MSCW appearing immediately below the declaration for that procedure. This forms a tree structured list which indicates the legitimate addressing environment of each procedure under dynamic conditions (Figs. 5 and 6B). This list is searched by the hardware to update the Display Registers' contents whenever a procedure entry or exit occurs.

The entry and exit mechanism of the Processor hardware automatically maintains both stack lists to reflect the current status of the program. Therefore, the system is able to respond to, and return from, interrupts conveniently. Interrupt response is handled as a procedure entry. Upon recognition of an interrupt condition, the hardware causes the stack to be marked, inserts into the stack an indirect reference word (address couple) pointing to the interrupt handling procedure, inserts a literal constant to identify the interrupt condition, and then causes an entry into the operating system interrupt-handling procedure. The Display Registers will track with the entry into the interrupt-handling procedure to make all legitimate

variables visible. Also upon return, the Display Registers track back to the environment of the former procedure, making all of its variables visible again.

Multiple stacks and re-entrant code

The B6500/B7500 stack mechanism provides a facility to handle several active stacks. These stacks are organized into a single tree structure. The trunk of this tree structure is a stack which contains certain operating system global variables, and contains all of the Segment Descriptors describing the various procedures within the operating system.

Let us make a distinction between a program, which is a set of executable instructions, and a job, which is single execution of a program for a particular set of data. As the operating system is requested to run a job, a level-1 branch of the basic stack is created. This level-1 branch is a stack which contains only the Segment Descriptors describing the executable code for the named program. Emerging from this level-1 branch is a level-2 branch, a stack to contain the variables and data for this job. Thus, starting from the job's stack and tracing downward through the tree-structure, one would find first the stack containing the variables and data for the job (at level 2), the program code to be executed (at level 1), and finally the operating system's stack at the trunk (level 0).

A subsequent request to run another execution of an already-running program would require that only a level-2 branch be established. This level-2 stack branch would sprout from the level-1 stack that describes the already running program. Thus two jobs which are different executions of the same program will have a common node, at level 1, which describes the executable code. It is in this way that program code, which is not modifiable, is re-entrant and shared. It comes about simply from the proper tree-structured organization of the various stacks within the machine. Thus all programs within the system are re-entrant, including all user programs as well as the compilers and the operating system itself.

The B6500/B7500 stack mechanism also provides the facility for a single job to split itself into two independent jobs. It is anticipated that the most common use of this facility will occur when there is a point in a job where two relatively large independent processes must be performed. This kind of splitting could be used to make full use of a multiprocessor configuration, or simply to reduce elapsed time by multiprogramming the independent processes.

This kind of program splitting becomes almost literally "reproduction by budding" in the B6500/B7500 system. A split of this type is handled by establisting a new limb of the tree structured stack, with the two independent jobs sharing that part of the stack which was created before the budding was requested. The process is recursively defined, and can happen repeatedly at any level. An implementation restriction limits the total number of separate stacks to 1024.

This tree-structured organization for handling multiple stacks is referred to as the Saguaro Stack System.

Linkage of stack branches is achieved through a single array of data descriptors, the stack vector array (Fig. 7). A data descriptor is entered into the array for every stack branch as it is set up by the operating system. This data descriptor, the stack descriptor, serves to describe the length of the memory area assigned to a stack branch, and its location in either main memory or on disk.

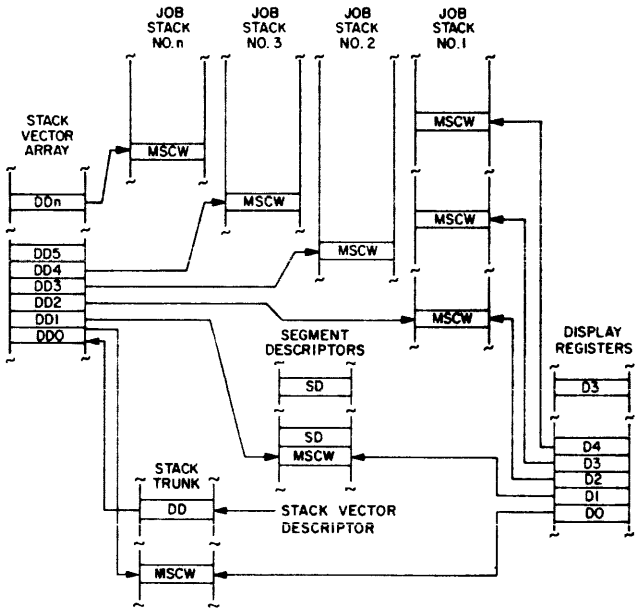


Figure 7 – Multiple linked stacks

A stack number is assigned to each stack branch to indicate the position of its stack descriptor within the stack vector array. The stack number is used as an index value to locate the related stack descriptor from the stack vector array for subsequent reference.

The stack vector array's size and location in memory is described by the stack vector descriptor. This descriptor is located in a reserved position of the stack's trunk (Fig. 7). All references to stack branches are made through the stack vector descriptor which is indexed by the value of the stack number

to select the stack descriptor for the referenced stack.

A Presence Bit Interrupt is incurred upon making reference to a stack which is not present in memory. This Presence Bit Interrupt facility provides the means to permit stack overlays and recalls under dynamic conditions. Idle or inactive stacks may be moved from main memory to disk as the need arises, and when subsequently referenced will cause a Presence Bit Interrupt which triggers the operating system to recall the non-present stack from disk.

Referencing a variable within the current addressing environment of an active procedure is accomplished through the use of the address couples contained in the IRW and the address couple field of the Program Control Word (PCW) as shown in Fig. 1. Both references are made relative to the Display Registers specified by the address couple. The address couple and Display Registers are usable only for addressing variables within the scope of the current addressing environment. Reference to variables beyond the scope of the current environment is accomplished by a stuffed IRWS. This causes the addressing to be accomplished by addressing relative to the base of the stack (BOS) in which the variable is located.

The IRWS contains information specifying the stack number (Stack No.), the location (DISP) of the related MSCW, and the displacement (δ) of the parameter relative to the MSCW. The absolute memory location of the sought parameter is formed by adding the contents of DISP and δ to the base address of the referenced stack. The base address of the stack is determined by accessing the stack descriptor as described previously. The information contents of the stuffed IRWS with the exception of δ, is dynamic in nature and must therefore be accumulated as the program is executed. The contents of the stack number (Stack No.) and DISP fields are entered into the IRWS by a special hardware operator which is invoked by the software whenever the program attempts to pass a parameter by name.

ACKNOWLEDGMENTS

Recognition for the stack concepts and operating philosophy of Burroughs' B6500/B7500 system must be extended to many system designers engaged in both the B5500 and B6500/B7500 programs. Among the contributors, special mention should be made for B. A. Creech, Burroughs Corporation, and R. S. Barton, W. M. McKeeman, consultants.

REFERENCES

1 *Burroughs' B5500 information processing system reference manual*
Burroughs Corporation 1964

2 *A narrative description of the Burroughs' B5500 disk file master control program*
Burroughs Corporation 1965

3 B RANDALL L J RUSSELL
ALGOL 60 implementation
Academic Press 111 5th Avenue New York 1964

A compact, economical core memory with all-monolithic electronics

by ROBERT W. REICHARD and WILLIAM F. JORDAN, JR.
Honeywell Computer Control Division
Framingham, Massachusetts

INTRODUCTION

The computer memory business has been plagued at various times during the past 8 years with a cliche that the art of core memories would be exhausted within 5 years. This paper describes the attainment of new standards of size and cost and new design features intended to further postpone this elusive demise.

The new standards include greater reductions in cycle time, volume, and selling price than had been hoped for. These significant improvements involved the following factors:

Use of monolithic integrated circuits for *all* major electronic functions related to the core stack;

Integration of a core driver transistor with decoding/timing logic circuitry;

A novel packaging concept without wired backboard;

A high-performance power supply subsystem.

Over the past decade, the cost per bit and cycle time of state-of-the-art core memories have simultaneously been halved every 2½ years. In large measure, this is due to the continuing competitive market and to the appearance of new suppliers, some of whom have yet to become significant factors in the market. Some short-lived competitors grasp an occasional opportunity and depart, affecting the market by their having appeared.

The core memory business may not have attained a semblance of maturity and stability. Opportunism has been amply demonstrated as an unsatisfactory business approach in the long run. A classic approach to design, product development, and release to manufacture has been recognized as desirable. There is a paradox here in that, as the market grows and larger production runs become necessary and possible, tooling requirements increase in scope and duration and the conception-to-production cycle tends to increase. Perhaps this is the reason that several of the major suppliers in the core memory business have been able to continue in business despite the fact that they are essentially specialty-houses with platoons of project engineers. However, there seems not to be general tacit admission that the economy will no longer support such frivolity; or more properly, that such approaches can no longer compete against that of a disciplined organization.

The logical conclusion from this set of constraints is that design capabilities and requirements must be projected far into the future, and an organization must be willing to back long, expensive programs which generally admit to rather high risk. The success of one or more suppliers in doing this can mean only jeopardy to those attempting to operate in the old "job-shop" manner.

Product requirements

A planned, disciplined approach with stated objectives of main-frame capacities, fast cycle time, economy, fast delivery, modularity, and small size will be discussed in this presentation.

An obvious, evolutionary tendency has been for memory capacities to become larger in order to meet the needs of ever more powerful hardware and software products. The system described provides a maximum capacity of approximately 300 kilobits—as 8192 words of 36 bits, or 16,384 words of 18 bits. The limiting capacities are a result of cost-speed-capacity trade-offs as well as the limit of signal/noise ratio affected by sense-winding geometry and length. This system, with its power supply unit, is shown in Figure 1.

Detailed studies of the maximizing the performance/cost index of core memory systems dictated that the mode of operation of this memory be that known currently as 2½D. The technique dates back more than a decade, but only recently has become economically attractive, due to the advent of integrated circuits.

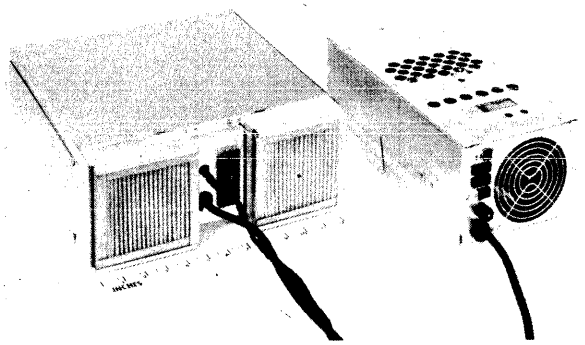


Figure 1 – 300-Kilobit memory system and power supply

The technique is an elemental implementation of a coincident-current memory in which one axis of read-write circuitry is replicated for each bit; the result is repetitive assemblies, but of drive lines which are short, since they need not be continuous through the stack. This permits a new degree of freedom in optimizing the aspect ratio of bit areas for one or another parameter. A further gain is the obvious dispensing with inhibit windings, enabling smaller cores for a given size wire, and the saving of time formerly allocated to insuring time overlap on inhibit and write currents, plus time required for recovery of sensing circuitry from the inhibit transients.

Another goal stated at the initiation of this product development was the requirement for reasonably fast delivery, to order, of a variety of capacities, which is at odds with the concept of a hard-wired custom core stack. This last item can be expedited by the act of inventorying tested planes and stacking to order or inventorying an array of stacks; but clearly a preferable route is to avoid stacking in its traditional sense. In the subject memory no hard-wired stacking ever is done; the implementation of an assembly of planes is achieved by pressure-contact connectors making a one-to-one connection between facing surfaces of adjacent boards. Sixty-four such one-to-one connections are made by each proprietary connector. The resultant connections not only achieve the linking of the long drive axis but also serve to distribute dc power and logic signals through the resultant assembly.

The results of using stackable planes include not only the obvious ones of less restrictive inventory capability and/or less dependence upon the supplier but also lower cost, since stack test and tedious stack repair are eliminated. Maintenance and repair are also simplified since plane replacement is practical, and a true capability for incremental field expansion is made possible.

Modularity of capacity was also a prestated goal at the initiation of this design. As with any manufactured product, a considerable degree of standardization is necessary to achieve economical manufacture. The modularity of the present equipment is fixed by the design at 32,768 bits, or the content of one plane. Thus, increments of capacity are 4 bits in 8K memories or 2 bits in 16K memories. Inasmuch as the 4K memory uses half-density planes, the modularity is also 4 bits at that capacity. Only five other major electrical circuit modules are required, with a total diversity of eight assemblies, to implement systems of 33K through 300K bits of storage. Standardization is further attested by the fact that only seven IC flat packs are used throughout. These factors result in a system which requires minimal inventory vis-a-vis a broad range of capacities that can be assembled in short periods of time.

Electrical design

In present-day state-of-the-art electronic equipment it is becoming more and more difficult to dissociate mechanical design from electrical design. In this design it was found possible to avoid artificial delineations and in fact to capitalize upon the interdependence of spatial and electrical characteristics both in a miscroscopic (e.g., etched conductors) and macroscopic (e.g., organizational) sense. Thus, the following discussion treats those areas in which the electrical requirements predominate.

Packaging organization of this memory system was conceived to emphasize an intimate, orderly relationship between data logic and storage. All of the circuitry associated with the regeneration of data from a core plane is located on peripheral plug-in cards. The cards and cores are mounted in a coplanar relationship permitting an orderly stackup of complete regeneration channels.

A 4-bit, 8192-word data plane is illustrated in Figure 2. The core plane contains conventional 20 mil o-d cores requiring nominally 800-mA full-drive current, in a conventional $2\frac{1}{2}$ D three-wire rectilinear array. The long lines are re-entrant such that when current linking coincidence occurs in a selected core, anti-coincidence occurs in that core corresponding to the second intersection. Furthermore, the two positions of each re-entrant line link separate sense windings, two of which are interleaved within each bit area. This results in sense-line noise of 64 delta-pairs on the long lines and only 8 pairs on the short lines. Printed-circuit sense wiring pairs connect to monolithic sense amplifiers mounted on the data cards. In addition to optimal differential terminators, a resistor is provided on

each data card to optimize the common mode termination. High-level TTL gates are also mounted on the data cards for storing, gating, and output transmission line driving functions.

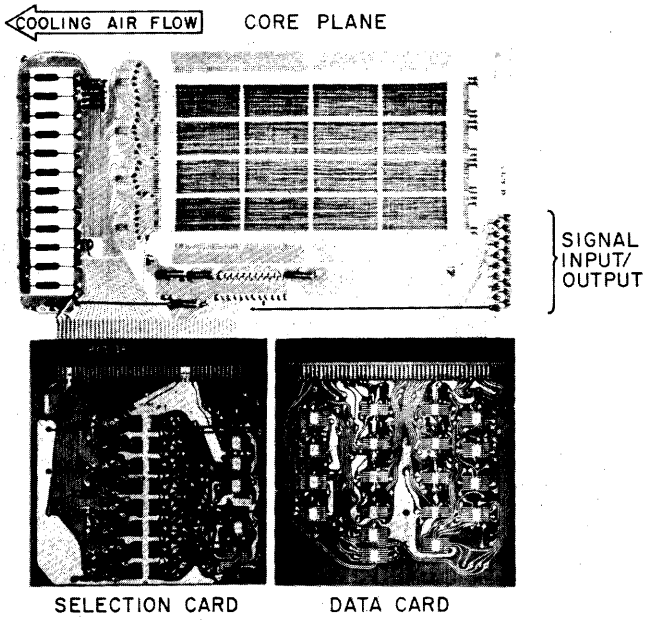


Figure 2 – Organization and construction of data plane

Adjacent to the data card is the selection card, which contains all of the decode-drive circuitry required for the short drive selection axis. An 8192-word selection card contains circuitry for four 4×4 matrices; a 16,384-word selection card contains two 4×8 matrices. Storage for the associated address bits is provided by R-S TTL flip-flops mounted on each selection card. Although these address flip-flops are repeated on tach plane their ulilization is not extravagant. The redundant storage actually serves as local modular buffering for the double rail address signals. The organization of address storage is unique in this unit. Address registers respond to their inputs at all times except when the memory is busy. This is the so-called open-ended address register configuration. "Look ahead" gating is also used to shorten the address delays by a technique similar to that of feed-forward in servo systems.

The long-axis drive and control circuitry is packaged on the top and bottom planes to sandwich the core stack. The top plane contains integrated selection diodes for the long-axis 16×16 matrix. A control card and a long-axis selection card are both mounted on each end plane. The control card contains logical gates and an electrical delay line for the generation of memory timing signals. Each long-axis selection card contains 16 switch-sink pairs for one coordinate of the 16×16 matrix.

The key to this packing desity is the very compact decode-drive circuitry which is constructed of monolithic chips. One chip contains two 400-mA switch pairs with decoding for eight or fewer address bits. Since logical ground and the sink emitters are common, this function has exactly 14 pins for use in a standard flat pack. The smaller matrices required on the short axis of the $2\frac{1}{2}$ D memory organization also efficiently employ this function; unused address inputs are used for the modulation and zoning of data.

A line selection matrix organization has been conceived which relieves some of the integrated-circuit liabilities: voltage breakdown, pin limitation, and accidental destruction. This arrangement of decode-drive chips with a memory current-limiting resistor prevents catastrophic damage from inadvertent simultandous activation of read and write. It avoids the requirement of saturating a switch to a memory drive voltage and therefore avoids the need of a higher-voltage supply. Any output may be shorted to ground without damage. The drive current resistor is time-shared for the read and write halves of the memory cycle. The single resistor (having full duty cycle dissipation capability) is clamped to limit the IC voltage requirement. A technique is employed with the long-axis address selection matrix for reducing the loading effect of drive line bus capacity; integrated diodes are used to segment the 16-line busses into four groups of four drive lines.

Circuit layouts were judiciously considered to satisfy the problems of signal transmission, noise, and termal management. On the selection board, driver flat packs are mounted on a wide copper lamina with a thermal compound. The row of flat packs is perpendicular to the cooling air stream to avoid cumulative heating effects, since, in the worst case, each flat pack can dissipate some 600 milliwatts. The lamina is expanded, filling the unused area on the selection board to enlarge the cooling area. The complete regeneration path is organized with the cooling air stream passing over the heat-sensitive cores and sense amplifiers first, then the diode matrices and selection circuit, and finally the very-high-dissipation drive resistors.

Care was taken to employ as few types of materials and assemblies as possible. As part of this theme, a scheme for replacing odd and even core planes with just one assembly was conceived. Three types of card layouts and three types of board layouts satisfy all layouts and no logical hook-up wiring, the optional system characteristics – signal inversion,

data levels, and partitioning zones – are accomplished by the proper placement of jumpers on the circuit cards.

From an electrical point of view, it would have been tempting to use multilayered cards with ground planes. This expensive approach was avoided, however, by carefully laying out minimum signal lines and filling any unused area with ground or supply bus patterns. The data card has been especially considered to insure balanced sense lines and the absence of electromagnetic or electrostatic coupling. On the core planes the sense lines are surrounded with ground peninsulars for current-free electrostatic shielding. The sense boundary connector pins on the data card are connected on one end for the same reason.

The creation of ground noise is minimized on the selection board by an alternate ac circuit return which is laid out for close coupling to the input drive current paths. The mutually cancelling effects of flux linkage from opposing fields leaves only magnetic leakage inductance to impede the flow of current.

The degree to which a system is implemented by repeated subassemblies is an important diagnostic characteristic. High repeatability decreases the needed spare parts inventory. Repeatability of disconnectable subassemblies permits transposition of parts for localizing faults, and comparison for tracing. In this system, all of the signal paths (including drive currents) associated with a data bit are limited to a specific plane level for orderly modularity. All of the active circuitry is located on plug-in cards. Many of the selection diode flat packs (36) are on exposed surfaces of the assembly. The remaining (52 in an 8K × 24) can be reached by disassembling the stack. All of the flat packs and all of the plug-in cards are repeated.

Mechanical implementation

As was indicated previously, the memory module described herein provides 300 kilobits of storage in substantially less than 1 cubit foot in a package which includes cooling fans and filters. In actuality the package contains some 20% unused volume, so that the equipment represents very high functional packaging density for a high-performance commercial memory. One fact in the space economy is the relative lack of discrete components. Another very important one is the lack of a conventional backboard. Two proprietary connectors are instrumental in achieving this. One is the stacking connector mentioned previously, which provides 64 connections on 0.050-inch centers, and incorporates precise aligning pins. The contacts are precious metal, as are the pads on the mating PC boards. The other connector is an 88-pin edge connector with a double rank of precious metal-tipped contacts on 0.100-inch centers, and provision on the affixing end for mechanical and solder retention to a PC board. The application of this connector is to effect an edge-to-edge connection of numerous close-spaced contacts with a minimum of hardware. These contacts also mate with precious metal-plated pads on another PC board. This edge connector is in practice affixed to each plug-in board, and is therefore replaceable if ever necessary. Thus, there is no backboard in the conventional sense and most interboard connections have essentially a short, straight run with minimal connector capacity and minimal length, hence minimal inductance.

Figure 3 indicates the internal construction of the system.

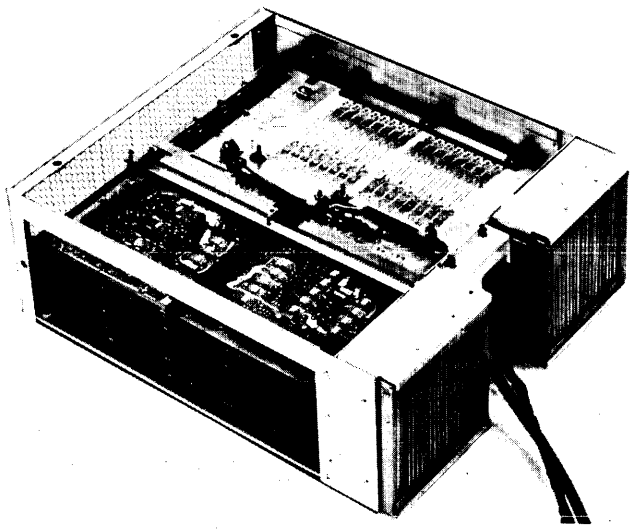


Figure 3 – View of internal construction of memory system

There is some conventional wiring present in this system, and that is, in addition to the obvious power wiring, the input and output signal wiring from the I/O connector to each board in the "core stack." These wires are all twisted pairs, affixed to a crimp-on, poke-home contact at the I/O connector end and at the other end to a crimp-on contact which slides onto a 20 × 30-mil rectangular pin affixed to the stack boards. There are no hand-soldered wires in the unit except in the ac fan power wiring. Note that the signal leads enter and exit via the stack boards even though all electronic functions are performed on the plug-in boards. This is done so that insertion and extraction of boards is kept simple and that the number of connectors is kept to a minimum. That signal paths are lengthened by this procedure may be questioned, but

any added length is over well-controlled and absolutely repeatable signal paths.

A single, dense I/O connector containing 200 pins is utilized. A central jackscrew and positive alignment pins insure against damage to pin or shell by mismating. The mating shell and loose pins furnished to a customer with each unit may be crimped or soldered, and no tools are necessary to insert the wired contacts.

High-performance card extenders were developed to permit operation of any plug-in board in position for maintenance checks without compromising system performance. This is achieved through a simplified multilayer board technique. The added ground planes reduce coupling effects between lines, and lower the lime impedance to avoid discontinuities which have previously restricted the use of extenders.

Results achieved

The memory system described here is the Honeywell Computer Control Division ICM-500 μ-Store, a product first delivered in January, 1968 and now being produced in quantity. This system, with capacities as great as 8192 words of 36 bits, provides 600-ns cycle time and 300-ns access time. It is packaged in a sturdy but simple enclosure about 0.6 cubic foot in volume and 25 pounds in weight. This enclosure includes flushing fans and filters and may be mounted via either of two surfaces in any of several orientations.

Power consumption of the largest module is approximately 350 watts. More than half of the system power is dissipated in the set of drive resistors, all of which are located at the exhaust end of the cooling air stream. Four supply voltages are required, all referenced to ground. A proprietary power supply furnishes all of these voltages, with appropriate regulation, for one maximal module. It also provides the usual features of line-voltage sensing and low-line-voltage indication, with provision for enabling orderly and non-destructive shutdown as well as startup, overvoltage and overcurrent protection, remote-sensing for termperature compensation of drive voltage, and thermal overload protection. This supply is operable from a nominal 115-volt line, 50 through 400 Hz. It weighs only 45 pounds and occupies about one-half cubic foot.

By the application of reliability forecasting techniques refined via similar predecessor products, an MTBF of some 25,000 hours is forecast for this system and a unit is presently on life test to commence accumulation of supporting data. All logical and drive circuits are immune to failure due to accidental grounding, and the selection matrix design is such that excessive read and write currents cannot flow simultaneously and cause destruction to components or to stored data. In addition, the fault-localizing time is short since functional subunits are interchangeable. The availability of high-performance extender boards also enables functional boards to be extended for signal tracing without compromising operational speed. Test points, accessible without any disassembly or unplugging of cards, also facilitate the localizing of signal interface problems.

ACKNOWLEDGMENTS

The authors wish to acknowledge contributions to the development of this product by personnel of the Memory Products Department at Honeywell, Computer Control Division, particularly Mr. Dana W. Moore, and by members of related areas.

A progress report on large capacity magnetic film memory development

by JACK I. RAFFEL, ALLAN H. ANDERSON, THOMAS S. CROWTHER, TERRY O. HERNDON and CHARLES E. WOODWARD

Massachusetts Institute of Technology
*Lincoln Laboratory**
Lexington, Massachusetts

INTRODUCTION

In 1964 we proposed an approach to magnetic film memory development aimed at providing large, high-speed, low-cost random-access memories.[1] Almost without exception, all early attempts at film memory design emphasized speed with little consideration for the potential of batch-fabrication to reduce costs. Based on our earlier work in building the first film memory in 1959,[2] and a 1,000 word, 400 nsec model for the TX-2 computer in 1962,[3,4] we had reached some fundamental conclusions about the compatibility of high speed and low cost for destructive-readout film memories.

It seemed clear to us that in order to provide significantly more storage capacity per dollar it was necessary to achieve very high bit densities, to eliminate as far as possible internal connections, to fabricate access wiring integrally with the storage medium if possible, and to provide very wide magnetic operating tolerances in order to achieve adequate yields for arrays of 10^5 bits or more. It was also clear that for small memories the cost of storage elements is essentially irrelevant and that the area of significant impact for batch-fabricated films should be in large memories, i.e., greater than one million bits.

There were at least five areas of major uncertainty when this proposal was made:

(1) The signal to be detected in the presence of both random noise and parasitic transients was lower than in any previous magnetic store.
(2) Techniques did not exist for forming precise, dense (2 mil width, 2 mil space) lines over large areas (10″ × 1.5″) cheaply and reliably.
(3) Routine evaporation of uniform magnetic films over large areas with elimination of defects greater than 0.5 mil in any dimension remained to be achieved.
(4) Connection to many lines, even at the periphery, at these densities appeared difficult.
(5) Finally, processing and testing of such large arrays had to be sufficiently automated to justify the pains taken to achieve high density.

None of the questions raised here has been answered definitively, but the completion of a one million bit prototype memory (shown in Fig. 1), which is to be installed in the TX-2 computer, has provided us with enough encouragement on each of these points to justify a high degree of confidence and to open up exciting possibilities for extending these techniques further than had originally been anticipated.

The most interesting feature of the memory for system applications is the parallel-access to 352 bits. This multiword access should be useful for a number of applications such as parallel processing of the Illiac IV variety, list processing, searching, and display buffering. Although for the present TX-2 will handle a single subword of 44 bits on each memory access, the memory bussing arrangement is such that access may be made to each of the eight subwords during the memory cycle of less than one microsecond.

Memory design

A. Structure and organization

The basic structure of the memory is shown in Fig. 2. A single layer composite magnetic film is operated in a rotational destructive-read-out mode with two access wires.[2] Each bit is composed of two 2 × 6 mil intersections of the word and digit lines with a density of 12,500 bits/in^2. Magnetic film structures which provide flux closure in the hard, easy, or both

*Operated with support from the U.S. Air Force.

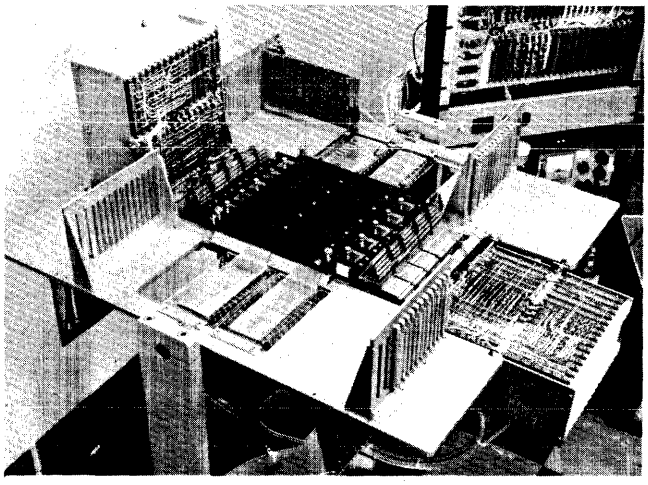


Figure 1—One million bit magnetic film memory. The memory stack is in the center, under the dark plate, with the word line connections and diode matrix at the top and bottom. Word access circuitry is in the two card files. One digit card is plugged into digit lines at the upper right-hand corner. A fully populated memory will have digit cards in the sockets on all four corners, top and bottom. The digit lines shown have not yet been connected to sockets.

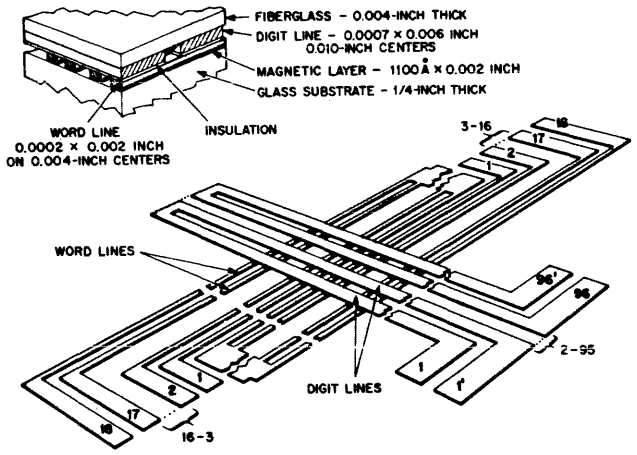
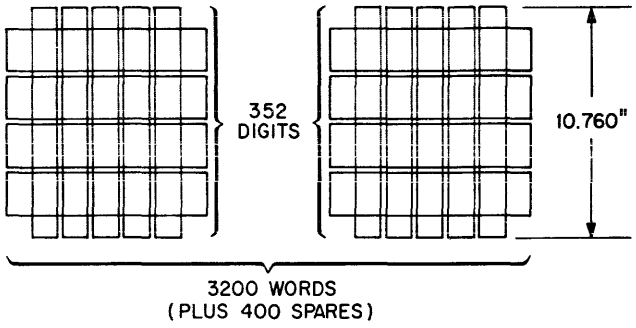


Figure 2—Detail of memory structure and arrangement of access lines. Two 2 × 6 mil intersections form one memory bit

directions were considered but rejected when adequate margins were obtained with the single layer. Although the open structure has fabrication advantages, the closed structures remain of interest for future work.

The organization of the one million bit memory is shown in Fig. 3. The ten inch long glass substrates each have 360 lines of 2 mil width which have been formed by etching vacuum evaporated magnetic and copper layers. Storage is in the magnetic layer while the copper forms the word line. Forty lines on each substrate are redundant and can replace other



352 SENSE AMPLIFIERS
352 DIGIT DRIVERS
100 WORD SWITCHES
64 WORD DRIVERS
PROTOTYPE LARGE CAPACITY MEMORY
1.1×10^6 BITS

Figure 3—Memory organization. Five substrates and four digit pieces comprise each half of the memory

defective lines through simple wiring on the edge connectors. Advantage has been taken of the low back voltage of the storage element to minimize stack interconnections by addressing many bits, up to 384, on each word line. The digit-lines are formed from pairs of 6 mil wide copper lines on 10 mil centers which have been etched from copper-clad fiberglass in a hairpin configuration as shown in Fig. 2. Two digit pairs are connected in a bridge connection to the digit driver and sense amplifier as shown in Fig. 4. Stack assembly is accomplished by pressing the glass substrates against the digit lines with no critical registration. The TX-2 memory, as shown in Fig. 3, uses ten substrates and 352 digit lines to form a 3200 × 352 stack, although to the computer it appears as a 25,600 × 44 memory. Since read-out is destructive, each digit-line requires a sense amplifier and digit driver. It was recognized at the outset that the overall costs for the prototype would be dominated by the digit circuitry and that an economical module size would use longer digit lines and more substrates.

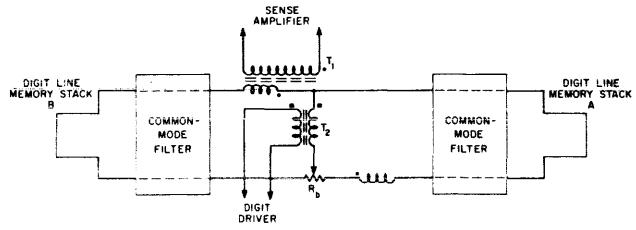


Figure 4—Digit coupling circuit

B. Digit circuitry

If one accepts the premise that low cost depends crucially on batch-fabrication whose effectiveness in turn depends on high density, the ultimate lower limit on bit size and signal energy becomes a determining factor in memory design. This limit is set by random noise considerations. It has been shown[5] that the mean time between failures for an M digit memory with a cycle time T and peak-signal to rms-noise ratio A/σ is given by:

$$\text{MTBF} = \frac{2T}{(1 - \text{erf}\, A/\sqrt{2}\sigma)\, M}$$

This relationship gives a required signal-to-noise ratio of 8.3:1 for a MTBF of one year for a 1 μsec, 400 bit memory. In this design the bit size was reduced to the limits of our then existing fabrication technology with a resultant bit signal amplitude of 130 μvolts, 35 nsec wide, at the sense line terminals. This small signal in the presence of rather large drive currents imposed stringent requirements on the sense amplifier and the coupling circuitry between it and the digit lines. The essential features of the coupler are shown in Fig. 4. The two halves of the digit line pair are connected in series to the sense transformer after passing through common mode filters. The digit current is transformer coupled to the two digit halves so as to feed them in parallel, thus providing a cancellation of the two digit currents at the sense transformer. A small potentiometer compensates for mismatch in digit line resistances. The sense system bandwidth is 14 MHz, the noise figure is 6 dB, and the peak-signal to rms-noise ratio is 27:1. A synchronous clamp samples the sense amplifier output just before strobe time and establishes a relative base-line reference from which to measure signal excursion, thus eliminating the effects of low frequency components in the digit transient. However, this introduces a random noise component in the baseline which leads to a total augmentation of the noise by $\sqrt{2}$, reducing the signal-to-rms-noise ratio to 19:1.

Two synchronous clamps are used as a SPDT switch to direct the sense amplifier output to the appropriate side of the strobe flip-flop, dependent on which half of the memory is being addressed. These strobe flip-flops are also the buffer storage and are arranged in a rectangular array of 44 bits by eight subwords. Any one of the subwords can be selected for writing into or reading out of the memory by external access circuitry.

The polarity switch for the digit pulse is a flip-flop which is transformer-coupled to the digit lines. Power is applied from a pulser shared by four digit circuits. The flip-flop state, and so the output polarity, is determined by the strobe-buffer flip-flop. Transition times are 25 nsec for the operating digit current of 190 ma $\pm$ 40 ma. Four complete digit channels are packaged on one card which plugs directly into sockets on the digit lines as shown in Fig. 1.

C. Word circuitry

Word-lines are connected in groups of eighteen lines (two of which are spares) on the substrate. The common end of a group is selected by a transistor switch and the lines driven through a diode matrix. Because of packaging considerations, the 3200 lines are driven by two 32 × 50 matrices. Word current amplitude is 500 ma with rise and fall times of approximately 35 nsec.

D. Nonrandom noise

Great care is necessary in the stack design and fabrication to minimize all digit-line difference mode voltages other than the signal.[6] At signal time, noise may be generated by either inductive or differential capacitive coupling between word and digit-lines, both of which may be caused by line defects and spacing nonuniformities. Maximum allowable noise of about one half signal amplitude is determined by random noise and strobe threshold uncertainty. Noise due to group-switch selection voltages and the digit transient largely determine the memory timing.

E. Timing

The access time of the memory from change of address to information output from the buffer flip-flops is about 450 nsec. The largest contribution to this delay is the transient on the sense-line due to group-switch voltage transitions. The circuit-limited cycle time for read-rewrite or clear-write is 600 nsec. Recovery from the digit-pulse transient limits the total cycle time to 1 μsec with the digit transient overlapping the group-switch transient.

Production techniques

A. Film preparation

In order to provide the very high single-bit margins necessary to obtain good yields on 100,000 bit arrays, a wall coercive force specification was chosen which was well above the maximum digit writing field. The film specifications are $H_c \geq 15$ Oe, intrinsic $H_{k_i} \leq 20$ Oe, $\alpha 90 < 5°$, skew $\beta < 2°$, and thickness d = 1200 $\pm$ 150Å. The best method found to obtain these characteristics was to deposit a two-layer film,[7] having typical characteristics shown in Table I.

TABLE I – Characteristics of Composite Magnetic Film

	d	H_k	H_c	$\alpha 90$	β
First layer 50% Co 47% Ni 3% Fe	800Å	25 Oe	25 Oe	±3°	±0.5°
Second layer 83% Ni 17% Fe	400Å	3 Oe	2 Oe	±3°	±10°
Composite	1200Å	15 Oe	15 Oe	±3°	±1.0°

The large skew of the second layer is due to the off-center position of its melt. The skew of the composite film, being largely determined by the thicker, higher H_k first layer, is within specification. The substrate is ¼" soft glass, 10.76" x 1.6", optically polished on one surface. It is cleaned in detergents, spray-rinsed, then dip-rinsed in distilled water before air-drying in a filtered-air bench. No ultrasonic cleaning is used as this may cause pitting of the surface. A drum holds fourteen substrates, and evaporation takes place from rf induction heated melts. A 24" source-to-substrate distance reduces film thickness variation to ± 2.5% over the center 8" of the substrate occupied by memory elements. A layer of Cr about 100Å thick is first deposited on the substrate to improve adhesion. The magnetic layers are deposited at a substrate temperature of 335°C and then 5 μ of copper at 150°C. The rate of copper evaporation must be kept below 16 μ per hour to eliminate spattering tiny balls of molten copper onto the substrate.

B. Fabrication

Each substrate has 360 word lines in twenty groups of eighteen lines. Each group is terminated in a common pad at one end and each line in a separate pad at the opposite end. Groups are interleaved so that 180 lines are connected to diodes at each end. The arrangement of connection pads is shown in Fig. 2.

Either holes in the line or bumps on the edge of a line may cause word noise. A good word line must have no defect larger than 0.5 mil in its largest dimension. After experiencing considerable difficulty in obtaining the desired line quality with photoexposure of a resist through a mask, a mechanical scribing technique was developed for line definition. The substrate with its magnetic alloy and copper coating is coated with a thin layer of photoresist. The connection pads at the ends are photoexposed from a master pattern and the resist developed and cured. The word-line pattern of two mil lines on four mil centers is then scribed in the photoresist, without penetrating the copper, using a diamond tool. The automated scribing machine which controls the tool has been built on a coordinatograph (Fig. 5). After a setup time of ten minutes, the machine operates unattended for the 2½ hours required to scribe one substrate. The metal layers are etched using standard procedures. The line edge smoothness is excellent. A typical substrate has perhaps three unacceptable nicks or opens and 25 unacceptable bumps or shorts. Scribing defects are nearly always attributable to defects in the copper layers which may cause the tool to jump or slide over some resist, thus accounting for the preponderance of bumps. These are easily repaired after etching by direct removal with a special scribing tool. The ends of the substrates are gold plated and the active area coated with a 0.4 mil layer of photoresist which insulates the word from the digit lines.

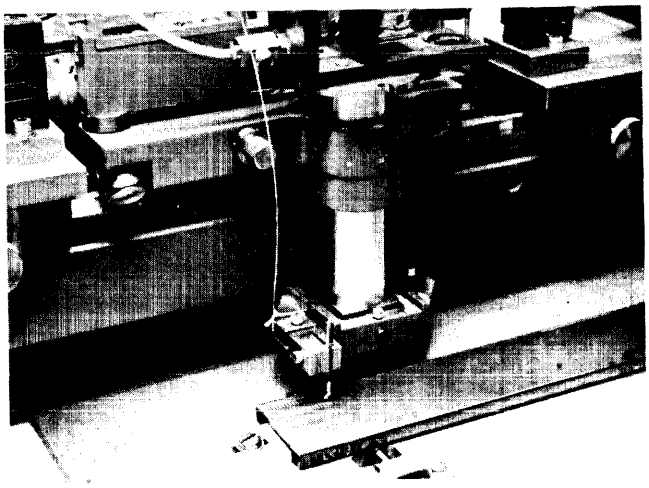


Figure 5 – Scribing machine with a substrate being scribed. The photo-exposed word-line pads can be seen

Connection to the word and group pads is by a pressure connection so that connections to a substrate in a tester or the memory can be made very easily. The connector assembly uses spring-loaded pin connectors to which the diodes of the selection matrix are wired. Substitution of spare for defective lines is done on this connector.

The digit line patterns are eighteen inches long, two inches wide, and consist of 192 lines, six mils wide on ten mil centers (96 pairs). Half this length is active digit line, the other half being required for fan out for direct connection to the digit cards as is shown in Fig. 1. The memory uses eight such patterns. Digit conductors must be entirely free of shorts and opens and can have no holes in a line greater than one mil in diameter. The digit line is made by etching half-ounce (0.7 mil) copper laminated to five mil glass-epoxy. Exposure of the resist is by a projection printing technique in which the photomaster is spaced 25 mils away from the resist layer. A

traveling front-surface mirror is used to paint the master and substrate with collimated light.[8] This eliminates all degradation of the photomaster, and by projecting the light left and right a few degrees off the normal any remaining dust particles are undercut by the light and prevented from causing flaws in the resist coating. The photomasters are generated actual size by scribing emulsion-coated glass plates on the automatic scribing machine.

C. Assembly

The two halves of the memory shown in Fig. 3 are assembled on two sides of a ground plane as can be seen in Fig. 1. A resilient material spaces the digit lines from the ground plane while the substrates are pressed against the digit lines by a cover plate with air bags. The resilient backing and air bags are necessary to achieve close spacing between word and digit lines with loose tolerances on digit-line material and substrate thicknesses. The stack is assembled under clean conditions to eliminate dust particles which can cause spacing irregularities. Word connections are made on two sides of the stack and digit cards plug into digit-line sockets on the other two sides.

D. Testing procedure

The testing procedure is designed to eliminate as many defective substrates as possible before the more expensive processing steps of complete inspection, repair, and electrical testing. The composite film substrates have wide operating margins with uniform large-scale characteristics. Furthermore, the causes of errors at individual bit positions are well known and easily detected. These two factors make simple preliminary screening effective. After the substrate has been coated, it is tested in a B-H looper which measures average total flux, H_k, H_c, dispersion and skew. The pinholes greater than one mil in diameter are counted and adhesion checked. After scribing and etching the film is again looped, line resistance checked, and line edge quality evaluated. The substrate transparency and regularity of the word lines make it possible to see small defects with the unaided eye. At this time curves of signal vs digit current of several bits may also be made as a check on the B-H loop data.

All individual bit errors are attributable to physical defects such as scratches, pits, or dirt on the glass or holes in the layer of magnetic material all of which may decrease both signal amplitude and operating margins. Defects in the copper line edge larger than 0.5 mil may cause undesired inductive coupling of word current into the digit line. The effect of a defect is often quite critically dependent on its position with respect to the digit line; a translation of two mils may be the difference between a good and bad bit. Word lines are continuous and to simplify memory assembly no attempt is made to maintain any word to digit line registration. In final testing, therefore, it is assumed that any defect may occur at the worst position.

Fig. 6 is a photograph of the automatic pulse tester mechanism. The 360 lines of a substrate are accessed with a diode matrix. Only eight digit channels are provided and these digit lines are mechanically indexed automatically along the substrate to test the 384 possible digit positions. Several different worst-case tests are applied in sequence to each word at each digit position. The signal is sampled and tested at one threshold level, and on a failure the address and test number are printed. A test including 1,000 adjacent word disturbs at a ⅓ MHz rate, each test repeated sixteen times, requires about 45 minutes per substrate. To obtain worst position testing of all possible defects it is necessary to do this test several times at increments of several thousandths of an inch. Although the number of disturbs is lower than desirable, the test is made at a digit current level for disturbing which is 150% of the digit write current. It would be relatively straightforward to use higher clock speeds to decrease the total test time.

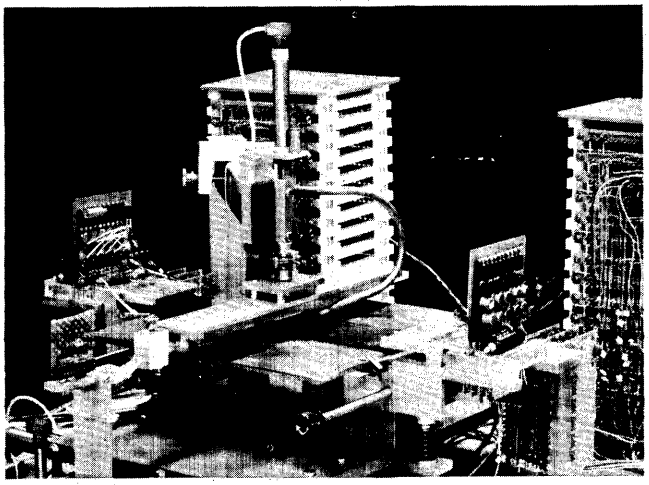


Figure 6—Substrate holding and positioning mechanism of the electrical tester. Connection is made to the substrate with spring-loaded pin connectors

The defect registration problem and difficulties with word noise in the tester have caused us to supplement the electrical test with optical inspection although improvements to the tester are possible which would make it alone sufficient. Using vertical illumination and looking through the glass substrate the operator views the magnetic layer at 100 power in a comparator as the substrate is moved along its

length on a motor-driven table. All defects except copper holes can be seen and are recorded. Although it is fatiguing, the substrate can be completely scanned in two hours. With presently planned electro-optical detection this time will be reduced to about one hour with better accuracy and minimal operator intervention. Acceptance of a substrate could be done after optical inspection alone, however, at present margins are examined in the tester at each defect position and a complete scan is performed as a double check. Optical inspection would clearly be difficult with multilayer closed structures and impossible with nontransparent substrates.

E. Yields

In interpreting yield data it is important to recognize that processing parameters and specifications for the runs included have been barely stabilized. However, the yield question is so fundamental to the overall objective of reduced costs that it is essential to provide some hard data no matter how provisional or unrepresentative of ultimate possibilities.

The yield data is taken from ten successive production runs of the evaporator starting with the first one in which composite films were evaporated to memory specifications. Of the total 131 substrates, 22 were rejected for thin copper and copper surface imperfections, 16 did not meet magnetic specifications, and one was damaged. These 39 were rejected before any processing; during processing 50 were rejected before final test for the following reasons:

(1) spatter bumps on copper surface—too high an evaporation rate	16
(2) poor adhesion	7
(3) damaged in processing	9
(4) poor line edges due to a bad scribing tool	16
(5) other	2

Of the remaining 42, 23 were rejected in final testing, 21 for excessive defects and 2 for poor magnetic characteristics. The 19 acceptable represent 15% of the total and 45% or those reaching final test. It is quite certain that the large attrition due to poor copper, poor line edges, and damage can be significantly reduced. In one run in which there was no loss due to evaporation or processing defects, fourteen substrates were processed and nine were acceptable.

The importance of providing redundancy should be emphasized. In the acceptable substrates two to eight lines were defective. Some of the 21 substrates rejected for excessive defects would have been acceptable if line substitutions were permitted on a 32 instead of a 16 line-group basis since spare lines are committed to a group by the bussing on the substrate.

Costs

Extrapolating costs from a prototype built using experimental techniques is always difficult. In particular, little effort has gone into minimizing the costs of associated electronics. On the other hand, it is possible to make some reasonable projections of material and direct labor costs for completely tested memory arrays based on present processing techniques. A single substrate cost under $100 appears to be a reasonable estimate (<0.1¢ per bit) assuming a potential yield of 50%. Present digit circuit costs are approximately $30 for parts and four man-hours/channel (3200 bits). Word circuit costs per word line are approximately $1.00 for parts and 0.1 man-hour (352 bits). It should be possible to make substantial reductions in the labor involved in all phases of the fabrication process; parts costs would also be substantially reduced for quantity purchases. The use of integrated semiconductors could radically reduce circuit costs.

Work has already begun on a memory which will use the techniques described above with digit lines extended by a factor of five in length to provide a better economic balance between digit circuit and other stack and circuit costs

Future developments

The processing machinery to accomplish the extended stack is sufficiently well developed that the main development effort can be applied to exploring possibilities for achieving perhaps another order of magnitude increase in word-line density. Preliminary scribing and etching have produced lines as narrow as 0.1 mil with good edge definition over large areas. At line widths corresponding to some five wavelengths of light the potential for batch-fabrication of multimillion bit arrays is accompanied by formidable problems in signal detection and interconnection. The technology of merging integrated semiconductor access circuitry and high-density film arrays presents considerable challenge with enormous dividends. It becomes reasonable, for the first time, to seriously consider the possibility of providing random-access static storage which is economically competitive with rotating mass memories.

ACKNOWLEDGMENT

Acknowledgment is made of significant contributions to this program by Robert Berger, Gilbert Gagnon, Elis A. Guditz, Charles Hoover, and Mark Naiman.

REFERENCES

1 J I RAFFEL
Future developments in large magnetic film memories
J Appl Phys 35 No 3 part 2 p 748-753 March 1964

2 J I RAFFEL
Operating characteristics of a thin film memory
J Appl Phys Supplement 30 No 4 p 60S-61S April 1959
3 J W FORGIE
A time- and memory-sharing executive program for quick-response on-line applications
Proceedings of the 1965 Fall Joint Computer Conference
4 J I RAFFEL A H ANDERSON H BLATT T S CROWTHER and T O HERNDON
The FX-1 magnetic film memory
MIT Lincoln Laboratory Technical Report No 278 August 1962
5 H BLATT
Random noise considerations in the design of magnetic film sense smplifiers
MIT Lincoln Laboratory Group Report 1964-6 August 1964
6 J I RAFFEL A H ANDERSON T S CROWTHER T O HERNDON and C E WOODWARD
A million bit memory module using high-density batch-fabricated magnetic film arrays
To be published Proceedings of the 1968 Intermag Conference
7 T S CROWTHER
Specifications and yields of composite magnetic films for a high density memory
To be published Proceedings of the 1968 Intermag Conference
8 E A GUDITZ T O HERNDON W J LANDOCH and G P GAGNON
Large area high density precision etched wiring
To be published Proceedings of the 1968 Electronic Components Conference Washington D C May 1968

A fast 2½ D mass memory

by C. C. M. SCHUUR
Philips' Gloeilampenfabrieken N.V.
Eindhoven, the Netherlands

INTRODUCTION

The mass memory described in this paper is a randomly addressable magnetic core memory having a storage capacity of 0.5 Megabytes.

Because of the many inherent advantages to be gained, such as low core-stringing costs, high speed and negligible heat dissipation within the core stack, a 2½ D selection organization is used. Since the principle of this organizational structure has been adequately covered in recent literature[1,2,4,5], it is assumed known and will not be discussed further herein.

In order to obtain maximum flexibility, the memory has been designed for five modes of operation as follows:

(1) Read	cycle time	1.3 /μs
	access time	1.2 /μs
(2) Write	cycle time	1.2 /μs
(3) Read-Restore	cycle time	2.5 /μs
	access time	1.2 /μs
(4) Read-Modify-Write	cycle time	2.5 /μs
	access time	1.2 /μs
(5) Clear-Write	cycle time	2.5 /μs

The values listed above are very conservative. Witl the prototype, which is a completely populated memory, a cycle time of 2 usec and an access time of 1 usec can be obtained under adverse conditions. These conditions are:

a) Tamb = 5 °C − 45 °C. b) worst case voltage tolerances of ± 10%. c) full worst pattern along word lines and bit lines. There are no limitations imposed on the sequence in which addresses are selected, or the number of times each individual address is selected sequentially in any of these five operating modes.

The design of the input and output circuits is such that the corresponding inputs or outputs of several memories can be linked by means of a single cable. By linking corresponding address bit inputs it is possible to increase the number of bytes per memory word. By linking corresponding data inputs and outputs the number of storage locations can be increased.

All the memory inputs and outputs are provided with gates. The gates can be controlled by the computer so that it is possible to use several memories or banks of memories interleaved, which greatly increases the effective byte transfer rate. Alternate memories or banks can be addressed every 100 ns.

As is evident from the foregoing although the storage capacity is limited, the memory has been developed for use with larger mass memory systems Small mass memories whose design is such that several of them can be combined to form a larger entity offer some significant advantages. They are relatively easy to mass produce, particularly the core stacks, and enable large mass memory systems to be tailored to the needs of the customer. When the memory modules in such a system are self-contained it is possible to effect repairs on a particular unit without having to switch off the whole system. One drawback of small 2½ D mass memories is that they require more electronic components per bit, however measures can be taken to overcome this. Because of their organizational structure, 2½ D memories of a given size and speed are cheapest when the word lengths are kept as short as possible. By choosing a word length of one byte (nine bits) a small mass memory can be made, without sacrificing the inherently low cost per bit of a larger 2½ D mass memory having longer words.

Should a mass memory of only limited storage capacity be required it will not be possible to combine several standard mass memories to obtain more practical word lengths. However, it will be shown later that by making the most of the separate sense wires and the arrangement of word-line selection switches, words of up to 144 bits (16 bytes) in length can be made with the standard core stack. Under these operating conditions the byte transfer rate per memory increases from 0.4 Megabytes/s to 2.6 Megabytes/s. It will be realized that a corresponding increase in the

necessary electronics is thus required, an increase in power consumption also occurring. A new interconnection technique has made it possible to keep the core stacks very compact. In the event of failure, which is very unlikely, the defective part of the stack can be replaced in a short time.

The word-line selection diodes (8704) can also be replaced quickly and easily. They are mounted on small printed-circuit cards which can be plugged into sockets mounted at the rear of the stack container, rather than on the stack itself or on the matrix planes.

The dimensions of the core stack and its associated electronics are 98 cm (39 in) height by 45 cm (18 in) width by 50 cm (20 in) depth. If power supplies are added these dimensions become 147 cm (59 in) height by 45 cm (18 in) width by 50 cm (20 in) depth. Since the heat dissipation within the core stack is low, ambient temperatures of up to 40°C can be tolerated; the lower temperature limit is 10°C.

The standard memory (520,000 words of 9 bits) will be sold for well under 2 ct (U.S.)/bit.

Core stack

The n^{th} bits of all words are arrayed schematically in a rectangle of 256 bit-lines by 2048 word-lines. Nine of these rectangles make up the complete matrix of 2304 bit × 2048 word lines as shown in Fig. 1. Because a matrix of this size with 30 mil cores at 30 mil centers would be unwieldy it has been divided into 36 submatrices. The nine sub-matrices in each column are arranged so as to make four sub-stacks. Each substack contains 130,000 words each of nine bits. The word-lines, split into nine sections by the division into sub-matrices, can easily be reconnected by dip-soldering. A number of sockets mounted on each substack locate the cards carrying the word-line selection diodes. Conventional wiring is used between the socket pins and the word-line terminals on the sub-stack.

Reconnecting the bit-lines separated by the subdivision is more complicated. Both ends of each bit-line in each sub-matrix are connected to conductors printed on polyimide foils. The correlated bit-lines of corresponding matrices in two adjacent substacks can be interconnected by simply pressing the foils together, as shown schematically in Fig. 2.

The two outermost sub-stacks are connected by pressing their foils against glass epoxy strips, on which are printed the same patterns of conductors as on the polyimide foils. These conductors are then joined by conventional wiring to the pins of the sockets for the cards carrying the bit-line selection diodes, which are mounted on the main stack frame. The frame also carries the clamping devices for pressing the polyimide foils together and the rails along which the sub-stacks

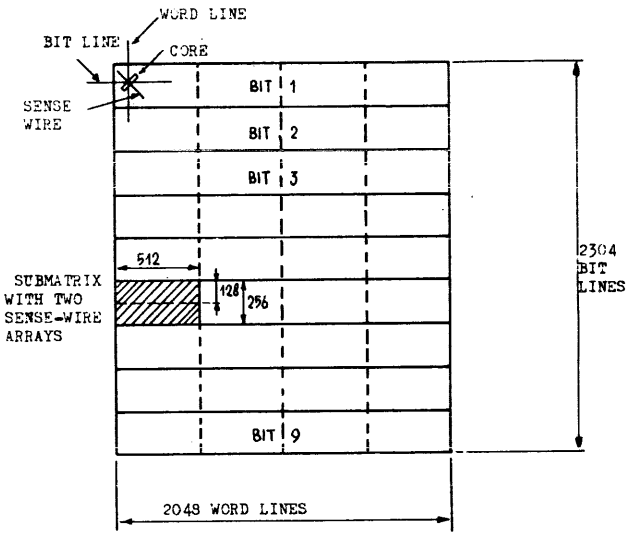


Figure 1 – Schematic representation of the core array

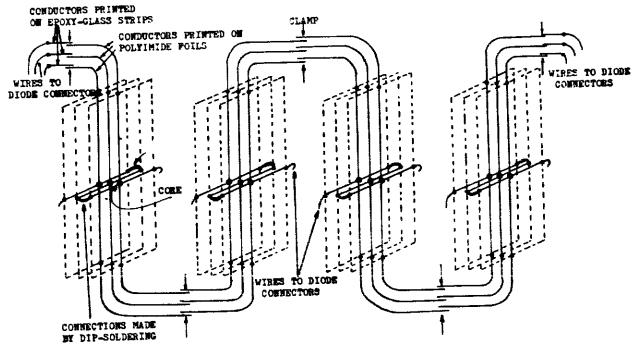


Figure 2 – Arrangement and interconnection of submatrices

slide into position. A main stack frame containing four sub-stacks is shown in Figs. 3a and 3b, a substack in Fig. 4.

Typical drive conditions for the cores used in this memory are:

Full current	700 mA (at 25°C)
Rise time	0.1 /μs
Pulse width	0.4 /μs

Under these conditons, the typical response values are:

rV_1	53 mV
wV_z	6 mV (disturb ratio = 0.5)
Peaking time	0.2 /μs
Switching time	0.4 /μs

Selection of bit lines and word lines.

As mentioned earlier diode-matrix selection is used in this memory. For the selection of nine bit-lines, one out of each of the nine groups of 256 bit-lines, nine diode matrices of 16 by 16 diodes are required. The rise time of the drive current through a bit-line is

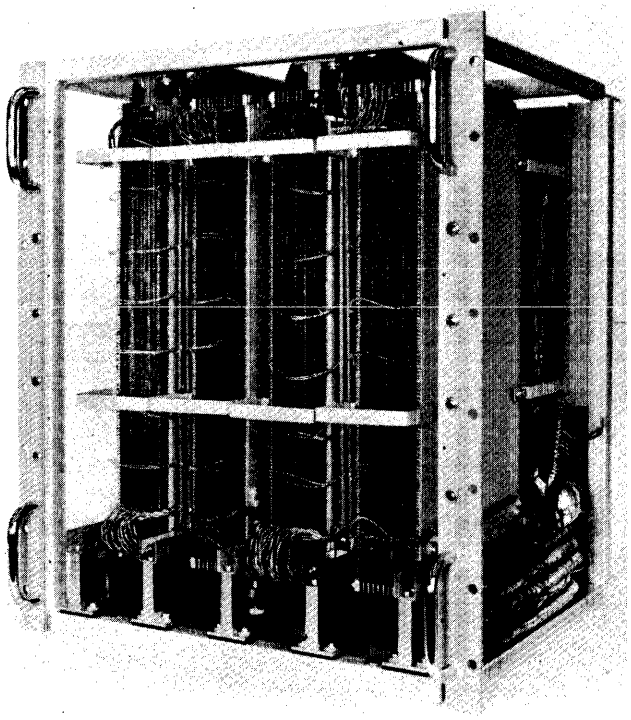


Figure 3a – Front view or core stack

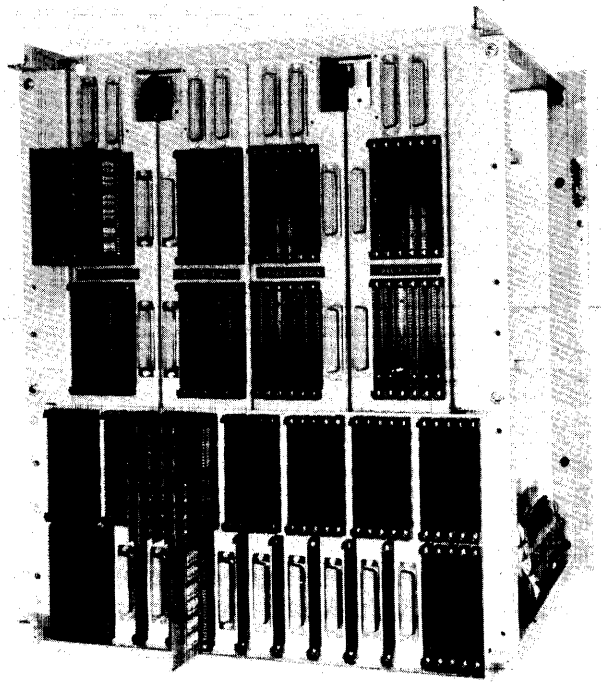


Figure 3b – Rear view of core stack

approximately 250 ns. To prevent deterioration of the "one" signal owing to slowly increasing word-line drive currents, the rise time of these currents should be less than 300 ns. The word lines are thus divided into eight groups of 256 lines, each having its own diode matrix consisting of 16 × 16 diodes.

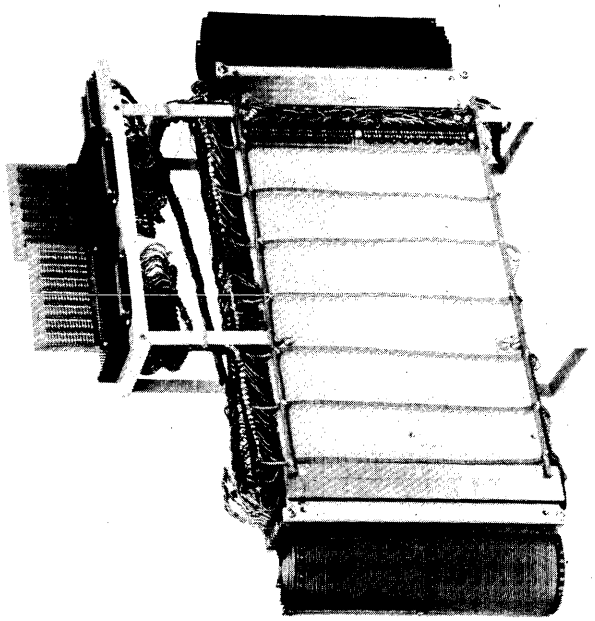


Figure 4 – Substack

A complete circuit for the selection of one line from a group of four is shown in Fig. 5. The circuit is quite conventional and straightforward. To reduce the possibility of amplitude differences occurring in the read and write currents, a single voltage source and one set of resistors are employed. The voltage source is floating and balanced with respect to earth by means of a voltage divider, so that the voltage variation on the selection lines is only 6 V thus minimizing the noise picked up by the sense wire. Capacitors shunted across the current determining resistors improve the current waveform. Auxiliary switches are employed to ensure short capacitor discharge times.

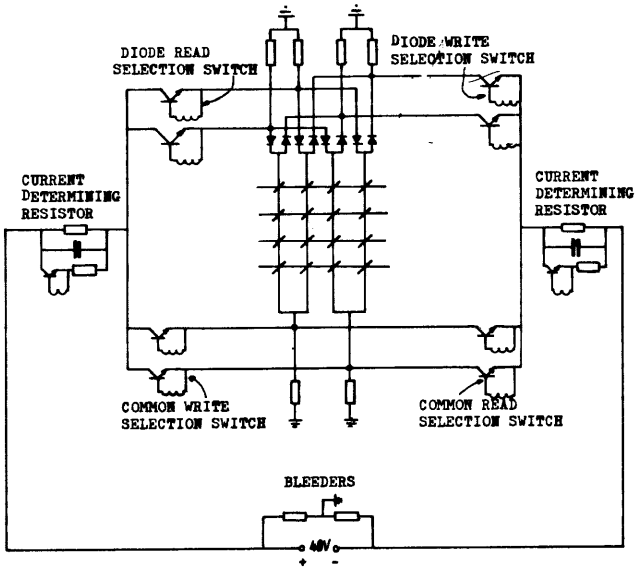


Figure 5 – Selection switches and diode matrix

The most suitable coupling devices between switches of this type and their drives are transformers since they are particularly useful as a base current supply source; moreover they permit the use of very simple drive circuits.

For selecting one line from a total of 256 lines, 64 selection switches are required. These form a self-contained unit mounted on a single card. Nine such units are required for the bit-line, and eight for the word-line selection.

Drive circuits and address decoding

Fig. 6 shows a combination of selection switches, drive switches and address decoders in simplified form. Two such combinations are employed in the memory, one selects the bit-lines, the other the word-lines.

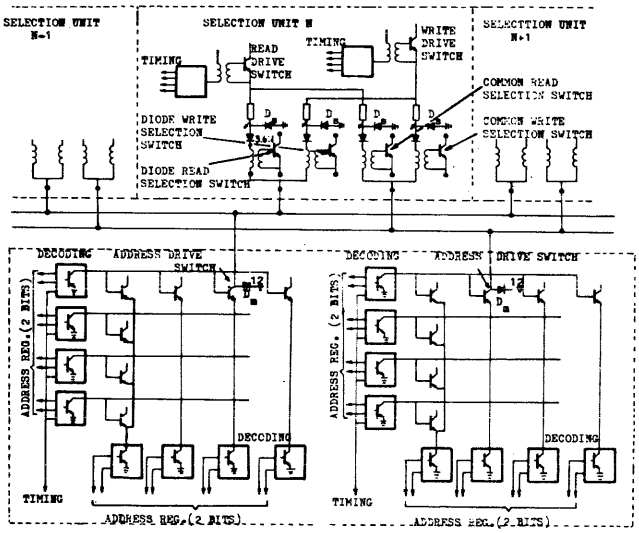


Figure 6 – Drive circuits of selection switches

In each 4 × 4 matrix of address drive switches, each switch is responsive to two sets of address decoders, thus two sets of read- and write-selection switches are pre-selected. This occurs in each line-selection unit connected to one of these driver matrices. The final decision whether or not the read- or write-selection switches are to be driven, is left to the read- and/or write-drive switch in each selection unit. Three functions can thus be assigned to the read and write-drive switches:

(1) selecting either the read- or write-selection switches in all selections units, governed by the command pulses from the timing unit.
(2) selecting one unit from eight word-line selection units on the basis of three address bit levels;
(3) digit switching in the bit-line selection units.

Current phasing as a means of core selection though not used here, could also be controlled by the read- and write-drive switches.

Saturation of a selection-switch drive transformer which may occur after several successive switching operations, is prevented by diodes D_s and D_m which ensure rapid demagnetization of the particular transformer after the termination of a read- or write-drive pulse.

With both the address-drive and read- or write-drive switches broken, the primary winding of the transformer previously selected is isolated. Point A tends to become negative and is clamped to earth by D whilst point B swings positively and is clamped to +12 V by D, resulting in a constant voltage of some 15 V being applied to the primary. This accelerates the decay of the magnetizing current through the winding.

Sense wires and amplifiers

Unlike other 2½ D mass memories described in the literature, the memory under discussion employs a separate wire for sensing the interrogated bits. The path followed by the two pairs of sense wires is shown in Fig. 7a. For the sake of simplicity a 2× (128 × 512) sub-matrix is reduced in the figure to 2 × (8 × 32). By arranging the cores so that their axes are mutually parallel, good balancing and easy threading of the sense winding are obtained. The manner in which several sense wires are combined to form one sense winding is shown in Figs. 7b and 7c. Because of the method used to terminate the sense wires in this memory, perfect balancing of the sense winding is required. The type of termination employed eliminates distortion of the output signal owing to phase differences at the inputs of the differential preamplifiers.

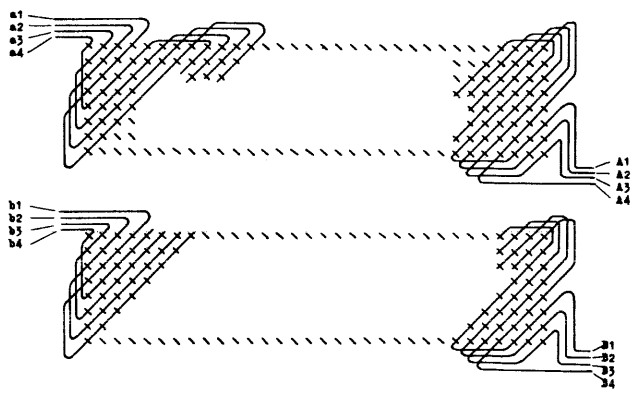


Figure 7a – Sense wire through core array

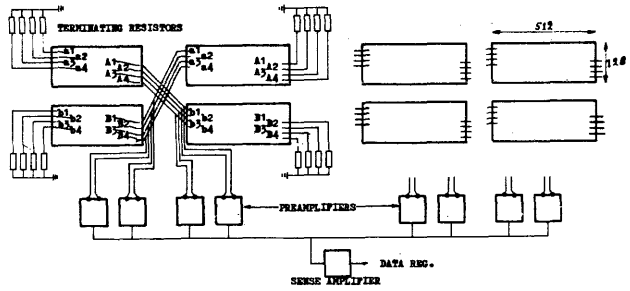


Figure 7b – Combination of sense wires and preamplifiers

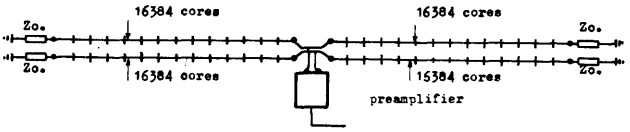


Figure 7c – Connection of sense wires to one preamplifier

One pre-amplifier is capable of handling as many as 65,000 bits, thus only eight pre-amplifiers are required per word-bit. Despite the large number of cores per sensing circuit, only 128 core pairs on each selected bit-line and 32 core pairs on every selected word-line contribute to the noise level. In order to limit the level of noise, only those preamplifiers coupled to the selected cores at a given instant are operational. The signal-to-noise ratio is improved by initiating the bit-line currents during a given operation some 250 ns before the word-line current.

In order to compensate for delay and attenuation of the signal on the sense wire, the threshold level and strobing time of the sense amplifier are required to vary as a function of the address.

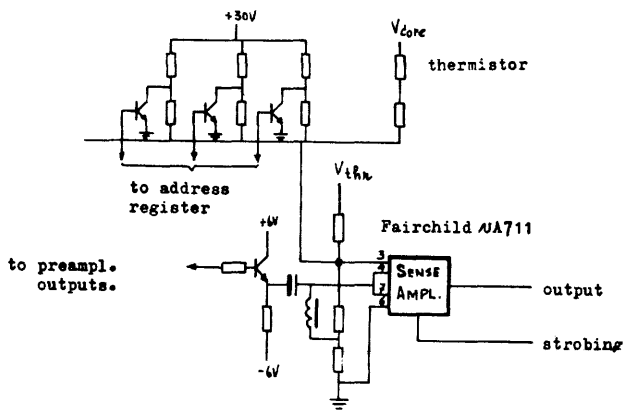


Figure 8 – Circuit to vary sense amplifier threshold as a function to the address

Variation of the threshold level is accomplished by a circuit (Fig. 8) which is actually a digital-to-analogue converter. The variation constitutes a correction of the main adjustment which is determined by the setting of the manually variable voltage V_{thres}. With a threshold circuit of this type it is relatively simple to feed back a fraction of the core driving voltage to the threshold circuit, so that a considerable improvement in operating tolerances is obtained. This could be done by a resistor network which includes a thermistor to render the threshold level insensitive to variations of the core driving voltage, arising from temperature changes in the core stack.

The sense amplifier strobe time can be varied as a function of the address by means of the circuit shown in Fig. 9a. In the circuit, V_A is the output voltage of a digital-to-analogue converter, whilst a sawtooth voltage V_B is applied to the other input of the long-tailed pair.

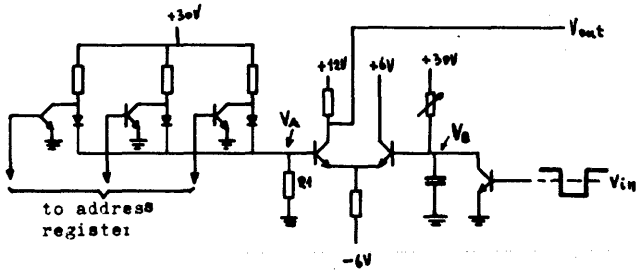


Fig. 9a.

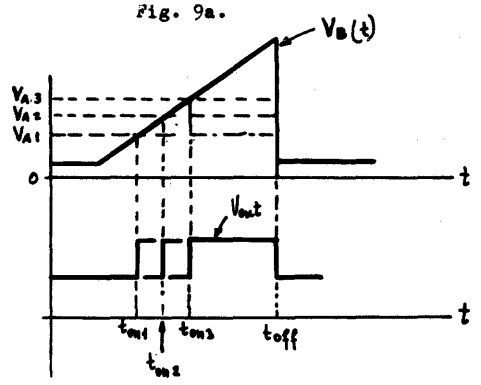


Figure 9a,b – Circuit to vary sense amplifier strobing as a function of the address

The resulting output voltage is a positive-going pulse (Fig. 9b) of which t_{on} varies with the address of the selected word, and t_{off} is constant.

Reference to the oscillograms of Fig. 10 shows the output voltages from the various addresses, together with the bit and word currents and strobing and data register outputs. Experience gained during manufacture of the sub-matrices has completely vindicated the choice of an extra wire for sensing. The cost of the extra wire has proved to be less than the cost of the transformers that would otherwise be required (3). The external core-stack wiring is much simpler and much less critical with regard to noise pick-up.

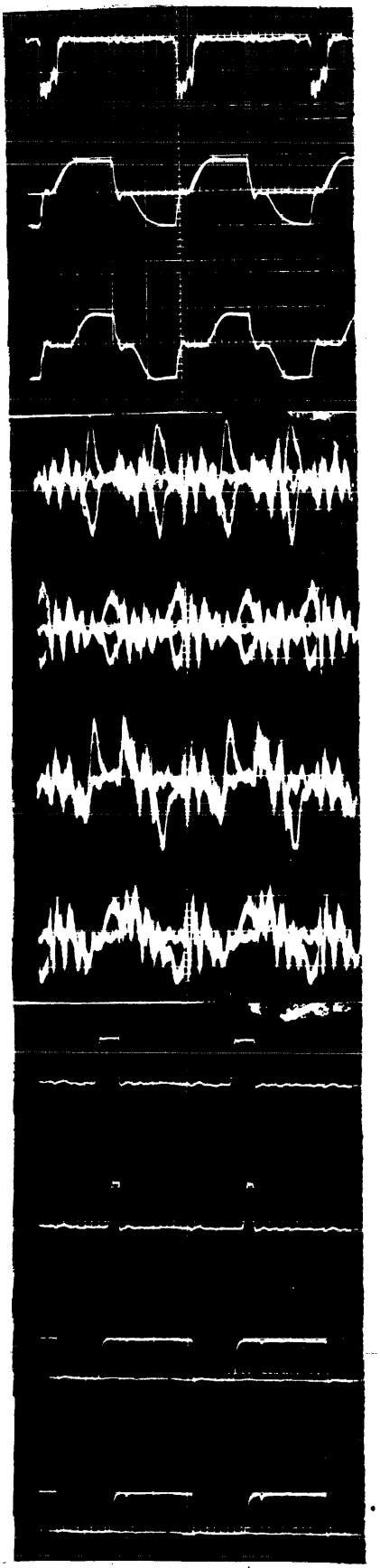


Figure 10—Read command

Longer words

Two methods can be adopted in the memory to increase the number of bits per word. The length of the words is limited by the number of sense wire pairs (144).

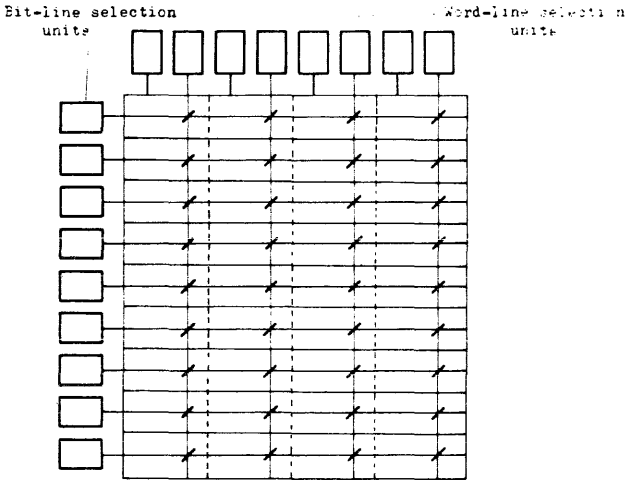


Figure 11—Combination of bit-line and word-line selection units to interrogate 36 bits at a time in an essentially 9-bit organization

One technique is illustrated in Fig. 11. Here, use is made of the fact that eight separate word-line selection units are employed. When nine bit-lines are driven concurrently with four word-lines rather than one, 36 bits can be interrogated at a time. This is possible if the bits are situated on 36 different sense wires and the output voltages of the sense wires are staticized by 36 different registers. A word of 36 bits has to be stored in four shifts of nine bits, each shift occurring in 1.2 μs; this operation is done independently by the memory. The memory is thus rearranged to produce 130,000 words each of 36 bits with an access time of 1.2 μs and a cycle time of 1.3 μs + 4.8 μs = 6.1 μs. The bit transfer rate increases from 3.6×10^6 bits/s to 5.9×10^6 bits/s. When for instance two word-lines are driven concurrently with nine bit-lines, 18 bits can be interrogated at once the storage time being 2.4 μs. It is thus possible with this technique to multiply the number of bits per word without a proportional increase in heat dissipation occurring. A drawback is the multiplication of the cycle time, however the short access time is maintained.

It is also possible to increase the number of bits per word by increasing the number of bit-line selection units. Referring to Fig. 1, it can be seen that 18 bit selection units each of which handle 128 bit-lines, can be connected to 2304 bit-lines. The nine diode matrices of 16×16 diodes have to be altered to 18 diode matrices of 8×16 diodes. This is however, a relatively simple matter and involves merely changing the wir-

ing in the main stack frame, the wiring or construction of the sub-stacks being unaffected. The conversion is completed by effecting the necessary modifications to the electronic circuitry. A memory of 260,000 words each of 18 bits, having an access time of 1.3 μs and a cycle time of 2.5 μs is thus obtained. Fig. 12 shows how 36 bit-line selection units each handling 64 bit-lines, can be connected to 2304 bit-lines. In

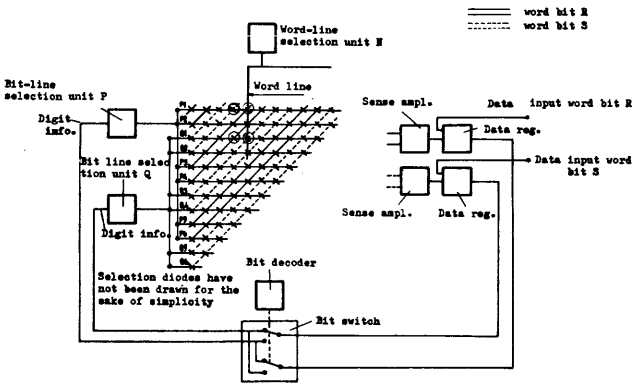


Figure 12 – Combination of bit-line selection units and sense wires in a 36-bit organization

the figure part of a sense wire array of 128 × 512 addresses is shown. P and Q are selection units for the 64 bit-lines and N is a word-line selection unit. A combination of two corresponding bit-lines for example p_1 and q_1 with any word-line, selects a pair of cores located on two different sense wires. The bit-line selection units P and Q are not however, correlated solely to a particular word-bit *R* or *S*. By suitable addressing, P can select a core of either word-bit S or R, Q then selects a core of either word-bit R or S. The address has thus to be decoded, the output of the decoder being fed to the digit switch whose function is to direct the digit information into the correct channels. One item in the stack which has to be changed is the wiring between the bit selection diodes and the bit-lines, however, the standard sub-stacks can be used without any wiring or constructional changes. A memory is thus realized which has a capacity of 130,000 36-bit words, an access time of 1.2 μs and a cycle time of 2.5 μs. The two methods described in the foregoing can be combined to make a mass memory of 32,000 words each of 144 bits, with an access time of 1.2 μs and a cycle time of 6.1 μs. In such a system the bit transfer rate is 23.6×10^6 bits/s.

Power supplies and protections

The power supplies are designed to operate from a three-phase a.c. mains supply. All the d.c. outputs are protected against excess voltage and currents and falls in voltage, whilst thyristors are included to reduce energy losses. The memory can be switched on and off without destroying information stored in the cores. When a memory is switched off owing to failure of one of the d.c. outputs, an aid to locating the fault is provided by pilot lamps which indicate the particular d.c. output that has failed. When several memories are combined to form a single bank and one breaks down, the inputs of the other memories are immobilized, however their power supplies remain switched on.

Tolerances

Operating tolerances of a memory system can be represented by means of an n-dimensional Schmoo diagram, where n is almost limitless. Only a few of these variables are of practical interest. For this memory a two-dimensional Schmoo diagram has been made, having the sense amplifier threshold plotted along one axis and the voltages corresponding to the core drive currents along the other (Fig. 13). In plotting the graph all other voltages were set to their nominal values (–6 V, +12 V and 30V). During the measurements the ambient temperature of the memory was kept at the upper limit of 45°C, at which level the noise is at a maximum.

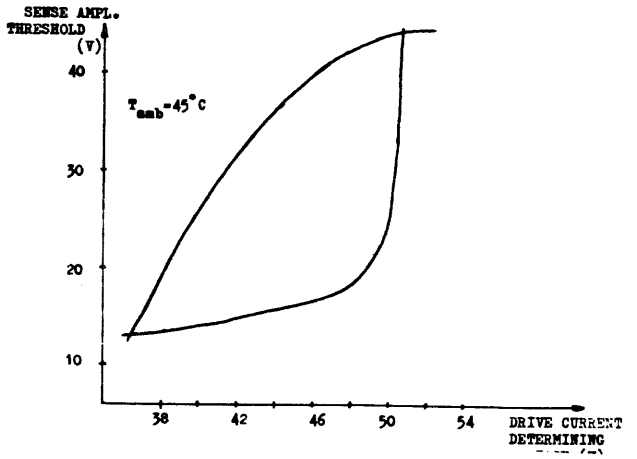


Figure 13 – Schmoo diagram. No feed-back from core drive voltage to sense amplifier threshold

Mechanical design

All electronic components are mounted on double-faced, epoxy-glass printed circuit cards. The cards carrying the selection and drive switches, read amplifiers and timing circuits, measure 32 × 25 cm. (12½ × 10 in). The cards slide into a compartment which can be mounted in a standard (19 in) rack.

Cable terminating resistors and cable amplifiers are mounted on cards of 32 × 13 cm (12½ × 5 in) which are housed in a second compartment also containing the cable connectors.

To take full advantage of the low heat dissipation within the stack container-4.5 W in a volume of 84 litres (22 U.S. galls)-the container is situated at the base of the cabinet.

The compartments housing the memory electronics are located immediately above the stack container.

The power supplies are accommodated in two such compartments which occupy the upper part of the cabinet together with another unit housing the transformers and certain control circuits. Disposition of the various sub-assemblies in this manner renders forced cooling unnecessary. The space occupied by a complete memory (less cabinet) measures 147 cm (59 in) height by 45 cm (18 in) width by 50 cm (20 in) depth.

CONCLUSIONS

Well-designed and properly terminated sense wires enable the manufacture of a fast and relatively inexpensive mass memory to be realized.

Although the application of sense wires usually implies a limitation to one particular word length for a given storage capacity, it transpires that by purely electronic means, a large range of word lengths can be covered using standard sub-stacks. This is a very important factor in mass memory design since the core stack is invariably the most difficult and most expensive part to develop and manufacture. Provided therefore that sufficient attention is given to the interface circuitry, small economic mass memories can be manufactured on a mass production basis to form part of larger memory systems.

ACKNOWLEDGMENT

Thanks are expressed to Mr. P. J. Baker of our Technical Publication Dept. who edited the manuscript.

REFERENCES

1 R J PETSCHAUER G A ANDERSEN
W J NEUMANN
A large capacity low cost core memory
Presented at the IFIP Congress in New York N Y May 1965

2 T J GILLIGAN P B PERSONS
High Speed Ferrite 2½ D Memory
Proceedings - Fall Joint Computer Conference 1965.

3 A M PATEL J W SUMILAS
A 2½ F ferrite memory sense amplifier
IEEE JOURNAL of Solid State Circuits, Vol. SC-1 No. 1 September 1966

4 J REESE BROWN
First- and second-order ferrite memory core characteristics and their relationship to system performance
IEEE Transactions on Electronic Computers vol EG-15 no 4

5 *2361 Core storage original equipment manufacturers information*
IBM Publication SRL - 28 - 822 - 6869 - 1

A magnetic associative memory*

by TSE-YUN FENG**
Syracuse University
Syracuse, New York

INTRODUCTION

Hundreds of technical papers and reports on associative systems have been published since 1956 when the first electronic associative memory was reported. So far the use of such a system is still very limited because of its cost. But the tremendous technological advances made in the batch fabrication of thin films and integrated circuits for the last few years would evidently reduce the absolute cost of an associative system and enhance its usefulness.

Much work has been done in the area of magnetic associative memories.[1-19] However, most of the memory schemes proposed have a low "per-bit mismatch-to-match signal ratio"*** which is one of the principal factors in determining the ultimate size (or speed) of the memory. In these systems the delta noise due to the reversible switching of the magnetic material is compensatable, thus with appropriate compensation techniques, it is possible to make the per-bit mismatch-to-match signal ratio independent of the signal-to-noise ratio of the magnetic device. As a result, the per-bit mismatch-to-match signal ratio is greatly increased. It can also be shown that an associative memory may have the ability of detecting the neighboring codes of a given code if a similar signal compensation technique is employed.

In the following scheme the implementation can be achieved by many kinds of devices. However, a 4×4 memory model using multiaperture cores was constructed and tested. Experiments performed on the memory model confirmed the effectiveness of the compensation techniques.

*The work reported here was supported by the United States Air Force Contract AF 30(602)-3546.

**Formerly with the University of Michigan, Ann Arbor, Michigan.

***The "per-bit mismatch signal" is the signal produced at a word sense wire when only one bit of the word is interrogated and the information stored in the bit is different from that of interrogated bit. Similarly, the "per-bit match signal" may be defined as the word signal produced when the information stored in the bit matches that of the interrogation bit. In most cases it is the signal-to-noise ratio of the associative cell.

Memory scheme

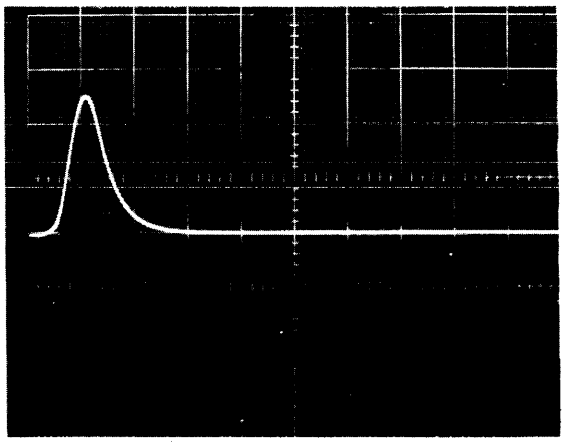
For a magnetic core if the maximum induced voltage due to the irreversible switching of flux is normalized to 1 and that due to the reversible switching (delta noise) is δ, the maximum possible signal to noise ratio is

$$\frac{1+\Delta\delta}{\delta} > p, \tag{1}$$

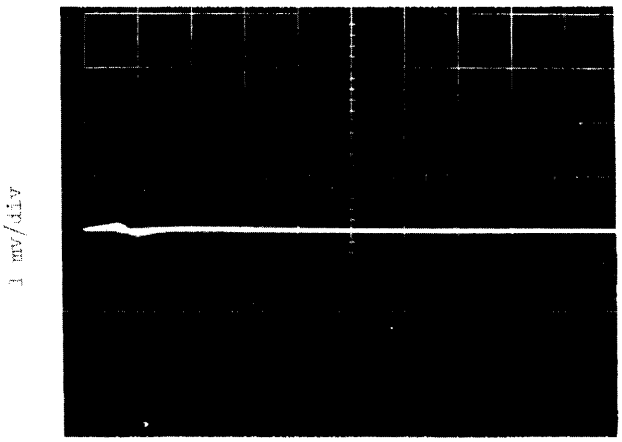
where p is the largest integer to satisfy this relation. If parallel-by-bit interrogation is assumed, p is also the maximum possible number of bits per word that an associative memory may have before the noise masks the information-bearing signal. Equation 1 indicates that in order to increase p, δ must be small. Since δ is an inherent property of the magnetic material and cannot be eliminated, it must be compensated externally. Suppose that there is a noise generator that would produce a noise of δ when interrogated. Its sense wire is connected in series with a memory bit but opposite in polarity. The output during interrogation would then be $(1+\delta)-\delta=1$ when the memory bit changes state, and is $\delta-\delta=0$ when the memory bit does not switch. The per-bit mismatch-to-match signal ratio would for this case be infinite. In reality, the total compensation of delta noise is very difficult (if not impossible). This is mainly due to the nonhomogeneity of the material and other factors involved in the fabrication of the memory devices. Suppose that the maximum noise that cannot be compensated by the noise generator is $\epsilon(\epsilon < \delta)$. Then the per-bit mismatch-to-match signal ratio would be in the worst case:

$$\frac{1-\epsilon}{\epsilon} > p \tag{2}$$

Experimental result shows that the per-bit mismatch-to-match signal ratio is greatly increased to $\frac{52}{0.1}=520$ by the noise compensation technique (Fig. 1) as compared to $\frac{52}{4}=13$ for the uncompensated case (Fig. 2).

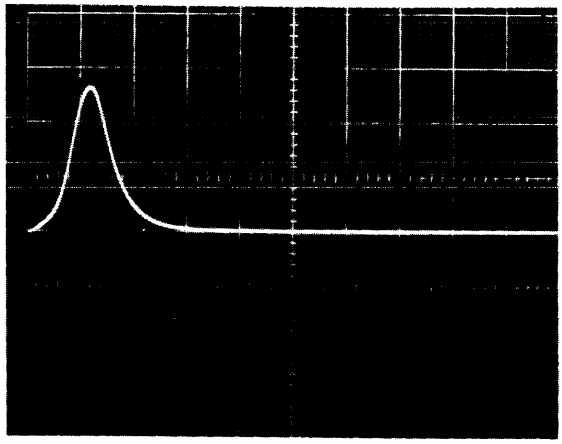


(a) Mismatch signal voltage

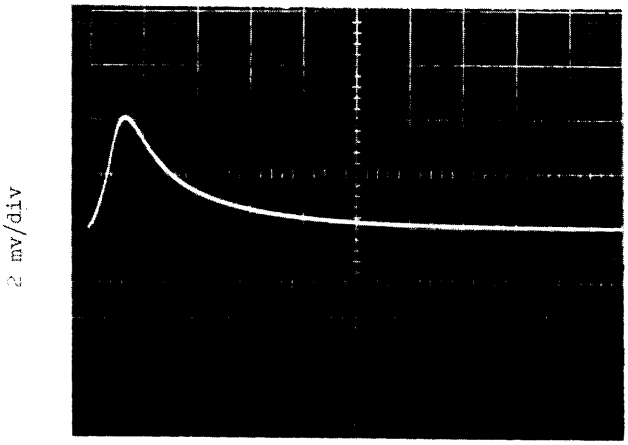


(b) Match signal voltage

Figure 1 – The per-bit mismatch and match signal voltages



(a) Signal Voltage



(b) Noise Voltage

Figure 2 – The signal and noise voltages of a multiaperture core

Figure 3 shows the circuit configuration of the associative memory with noise compensation. The two states of an associative cell are defined by Fig. 4. Thus, each cell may consist of a flip-flop and a few gates or two multiaperture cores (or their equivalent). The vertical lines of the 4 × 4 memory are the interrogation wires and the horizontal lines are the row sense wires. The memory interrogation pulses have three states as given by Table I. There is no 11 state since both interrogation wires are never energized simultaneously during the searches. The symbol Ø is used here for the masked bits. The relations between input, store, and output may be expressed by Table 2. The choice of O output for a matched condition simplifies the detection structure of the associative memory.

In Fig. 3 the memory has four 4-bit words with C_{11} C_{12} C_{13} $C_{14} = 1100$, C_{21} C_{22} C_{23} $C_{24} = 0110$, C_{31} C_{32} C_{33} $C_{34} = 1110$, C_{41} C_{42} C_{43} $C_{44} = 0100$, as defined by Fig. 4. The row noise generator uses the same types of cells as those in the memory but for each cell only the portion that would generate delta noise is interrogated and sensed. Of course, this is not always necessary. For example, when transfluxors are used in the scheme, each memory cell requires two transfluxors, but each noise cell requires only one transfluxor which is in the blocked state.

Equality search

As an illustration for equality search, suppose that a pattern of 1110 is stored in the association register (Fig. 3). Then, during the interrogation period the interrogation wires I_{11}, I_{21}, I_{31}, I_{40} will be energized. The normalized induced voltages at the row sense wires are $N_1 = 1 + 4\delta$, $N_2 = 1 + 4\delta$, $N_3 = 4\delta$, $N_4 = 2 + 4\delta$. But these voltages are balanced by a voltage of (-4δ)

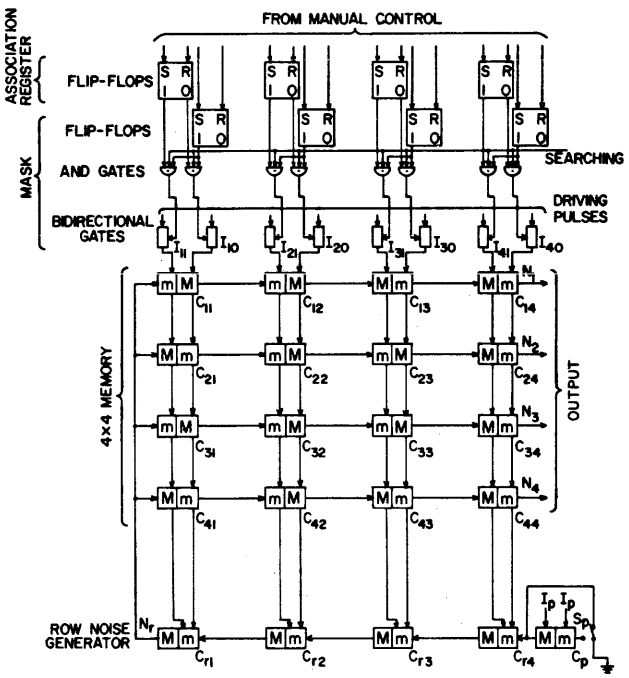


Figure 3 – Circuit and configuration of a noise-compensating and error checking associative memory

M: Signal output when interrogated (1 + δ)

m: Noise output when interrogated (δ)

Figure 4 – Two states of an associative cell

induced at the sense wire N_r. Therefore, the net voltages relative to ground appearing at the N wires are: $1 \pm 4\epsilon$, $1 \pm 4\epsilon$, $0 \pm 4\epsilon$, $2 \pm 4\epsilon$ and ϵ represents the maximum noise that cannot be compensated by the noise generator. Hence, for an exact match there will be no (or little) voltage at a sense wire and there will be varying degrees of voltage differences in all other cases. If these voltages are normalized by a threshold device, then we have $N_1 = 1$, $N_2 = 1$, $N_3 = 0$, $N_4 = 1$. 0 indicates equality match.

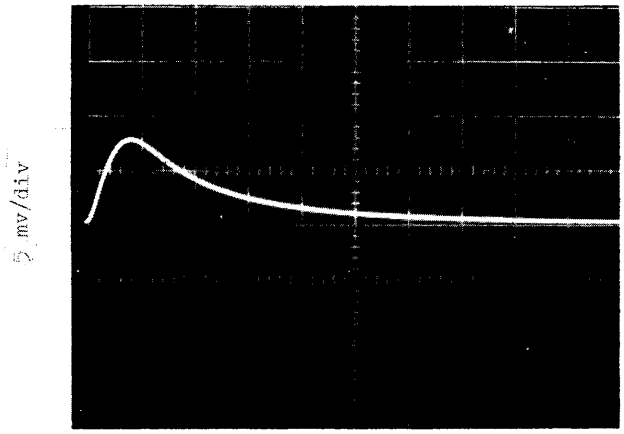
Experimental results on equality search are shown in Fig. 5. Figure 5(a) is the 4-bit match signal without noise compensation. After noise compensation the 4-bit match signal is reduced to that shown in Fig. 5(b).

TABLE I – Interrogation pulse states

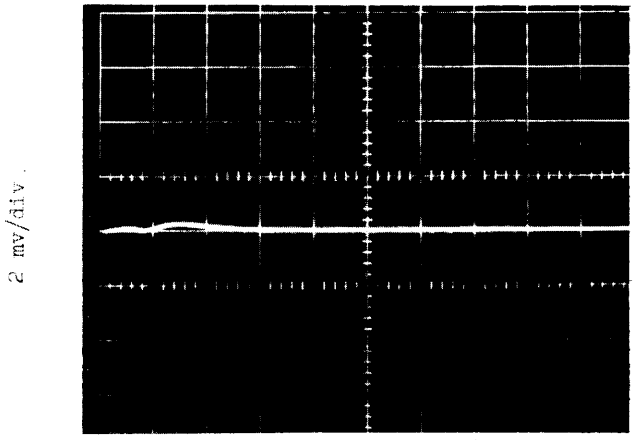
Input of i-th Bit	Interrogation Pulses	
	I_{i1}	I_{i0}
0	0	1
1	1	0
ø	0	0

TABLE II – Relations between memory input, store, and output

Input	Store	Output
0	0	0
0	1	1
1	0	1
1	1	0
ø	0	0
ø	1	0



(a) 4-bit match signal without noise compensation



(b) 4-bit match signal with noise compensation

Figure 5 – Match signals from a 4-bit word

Inequality search

Suppose that 1110 is still stored in the association register and we search for inequality responses. The function of the memory is the same as before. In order to detect inequality condition by the same circuit used for equality search AND gates are added. Figure 6 shows one possible arrangement. For equality searches the upper AND gate of each word is activated so that both the N and N′ wires have the same state. But during the inequality search period the negation line is activated so that the states of the N′ wires are inverted from that of the N wires. Thus, we have $N_1'=0, N_2'=0, N_3'=1, N_4'=0$ instead of 1101 for the previous case. Here we have multiple responses for inequality match.

Figure 7 shows the 1-bit mismatch signals from a 4-bit word.

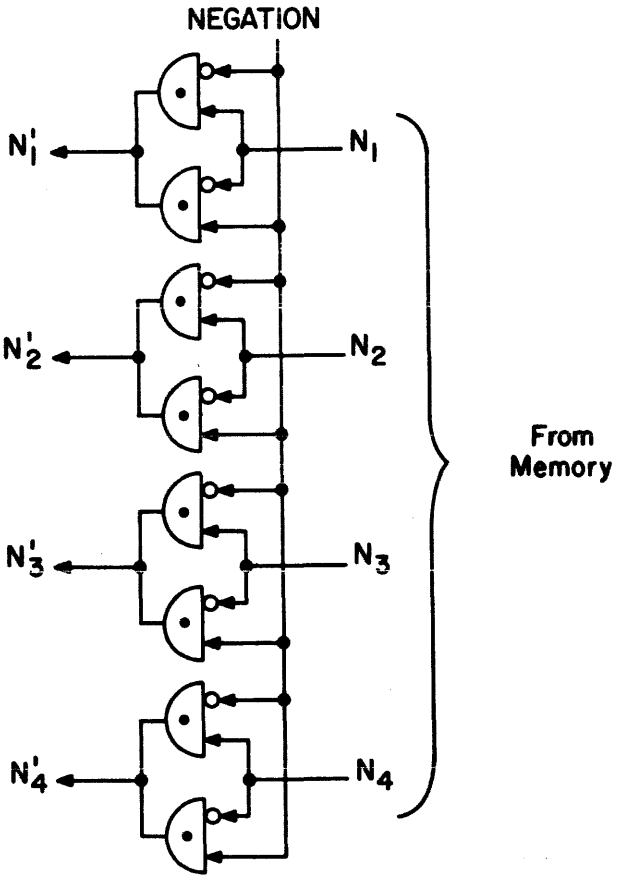
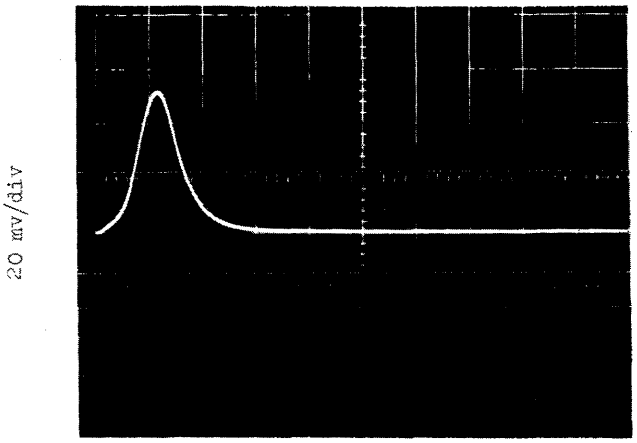


Figure 6 – And gates for simple searches

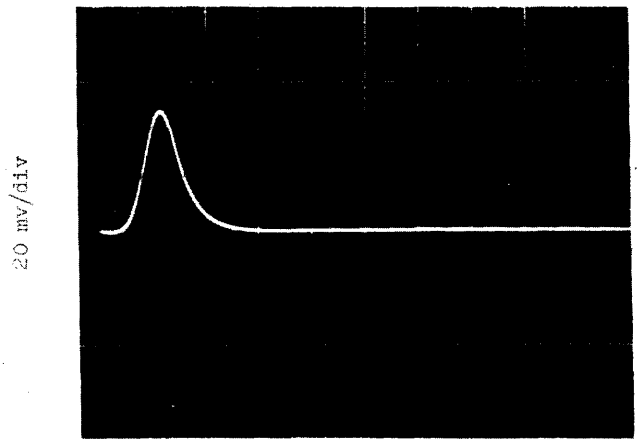
Similarity search

Suppose that the same pattern 1110 is stored in the association register as before but the third bit of the search word is masked off, i.e., the word to be searched is 1100. (Such a search when applied to one field is known as the similarity search.) Then during the interrogation period the interrogation wires I_{11}, I_{21}, I_{40} will be energized. The new output voltages from the sense wires will be now $0 \pm 3\epsilon$, $1 \pm 3\epsilon$, $0 \pm 3\epsilon$, $1 \pm 3\epsilon$, and the states of the N[wires are $N_1'=O$, $N_2'=1$, $N_3'=O$, $N_4'=1$. Again we have multiple responses to our search.

The output voltages from the word sense wires are similar to those shown in Fig. 5(b) and Fig. 7(b).



(a) 1-bit mismatch without noise compensation



(b) 1-bit mismatch with noise compensation

Figure 7 – 1-bit mismatch signals from a 4-bit word

Proximity search

The proximity search is defined here as the match such that the interrogated field of the memory word is different from that of the search word by a predetermined distance.* Such a match is very useful, particularly when redundancy techniques are applied

*The distance between two code points in an m-dimensional space is given by the number of coordinates (or bits) by which they differ.

to the associative memory to increase its reliability. Thus, the proximity match for distance 1 would detect single errors in the system. The proximity match for higher distances would detect higher-order errors provided that it is permitted by the per-bit mismatch-to-match signal ratio and the redundant codes used. The proximity search is also useful in some pattern recognition schemes.

Referring to Fig. 3 and the illustration on equality search we may summarize the results in Table III. Table III indicates that the memory words No. 1 and No. 2 differ from the search word by a distance of 1, thus their signal outputs are $1 \pm 4\epsilon$. Similarly, signal outputs from words No. 3 and No. 4 indicate a match (distance 0) and a distance of 2 respectively. Thus, if we introduce a cell which would generate a signal of (-1) in the row noise generator to balance the signal outputs, the same detection circuit would be able to detect the proximity match for distance 1. This is done by connecting a cell C_p to the row noise generator shown in the lower right corner of Fig. 3. The proximity–match switch S_p will be closed to the C_p sense wire which has an induced voltage of (-1)* during the interrogation period. The result of proximity match for distance 1 is shown in Table IV. The ζ term in the signal output comes from the switching signal compensation. The compensated signal voltage ζ was measured to be about 2 mv (Fig. 8) in our experiments.

TABLE III – Output of equality search

Search Word			Signal Output
Memory Word	#1	1 1 0 0	$1 \pm 4\epsilon$
	#2	0 1 1 0	$1 \pm 4\epsilon$
	#3	1 1 1 0	$0 \pm 4\epsilon$
	#4	0 1 0 0	$2 \pm 4\epsilon$

TABLE IV – Output of proximity search

Search Word		1 1 1 0	Signal Output
Memory Word	#1	1 1 0 0	$0 \pm 4\epsilon \pm \zeta$
	#2	0 1 1 0	$0 \pm 4\epsilon \pm \zeta$
	#3	1 1 1 0	$-1 \pm 4\epsilon \pm \zeta$
	#4	0 1 0 0	$1 \pm 4\epsilon \pm \zeta$

Figure 9 shows the experimental results on proximity match for distances 1 and 4 from a 4-bit word.

*Strictly speaking the induced voltage is $(-1 \pm \epsilon)$.

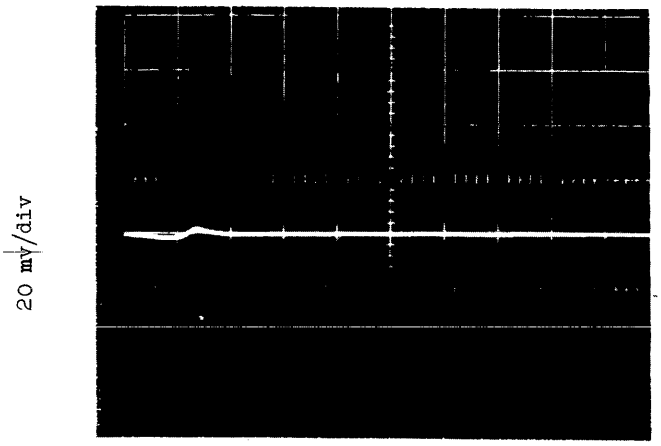
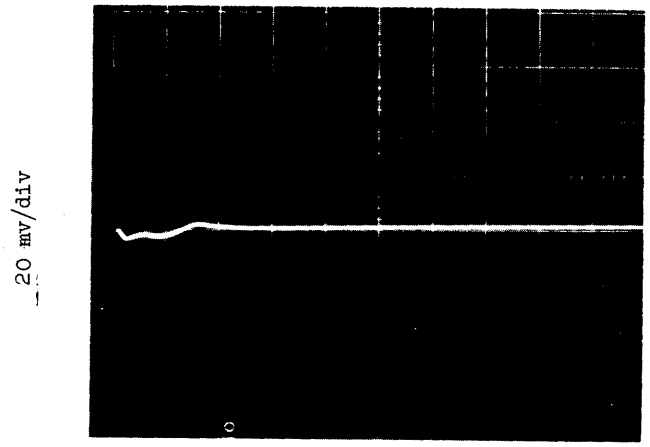
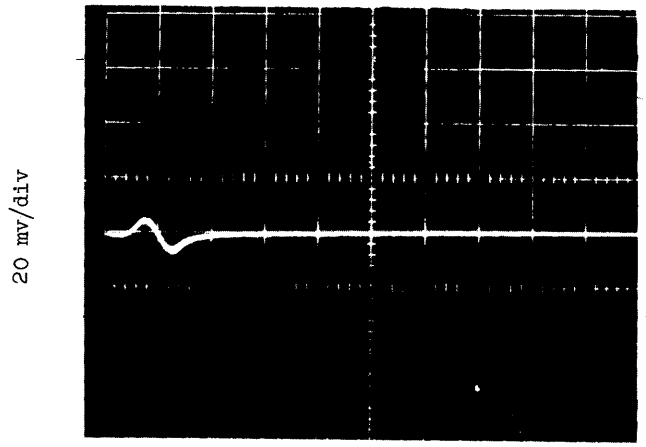


Fig. 8 Compensated signal voltage



(a) Proximity match for distance 1



(b) Proximity match for distance 4

Fig. 9 Proximity match signals for distances 1 and 4

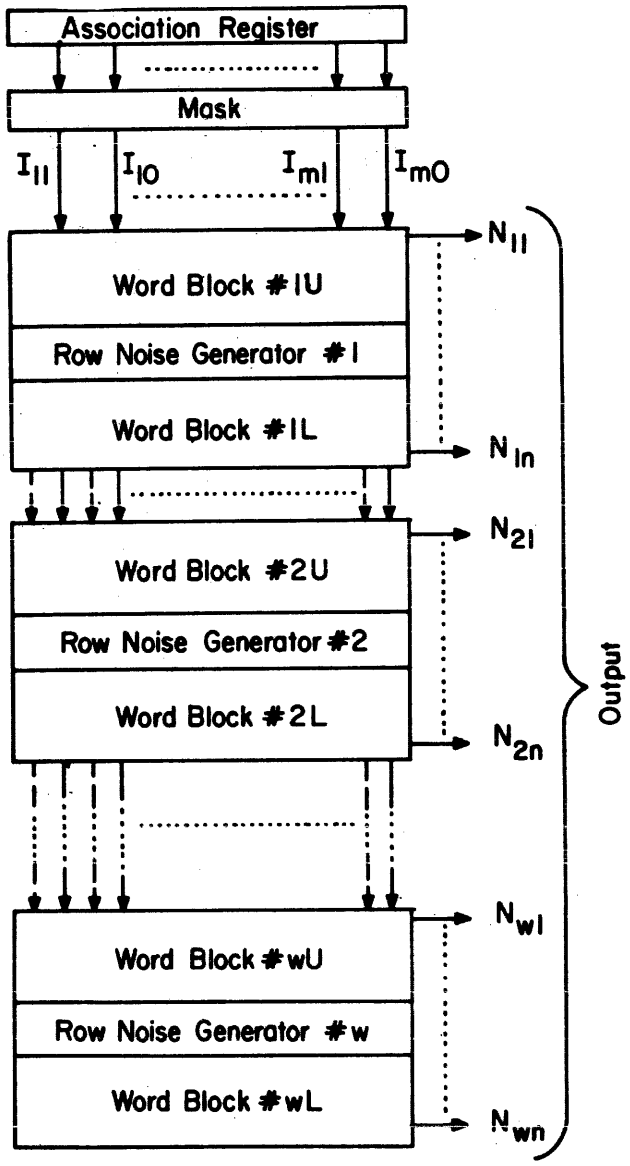


Figure 10—A noise-compensating and error-checking associative memory

The complete associative memory scheme

In the previous sections an associative memory with noise and signal compensations have been described. The experimental results as shown by the oscillograms were exceptionally good. However, due to the small number of words involved, the propagation delay and attenuation effects in this model were negligible. A complete associative memory scheme taking the propagation and attenuation effects into consideration is shown in Fig. 10. The configuration of Fig. 3 is only a portion of Fig. 10 which consists of an association register, a mask, the word block No. 1U and the row noise generator No. 1.

CONCLUSION

This paper demonstrates the effectiveness of the compensation scheme. Evidently the memory scheme can be used in large systems with high reliability.

REFERENCES

1 A APICELLA J FRANK
BILOC - A high speec NDRO one core per bit associative element
Intermag 1965

2 J R BROWN JR
A semipermanent magnetic associative memory
Proceedings of the Conference on Nonlinear Magnetics 1961

3 W F CHOW L M SPANDORFER
Plated wire bit steering for logic and storage
Proceedings of the 1967 Spring Joint Computer Conference

4 W F CHOW
Plated wire content-addressable memories with bit-steering technique
IEEE Trans on Electronic Computers October 1967

5 R H FULLER J C TU R M BIRD
A woven plated-wire associative memory
National Aerospace Electronics Conference 1965

6 J GOLDBERG M W GREEN
Large files for information retrieval based on simultaneous interrogation of all items
Proceedings of the Symposium on Large-Capacity Memory Techniques for Computing Systems 1961

7 R T HUNT D L SNIDER J SUPRISE H N BOYD
Study of elastic switching for associative memory systems
Contract AF 30(602)-3103 Report 1964

8 J R KISEDA H E PETERSEN W C SEELBACK M TEIG
A magnetic associative memory
IBM Journal April 1961

9 R R LUSSIER R P SCHNEIDER
All magnetic content addressed memory
Electronic Industries March 1963

10 J E MCATEER J A CAPOBIANCO R L KOPPEL
Associative memory system inmplementation and characteristics
Proceedings of the 1964 Fall Joint Computer Conference

11 W L MCDERMID H E PETERSEN
A magnetic associative memory system
IBM Journal January 1961

12 R C MINNICK
Magnetic comparators and code converters
Symposium on the Application of Switching Theory in Space Technology 1962

13 M NAIMAN
Content-addressed memory using magnetoresistive readout of magnetic thin films
Intermag 1965

14 J I RAFFEL T S CROWTHER
A proposal for an associative memory using magnetic films
IEEE Trans on Electronic Computers October 1964

15 A D ROBBI R RICCI
Transfluxor content-addressable memory
Intermag 1964

16 C A ROWLAND W O BERGE
A 300 nanosecond search memory
Proceedings of the 1963 Fall Joint Computer Conference

17 D O SMITH K J HARTE
Content-addressed memory using magneto- or electro-optical interrogation
IEEE Trans on Electronic Computers February 1966 and June 1967

18 G T TUTTLE
How to quiz a whole memory at once
Electronics November 15 1963

19 E L YOUNKER C H HECKLER JR D P MASHER J M JARBOROUGH
Design of an experimental instantaneous response file
Proceedings of the 1964 Fall Joint Computer Conference

Selection and implementation of a ternary switching algebra*

by ROBERT L. HERRMANN
Aerospace Division; Honeywell Incorporated
Minneapolis, Minnesota

INTRODUCTION

During the last few years there has been a growing interest in non-binary switching algebras. A number of such algebras have been proposed,[1-6] but so far they have had little practical application. Requirements for a practical switching algebra include economical circuits for the basic operations, the ability to implement an arbitrary function with a resonable number of basic circuits, and a suitable set of rules to allow the logic designer to form and manipulate expressions without undue mathematical gymnastics. These are rather vague criteria for a "practical" switching algebra, but they do serve as guidelines for attempts to establish suitable algebras.

Among non-binary switching algebras, ternary switching algebras have created the most active interest. The possibility of using positive, zero, and negative voltages or currents for representing three logic values appears appealing. The information transmitted by each wire in a system may be increased without employing complication level detectors and without sacrificing noise rejection. Complementary transistor configurations can be used to perform the logic functions.

Any single valued function in an algebra with a finite number of discrete *truth values*, including the basic operations of the algebra, may be expressed in tabular form by a *truth table*. Hence, all possible basic operations for a ternary switching algebra may be expressed in this way. Furthermore, such a table completely defines the function. To establish a practical ternary switching algebra, a suitable set of operations must be selected from the set of all possible operations. Methods employed by the author to select a suitable set are discussed in this paper.

In the following discussion, the integers 0, 1, and 2 denote the three truth values (constants), capital letters A, B, ... denote variables that assume the values 0, 1, or 2, and the symbols α, δ, f_n, $\oplus$, $\cdot$, and superscripted variables denote functions of one or more variables. The symbol = is used in its usual sense to denote equality of two expressions. An *n-ary operation* is a function of n variables that is one of the basic functions defining an algebra.

Manipulating functions within an algebra is governed by a set of theorems that apply to the particular albegra. As stated above, one of the desirable features of a practical switching algebra is the ability to manipulate expressions to reduce the amount of hardware required to implement a given function. Theorems that aid manipulation of expressions should therefore play an important role in the selection of a suitable ternary switching algebra.

Algebraic rules

Before embarking on a discussion of algebraic rules, it is necessary to define the constituents of such an algebra. They are:

1) The set of constants {0, 1, 2}
2) A set of symbols {A, B, ..., N} to represent variables.
3) A set of basic operations on one or more variables with appropriate symbols denoting mappings into the set of constants.
4) A set of rules governing the format of expressions in the algebra.

The approach taken in this paper to determine a suitable ternary switching algebra is to first establish a desirable set of manipulation rules, and then to search for a set of basic operations that best satisfies these rules.

The truth table is most straightforward and easily understood method of representing a function. Many algebraic rules may be introduced as constraints on the truth tables representing the basic operations.

The associative law for a single binary operation α

$$A\alpha(B\alpha C) = (A\alpha B)\alpha C = A\alpha B\alpha C \qquad (1)$$

*This paper is based on a thesis for the degree of Master of Science in Electrical Engineering submitted by the author to the faculty of Purdue University, West Lafayette, Indiana, in August 1964.

means that the order of evaluation of several consecutive applications of the operation is immaterial. As a direct result, the binary operation α can become a generalized n-ary operation $\alpha(x_1, x_2, \ldots, x_n)$. The commutative law

$$A\alpha B = B\alpha A \tag{2}$$

states that the order of the operands to which an operation α is applied is immaterial. A binary operation for which these two laws hold may be generalized to a *single level* n-ary operation. If these two laws are adopted as necessary criteria for the operations to be selected, only unary and binary operations need be considered. The binary operations AND and OR of conventional switching algebra are generalized in this way, as evidenced by the existence of multi-input AND and OR gates. These two laws also play an important role in manipulation of algebraic expressions.

The idempotent law

$$A\alpha A = A \tag{3}$$

can also be expected to aid simplification of algebraic expressions by sometimes reducing a multiplicity of a variable to a single occurrence. With these three laws as constraints, the number of possible operations is reduced to a reasonable number.

The effect of these laws on the truth table of a general binary operation α will first be considered. The idempotent law requires that the diagonal of the truth table matrix consist of the constants 0, 1, and 2, in that order. The commutative law requires that the matrix by symmetric. Table I illustrates these constraints on the truth table.

A \ α \ B	0	1	2
0	0	a	b
1	a	1	c
2	b	c	2

TABLE I. Generalized Commutative and Idempotent Binary Operation

The effect of the associative law on the truth table is not as obvious and can best be determined by trial-and-error testing of the associative law on the 27 possible truth tables of the form shown in Table I. This was done by the author with the aid of a computer program.[7] Nine of the remaining 27 operations were found to be associative. The truth tables for these nine are shown in Table II.

One or more of these nine binary operations, along with one or more suitable urary operations, must be selected to form a suitable set of basic operations.

Laws involving pairs of binary operations are considered next. This will determine which combinations of binary operations will work well together in terms of expression manipulation and simplification. The distributive and anti-distributive laws for a pair of binary operations α and δ

$$(A\alpha B)\ \delta\ (A\alpha C) = (A\delta A)\ \alpha\ (B\delta C) = A\ \alpha\ (B\delta C) \tag{4}$$
$$(A\delta B)\ \alpha\ (A\delta C) = (A\alpha A)\ \delta\ (B\alpha C) = A\ \delta\ (B\alpha C) \tag{5}$$
$$(A\alpha B)\ \delta\ (A\alpha C) = (A\alpha A)\ \delta\ (B\alpha C) = A\ \delta\ (B\alpha C) \tag{6}$$
$$(A\delta B)\ \alpha\ (A\delta C) = (A\delta A)\ \alpha\ (B\delta C) = A\ \alpha\ (B\delta C) \tag{7}$$

occur frequently in useful algebras, and may be useful in a switching algebra. They aid in manipulating and simplifying expressions by reducing the number of occurrences of a variable or subexpression. Idempotency is assumed in writing Equations (4) through (7). Equations (4) and (5) are the distributive laws; Equations (6) and (7) are the anti-distributive laws.

As with the associative law, Equations (4) through (7) may be tested for validity on each of the 36 distinct pairs of binary operations from the set of nine in Table II. Twelve of the 36 distinct pairs were found by the author[7] to satisfy both Equations (4) and (5). Six additional pairs were found to obey either Equations (4) or (5), but not both, and none were found to satisfy Equations (6) or (7). Referring to Table II, the pairs that satisfy both Equations (4) and (5) are:

$$\begin{array}{ll} P_1 = \{\text{IIb, IIc}\} & P_7 = \{\text{IIa, IId}\} \\ P_2 = \{\text{IId, IIh}\} & P_8 = \{\text{IIa, IIe}\} \\ P_3 = \{\text{IIe, IIg}\} & P_9 = \{\text{IIf, IIh}\} \\ P_4 = \{\text{IIb, IIh}\} & P_{10} = \{\text{IIb, IIf}\} \\ P_5 = \{\text{IIc, IIg}\} & P_{11} = \{\text{IIc, IIi}\} \\ P_6 = \{\text{IId, IIe}\} & P_{12} = \{\text{IIg, IIi}\} \end{array} \tag{8}$$

These 12 pairs are not truly distinct; among the 12 pairs, *isomorphisms* exist. Two pair are isomorphic if one pair can be converted into the other simply by interchanging the constants 0, 1, and 2 along with the appropriate permutation of rows and columns in the truth table. The validity of an algebraic rule does not depend upon the choice of symbols to denote the constants as long as the appropriate symbols are used in the rule, so all pairs in an isomorphic set must obey the same total set of rules.

The 12 pairs of binary operations found to obey both distributive laws may be partitioned into four isomorphic sets of pairs.[7] Referring to Equation (8), these isomorphic sets are:

	0	1	2
0	0	0	0
1	0	1	0
2	0	0	2

(a)

	0	1	2
0	0	0	0
1	0	1	1
2	0	1	2

(b)

	0	1	2
0	0	0	0
1	0	1	2
2	0	2	2

(c)

	0	1	2
0	0	0	2
1	0	1	2
2	2	2	2

(d)

	0	1	2
0	0	1	0
1	1	1	1
2	0	1	2

(e)

	0	1	2
0	0	1	1
1	1	1	1
2	1	1	2

(f)

	0	1	2
0	0	1	2
1	1	1	1
2	2	1	2

(g)

	0	1	2
0	0	1	2
1	1	1	2
2	2	2	2

(h)

	0	1	2
0	0	2	2
1	2	1	2
2	2	2	2

(i)

TABLE II. Set of Associative, Commutative, and Idempotent Binary Operations

$$G_1 = \{P_1, P_2, P_3\}$$
$$G_2 = \{P_4, P_5, P_6\}$$
$$G_3 = \{P_7, P_8, P_9\}$$
$$G_4 = \{P_{10}, P_{11}, P_{12}\} \quad (9)$$

One pair in an isomorphic set may be preferable from an implementation standpoint if the symbols 2, 1, and 0 are assumed a priori to be represented by positive, zero, and negative voltages respectively.

So far only binary operations have been considered. Equally important is the selection of suitable unary operations. Using the truth table approach, there are $3^3 = 27$ possible unary operations. The identity mapping and the mappings into a single constant are obviously of no value in a switching algebra. Eighteen of the possible unary operations map into two of the three truth values, and five map into the full set {0, 1, 2}.

Unary operations in the "truncating" class that map into two truth values have been proposed by several other authors.[3,5,6] However, when one attempts to form a "sum-of-products" or "product-of-sums" form of expression for a function in an algebra using this class of unary operations, three unary operations are required and the "terms" in the expression are two-valued. As an example, consider the three unary operations J_0, J_1, and J_2 defined in Table III.

These three unary operations may be used in defining an expansion theorem, to be discussed later, that results in a sum-of-products form of expression for all possible functions (see Equation 16). Each term in such an expression is a logical product of a constant and unary functions "J" of each of the independent variables. With the exception of the constant, the constituents of each term can assume only the values {0, 2}. Ternary information is regained by the ternary logical sum of some terms which can assume the values {0, 1} and some which can assume the values {0, 2}, as dictated by the constants in each elementary product. It should be noted in passing that DeMorgan type laws exist for J_0 and J_2, but not for J_1, using the Post binary operations. (P_4 in Equation (8)).

A	A^1	A^2	A^3	A^4	A^5
0	1	2	2	1	0
1	2	0	1	0	2
2	0	1	0	2	1

TABLE III. Truncating Unary Operations

By using nontruncating unary operations, a similar approach to expressing functions may be taken using single applications of only two unary operations and retaining three valued terms in a sum-of-products expression.

The five remaining unary operations that map into the full set (0, 1, 2) are shown in Table IV.

A	$J_0(A)$	$J_1(A)$	$J_2(A)$
0	2	0	0
1	0	2	0
2	0	0	2

TABLE IV. Nontruncating Unary Operations

The superscripts denote application of the unary operation to the variable A. Adopting for the moment the symbol ′ to denote a general unary operation, the operations A^1 and A^2 in Table IV obey the law

$$((A')')' = A \tag{10}$$

and the operations A^3, A^4, and A^5 obey the law

$$(A')' = A \tag{11}$$

Assuming that one or more unary operations are necessary, their position within an expression should be able to be manipulated in conjunction with the selected set of binary operations. The laws that allow manipulation of unary operations in conventional switching algebra are the DeMorgan Laws:

$$\begin{aligned} (\overline{A+B}) &= \overline{A}\cdot\overline{B} \\ (\overline{A\cdot B}) &= \overline{A}+\overline{B} \end{aligned} \tag{12}$$

A set of laws of this general type should be sought in the selection of unary operations for a ternary switching algebra. A set of possible "DeMorgan-type Laws" for the unary operation ′ is shown in Equation (13).

$$\begin{aligned} (A\alpha B)' &= A' \ \alpha \ B' && \text{(a)} \\ (A\alpha B)' &= A' \ \delta \ B' && \text{(b)} \\ (A\alpha B)' &= A'' \ \alpha \ B'' && \text{(c)} \\ (A\alpha B)' &= A'' \ \delta \ B'' && \text{(d)} \\ (A\alpha B)' &= A' \ \alpha \ B'' && \text{(e)} \\ (A\alpha B)' &= A' \ \delta \ B' && \text{(f)} \\ (A\alpha B)' &= A \ \alpha \ B && \text{(g)} \\ (A\alpha B)' &= A \ \delta \ B' && \text{(h)} \\ (A\alpha B)' &= A \ \alpha \ B'' && \text{(i)} \\ (A\alpha B)' &= A \ \delta \ B'' && \text{(j)} \end{aligned} \tag{13}$$

Each of the above rules, as they are stated and with α and δ interchanged, were tested on each of the 12 pairs of binary operations in Equation (8) for each unary operation in Table IV. It was found that isomorphic sets G_1 and G_2 in Equation (9) satisfied Equation (13b) and its inverse (with α and δ interchanged) for one of the unary operations in Table IV, G_3 and G_4 satisfied only Equation (13b) or its inverse, but not both, for only one of the unary operations in Table IV. This would indicate that one of the pairs of binary operations in G_1 or G_2 should be selected for the binary operations.

Functional completeness

As yet, the required number of unary and binary operations has not been determined. Since each basic operation requires a fundamental circuit for implementation of the algebra, the total number should be kept small. An absolute minimum number may be undesirable if unnecessary complexity of expressions result. A compromise may be made if one or two additional operations significantly reduce expression complexity.

Functional completeness, the ability of an algebra to express all possible functions of k variables for all k, dictates a minimal set of operations. A direct proof of functional completeness of a given set of operations is mathematically involved,[1] but it may be proven indirectly by showing that every operation of a known functionally complete algebra, such as ternary Post Algebra,[8] may be produced by a finite expression in the algebra in question. The basic operations ⊕, ·,and A′ of Post Algebra are shown in Table V.

A⊕B	0	1	2
0	0	1	2
1	1	1	2
2	2	2	2

A·B	0	1	2
0	0	0	0
1	0	1	1
2	0	1	2

A	A′
0	1
1	2
2	0

TABLE V. Ternary Post Algebra

They are identical to the operations shown in Tables II(b), II(h), and the operation A^1 in Table IV. The two binary operations are also identical to one pair found to be idempotent, associative, commutative, and distributive, and belong to isomorphic set G_2

Two binary operations seem to be a good choice based on algebras that have already been proposed.[1-6] From the results of the distributive test listed in Equation (8), it is easily shown that no three of the operations in Table II are all distributive with each other. An algebra with a single binary operation such as conventional NAND logic could be established, but expression complexity would be increased. Based on the results of the DeMorgan law tests, the choice is between the pairs in isomorphic groups G_1 and G_2. It is not obvious from algebraic considerations which group contains the best pair of operations. Pair P_1 in group G_1 obeys the following additional rules:

$$\begin{aligned} 0\,\alpha\,A &= 0 \\ 2\,\alpha\,A &= A \\ 1\,\delta\,A &= A \\ 0\,\delta\,A &= 0 \end{aligned} \tag{14}$$

Pair P_4 in group G_2 obeys a somewhat similar set of rules:

$$\begin{aligned} 0\,\alpha\,A &= 0 \\ 2\,\alpha\,A &= A \\ 0\,\delta\,A &= A \\ 2\,\delta\,A &= 2 \end{aligned} \tag{15}$$

The Post binary operations, pair P_4, are by far the easiest to implement, and thus are preferable. These may be considered the functions min(A, B) and max (A, B) in the usual arithmetic sense,[5] and can be implemented with simple diode gates similar to those used in conventional logic. Post Algebra and variations based on its binary operations has been suggested by several other authors[3-6] for ternary switching algebras, so the idea certainly is not new. The essence of the author's approach is that an algebra is chosen a priori, but rather by a suitable examination of all possible algebras.

Selection of a suitable set of unary operations remains. Of the five unary operations in Table IV, A^3 obeys the DeMorgan Laws Equation (13b) and its inverse with α and δ interchanged. A^3 and the two Post binary operations do not form a functionally complete algebra, so one additional operation must be included. Either A^4 or A^5 may be used.

Before further consideration is given unary operations, the subjects of how to form and manipulate expressions must be introduced.

Expansion theorems

A general method of generating expressions in a given switching algebra is an expansion theorem.[1,3] A logic designer uses the Boolean expansion theorem,[9] perhaps without realizing it, to generate expressions in conventional switching algebra. One comused ternary expansion theorem[3] is:

$$f(A, B, \ldots, N) = \sum_{x_A=0}^{2} \cdots \sum_{x_N=0}^{2} f(x_A, x_B, \ldots, x_N) \cdot J_{x_A}(A) \cdot J_{x_B}(B) \ldots J_{x_N}(N) \tag{16}$$

where

$\sum_{x=0}^{n} F_x$ = logical sum of all the $F_x = F_0 \oplus F_1 \oplus \ldots F_n$

and

$J_{x_N}(N)$ = unary operations that satisfy the expansion theorem

This theorem is a generalization of the Boolean expansion theorem that results in a "sum-of-products" expression in conventional switching algebra. The terms $f(x_A, x_B, \ldots, x_n)$ are the constants 0, 1, or 2 obtained from the truth table representing f(A, B, . . . ,N). Lee and Chen[3] have proposed an algebra consisting of the Post binary operations $\oplus$ and $\cdot$, and the unary operations J_0, J_1, and J_2 defined in Table III. The required "J" operations are, unfortunately, of the "truncating" type that map into the set {0, 2}.

Another possible expansion theorem is:

$$f(A, B, \ldots, N) = \prod_{x_A=0}^{2} \cdots \prod_{x_N=0}^{2} f(x_A, x_B, \ldots, x_N) K_{x_A}(A) + K_{x_B}(B) \oplus \ldots \oplus K_{x_N}(N) \tag{18}$$

where

$$\prod_{x=0}^{n} F_x = F_0 \cdot F_1 \ldots F_n = \text{logical product of all } F_x \tag{19}$$

This theorem corresponds to the "product-of-sums" form of conventional switching algebra. Expansion theorem (16) only will be considered in the following discussion; a discussion of both would be too lengthy.

Expansion theorem (16), when applied to a given truth table, results in an equation of the form

$$f(A, B) = (0)(J_i(A)\, J_j(B) \oplus J_k(A) J_l(B) \oplus \ldots) \oplus (1)(J_m(A) J_n(B) \oplus \ldots) \oplus (2)(J_r(A) J_s(B) \oplus \ldots) \tag{20}$$

From Equations (15), which apply to $\oplus$ and $\cdot$, (20) reduces to

$$f = (1)\,(J_m(A) J_n(B) \oplus \ldots) \oplus (J_r(A) J_s(B) \oplus \ldots) \tag{21}$$

Denoting the subexpressions in Equation (21) by "U" and "V,"

$$f = (1 \cdot U) \oplus (V)$$

If the "J" functions in Equation (21) are simple variables with or without a single application of a basic unary operation, a sum-of-products expression of the form commonly encountered in binary switching algebra results; otherwise, it does not. This form is preferable since currently known methods of simplification, such as the Karnaugh Map and Quine's method, are applicable only to this form of expression.

From Equation (22) some general requirements of the subexpressions U and V may be inferred. Both U and V are basic sums-of-products with no constants.

For f = 0, both U and V must equal "0". For f = 1, either (1·U) or V or both must equal "1" and neither equal "2". Either U = 1 or U = 2 will satisfy (1·U) = 1. For f = 2, V must equal "2" since (1·U) can never equal "2". These requirements can be met by sum-of-products forms of $\bar{U}$ and $\bar{V}$ only if $\bar{U}$ and $\bar{V}$ are allowed to assume only the values {0, 2}. This is true with the operations chosen because a basic product assumes the value 2 for one and only one condition but can be either "0" or "1" under other conditions over which there is no control. This problem may be avoided by truncating U and V at the individual variable level, such as is done when using the unary operations J_0, J_1, and J_2 defined in Table III, or by handling Equation (22) in such a manner that the problem does not occur.

An equivalent form of Equation (22) may be written

$$f = (1 \cdot J_2(X)) \oplus J_2(Y) = T(X, Y) \tag{23}$$

where

$$U = J_2(X) \text{ and } V = J_2(Y)$$

The subexpressions X and Y now may be three valued with truncation performed at the expression level in a prescribed manner independent of the function rather than at the individual variable level. Since Equation (23) is independent of the function, a new operation T(X, Y) may be defined by Equation (23).

The truth table for T(X, Y) is shown below.

T(X, Y)	Y = 0	Y = 1	Y = 2
X = 0	0	0	2
X = 1	0	0	2
X = 2	1	1	2

TABLE VI. Definition of T(X, Y)

T(X, Y) may be electronically implemented as one of the basic "building blocks" for constructing a working digital system using ternary logic. It will occur once for each unique combinational function of any number of variables. It is not necessary at all in the few cases where the base algebra can express the function directly. T(X, Y) might by symbolically represented by Figure 1.

If A^3, which obeys the DeMorgan Laws, is chosen for one of the unary operations, the addition of A^5 allows writing simple "sum-of-products" expressions

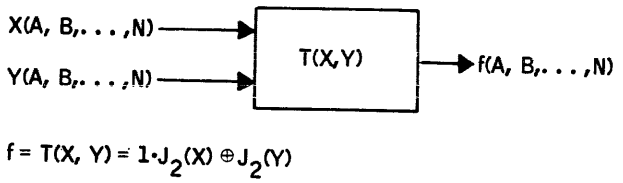


Figure 1 – Illustration of T(X,Y) function

that satisfy the requirements of X and Y. A^3 and A^4 would be the proper choice for expansion theorem (18). The operations ⊕, ·, A^3 and either A^4 or A^5 form a functionally complete algebra since

$$A^1 = (A^3)^5 \tag{24}$$

but cannot in itself produce simple sums-of-products for all possible functions. By appending T(X, Y) as defined above to the set of operations, the variable expressions may be written in sum-of-products form.

Simplification of expressions

Simplification of the subexpressions X and Y in Equation (23) may be accomplished by modifications of techniques already established for conventional switching algebra. The term *minterm* may be adopted form conventional switching algebra to denote a basic product in a ternary sum-of-products expression. The appropriate minterms are included in X to make X = 2 when f = 1, and in Y to make Y = 2 when f = 2. The minterms corresponding to f = 2 may be included in X as "don't care" conditions.

One method of forming and simplifying X and Y is a ternary version of the Karnaugh Map. This method directly applies the laws

$$A \oplus A^3 \oplus A^5 = 1 \tag{25a}$$

and

$$(A \cdot B) \oplus (A \cdot C) = A \cdot (B \oplus C) \tag{25b}$$

and indirectly depends on the associative, commutative, and idempotent properties of ⊕ and ·. As an example, consider the simple function f(A, B) defined by the truth table in Figure 2(a).

The subexpressions X and Y may be defined mathematically as:

$$X(A, B) = \sum_{x_1=0}^{2} \sum_{x_2=0}^{2} \{J_1 \cdot f(x_1, x_2)\} \cdot P_{x_1}(A) \cdot P_{x_2}(B) \tag{26a}$$

$$Y(A, B) = \sum_{x_1=0}^{2} \sum_{x_2=0}^{2} \{J_2 \cdot f(x_1, x_2)\} \cdot P_{x_1}(A) \cdot P_{x_2}(B) \tag{26b}$$

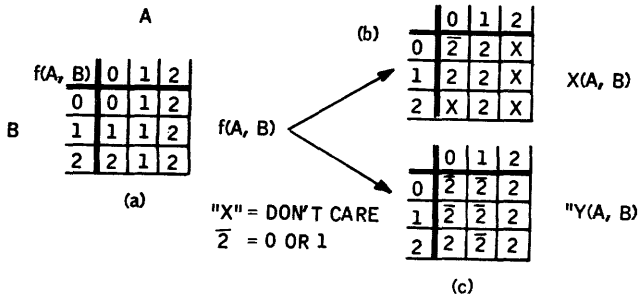


Figure 2 – Example of Karnaugh simplification

where J_1 and J_2 are as defined in Table III and

$$\begin{aligned} P_0(A) &= A \\ P_1(A) &= A^3 \\ P_2(A) &= A \end{aligned} \tag{27}$$

In words, Equations (26) simply state that all basic products or minterms are included in X that equal 2 when f = 1 and all minterms are included in Y that equal 2 when f = 2. In unsimplified form, X and Y for the example in Figure 2(a) are

$$\begin{aligned} Y(A,B) &= A^3B \oplus AB^3 \oplus AB^5 \oplus AB \\ X(A,B) &= A^3B^5 \oplus A^5B \oplus A^5B^3 \oplus A^5B^5 \end{aligned} \tag{28}$$

For simplification purposes, any or all terms in Y(A, B) may be included in X(A, B) as don't care conditions since if Y(A, B) = 2 the value of X(A, B) is immaterial (see Table VI).

In this example, if the term AB^5 in Y(A, B) is included in X(A, B), X and Y may be simplified to

$$\begin{aligned} X(A, B) &= A^5 \oplus B^5 \\ Y(A, B) &= A \oplus A^3B \\ f(A,B) &= T(X,Y) \end{aligned} \tag{29}$$

A Karnaugh map method may be applied to obtain the same result as shown in Figure 2(b) and 2(c). These reflect X(A, B) = 2 for f(A, B) = 1 and Y(A, B) = 2 for X(A, B) = 2. All "2" entries in the truth table for Y(A, B) are entered in the table for X(A, B) as "don't care" conditions. Simplification is done by groupings of 3^n terms.

The conditions for grouping terms differs slightly from conventional Karnaugh Maps. Groupings of 3^n terms instead of 2^n terms are used, and the conditions for "adjacency" are slightly more complex. For two variables, the groupings are obvious as shown in Figures 2(b) and 2(c).

For three or more variables, two minterms are "adjacent" if they differ in only one element of the basic product just as in the conventional case; these are not necessarily physically adjacent in the table as in the conventional case with more than four variables. Furthermore, *all three* terms that differ by one element must be present to effect a grouping, as dictated by Equation (25a).

Other simplification methods can be developed, based on methods used in conventional switching algebra, but are beyond the scope of this presentation. The Karnaugh Map method presented is one way to create suitable expressions to direct construction of a working digital system using ternary logic "building blocks."

Implementation

Before a useful digital system can be created, sequential techniques and suitable storage elements must be developed. The author cannot hope to more than merely scratch the surface of these areas. The discussion will be limited to consideration of a suitable storage element.

The obvious choice for a storage element is a ternary version of the flip-flop, if such a thing exists. First, the characteristics of such an element must be defined. It obviously must have three stable states and be capable of switching from one to another upon application of external stimuli. To be compatible with the algebra discussed above, it should have at lease one tri-level output Q, and preferably three outputs Q, Q^3, and Q^5. The required input stimuli must be obtained from the tri-level voltage signals provided by the other basic circuits.

The law

$$((A')')' = A \tag{30}$$

for the first unary operation in Table IV indicates that a tri-stable configuration can indeed be produced by cascading three circuits that implement the operation A^1, and connecting the last back to the first. This is analogous to cross-coupling two inverters in conventional logic to form a basic flip-flop. A basic ternary flip-flop is shown symbolically in Figure 3.

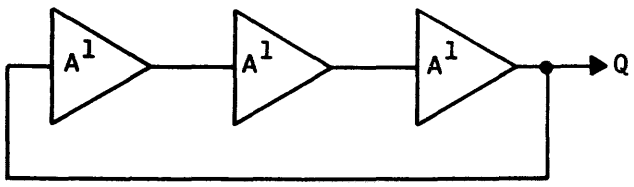


Figure 3 – Representation of a ternary flip-flop

Still needed is a method of controlling the state of the basic ternary flip-flop by external stimuli. There can be as many, if not more, variations as there are currently with binary flip-flops. Ternary diode gates inserted between each of the A^1 circuits in Figure 3 is

a method of control. Consolidation of the basic idea shown in Figure 3 into a single flip-flop package with appropriate inputs and outputs is left to the circuit designer.

An interesting variety of "clocked" flip-flops could result from the tri-level nature of the input signals. Positive and negative pulses about the center voltage level could perhaps be used to stimulate two entirely different actions within a ternary flip-flop, such as to load it and to preset it to a given state.

A set of basic circuits has been devised by the author to implement the operations A^1, A^3, A^4, A^5, and T(X, Y) using complementary-symmetry transistor techniques.

CONCLUSION

In the preceding sections, a suitable ternary switching algebra was determined by applying a reasonable set of constraints to the class of all possible ternary algebras. Based on the ability to manipulate, simplify, and implement algebraic expressions, the Post binary operations $\oplus$ and $\cdot$, the unary operations A^3 and A^4 or A^5, and the auxiliary binary operation T(X, Y) seem to be the best choice for a set of basic operations. The similarity between the theorems of this algebra and those of conventional switching algebra allow variations of conventional simplification techniques to be employed.

Sequential techniques, a necessary addition to the theory to be able to develop useful digital systems, were considered only briefly. Variations of conventional sequential techniques could perhaps be used to form the basis of a theory for ternary sequential systems. Primary emphasis was placed on the groundwork for development of a suitable storage element, a ternary version of the flip-flop. Much work remains to be done concerning sequential topics, including a comprehensive sequential theory, further development and implementation of ternary flip-flops, and suitable ternary memory systems.

Although the basic "inverter" circuits discussed are more complex than their binary predecessor, circuit complexity is becoming less and less a primary consideration as integrated circuit technology expands. Most current digital integrated circuits are limited not by circuit complexity, but by the number of terminals available on the physical package. A given number of signal terminals could convey more information in and out of a package if ternary circuits were used. This could possibly effect a savings in interconnection, especially where parallel transmission of large data words is used.

ACKNOWLEDGMENTS

The author is indebted to Professors R. R. Korfhage and S. Freeman of Purdue University for their guidance and suggestions while the author was performing the research leading to this paper.

REFERENCES

1 O LOWENSCHUSS
Nonbinary switching theory
1958 IRE National Convention Record part 4 pp 305-317

2 R D BERLIN
Synthesis of N-valued switching circuits
IRE Transactions on Electronic Computers vol EC-7 pp 52-56 March 1958

3 C Y LEE W H CHEN
Several-valued combinational switching circuits
Transactions of the AIEE (Communication and Electronics) vol 75 pt I pp 278-283 July 1956

4 E MUEHLDORF
Ternaere schaltalgebra
Archiv der Elektrischen Uebertragung vol XII pp 138-148 March 1958

5 M YOELI G ROSENFELD
Logical design of ternary switching circuits
IEEE Transactions on Electronic Computers vol EC-14 pp 19-29 February 1965

6 A MUKHOPADHYAY
Symmetric ternary switching functions
IEEE Transactions on Electronic Computers vol EC-15 pp 731-739 October 1966

7 R L HERRMANN
Development of a three-valued switching algebra
Master's thesis Purdue University Department of Electrical Engineering August 1964

8 E L POST
Introduction to a general theory of elementary propositions
American Journal of Mathematics vol 43 pp 163-185 1921

9 P C ROSENBLOOM
The elements of mathematical logic
New York Dover 1950

Application of Karnaugh maps to Maitra cascades

by GIUSEPPE FANTAUZZI
Fondazione U. Bordoni
Rome, Italy
and
Montana State University
Bozeman, Montana

INTRODUCTION

Statement of the problem

A Maitra cascade, as shown in Fig. 8, is a one dimensional cellular array whose cells have only one output and two inputs. At the output of the last cell the function is performed whose independent variables are introduced at the free inputs of the cascade cells. There are two different kinds of Maitra cascades according to whether a different binary variable is introduced or not at each independent input.[1] In the first case the cascade is an irredundant one, in the second it is said to be redundant. The most important results about the synthesis of Maitra cascade are given in Refs. 1-5. In Ref. 6 it is proved that a sufficient condition for a cellular cascade to reach its optimal synthesis possibility is that every cell can perform the set of five functions shown in Fig. 1a, 1b, 1c.

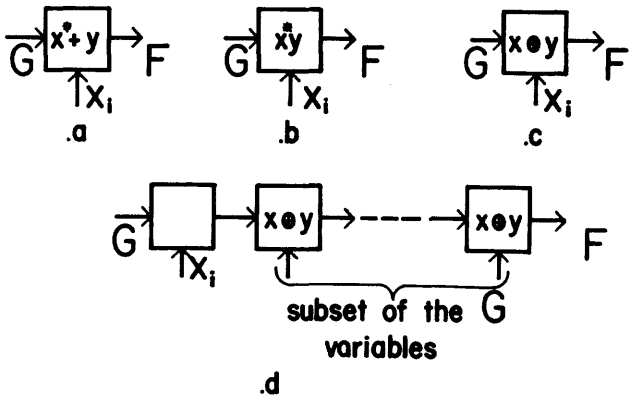


Figure 1 – Functions to be performed by the cells in a Maitra cascade

Four different algorithms have been published for the synthesis of Boolean functions by means of Maitra cascades. Three of them concern the irredundant Maitra cascades[2-4] while the fourth[5] can be used for the redundant ones, too (see also Ref. 6). The methods proposed by the authors of the above papers are different but the properties of the Boolean functions from which they are derived are quite similar. These properties can be summarized by the following two Propositions:

Proposition 1:

A completely specified binary function $F(x_1 \ldots x_n)$ is realizable by a Maitra cascade if both of the following hold:

I – There exists a variable x_i such that one of the following conditions is satisfied:

a: $F(x_1 \ldots x_n) = x_i^* + G(x_1 \ldots x_{i-1}x_{i+1} \ldots x_n)$
b: $F(x_1 \ldots x_n) = x_i^* G(x_1 \ldots x_{i-1}x_{i+1} \ldots x_n)$
c: $F(x_1 \ldots x_n) = x_i^* \oplus G(x_1 \ldots x_{i-1}x_{i+1} \ldots x_n)$
d: $F(x_i^* = 1)$ is a parity function

II – In the cases a,b,c, function $G(x_1 \ldots x_{i-1}x_{i+1} \ldots x_n)$ is Maitra realizable; in case d, $G = F^*(x_i = 1) \oplus F(x_i = 0)$ is Maitra realizable

Proof: See Refs. 5 or 6.(*)

Proposition 2:

When x_i satisfies conditions a,b,c of Proposition 1, F can be obtained at the output of a cell whose inputs are x_i and G, as it is respectively shown in Fig. 1a, 1b, or 1c. When x_i satisfies condition d, F is obtained at the output of the cascade shown in Fig. 1d. In this cascade, the function produced by the cell whose input is x_i, is selected as shown in Fig. 2. The exclusive-or cascade following the x_i cell must realize the odd parity function $F^*(x_i^* = 1)$(*)

(*) f^* stands either for f or for its negative, f.

(*) Methods given in Refs. 3-5 are derived directly from the statements of Propositions I and II. Maitra's method[2] is developed in a different way, but can be stated from the above propositions (see Section 6).

If the parity function is:	The function performed by the cell of X_i is:	and G is so defined:
$F(\bar{x}=1)$	xy	$F(0) \bullet F(1)$
$\bar{F}(\bar{x}=1)$	$\bar{x}+y$	$\bar{F}(0) \bullet F(1)$
$F(x=1)$	$\bar{x}y$	$F(0) \bullet F(1)$
$\bar{F}(x=1)$	$x+y$	$F(0) \bullet \bar{F}(1)$

Figure 2—Statement of properties of the cascade in the case d of Prop 1

Proof: See Refs. 3-5.

In order to test if a given function can be synthesized by a Maitra cascade,[3-5] it is necessary to look for a variable x_i satisfying one of the conditions a, b, c, d; then a function G with only n−1 variables is obtained. The same considerations used for F must be repeated for G. So, by applying the statement of Propositions 1 and 2 at most n−2 times, it is possible either to obtain the cascade realizing F, or to establish that there is no cascade realizing F. If, in the application of of the algroithm condition d is never used, the resulting cascade is irredundant; in the other case it is a redundant one.

From the above considerations it is clear that, for testing the Maitra realizability of a given function F, it is necessary to be able: (i) to look for a variable x_i and then (ii) to derive the new function G with one variable less than F. By using the Karnaugh map it is possible to solve both above problems in a simple way, both for redundant and for irredundant cascades. To show this, in the following sections some rules are given for testing on the Karnaugh maps for the x_i variable requested by Proposition 1 (Rules AI BI CI DI). Then four other rules are given for deriving the mapping of G from the one of F (AII BII CII DII). These rules are given in four different section according ω the condition a, b, c, d of Proposition 1 to be satisfied by x_i.

There are additional problems in the statement of an algorithm for Maitra cascades. These problems are concerned with the need of selecting variables where there is more than one x_i satisfying Proposition 1. They are not considered here because they are completely solved[4,5] and do not involve Karnaugh maps.

Rules A {B}

If a variable x_i satisfies condition a {b} of Proposition 1:

I—The Karnaugh mapping of $F(x_1 \ldots x_n)$ has the (n−1)-cube $[x_i^*]$ $\{[x_i^*]\}$ completely labeled {unlabeled}

II—The mapping of G is given in the subcube (*) $[x_i^*]$ $\{[\bar{x}_i^*]\}$

Proof: Obvious

Example:

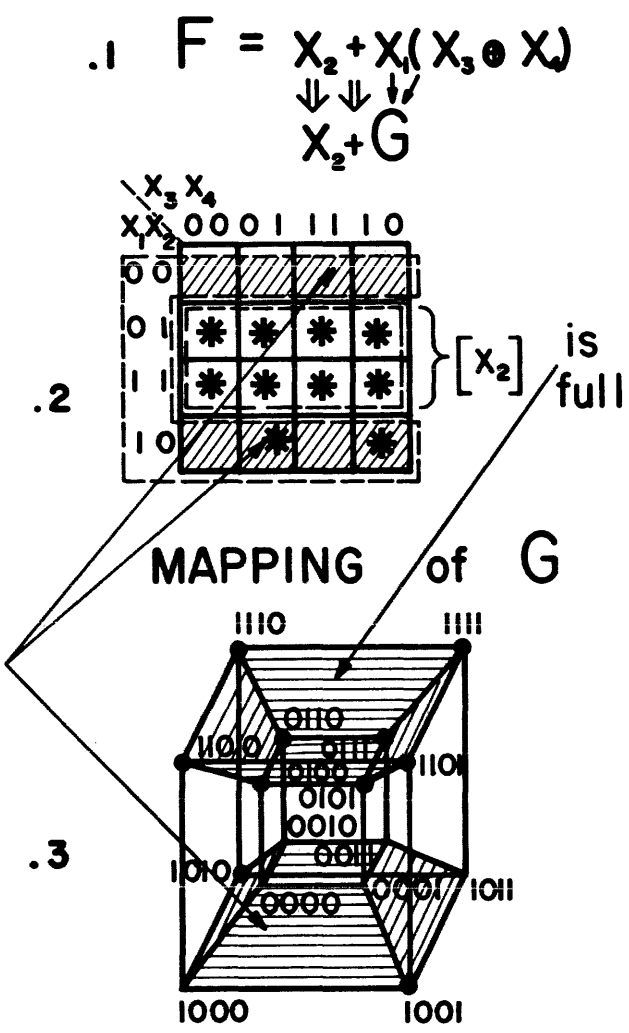


Figure 3—An example of application for rules A

Consider the example shown in Fig. 3.1 {4.1} whose Karnaugh mapping is given in Fig. 3.2 {4.2}. The 3-cube $[x_2]$ $\{[\bar{x}_2]\}$ is completely full {empty} and the mapping of the residual function G is given in 3-cube $[x_2]$ $\{\bar{x}_2]\}$. For the sake of clarity, in Fig. 3.3 {4.3} the mapping of F over a picture of 4-cube is given.

(*) In the remainder of this paper the (n−1)-cube is indicated by $[x^*]$ iff x_i is equal in all the coordinates of its vertices; if $x_i^* = x_i$ $\{x_i^* \ldots x_i\}$ the coordinate x_i is 1{0}

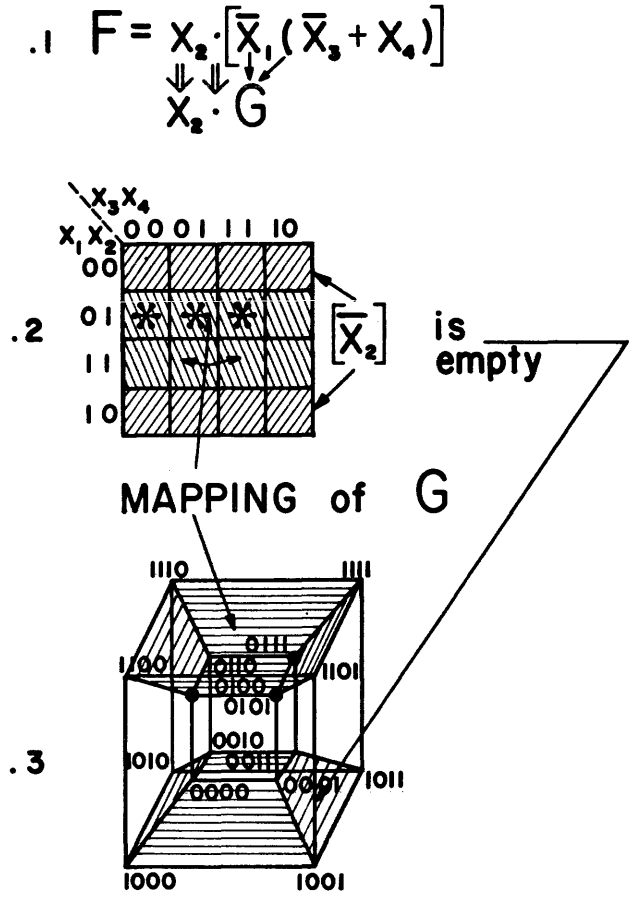


Figure 4 – An example of application for rules B

Rules C

If variable x_i satisfies condition c of Propositions 1:
I – The 2 (n − 1)-cubes $[x_i]$ and $[\bar{x}_i]$ are complementary, i.e., for any pair of vertexes $(\xi\xi')$ such that $\xi \epsilon [x_i]$ $\xi' \epsilon [\bar{x}_i]$ and the Hamming distance between ξ and ξ' is 1, one and only one of its elements is labeled.*
II - G is mapped on the (n–1)-cube $[x_i]$

Proof:

If condition c of Proposition 1 holds, F satisfies the following identity

$$f = \bar{x}_i G + x_i \bar{G}$$

Therefore, the function mapped on $[x_i]$ is the inversion of the function mapped on $[\bar{x}_i]$ and a minterm belonging {not belonging} to G cannot belong {must belong} to G. This implies that for every vertex of $[x_i]$ labeled, its adjacent vertex in $[\bar{x}_i]$ cannot be labeled and vice versa. End of proof.

Example:
An example is shown in fig. 5.

(*) This is the same as to state ξ and ξ' are adjacents but belonging to opposite (n–1)-cubes.

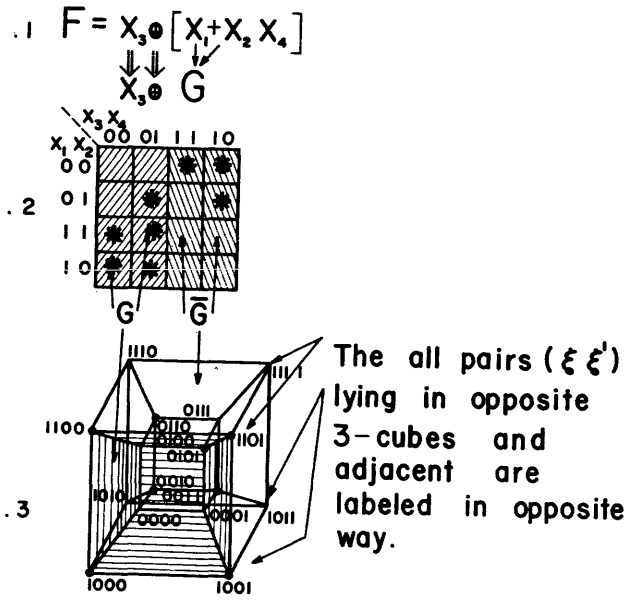


Figure 5 – An example of application for rules C

Rules D

If variable x_i satisfies condition d of Proposition 1, either an odd parity function or its inverse is mapped in the (n–1)-cube $[x_i^*]$. To test if this happens, the following properties can be used:

α - Both if an odd or an even parity function is mapped on $[x_i^*]$, there are 2^{n-2} vertexes labeled on it.

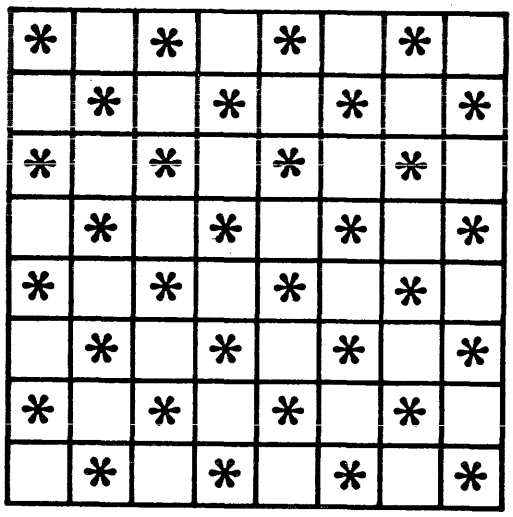
β - If the function mapped on $[x_i^*]$ is either a nondegenerate (i.e., depending on all the variables $x_i, \ldots x_{i-1}$ $x_{i+1} \ldots x_n$) odd parity function or its inverse, no labeled {unlabeled} vertex in $[x_i^*]$ has any of its adjacent vertexes belonging to $[\bar{x}_i^*]$ labeled {unlabeled} (see Fig. 6).

γ - If an odd parity function is mapped on $[x_i^*]$, vertex $00 \ldots 0x^* 00 \ldots 0$ is unlabeled. If an even parity function is mapped $00 \ldots 0x_i^* 0 \ldots 0$ is labeled.

By making use of the above conditions, the following rule can be derived·

I - For testing if a parity function is mapped on $[x_i^*]$ first count the number of labeled vertexes in $[x_i^*]$ If they are 2^{n-2}, reduce the function mapped on $[x_i^*]$ until it depends on all the coordinates of the map, then test condition β. If this condition is fulfilled, by using condition γ, test whether the function mapped on $[x_i^*]$ is an odd or an even parity function. There are two different rules for determining the mapping of G according to whether $F(x_i^* = 1)$ or $\bar{F}(x_i^* = 1)$ is an odd parity function.

II$\bar{\alpha}$ - If $F(x_i^* = 1)$ is an odd parity function, $G = F(x_i = 0) \oplus F(x_i = 1)$ is obtained by performing the



No labeled cell has an adjacent labeled cell and vice versa.

Figure 6 – Map of a parity function

exclusive-or of all the pairs ($\xi\xi'$) of vertexes the elements of which are both adjacent and belonging to opposite (n–1)-cubes $[x_i]$ and $[\bar{x}_i]$.

II$\bar{\beta}$ – If $F(x_i^* = 1)$ is even parity function, the coincidence operation must be performed instead of the exclusive-or.

Example:

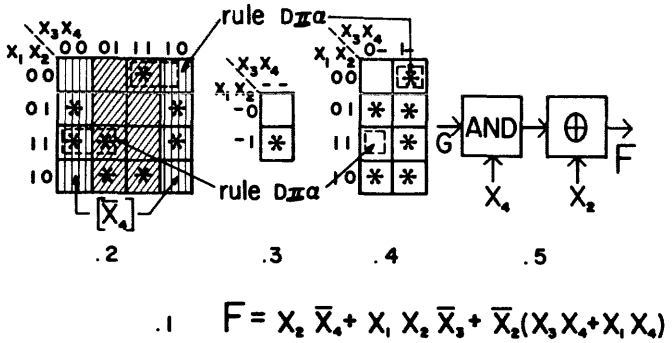


Figure 7 – An example of application for rules D

Rules DI DII are applied in the example shown in Fig. 7. The function F is mapped in Fig. 7.2. The function mapped on $[\bar{x}_4]$ is x_2 (odd parity function of one variable) as it is easy to see after reduction to irredundant form (Fig. 7.3). Cascade realizing F is shown in Fig. 7.5 whose function G is mapped in Fig. 7.4 obtained from Fig. 7.2 by applying rule DIIα

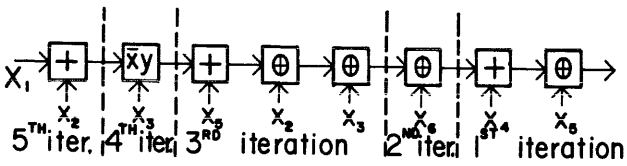


Figure 8 – A general example

An example

The function to be synthesized is mapped in Fig. 9. Variable x_4 satisfies condition d of Proposition 1 because the function mapped in 5-cube $[x_4]$ is equal to $\bar{x}_5$ and so it is the even parity function of one variable. The dependent input G of cell x_4 goes into is mapped in Fig. 10. This mapping has been obtained by applying rule DIIβ. Variable x_6 satisfies on the map of Fig. 10 condition c of Proposition 1. The new function G is mapped on the 4-cube $[\bar{x}_6]$ of Fig. 10 and is given in Fig. 11. Variable x_5 satisfies condition d of Proposition 1. In fact, the $4 = 2^{3-1}$ vertexes of the 3-cube $[x_5]$, after reduction (see Fig. 14), give the function $\bar{x}_3 \oplus x_2$. The new function G is given in Fig. 12 that is obtained by applying Rule DIIβ. Variable x_3 satisfies condition b of Proposition 1 on the mapping of Fig. 12. On the 2-cube $[x_3]$ of Fig. 12 is mapped the function $x_1 + x_2$ that can be realized by a cell. By applying Proposition 1 five times and using Propositions 2 at each step, a Maitra cascade has been obtained realizing F at its output (see Fig. 8).

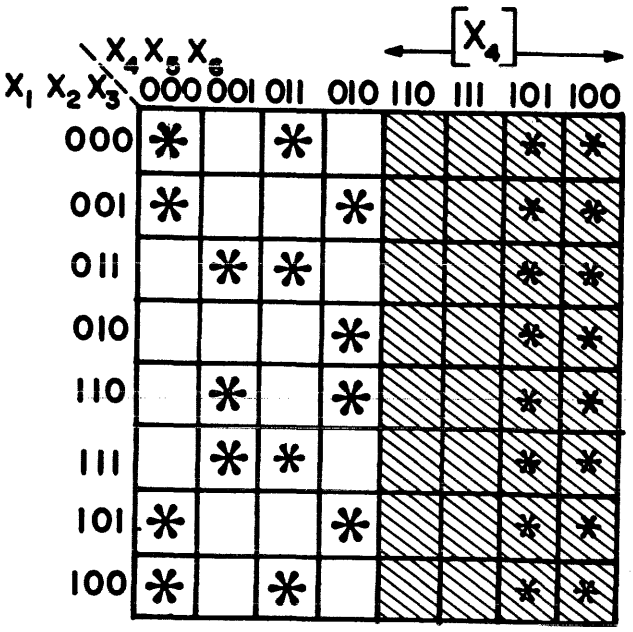


Figure 9 – Map of the function to be synthesized

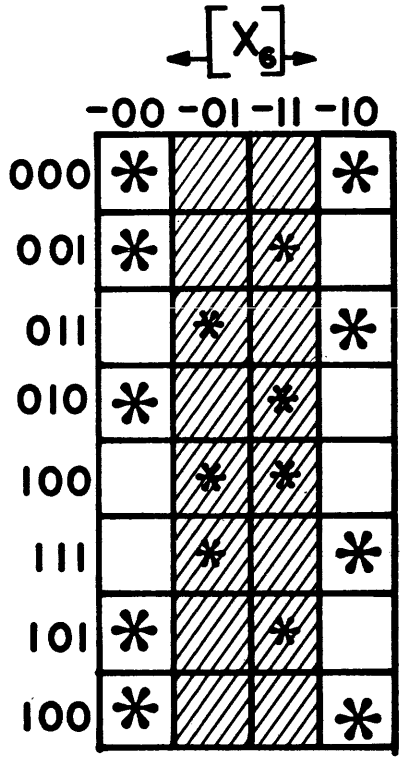


Figure 10 – First residual function

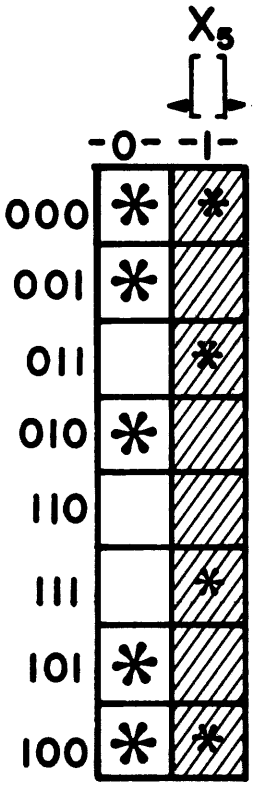


Figure 11 – Second residual function

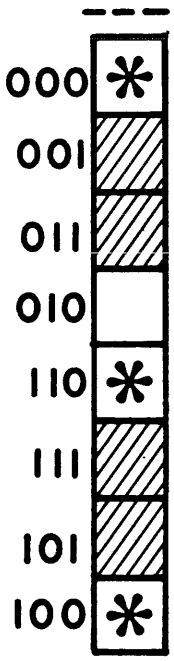


Figure 12 – Third residual function

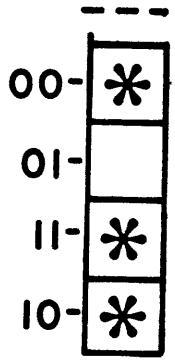


Figure 13 – Fourth residual function

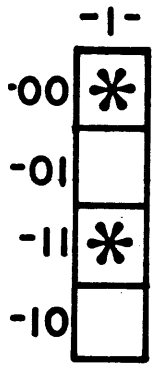


Figure 14 – Simplification of the function mapped in the 3-cube $[x_6]$ of fig 11

SUMMARY

In the studies on Cellular Logic, much work has been done with Maitra cascades. This is due to the fact that these cascades are useful as "elementary" module in more sophisticated cellular arrays. R. C. Minnick[1] has suggested the introduction of Karnaugh maps in the study of maitra cascades; he has given a casuistry for determining the Maitra realizability of a Boolean function with four variables by using its Karnaugh mapping. In this paper the application of Karnaugh maps to test for the Maitra realizability of any function with any number of variables is studied. A set of necessary and sufficient conditions is given to test the Maitra realizability by using only Karnaugh maps. The practical usefulness of the rules given here is limited only by the difficulty to study Karnaugh maps with more than 6-7 variables. Furthermore, the results given in this paper can be advantageously applied to the study of Rectangular Arrays.

CONCLUSION

The "realizability test" [2 p. 140] given by Maitra makes use of a geometrical representation of Boolean function that is a slight modification of the Karnaugh map. The algorithm introduced by Maitra is given without any mention of the Karnaugh maps and its itility is limited by the prior need to establish an ordering of the variables. Furthermore, it can be applied only for the irredundant cascades. By making explicit reference to the Karnaugh maps, some rules are obtained in this paper that can be applied for both redundant and the irredundant cascades. They do not require any ordering of the variables. These rules, in the particular case of the irredundant cascades, are similar to those of Maitra but they are derived in a different way and, when they are used for applying the algorithm given in Refs. 3 and 4, the cascade synthesizing F can be obtained. The results obtained in this paper can be applied to more general cellular arrays both rectangular and one dimensional as is shown in Ref. 7.

ACKNOWLEDGMENT

The author wishes to thank Prof. Robert C. Minnick for his help in improving this paper and Kathleen Wright who typed the manuscript.

REFERENCES

1 R C MINNICK et al
Cellular arrays for logic and storage
Final Report AFCRL-66-613 AD 643178 Stanford Research Institute Menlo Park California April 1966

2 K MAITRA
Cascade switching networks of two input flexible cells
IRE Transactions EC Vol EC-11 No 2 pp 136-143 April 1962

3 J SKLANSKY
General synthesis of tributary switching networks
IEEE Transactions on EC Vol EC-12 No 5 pp 464-469 October 1963

4 S LEVY R WINDER T MOTT JR
A note on tributary switching networks
IEEE Transactions EC Vol EC-13 No 2 pp 148-151 April 1964

5 H STONE A KORENJAK
Canonical form and synthesis of cellular cascades
IEEE Transactions on EC Vol EC-14 No 6 pp 852-862 December 1965

6 G FANTAUZZI
Catene di Maitra
Internal Report Fondazione Ugo Bordoni Rome Italy December 1966

7 G FANTAUZZI
Catene di Short
Internal Report Fondazione Ugo Bordoni Rome Italy December 1967

Universal logic circuits and their modular realizations

by S. S. YAU and C. K. TANG
Northwestern University
Evanston, Illinois

INTRODUCTION

In order to achieve the great economic advantage of utilizing integrated circuits in computer circuitry, it is desirable to design a circuit which can realize *any* logic function of a fixed number of variables by simply varying its input terminal connections. Such a circuit is called a *universal logic circuit (ULC)*. When the number of variables becomes large, a ULC may be too complex to be built in a single package economically. Hence, it is preferred to use ULC's of a small number of variables as the modules to build a ULC of a large number of variables. Such modules are called *universal logic modules (ULM's)*. In this paper, we shall first present a three-variable ULC, which has a fan-in for each logic gate not exceeding four, and consists of only 7 I/O pins. Then, we shall extend the ULC's to four or more variables. There are 12 I/O pins in a ULC of four variables, and several models with different fan-in limitations will be given. The logic gates in the ULC's may be all NAND or all NOR gates. Then, a simple technique for designing a ULC of any large number of variables using the ULC's of a small number of variables, say three variables, as the ULM's will be established. It will be seen that the ULC obtained by this technique will require a small number of ULM's. Moreover, the fault-detection tests for ULM's and a diagnostic procedure for locating all the faulty ULM's in the modular realization of a ULC realizing a given logic function will be presented. Finally, a method for improving the reliability of a ULC using an error-correcting code will be demonstrated.

Universal logic circuits of three variables

The problem of designing a ULC was first treated by Forslund and Waxman,[1] and later by Ellison, et al.[2] and Elspas, et al.[3] They employed the concept of equivalence classes to reduce the number of all possible logic functions of a given number of variables to the number of the equivalence classes. An equivalence class is a set of logic functions that may be obtained from a particular network by only manipulating the application of variables to the input terminals of the network. One of the most common constraints on these manipulations is that only true variables are available with the permutation of the variables at the input terminals permitted. With this restriction, Hellerman[4] partitioned the $2^{2^3} = 256$ three-variable logic functions into 80 equivalence classes. In order to reduce the number of equivalence classes, Forslund and Waxman[1] assumed that both true and complement variables are available at the input, and true and complement logic functions are both available at the output (two output terminals). In addition, biasing (to a logical 1 or 0) and duplication of input variables to the input terminals are also permitted. The equivalence classes defined this way reduces its number from 80 to 10 for three-variable logic functions. In this paper, the same constraints are to be placed on the manipulations of input terminals, except that only one output terminal will be required and that the biasing "0" and "1" are not necessary. We shall employ a different approach to obtain the ULC.

It is noted that a logic function f(x, y, z) of three variables x, y, z can always be expanded with respect to any two of the three variables x, y, z as follows:

$$f(x, y, z) = \bar{x}\,\bar{y}\,f(0, 0, z) + \bar{x}\,y\,f(0, 1, z) + x\,\bar{y}\,f(1, 0, z) + x\,y\,f(1, 1, z), \quad (1)$$

where the functions f(0, 0, z), f(0, 1, z), f(1, 0, z) and f(1, 1, z) are functions of z only, and each of these functions assumes one of the four values: z, z, 0 or 1. Hence, a circuit shown in Fig. 1 can realize any arbitrary three-variable logic function f(x, y, z) if the side terminals C_1 and C_2 are connected to x and y respectively and the four front terminals A_0, A_1, A_2, and A_3 are connected to the appropriate values z, z, 0, and 1. Based on (1) we obtain a ULC of three variables consisting of AND, OR and INVERTER gates shown in Fig. 1. It is noted that the biasing "0" and "1" are not necessary. In Fig. 1, for example, if f(0, 0, z) = 0, connecting terminal A_0 to biasing "0" is the same as connecting A_0 to input variable y;

and if f(0, 0, z) = 1, connecting terminal A_0 to biasing "1" is the same as connecting A_0 to input variable $\bar{y}$. Similar arguments apply to terminals A_1, A_2, and A_3.

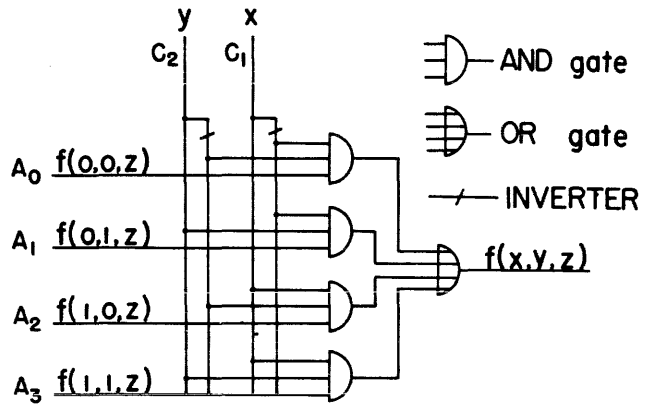


Figure 1 – A ULC of three variables consisting of AND, OR and NOT gates

It is well-known[5] that a two-level AND and OR circuit can be replaced by a NAND-gate circuit of the same configuration. Thus, the circuit shown in Fig. 2 is also a ULC of three variables employing NAND gates only.

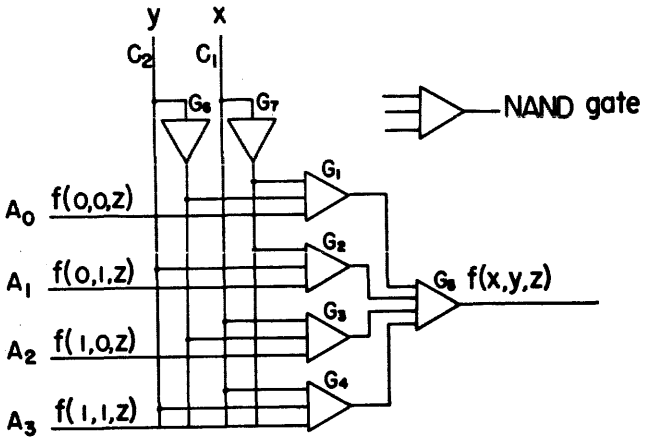


Figure 2 – A ULC of three variables consisting of NAND gates only

A ULC of three variables consisting of NOR gates can be obtained by using the dual relationship between NAND's and NOR's, and is given in Fig. 3. It is noted that the configurations of the ULC with NAND gates and the ULC with NOR gates are identical, and the only difference between these two circuits is the permutation of the input values for the front terminals. Another realization is shown in Fig. 4 using NAND, OR and INVERTER gates. This circuit is more desirable than those shown in Figs. 2 and 3, since it requires only 16 diodes and 3 transistors, while the circuits shown in Figs. 2 and 3 need 16 diodes and 7 transistors. Furthermore, the circuit shown in Fig. 4 is more reliable because fewer transistors are used. A similar circuit can be obtained using AND, NOR and INVERTER gates. In each of the above circuits, a total of 7 I/O pins is required.

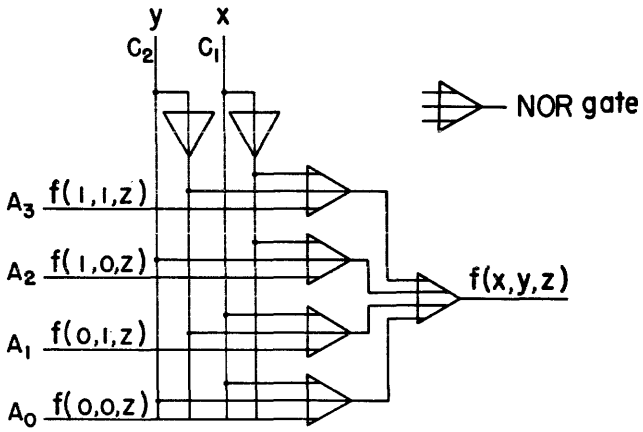


Figure 3 – A ULC of three variables consisting of NOR gates only

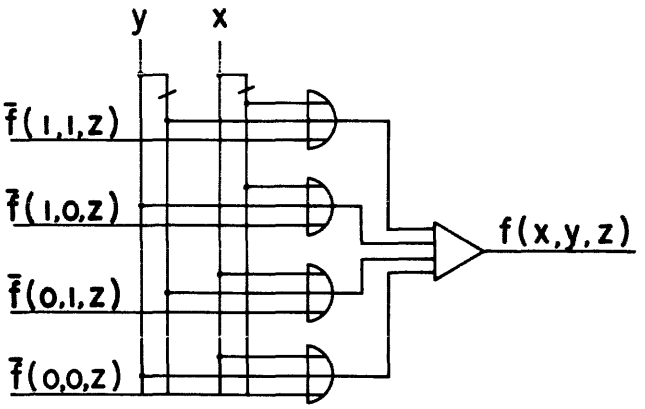


Figure 4 – A ULC of three variables consisting of OR, NAND and INVERTER gates

To evaluate the ULC given above, a comparison with the results given by Forslund and Waxman[1] is made as follows: Consider the circuit shown in Fig. 2 as a minimum-pin ULC. Since gates G_6 and G_7 are included in the ULC, the circuit has 7 pins, 7 gates, and 3 levels. The minimum-pin ULC of three variables given by Forslund and Waxman also has 7 pins, but it requires 10 gates and has 5 levels. The ULC given in Fig. 2 also has the advantage that only one complement input is required, whereas the minimum-pin ULC given by Forslund and Waxman requires two complement inputs. If the circuit shown in Fig. 2 is considered as a minimum-gate ULC, gates

G_6 and G_7 should be excluded from the ULC at the expense of adding two more pins. Consequently, it ends up with a minimum-gate ULC of 5 gates, 9 pins and 2 levels. The minimum-gate ULC of Forslund and Waxman has 6 gates, 9 pins and 4 levels, and can realize only the logic functions in nine out of the ten equivalence classes. It is seen that 5 is the absolute minimum number of gates required for any ULC of 3 variables, since the realization of the exclusive-or function of 3 variables alone requires a minimum of 5 NAND gates.[6]

Universal logic circuits of four and more variables

The problem of designing a ULC of four or more variables was also treated by Forslund and Waxman[1], using the same idea of equivalence classes as in the case of three-variable ULC. Due to the large amount of computations required, it is prohibitive to obtain such a ULC by that method. However, the approach used in the last section for obtaining the ULC of three variables can readily be extended to four or more variables. Since a logic function f(x, y, z, w) of four variables can be written in the form

$$\begin{aligned} f(x, y, z, w) = {} & \bar{x}\,\bar{y}\,\bar{z}\,f(0,0,0,w) + \bar{x}\,\bar{y}\,z\,f(0,0,1,w) \\ & + \bar{x}\,y\,\bar{z}\,f(0,1,0,w) + \bar{x}\,y\,z\,f(0,1,1,w) \\ & + x\,\bar{y}\,\bar{z}\,f(1,0,0,w) + x\,\bar{y}\,z\,f(1,0,1,w) \\ & + x\,y\,\bar{z}\,f(1,1,0,w) + x\,y\,z\,f(1,1,1,w), \end{aligned} \tag{2}$$

the ULC of four variables shown in Fig. 5 is obtained. It is noted that there is a NAND gate with a fan-in of 8 in this realization. Two other NAND realizations with smaller fan-in limitations and more gates are given in Figs. 6 and 7. Similar to the case of three-variable ULC's, the corresponding NOR realizations of Figs. 5-7 can easily be shown that they have the same configurations of the original NAND realizations with their input terminal connections permuted. The rule of permutation on the residue functions of one variable for the NOR realization is to replace 1 by 0 and 0 by 1 in the residue functions for a NAND realization. For instance, f(0, 1, 0, w) in Fig. 5 should be replaced by f(1, 0, 1, w) for the corresponding NOR realization.

The ULC's of five or more variables can be derived in a similar way. It can be shown that a ULC of n variables obtained by this method has p input pins, where

$$p = 2^{n-1} + n - 1. \tag{3}$$

With a fan-in limitation of four, this approach will yield a ULC of n variables, $n \geq 2$, which has q gates and ℓ levels, where

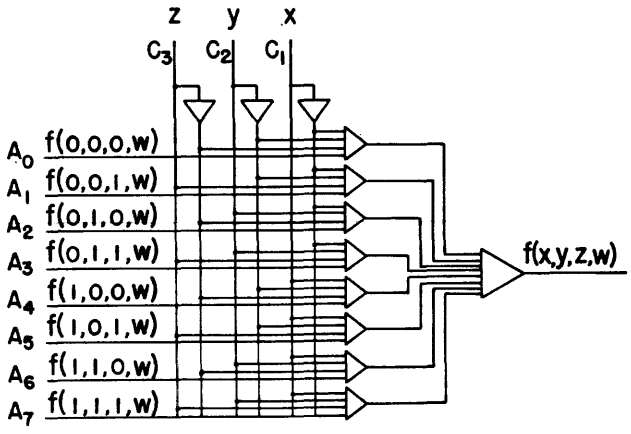


Figure 5 – A ULC of four variables with 3 levels and a fan-in 8

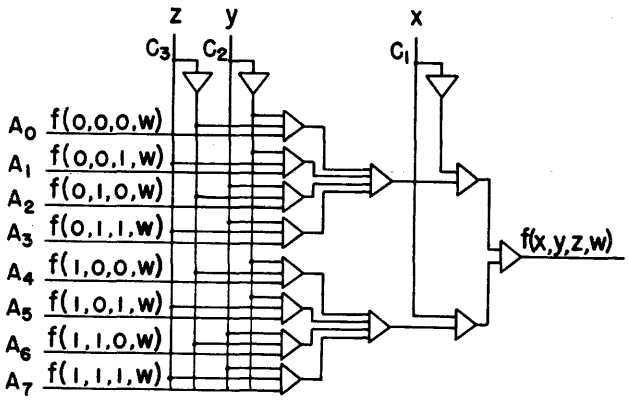


Figure 6 – A ULC of four variables with 5 levels and a fan-in 4

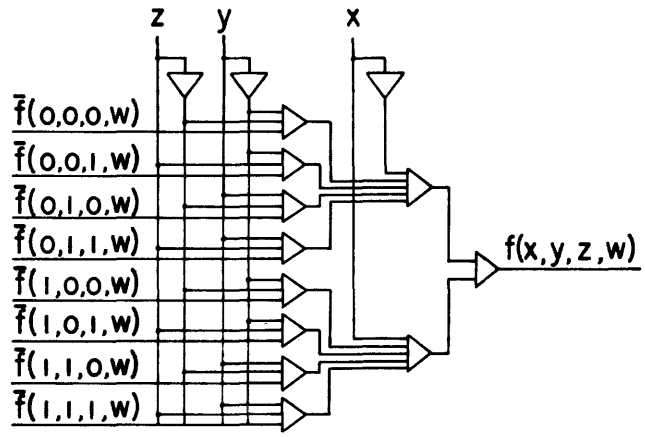


Figure 7 – A ULC of four variables with 4 levels and a fan-in 5

$$q = \begin{cases} \frac{5}{3}(2^{n-1} - 1) + (n - 1) & \text{when n is odd} \\ \frac{10}{3}(2^{n-2} - 1) + (n + 2) & \text{when n is even} \end{cases} \quad (4)$$

$$\ell = \begin{cases} n & \text{when n is odd} \\ n + 1 & \text{when n is even.} \end{cases} \quad (5)$$

For example, a ULC of five variables with a fan-in limitation of four is shown in Fig. 8. The numbers given by (4) and (5) can certainly be reduced if a larger fan-in is permitted. It is noted that for any n only one complementary input variable is required and all others can be true input variables in a ULC obtained by this method.

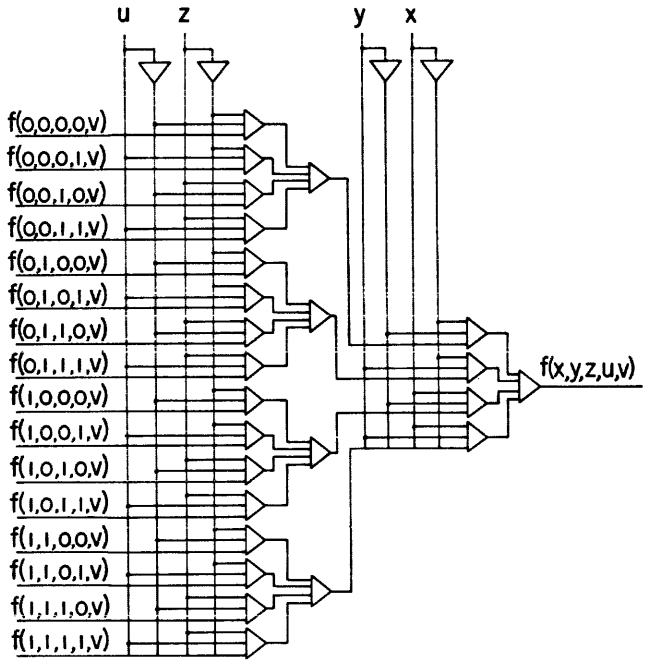


Figure 8 – A ULC of five variables

A comparison of the number of input pins required here and the upper bound of the number of input pins required for a ULC given by King[7] is shown in Table I. The upper bound given by King is for the ULC defined in a slightly different way, namely only true inputs are used, while in this paper one complementary input is allowed. It is seen that the values of p are considerably lower than King's upper bound. A lower bound on the number of required input pins is derived here with the restriction that only one complementary input is allowed. First we calculate the number of possible distinct connections of p input pins to the set of values $\{0, 1, x_1, x_2, \ldots, x_n, x_n\}$ such that every x_i, $1 \leq i \leq n$, is connected to at least one pin. It can be shown that this number is given by

The number p of input pins of a ULC and King's upper bound on p.

n	2	3	4	5	6
p	3	6	11	20	37
King's upper bound	6	11	20	37	70

TABLE I – The number p of input pins of a ULC and King's upper bound on p.

$$(n+3)^p - \binom{n}{1}(n+2)^p + \binom{n}{2}(n+1)^p - \binom{n}{3}n^p + \ldots + (-1)^n \binom{n}{n} 3^p.$$

The number of possible distinct connections must not be smaller than the number of logic functions of n variables. Hence, the following inequality is obtained.

$$(n+3)^p - \binom{n}{1}(n+2)^p + \binom{n}{2}(n+1)^p - \binom{n}{3}n^p + \ldots + (-1)^n \binom{n}{n} 3^p \geq 2^{2^n}.$$

The minimum p satisfying the inequality (6) is a lower bound on p, which is listed in Table II. It is noted that Elspas, et al, [3] have derived a ULC of 4 variables with a total of 9 I/O pins, and by decomposition a ULC of 5 variables with a total of 19 I/O pins was obtained. For $n \geq 6$, their result requires exactly the same number of input pins given by (3). They have also derived a lower bound for p, which is smaller than that listed in Table II, because all complementary inputs are allowed in their derivation.

TABLE II – The lower bound on p calculated according to (6).

n	2	3	4	5	6
Lower bound	3	5	7	12	21

Realization of a universal logic circuit using universal logic modules

We have shown in the last section that a ULC of any large number of variables can be found. However, it follows from (3)-(5) that the complexity of the ULC increases rapidly as the number of variables increases. From either economical point of view or maintenance point of view, it becomes prohibitive to build ULC's of various large numbers of variables in individual integrated circuit packages. Hence, we would like to present a technique for realizing a ULC of a large number of variables using identical ULC's of a small number of variables as modules, which are called universal logic modules (ULM's). Obviously, there are two great advantages of this technique. First, we only need a large quantity of identical

ULM's to build ULC's of various numbers of variables. Secondly, when there are faults in a ULC, we only need replace the faulty ULM's instead of the whole ULC.

To derive the modular realization of a ULC of n variables using ULC's of 3 variables as the ULM's (denoted by ULM-3's), let us first consider the case when n is odd. Since any logic function $f(x_1, x_2, \ldots, x_n)$ of n variables, $n \geq 3$, can be expanded to the form

$$\begin{aligned} f(x_1, x_2, \ldots, x_n) = {} & \bar{x}_1\bar{x}_2 f(0, 0, x_3, \ldots, x_n) \\ & + x_1 x_2 f(0, 1, x_3, \ldots, x_n) \\ & + x_1\bar{x}_2 f(1, 0, x_3, \ldots, x_n) \\ & + x_2 x_2 f(1, 1, x_3, \ldots, x_n), \end{aligned} \tag{7}$$

it can be realized by a ULM-3, provided that the side terminals C_1 and C_2 and the front terminals A_0, A_1, A_2 and A_3 shown in Fig. 1, 2, or 3 are connected to the input variables x_1 and x_2 and the residue functions $f(0,0,x_3,\ldots,x_n)$, $f(0,1,x_3,\ldots,x_n)$, $f(1,0,x_3,\ldots,x_n)$ and $f(1,1,x_3,\ldots,x_n)$ respectively. This ULM-3 forms the first level of the modular realization of the ULC. Since we can repeat this process to each of the residue functions, the second level of the modular realization consists of four ULM-3's whose side terminals are connected to the input variables x_3 and x_4 and front terminals connected to appropriate residue functions of $n-4$ variables. Continue this process until the residue functions become functions of the variable x_n. Because n is odd, and because each expansion reduces the number of variables of the residue functions by exactly 2, it requires a total of $(n-1)/2$ expansions. This implies that $f(x_1, \ldots, x_n)$ can be realized by using ULM-3's in a tree structure consisting of $(n-1)/2$ levels as shown in Fig. 9. It is seen that there are 4^{j-1} ULM-3's in the j-th level of the tree structure. Each of the front terminals of the ULM-3's in the last level is connected to one of the four values 0, 1, x_n and $\bar{x}_n$ defined by the corresponding residue function of variable x_n, which can be found as follows: Trace the path from the output terminal F to the front terminal in the last level in question in the tree structure, and use two bits to write the binary representation of the subscript h for the front terminal A_h of the ULM-3 in each level. Then, the concentration of the ½(n−1) 2-tuples in the order of the path forms the argument of the residue function for the front terminal. For instance, if the path from the output terminal to a front terminal in the last level in a modular ULC of 5 levels passes through the front terminals A_1, A_2, A_0, A_3, A_1 of the ULM-3's in the 1st, 2nd,...,5th levels respectively, the residue function for this terminal is $f(0,1,1,0,0,0,1,1,0,1,x_n)$. For convenience, we shall call the front terminal of a ULM-3 in the last level P_i if it is connected to the residue function with the binary argument whose decimal representation is i. It is obvious that there are 2^{n-1} front terminals of the ULM-3's in the last level for a modular ULC of n variables. Fig. 10 shows the modular realization of a ULC of 7 variables using ULM-3's.

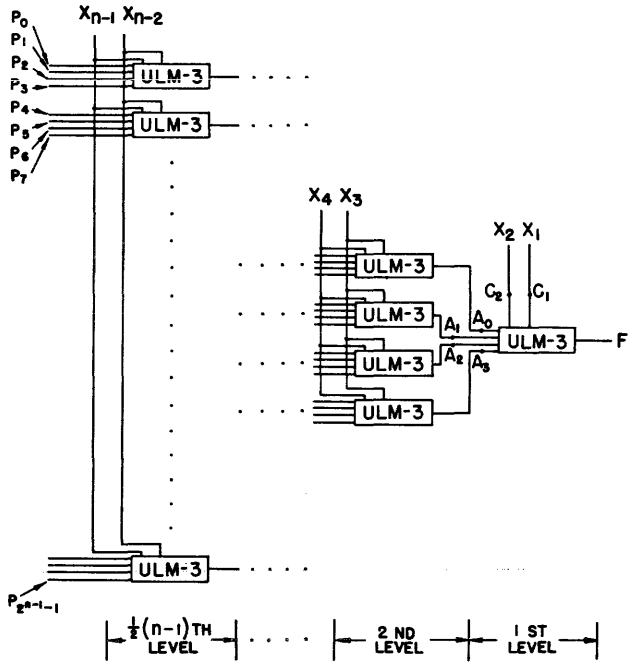


Figure 9—The modular realization of a ULC of n variables when n is odd

When n is even and when only ULM-3's can be used in the modular realization, only slight modification in the first level is required. Instead of expanding the logic function according to (7) for the first level, we only expand the logic function as follows:

$$f(x_1, x_2, \ldots, x_n) = \bar{x}_1 f(0, x_2, \ldots, x_n) + x_1 f(1, x_2, \ldots, x_n). \tag{8}$$

It is easily seen that (8) can be realized by a ULM-3, provided that the side terminals C_1 and C_2 are both connected to the input variable x_1, the front terminals A_0 and A_3 connected to the residue functions $f(0, x_2, \ldots, x_n)$ and $f(1, x_2, \ldots, x_n)$ respectively, and the connections for A_1 and A_2 are don't-care. Then, each of the residue functions in (8) is a function of an odd number of variables and hence can be realized by the previous method. The residue functions for the front terminals of the ULM-3's in the last level can be found in the same way as before except that only the first bit in the binary argument of the residue function corresponds to the subscript of the front terminal of the ULM-3 in the first level. The first bit is 0 or 1 depending upon whether the front terminal

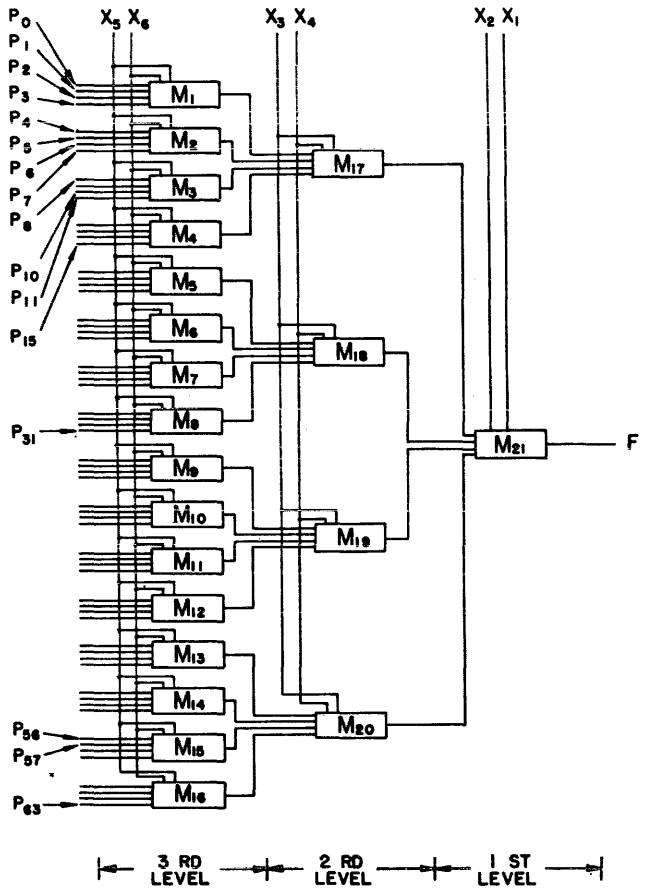


Figure 10—The modular realization of a ULC of 7 variables

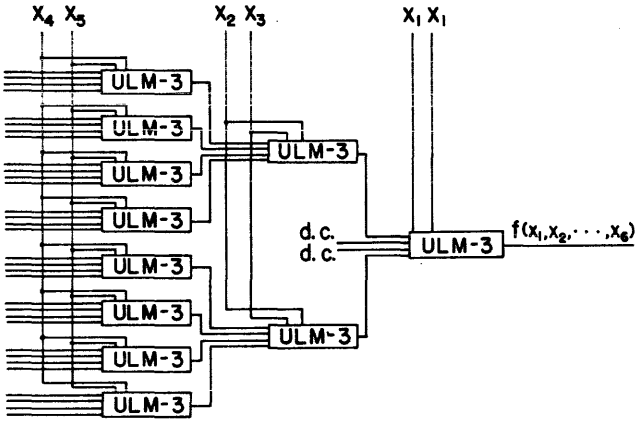


Figure 11—A modular realization of a ULC of six variables using ULM-3's only

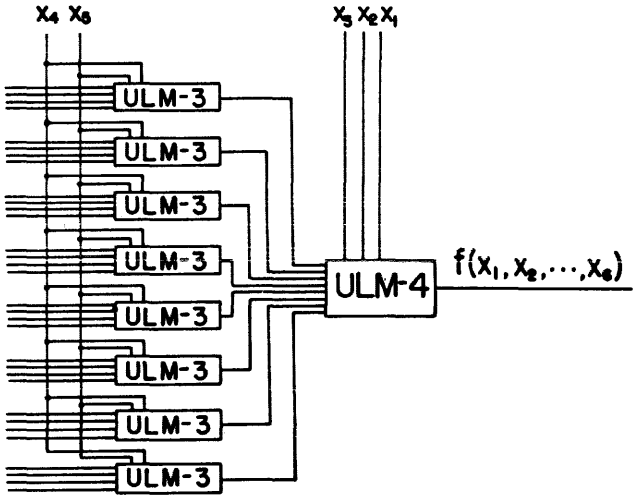


Figure 12—A modular realization of a ULC of six variables using ULM-3's and a ULM-4

of the ULM-3 in the first level in the path is A_0 or A_3 respectively. Fig. 11 shows such a modular realization of a ULC of 6 variables. It is noted that in this case we have not used the full capacity of the ULM-3 in the first level. In fact, if we are not restricted to use ULM-3's only in the modular realization, the three ULM-3's in the first and second levels can be substituted by a ULM-4 as shown in Fig. 12. The terminal connections and the residue functions for the front terminals of the ULM-4 in the last level can be found by considering the ULM-4 shown in Fig. 5 or 6 and the expansion of $f(x_1, x_2, \ldots, x_n)$ with respect to the variables x_1, x_2, and x_3.

The above modular realization technique can easily be extended to using ULM's of any variables. If only ULM's of a fixed number of variables can be used, it is often necessary to have some don't-care terminal connections to some of the ULM's and hence some of the ULM's are not utilized to their full capacity. It can be shown that if only ULM-4's are used in the modular realization, there will be no don't-care terminal connections to any ULM-4 for a ULC of n variables, where $n = 3\alpha + 4$ and α is a nonnegative integer. However, if both ULM-3 and ULM-4 are used in the modular realization, the don't-care terminal connections can always be avoided. Furthermore, it is noted that the tree structure of the modular realization of a ULC of n variables using ULM-k's always has 2^{n-1} front terminals in the last level for any k.

Fault-detection test for ULM's and a fault-diagnostic procedure for modular ULC's

The problem of developing a practical fault-diagnostic procedure for a logic circuit of a large number of variables is still far from being solved.[8-12] When integrated circuit packages are used for logical design,

so far there is no practical method of deriving a set of minimal fault diagnostic tests which can locate the faulty packages. At present, it is possible to find a set of tests to detect all faults and to locate *most* of the faults to within a reasonable number of packages.[13] In this section, we shall present the fault-detection tests for ULM's and a diagnostic procedure locating *all* the faulty ULM's in the modular realization of a ULC realizing a given logic function.

Fault-detection tests for ULM's

For a ULM built in an integrated circuit package, a defective unit usually means that a set of gates and possibly several connecting wires are burnt or broken. If we make no assumptions about the types of the faults in ULM's – whether they are due to single- or multiple-component failures, open- or short-circuit wires or gates, etc.—it is obvious that the fault-detection tests for ULM's must exhaust all possible combinations of the input terminals. Hence, for a ULM with p input terminals, where p is given by (3), it requires 2^p tests to detect all possible faults in a ULM. It is seen that a ULM-3 requires 64 tests* and a ULM-4 requires 2048 tests.

A diagnostic procedure to locate all the faulty ULM's in a ULC

Consider a ULC of n variables made of ULM-k's. Because a set of tests corresponding to all possible combinations of the n input variables will definitely detect all the faults in a logic circuit realizing a specific function, we need at most 2^n tests for detecting faults in the ULC realizing a given logic function.† If there are no restrictions on the type of possible faults in the ULM-k's, the 2^n tests also form the minimum test set.

Let a test applied to a ULC of n variables be represented by the n-tuple $(b_1, b_2, \ldots, b_n)$ of binary components, where b_i is the value of the input variable x_i, $i = 1, 2, \ldots, n$, employed in the test. Let T_i and T_i' be the tests $(b_1, b_2, \ldots, b_{n-1}, 0)$ and $(b_1, b_2, \ldots, b_{n-1}, 1)$ respectively, where $(b_1, b_2, \ldots, b_{n-1})$ has the decimal representation i. It follows from the tree structure of the modular realization of a ULC that if there are no faults in the ULC, the tests T_i and T_i' will make the output terminal F *logically connected* to the front terminal P_i in the last level of the tree structure, where P_i is connected to the residue function $f(b_1, b_2, \ldots, b_{n-1}, x_n)$. Hence, the output terminal F and the terminal P_i should have the same value under the tests T_i and T_i'. When F and P_i have different values under test T_i or T_i' or both, the terminal P_i is said to *have a faulty test*. Furthermore, when we say that *apply tests to terminal* P_i, it implies that apply tests T_i and T_i' to the ULC. Because of the tree structure of the ULC, it is obvious that there is one and only one path from ohe output terminal F to each terminal P_i, and the path contains one and only one ULM-k in each level of the ULC. If a ULM-k M_r (of any level) is the paths terminating at the terminals $P_i, P_{i+1}, \ldots, P_{i+d}$, we shall say that M_r *covers* the terminals $P_i, P_{i+1}, \ldots, P_{i+d}$.

Now, the diagnostic procedure to locate all the faulty ULM-k's in a ULC of n variables can be summarized as follows:

1) Apply tests to terminals P_i, $i = 0, 1, \ldots, 2^{n-1}-1$. If there exists no terminals with faulty tests, go to Step 4);otherwise, go to the next step.

2) Start from $\delta = 1$. Let L_δ be the set of all the ULM-k's in the δth level in which each ULM-k covers at least one faulty terminal. Apply the fault-detection tests to each of the ULM-k's in L_δ. If all the ULM-k's in L_δ are good, go to the next step. Otherwise, replace each faulty ULM-k in L_δ and apply test to *all* terminals P_i's covered by each replaced ULM-k. Record the terminals P_i's with the new faulty tests and those terminals with previous faulty tests not covered by the replaced ULM-k's, and go to the next step.

3) Increase δ by 1, and go to Step 2) when δ is smaller than the number of levels of the tree structure. Otherwise, the ULM-k's (in the last level) covering at least one terminal with a faulty test are faulty and should be replaced, and go to Step 4).

4) All the ULM-k's in the ULC are good for realizing the logic function under consideration.

To demonstrate this diagnostic procedure, let us consider the ULC shown in Fig. 10. We first apply tests to all P_i's. Assume that terminals P_0, P_4, P_5, P_6, P_7, P_{31}, P_{56}, and P_{57}, be the terminals with faulty tests after replacing M_{17}. Hence, we know that P_1, P_8, and assume that we find M_{21} is good. Then, we have to apply fault-detection tests to M_{17}, M_{18} and M_{20} since each of these ULM-3's covers at least one terminal with a faulty test. Suppose we find that M_{17} is faulty and that M_{18} and M_{20} are good. Then, replace

*If the faults are restricted to "stuck-at-1" and "stuck-at-0" types, and if we assume that only a single fault can occur at a time in a ULM[9,11], it can be shown that the set of minimum tests for a ULM-3 consists of only 8 tests.

†It is noted that passing the 2^n tests only guarantees that the ULC will realize the logic function under consideration, but does not guarantee that the ULC will realize *any* logic function of n variables correctly. Similar to the fault-detection tests for a ULM, the set of tests required to ensure a ULC realizing any logic function of n variables correctly will have 2^q tests, where q is the number of input terminals of the ULC. However, if we assume that there is only a single faulty module in a ULC at a time, then the minimum number of tests required to ensure the ULC realizing any logic function of n variables correctly is 2^{n+3}.

M_{17} and apply tests to terminals $P_0,P_1,...,P_{15}$. Let P_1, P_8, P_{10} and P_{11} be the terminals with faulty tests after replacing M_{17}. Hence, we know that P_1, P_8, $P_{10},P_{11},P_{31},P_{56},P_{57}$ and P_{63} are all the terminals with faulty tests. Since we reach the last level, we know M_1,M_3,M_8,M_{15} and M_{16} are faulty and should be replaced. This terminates the procedure.

Improving the reliability of the modular realization of a ULC by an error-correcting code

The reliability of the modular realization of a ULC can be improved by adding redundant ULM's using an error-correcting code. In this section, we shall demonstrate how to apply Hamming single-error-correcting code to increase the reliability of the modular realization of a ULC of n variables. The circuit is shown in Fig. 13 and the following notations are employed.

$$\begin{aligned} f &= f(x_1,x_2,x_3,...,x_n) \\ f_0 &= f(0,0,x_3,...,x_n) \\ f_1 &= f(0,1,x_3,...,x_n) \\ f_2 &= f(1,0,x_3,...;x_n) \\ f_3 &= f(1,1,x_3,...,x_n) \end{aligned} \tag{9}$$

The four blocks B_0,B_1,B_2 and B_3 are the modular realizations of the ULC's of n-2 variables and have the outputs f_0,f_1,f_2 and f_3 respectively. The Hamming single-error-correcting code with 4 information symbols is used, and its parity-check matrix H and generator matrix G are given by

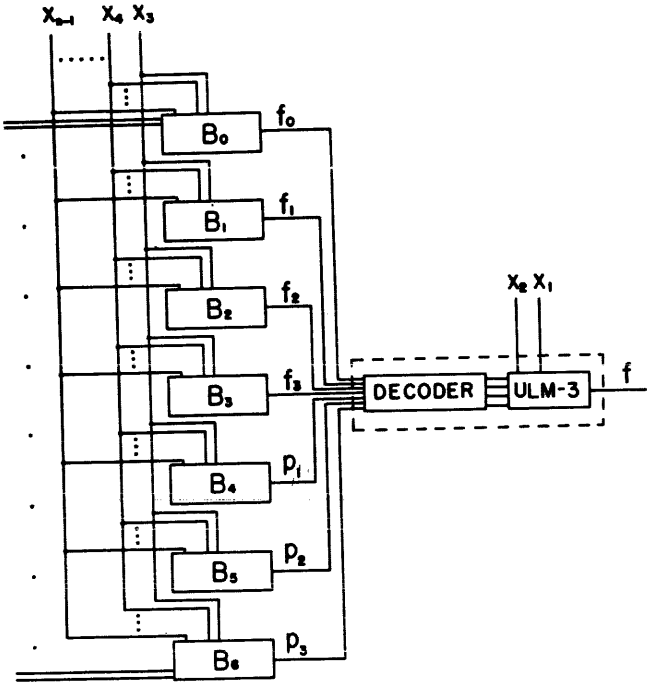


Figure 13—A modular realization of a ULC of n variables using Hamming single-error-correcting code

$$H = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} \tag{10}$$

$$\begin{matrix} p_1 & p_2 & f_0 & p_3 & f_1 & f_2 & f_3 \end{matrix}$$

$$G = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \tag{11}$$

The 4 information symbols to be encoded are f_0,f_1,f_2 and f_3, which are placed in the 3rd, 5th, 6th and 7th positions of the 7-bit code word respectively, while the remaining three positions are the parity check symbols p_1,p_2,p_3 as shown in (11). It follows from (10) and (11) that the parity-check symbols p_1,p_2 and p_3 can be expressed in terms of f_0,f_1,f_2 and f_3 as follows:

$$\begin{aligned} p_1 &= \bar{f}_0\bar{f}_1\bar{f}_3 + \bar{f}_0\bar{f}_1f_3 + f_0f_1f_3 + f_0\bar{f}_1\bar{f}_3 \\ p_2 &= \bar{f}_0\bar{f}_2f_3 + \bar{f}_0f_2\bar{f}_3 + f_0f_2f_3 + f_0\bar{f}_2\bar{f}_3 \\ p_3 &= \bar{f}_1\bar{f}_2f_3 + \bar{f}_1f_2\bar{f}_3 + f_1f_2f_3 + f_1\bar{f}_2\bar{f}_3 \end{aligned} \tag{12}$$

Since f_0,f_1,f_2 and f_3 are functions of the n-2 variables $x_3,...,x_n$, p_1,p_2 and p_3 are also functions of the same n-2 variables $x_3,...,x_n$. Thus, each of p_1,p_2,p_3 can be realized by using the modular realization of an ULC of n-2 variables. The three ULC's of n-2 variables for p_1,p_2 and p_3 are represented by the blocks B_4,B_5 and B_6 as shown in Fig. 13. The 7 signals f_0,f_1,f_2,f_3, p_1,p_2,p_3 are then fed to a decoder followed by a ULM-3 which produces the final output $f(x_1,\ldots,x_n)$. The decoder and the ULM-3 connected to the output terminal have to be of high reliability. It is found that the decoder will have 7 exclusive-OR gates, 3 INVERTER's and 4 AND gates. The decoder can be implemented together with the ULM-3 in a single reliable package. Let a block containing faulty ULM's be called a *faulty block*. It is seen that such a ULC of n variables will give correct output if there is only one faulty block. It is also noted that the ULC will still give correct output for the case that there are more than one faulty block, provided that only one erroneous block signal will show up at a time (under any input combination). If there exists a faulty block in the ULC, the easiest way to detect this faulty block is to add three output terminals to the decoder showing the syndrome of the code words. The faulty block can be located automatically

by simply reading the syndrome when the first fault occurs during the use of the ULC, and no separate test is required. The increase of cost for implementing this scheme is that for any $n > 3$, we have to add 75% redundant ULC's of n-2 variables and one highly reliable decoder-ULM-3 package. It is notes that the method illustrated above can easily be extended to the use of Hamming code with more than four information bits. Furthermore, the error-correcting code that can be used for increasing the reliability of the ULC is not restricted to the Hamming code, and the number of errors that can be corrected is not restricted to a single one.

DISCUSSION

In this paper, we have presented simple design techniques for ULC's, which are especially suitable to the use of integrated circuit packages for implementation. Various effects, such as the number of pins, the number of logic gates and the number of logic levels in a package, on the design of ULC's have been considered. For the ULC's obtained by the modular realization method, a diagnostic procedure for locating all the faulty ULM's in a faulty ULC has been established. Furthermore, a method for improving the reliability of a ULC using error-correcting codes has been demonstrated.

It is noted that an important practical advantage of using a ULC to realize a given logic function is that we need not find the minimal sum or minimal product of the logic function, which, however, is required for conventional realization methods. The only simplification process necessary to be applied to the logic function is to detect whether it can be written in a form which involves fewer variables. This result is used to determine a ULC of the smallest number of variables for realizing the given logic function.

It should be pointed out that the ULC's considered in this paper are restricted to realizing any single logic function. A natural extension of this study is to consider the design of a multiple-output ULC for realizing any set of m logic function. One way to obtain such a multiple-output ULC is simply to connect the m ULC's, each of which realizes one of the m logic functions, in the form of sharing the common input-variable terminals. It is quite unlikely that a multiple-output ULC with fewer I/O terminals can be obtained, because in general there are no fixed relations among the m logic functions to be realized.

ACKNOWLEDGMENT

The work reported here was supported in part by the U. S. Office of Scientific Research under Grant No. AF-AFOSR-1292-67.

REFERENCES

1 D C FORSLUND R WAXMAN
The universal logic block (ULB) and its application to logic design
Conference Record of 1966 Seventh Annual Symposium on Switching and Automata Theory IEEE Publication 16C40 pp 236-250

2 J T ELLISON B KOLMAN A P SCHIAVO
Universal function modules
UNIVAC Tech Rept Contract No AF19(628)-6012 (DDC AD-655395) April 1967

3 B ELSPAS et al
Properties of cellular arrays for logic and storage
Stanford Research Institute Scientific Report 3 Contract No AF-19-628-5828 (DDC AD-658832) pp 59-83 June 1967

4 L HELLERMAN
A catalogue of three-variables OR-INVERT and AND-INVERT logical circuit
IEEE Transaction on Electronic Computers vol 12 pp 198-223 1963

5 R B HURLEY
Transistor logic circuits
New York Wiley 1961

6 R A SMITH
Minimal three-variable NOR and NAND logic circuits
IEEE Transactions on Electronic Computers Vol EC-14 No 1 pp 79-81 February 1965

7 W FRANK KING III
The synthesis of multipurpose logic devices
Conference Record of 1966 Seventh Annual Symposium on Switching and Automata Theory IEEE Publication 1640 pp 227-235

8 J D BRULE R A JOHNSON E KLETSKY
Diagnosis of equipment failures
IRE Transactions on Reliability and Quality Control vol ol RQC-9 pp 23-34 1960

9 J M GALEY R E NORBY J P ROTH
Techniques for the diagnosis of switching circuit failures
IEEE Transactions on Comm and Elect Vol 83 No 74 pp 509-514 1964

10 H Y CHANG
An algorithm for selecting an optimum set of diagnostic tests
IEEE Transactions on Electronic Computers Vol EC-14 No 5 pp 705-711 1965

11 D B ARMSTRONG
On finding a nearly minimal set of fault detection tests for combinational logic nets
IEEE Transactions on Electronic Computers vol EC-14 no 1 pp 66-73 1966

12 W H KAUTZ
Fault diagnosis in combinational digital circuit
First Annual IEEE Computer Conference Digest IEEE Publication 16C51 pp 2-5 1967

13 *Logic partitioning in LSI*
Panel Discussion L M Spandorfer (moderator) IEEE Computer Group News vol no 6 p 16 May 1967

Sorting networks and their applications

by K. E. BATCHER
Goodyear Aerospace Corporation
Akron, Ohio

INTRODUCTION

To achieve high throughput rates today's computers perform several operations simultaneously. Not only are I/O operations performed concurrently with computing, but also, in multiprocessors, several computing operations are done concurrently. A major problem in the design of such a computing system is the connecting together of the various parts of the system (the I/O devices, memories, processing units, etc.) in such a way that all the required data transfers can be accommodated. One common scheme is a high-speed bus which is time-shared by the various parts; speed of available hardware limits this scheme. Another scheme is a cross-bar switch or matrix; limiting factors here are the amount of hardware (an m X n matrix requires m X n cross-points) and the fan-in and fan-out of the hardware.

This paper describes networks that have a fast sorting or ordering capability (sorting networks or sorting memories). In $(\frac{1}{2})p(p+1)$ steps 2^p words can be ordered. A sorting network can be used as a multiple-input, multiple-output switching network. It has the advantages over a normal crossbar of requiring less hardware (an n-input n-output switching network can be built with approximately $(\frac{1}{4})\, n(\log_2 n)^2$ elements versus n^2 in a normal crossbar) and of having a constant fan-in and fan-out requirement on its elements. Thus, a sorting network should be useful as a flexible means of tieing together the various parts of a large-scale computing system. Thousands of input and output lines can be accommodated with a reasonable amount of hardware.

Other applications of sorting memories are as a switching network with buffering, a multiaccess memory, a multiaccess content-addressable memory and as a multiprocessor. Of course, the networks also may be used just for sorting and merging.

Comparison elements

The basic element of sorting networks is the comparison element (Figure 1). It receives two numbers over its inputs, A and B, and presents their minimum on its L output and their maximum on its H output.

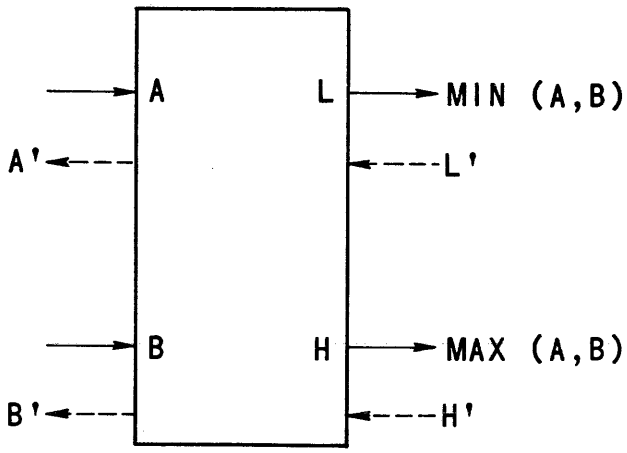


Figure 1 – Symbol for a comparison element

If the numbers in and out of the element are transmitted serially most-significant bit first the element has the state diagram of Figure 2. A reset input places the element in the A = B state and as long as the A and B bits agree it remains in this state with its outputs equal to its inputs. When the A and B bits disagree the element goes to the A < B or the A > B state and remains there until the next reset input. In the A > B state the output H equals the input A and the output L equals the input B. In the A < B state the opposite situation occurs.

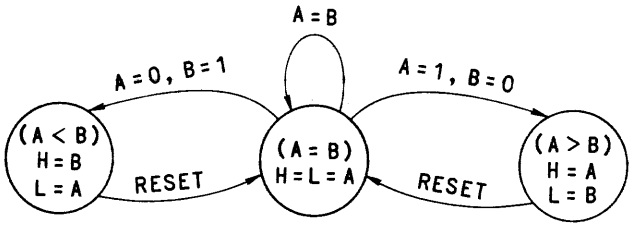


Figure 2 – State diagram for a serial comparison element (most-significant-bit first)

A serial comparison element can be implemented with 13 NORS and can be put on one integrated-circuit chip. When used in sorting networks each H and L output will feed an A or B input of another element so the fan-out is constant regardless of network size; this fact could be used to simplify the design of the chip. With several of the currently available logic families speeds of 100 nanoseconds/bit with a propagation delay from inputs to outputs of 40 nanoseconds are easily achieved.

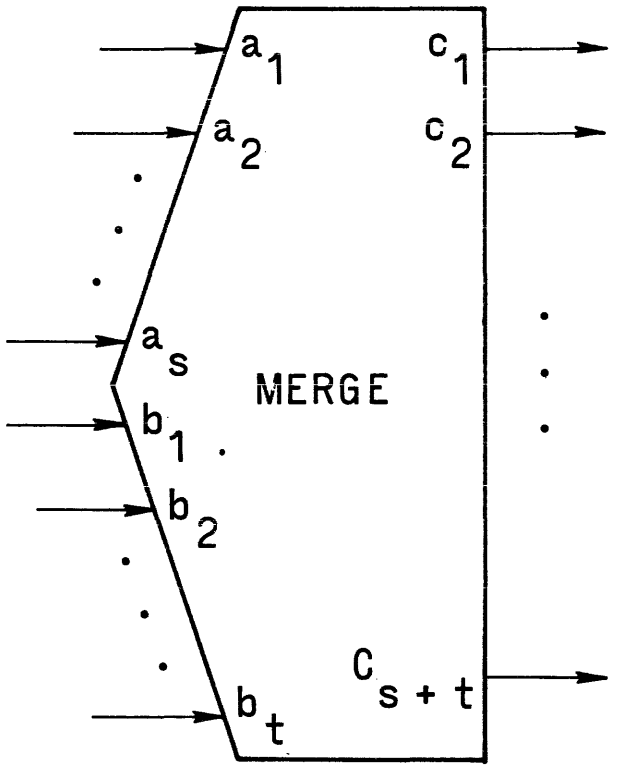
Faster operation can be attained by treating several bits in parallel in each step with more complex comparison elements.

Some of the applications described below will require "bi-directional" comparison elements. Besides the A and B inputs and the H and L outputs there are H′ and L′ inputs and A′ and B′ outputs (see Figure 1). If A > B then B′ = L′ and A′ = H′, if A < B then B′ = H′ and A′ = L′, otherwise A′ and B′ are left undefined. Information flows from left-to-right over the solid lines and from right-to-left over the dotted lines.

Odd-even merging networks

Merging is the process of arranging two ascendingly-ordered list of numbers into one ascendingly-ordered list. Figure 3 shows a symbol for an "s by t" merging network in which the s numbers of one ascendingly-ordered list, $a_1, a_2, \ldots, a_s$ are presented over s inputs simultaneously with the t numbers of another ascendingly-ordered list, $b_1, b_2, \ldots, b_t$ over another t inputs. The s + t outputs of the merging network present the s+t numbers of the merged lists in ascending order, $c_1, c_2, \ldots, c_{s+1}$.

A "1 by 1" merging network is simply one comparison element. Larger networks can be built by using the iterative rule shown in Figure 4. An "s by t" merging network can be built by presenting the odd-indexed numbers of the two input lists to one small merging network (the odd merge), presenting the even-indexed numbers to another small merging network (the even merge) and then comparing the outputs of these small merges with a row of comparison elements.[1] The lowest output of the odd merge is left alone and becomes the lowest number of the final list. The i^{th} output of the even merge is compared with the $i + 1^{th}$ output of the odd merge to form the $2i^{th}$ and $2i + 1^{th}$ numbers of the final list for all applicable i's. This may or may not exhaust all the outputs of the odd and even merges; if an output remains in the odd or even merge it is left alone and becomes the highest number in the final list.



$$a_1 \leq a_2 \leq \cdots \leq a_s$$

$$b_1 \leq b_2 \leq \cdots \leq b_t$$

$$c_1 \leq c_2 \leq \cdots \leq c_{s+t}$$

Figure 3 – Symbol for an "s by t" merging network

Appendix A sketches the proof of this iterative rule. Figure 5 shows a "2 by 2" and a "4 by 4" merging network constructed by this rule.

A "2^p by 2^p" merging network constructed by this rule uses $p.2^p+1$ comparison elements. The longest path goes through p+1 comparison elements and the shortest path through one element. Doubling the size of a merge only increases the longest path by unity so the merging time increases slowly with the size of the network.

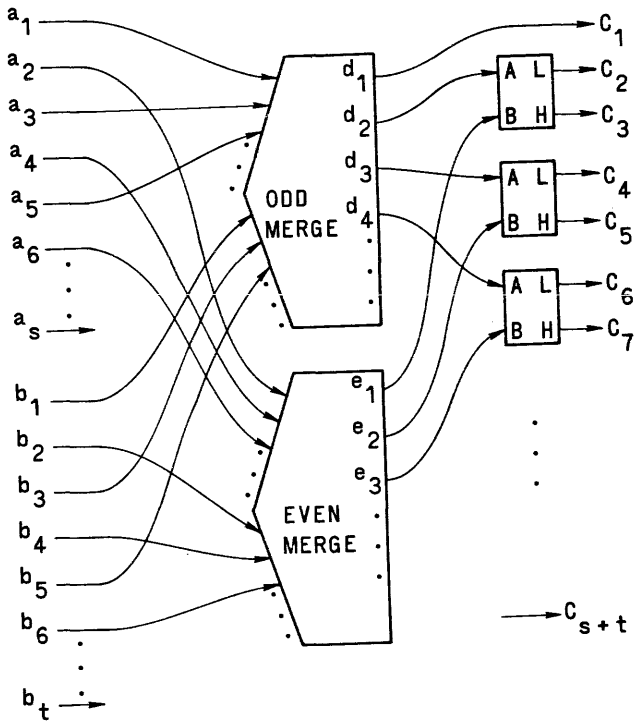


Figure 4 – Iterative rule for odd-even merging networks

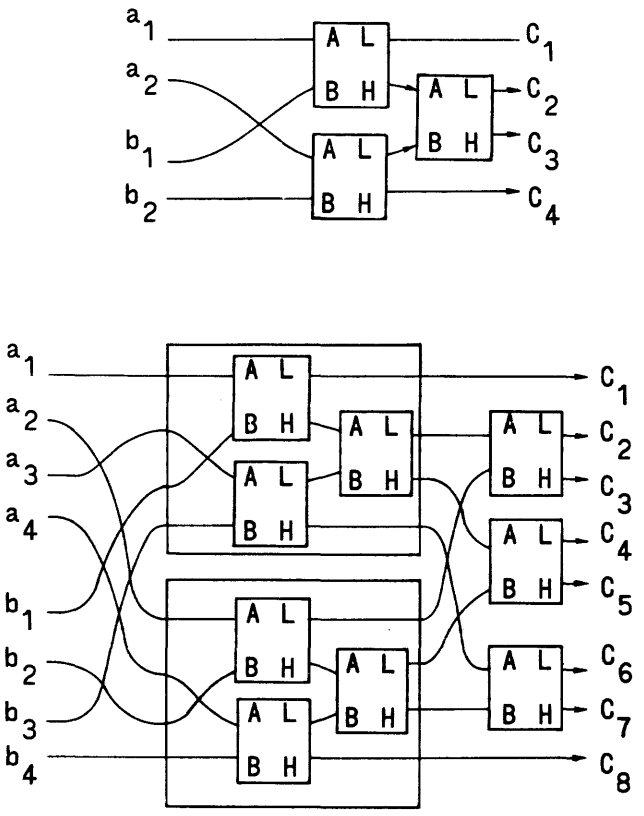


Figure 5 – Construction of "2 by 2" and "4 by 4" odd-even merging networks

Bitonic sorters

Another way of constructing merging networks from comparison elements is presented here. While requiring somewhat more elements than the odd-even merging networks, they have the advantage of flexibility (one network can accommodate input lists of various lengths) and of modularity (a large network can be split up into several identical modules).[2]

We will call a sequence of numbers *bitonic* if it is the juxtaposition of two monotonic sequences, one ascending, the other descending. We also say it remains bitonic if it is split anywhere and the two parts interchanged. Since any two monotonic sequences can be put together to form a bitonic sequence a network which rearranges a bitonic sequence into monotonic order (a bitonic sorter) can be used as a merging network.

Appendix B shows that if a sequence of 2n numbers, $a_1, a_2, \ldots, a_{2n}$ is bitonic and if we form the two n-number sequences:

$$\min(a_1, a_{n+1}), \min(a_2, a_{n+2}), \ldots, \min(a_n, a_{2n}) \quad (1)$$

and

$$\max(a_1, a_{n+1}), \max(a_2, a_{n+2}), \ldots, \max(a_n, a_{2n}), \quad (2)$$

that each of these sequences is bitonic and no number of (1) is greater than any number of (2).

This fact gives us the iterative rule illustrated in Figure 6. A bitonic sorter for 2n numbers can be constructed from n comparison elements and two bitonic sorters for n numbers. The comparison elements form the sequences (1) and (2) and since each is bitonic they are sorted by the two n-number bitonic sorters. Since no number of (1) is greater than any number of (2) the output of one bitonic sorter is the lower half of the sort and the output of the other is the upper half.

A bitonic sorter for 2 numbers is simply a comparison element and using the iterative rule bitonic sorters for 2^p numbers can be constructed for any p. Figure 7 shows bitonic sorters for 4 numbers and 8 numbers.* A 2^p-number bitonic sorter requires p levels of 2^{p-1} elements each for a total of $p \cdot 2^{p-1}$ elements. It can act as a merging network for any two input lists whose total length equals 2^p.

Large bitonic sorters can be constructed from a number of smaller bitonic sorters; for instance, a 16-number bitonic sorter can be constructed from eight 4-number bitonic sorters, as shown in Fig. 8. This allows large networks to be built of standard modules of convenient size.

*-Readers may recognize the similarity between the topologies of the bitonic sorter and the fast-fourier-transform.

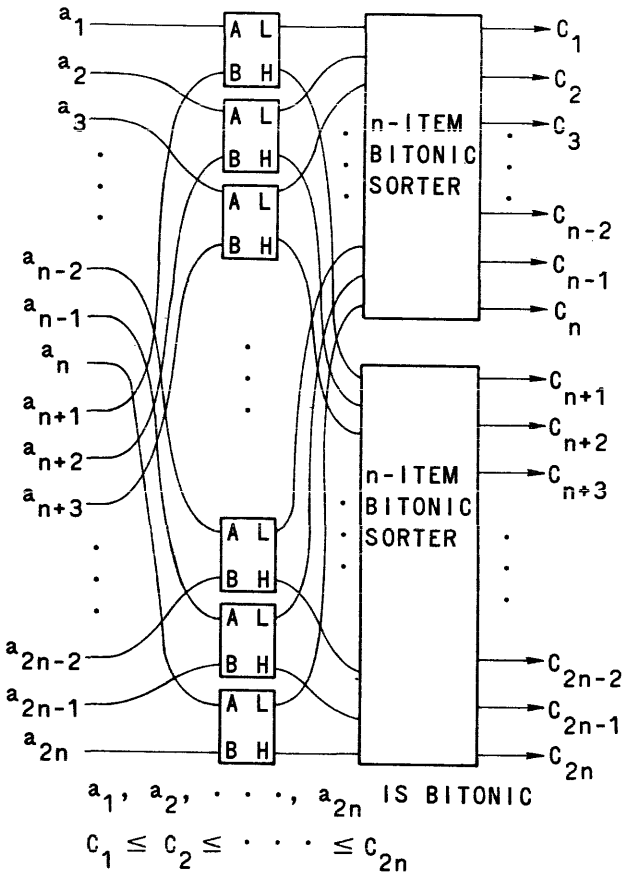


Figure 6 – Iterative rule for bitonic sorters

Sorting networks

A sorter for arbitrary sequences can be constructed from odd-even merges or bitonic sorters using the well-known sorting-by-merging scheme: The numbers are combined two at a time to form ordered lists of length two; these lists are merged two at a time to form ordered lists of length four, etc. until all numbers are merged into one ordered list.

To sort 2^p numbers using odd-even merges requires 2^{p-1} comparison elements followed by 2^{p-2} "2-by-2" merging networks followed by 2^{p-3} "4-by-4" merging networks, etc., etc. The longest path will go through $(\frac{1}{2})p\,(p+1)$ elements and the shortest path through p elements. The network requires $(p^2-p+4)2^{p-2}-1$ comparison elements.

To sort 2^p numbers using bitonic sorters requires $(\frac{1}{2})p(p+1)$ levels each with 2^{p-1} elements for (p^2+p) 2^{p-2} elements. Each path goes through $(\frac{1}{2})p(p+1)$ levels.

A sorter for 1024 numbers will have 55 levels and 24,063 elements with odd-even merges or 28,160 elements with bitonic sorters. With a 40 nanosecond propagation delay per level the total delay is 2.2 microseconds. Serial transmission of the bits would require about this much time between successive bits of the numbers unless re-clocking occurs within the

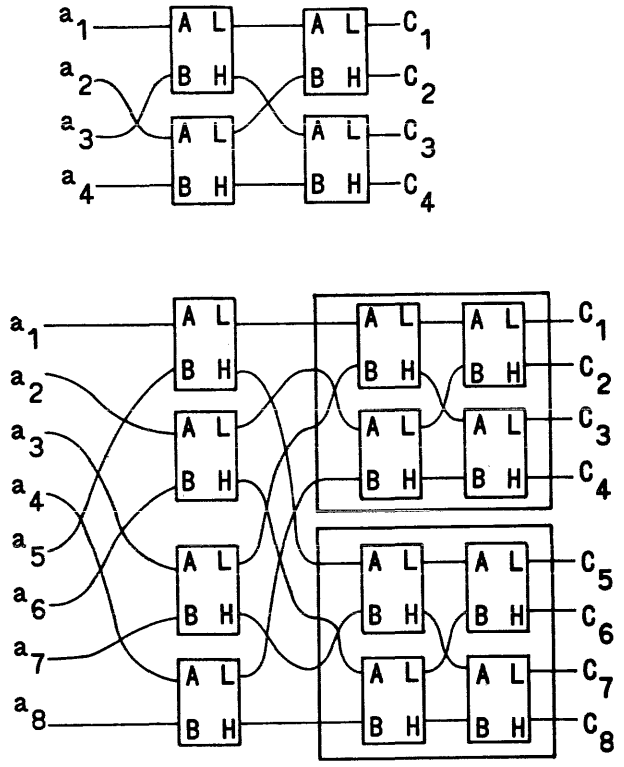


Figure 7 – Construction of bitonic sorters for 4 numbers and for 8 numbers

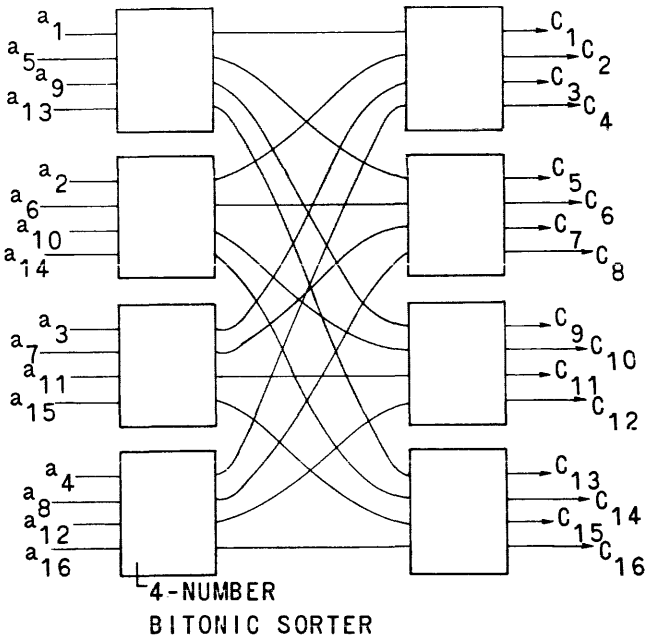


Figure 8 – A 16-number bitonic sorter constructed from eight 4-number bitonic sorters

network. Parallel-input-parallel-output registers of 1024 bits each can be placed between certain levels to perform this task or the re-clocking may be incorporated within each comparison element with a pair of flip-flops on the outputs. The latter scheme does not add to the terminal count of the comparison element so the cost of the added flip-flops on the comparison element chip is small. One can use any of the familiar techniques for driving shift registers such as the "A-B" technique where successive levels are clocked out-of-phase with each other. With present circuit and wiring techniques a bit rate of 10 megahertz may be possible with 50 nanosecond delay per level (2.75 microsecond delay from input to output of a 1024-word sorter).

With re-clocking in the elements and odd-even merges extra elements are needed to balance the unequal-length paths. Bitonic sorters do not have this problem.

Applications

The fast sorting capability of these networks allows their use in solving other problems where large sets of data must be manipulated. Some of these applications are sketched below.

Switching network

A sorting network can connect its input lines to its output lines with any permutation. The connection is made by numbering the output lines in order and presenting the desired output address for each input line at the input. The sorting network sorts the addresses and in the process makes a connection from each input line to its desired output line for the transmission of data. Bi-directional paths will be obtained if bi-directional comparison elements are used.

An alternative permuting network has been shown in the recent literature[3] which has less elements $[(p-1)2^p+1$ versus $(p^2-p+4)2^{p-2}-1$ for permuting 2^p items] but a more complex set-up algorithm.

Switching network with conflict resolution

The aforementioned switching network assumes each input wants a unique output line. In many applications conflicts between inputs occur and must be resolved by inhibiting conflicting inputs. Figure 9 sketches an m-input, n-output network that performs this task. Each input line inserts a word containing the output address desired (or zeroes if the line is inactive), a control bit equal to 1 and a priority number into an m-item sorting network with bi-directional elements. This orders the items so input items with the same output address are grouped together and ordered by their priority number. The ordered set of m-input items is merged with a set of n items, each containing a fixed output address and a control bit equal to O.

At the right side of the m by n merge the m + n items are in one ordered list; each address-inserter item will be directly below any input items with the same address. The adjacent word transfer network, looking at the control bits, connects each address-inserter item to the input item directly above it if one exists (the input item with lowest priority number is picked in each case). The elements in the sort and the merge are bi-directional so two-way paths are formed from input to output. The adjacent word transfer sends back signals over each path to signal each input and output line whether or not a connection has been established. Data can then be transmitted over each of the connected input lines.

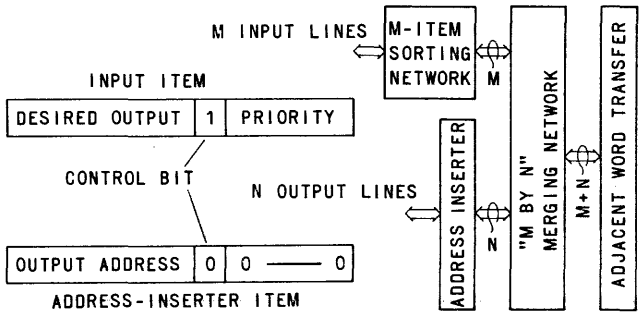


Figure 9—An m-input, n-output switching network with conflict resolution

Multi-access memory

Re-clocking delays in the comparison elements give a sorting network some storage capability which can be augmented if needed with shift registers on the outputs. When the output lines are fed back to the input lines a recirculating self-sorting store is created (Figure 10). In each recirculation cycle word positions are changed to keep the memory in order.

Inputs to the memory can be made by breaking the recirculation paths of some words and inserting new words. To prevent destroying old information during input we use the convention that words with all bits equal to "one" are "empty" and contain no information; these will automatically collect at the "high-end" of memory where input lines can use them to insert new words.

Outputs from the memory can be accommodated by reserving the most-significant-bit (MSB) of each word; "1" for normal words and "O" for words to be outputted. Words for output will automatically collect at the "low end" of memory where output lines can read them. Selection of which words to output is accommodated by reserving the least-significant-bit (LSB) of each word; "1" for normal words and "O"

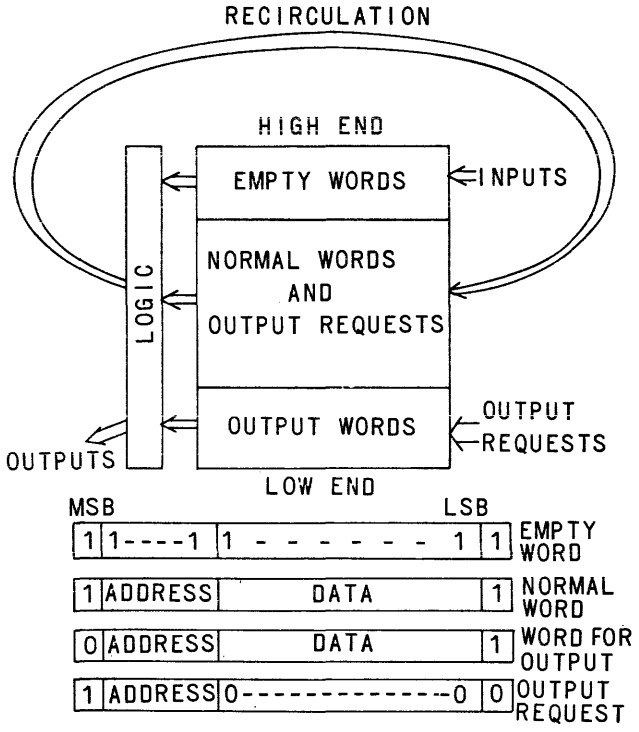


Figure 10 – A multi-access memory

for "output requests". Logic between adjacent words causes an output request to affect the word directly above it.

During one recirculation cycle new words and output requests are entered into memory. During the next recirculation cycle all words are recirculated with no new entries. At the end of the cycle the LSB of each word will precede the MSB of the same word (no reordering occurs in the second cycle). Output requests are identified by a "O" in the LSB and for each request logic performs the following action: if the word above the request is a normal word ("1" in the LSB) change its MSB to a "O" and empty the request (change all its bits to "1" as they fly by), if the word above the request is another request change the MSB of the first request to "O". During the following recirculation cycle the selected words and unfulfilled requests flow to the low end of memory and are read by output lines. Because the request itself is outputted if no word is found, as many outputs as original requests occur. If the original requests were in order the outputs directly correspond to them (a second sorting network can put the original output requests in order).

In use the more-significant part of each word is used as an address and the rest as data. To request a certain address an output request is sent in with that address and zeros for data. The word returned will be at that address or a higher address if the requested address is empty.

While a complete cycle may be long in this memory (50-bit words at 100 nanoseconds/bit = 5 microseconds/recirculation = 10 microseconds/complete cycle) many inputs and outputs can be accommodated in each cycle. An effective rate of 100 nanoseconds/word is achieved with 100 inputs and outputs.

Such a memory could be useful as the "common memory" of multiprocessors. The self-sorting capability could be useful for keeping "task lists" up to date and performing other housekeeping tasks.

Other uses may be as a message "store-and-forward" system and as a switching network with buffering capability. In these uses each output device is given a unique address which it continually interrogates; input devices send their data to these addresses.

Multi-access content addressable memory

By adding facilities for shifting the bits within the words in the aforementioned memory different fields of the words can be brought into the more-significant portions which govern the ordering of the words. Addressing can then take place on any part of the words. As long as the same field positions are being searched more than one search can be accommodated simultaneously.

Multi-processor

By adding processing logic to perform additions, subtractions, etc., on groups of adjacent words of a sorting memory one can implement a multi-processor. The sorting capability is used to transmit operands between processors. Merely by changing address fields the multiprocessor can be reconfigured quickly. Such a multi-processor can keep up with the "dynamic topology" of certain real-time problems.

To simplify the processing logic one might use the same network or another network to perform table look-up arithmetic. It is possible to have all the processors search the same tables simultaneously.

SUMMARY

Sorting networks capable of sorting thousands of items in the order of microseconds can be constructed with present-day hardware. Such fast sorting capability can be used to manipulate large sets of data quickly and solve some of the communications problems associated with large-scale computing systems.

Standard modules of convenient sizes can be picked and used in any size network to lower the cost. Large-scale integration can be applied if the problem of laying out the rather complex topology of the network can be solved. Studies of this problem are being conducted at Goodyear Aerospace.

APPENDIX A – SKETCH OF PROOF OF ITERATIVE RULE FOR ODD-EVEN MERGING

Let $a_1, a_2, a_3, \ldots$ and $b_1, b_2, b_3, \ldots$ be the two ordered input sequences. Let $c_1, c_2, c_3, \ldots$ be their ordered merge, $d_1, d_2, d_3, \ldots$ be the ordered merge of their odd-indexed terms and $e_1, e_2, e_3, \ldots$ be the ordered merge of their even-indexed terms.

For a given i let k of the $i+1$ terms in $d_1, d_2, d_3, \ldots, d_{i+1}$ come from $a_1, a_3, a_5, \ldots$ and $i+1-k$ come from $b_1, b_3, b_5, \ldots$ The term d_{i+1} is greater than or equal to k terms of $a_1, a_3, a_5, \ldots$ and therefore is greater than or equal to $2k-1$ terms of $a_1, a_2, a_3, \ldots$ Similarly it is greater than or equal to $2i+1-2k$ terms of b_1, b_2 $b_3, \ldots$ and hence $2i$ terms of $c_1, c_2, c_3, \ldots$ Therefore

$$d_{i+1} \geq c_{2i}. \tag{A1}$$

Similarly from consideration of the i terms of e_1, e_2 $e_3, \ldots, e_i$ the inequality

$$e_i \geq c_{2i} \tag{A2}$$

is obtained.

Now consider the $2i+1$ terms of $c_1, c_2, c_3, \ldots, c_{2i+1}$ and let k come from $a_1, a_2, a_3, \ldots$ and $2i+1-k$ come from $b_1, b_2, b_3, \ldots$ If k is even we have that c_{2i+1} is greater than or equal to:

k terms of $a_1, a_2, a_3, \ldots$
$(\frac{1}{2})$k terms of $a_1, a_3, a_5, \ldots$
$2i+1-k$ terms of $b_1, b_2, b_3, \ldots$
$i+1-(\frac{1}{2})$k terms of $b_1, b_3, b_5, \ldots$
$i+1$ terms of $d_1, d_2, d_3, \ldots$

and similarly c_{2i+1} is greater than or equal to i terms of $e_1, e_2, e_3, \ldots$
so

$$c_{2i+1} \geq d_{i+1} \tag{A3}$$

and

$$c_{2i+1} \geq e_i \tag{A4}$$

If k is odd, (A3) and (A4) still hold.

Since every item of $d_1, d_2, d_3, \ldots$ and $e_1, e_2, e_3, \ldots$ must appear somewhere in $c_1, c_2, c_3, \ldots$ and $c_1 \leq c_2 \leq c_3 \leq \ldots$ inequalities (A1), (A2), (A3) and (A4) imply that

$$c_{2i} = \min(d_{i+1}, e_i) \tag{A5}$$

and

$$c_{2i+1} = \max(d_{i+1}, e_i). \tag{A6}$$

APPENDIX B – SKETCH OF PROOF OF ITERATIVE RULE FOR BITONIC SORTERS

Let $a_1, a_2, a_3, \ldots, a_{2n}$ be bitonic. Let $d_i = \min(a_i, a_{n+i})$ and $e_i = \max(a_i, a_{n+i})$ for $1 \leq i \leq n$. We want to prove that $d_1, d_2, \ldots, d_n$ and $e_1, e_2, \ldots, e_n$ are each bitonic and

$$\max(d_1, d_2, \ldots, d_n) \leq \min(e_1, e_2, \ldots, e_n). \tag{A7}$$

If $a_1, a_2, a_3, \ldots, a_{2n}$ is split into two parts and the parts interchanged $d_1, d_2, \ldots, d_n$ and $e_1, e_2, \ldots, e_n$ undergo a similar interchange. This does not affect the bitonic property nor affect (A7) so it is sufficient to prove the proposition for the case where

$$a_1 \leq a_2 \leq a_3 \leq \ldots \leq a_{j-1} \leq a_j \geq a_{j+1} \geq \ldots \geq a_{2n} \tag{A8}$$

is true for some j $(1 \leq j \leq 2n)$.

Reversal of the terms of sequences does not affect the bitonic property nor maximums and minimums so it is sufficient to assume $n < j \leq 2n$.

If $a_n \leq a_{2n}$ then $a_i \leq a_{n+i}$ so $d_i = a_i$ and $e_i = a_{n+i}$ for $1 \leq i \leq n$ and the proposition holds.

If $a_n > a_{2n}$ then from $a_{j-n} \leq a_j$ we can find a k such that $j \leq k < 1n$, $a_{k-n} \leq a_k$ and $a_{k-n+1} > a_{k+1}$ (the sequence $a_j, a_{j+1}, a_{j+2}, \ldots, a_{2n}$ is decreasing while the sequence $a_{j-n}, a_{j+1-n}, a_{j+2-n}, \ldots, a_n$ is increasing). Then

$$\left.\begin{aligned} d_i &= a_i \\ e_i &= a_{i+n} \end{aligned}\right\} \text{ for } 1 \leq i \leq k-n \tag{A9}$$

and

$$\left.\begin{aligned} d_i &= a_{i+n} \\ e_i &= a_i \end{aligned}\right\} \text{ for } k-n < i \leq n. \tag{A10}$$

The inequalities

$$d_i \leq d_{i+1} \text{ for } 1 \leq i < k-n, \tag{A11}$$

$$d_i \geq d_{i+1} \text{ for } k-n < i < n, \tag{A12}$$

$$e_i \leq e_{i+1} \text{ for } k-n < i < n, \tag{A13}$$

$$e_n \leq e_1, \tag{A14}$$

$$ei \leq e_{i+1} \text{ for } 1 \leq i < j-n, \tag{A15}$$

and

$$e_i \geq e_{i+1} \text{ for } j-n \leq i < k-n \tag{A16}$$

can be shown which prove that $d_1, d_2, \ldots, d_n$ and $e_1, e_2, \ldots, e_n$ are bitonic and $\max(d_1, d_2, \ldots, d_n) = \max(a_{k-n}, a_{k+1}) \leq \min(a_k, a_{k-n+1}) = \min(e_1, e_2, \ldots, e_n)$.

ACKNOWLEDGMENTS

The help of D. L. Rohrbacher, P. A. Gilmore and others at Goodyear Aerospace is gratefully acknowledged.

Part of this work was supported by Rome Air Development Center under Contract AF30(602)-3550, F. Dion, Administrator.

REFERENCES

1 K E BATCHER
A new internal sorting method
Goodyear Aerospace Report GER-11759 1964

2 K E BATCHER
Bitonic sorting
Goodyear Aerospace Report GER-11869 1964

3 L J GOLDSTEIN S W LEIBHOLZ
On the synthesis of signal switching networks with transient blocking
IEEE Transactions EC-16 5 637-641 1967

The Sylvania data tablet: A new approach to graphic data input

by JAMES F. TEIXEIRA and ROY P. SALLEN
Sylvania Electronic Systems
Waltham, Massachusetts

INTRODUCTION

Advances in the computer field have been spectacular and profuse. Sophisticated software can lead multiple users through a time-sharing maze while hardware developments offer increasing speed and flexibility. Research into the man-machine interface, however, uncovered serious deficiencies in equipment available to input graphical data into a computer. This fact prompted work by H. Teager at MIT and others, and by the Rand Corporation which resulted in an ingenious device known as the Rand Tablet.[1] This equipment allows the user to enter two-dimensional information with a free moving electronic pen and flat, pad-like tablet. One simply prints alphanumeric data or sketches circuit diagrams, structural diagrams, or waveforms on the tablet. The computer is continuously supplied with X-Y pen coordinates and can therefore "see" what the operator has drawn.

The Rand Tablet is inherently a digital device. Coordinate information is supplied by a 1024 by 1024 gridwork of fine conductive lines which are digitally excited by time coded pulses. A high input impedance pen and associated circuitry sense the polarity and timing of the pulses immediately beneath the pen tip and generate the corresponding 10 binary bits of coordinate position for each axis.

Several years ago Sylvania looked into the possibility of applying *analog* techniques in this area in the hopes that high resolution might be attained along with certain other advantages. It was realized that accuracy would probably suffer in this approach but it could still be commensurate with that of most computer driven displays. Therefore, we set an initial goal of 12-bit resolution and one per cent accuracy in each axis. In addition, a coarse measure of pen height above the tablet surface was desired, to give the operator some Z axis capability. Other intended features included: 1) transparency, so that the tablet could be placed over a cathode ray tube and the unit used as a 'light pen'; 2) an ink capability in the pen to generate hard copy by placing a sheet of paper over the tablet; and 3) simultaneous analog voltages representing the coordinate positions to allow additional flexibility in interfacing with display systems.

The result of Sylvania's work in this field was the introduction in early 1966 of the Sylvania Data Tablet Model DT-1 (Fig. 1), a device which has the above-mentioned features. An electronic pen using a conventional ball-point pen cartridge, a writing panel measuring approximately 16″ × 20″ × ¾″, with an 11″ square writing aperture, and a 17″ × 18″ × 7″ electronic package comprise the complete Data Tablet unit. Ten-foot cables connect the writing panel and pen to the electronic package.

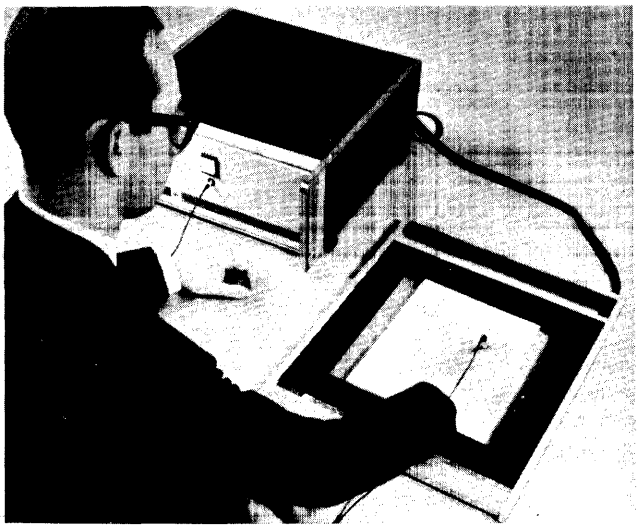


Figure 1 – The Sylvania data tablet

General method of operation

The writing panel consists of a thin, transparent conductive film bonded to a heavy glass base plate and overlayed by a thin glass protective sheet. An electric field is established on the film and the high input

impedance pen capacitively couples to it. Two-dimensional position information is obtained from the writing panel by *phase* measurements performed on the signals picked up by the pen.

A drive network excited by the electronics package drives the film at discrete points along its circumference in such a manner that a traveling wave (in a mathematical sense) is established parallel to each orthogonal axis. This wave has the property that its phase is a linear function of position as in the relationship:

$$V = K \sin(\omega t - \alpha X) \quad (1)$$

were α, ω are constants and X is the position coordinate. (It should be mentioned that a true propagating wave does not exist on the writing panel since a frequency in the hundreds of megahertz would be required to give a significant phase shift along an eleven inch path. Actually, only one kilohertz is used as the phase shifted frequency.)

The relationship in Eq. (1) applies only to the X axis, but a similar function exists for the Y axis. Confusion between the two in the electronics package is prevented by frequency multiplexing. Each axis is excited by a one kilohertz sinusoid presented as a suppressed carrier modulation on a higher frequency. The carrier frequencies are approximately 100 kHz and of course are different for the X and Y axes. Circuitry within the electronics package separates the pen signals into X and Y channels and extracts the phase information. The desired phase is observed in the envelope waveform of the signals while the axis information is contained in the carrier frequencies.

Z axis position is sensed by an amplitude sensitive circuit which compares the peak-to-peak signal from the pen with fixed threshold voltages. Since the pen signals fall off rapidly with height, this technique is very effective. In fact, the threshold setting corresponding to contact with the writing panel surface can easily be set within 5 or 10 thousandths of an inch, thereby eliminating the need for a mechanical contact switch in the pen.

Ideal excitation of the surface

Use of phase measurements to obtain position as in the Sylvania Data Tablet is apparently a fairly new concept or at least one which was not developed fully in the past. Its greatest advantage in this application (over other conductive film techniques dependent upon voltage gradient) is that capacitive coupling to the conductive film by the pen can be used even through dielectric layers such as a protective glass covering or sheets of paper. The latter item and the fact that the pen used in the Data Tablet contains a standard ball-point pen stylus offers the ability to produce a hard copy simultaneously with the data entry.

The underlying principles in the phase shift concept are best illustrated mathematically using an idealized model at first. Fig. 2 shows a square conductive sheet of finite, uniform resistivity. In the ideal case, the sheet is excited by distributed current sources as shown in the figure. Although X and Y axes are labeled, only phase variations along the X axis will be considered for the moment.

Ideally, the voltage on the sheet (for X axis phase shift) would have the form:

$$V(X,Y,t) = \cos(\omega t - \alpha X) \quad (2)$$

giving a linear phase shift along the X axis with no contribution from the Y axis. Since the sheet is a source-free region, however, this realtionship cannot be established as it does not satisfy Laplace's equation:

$$\nabla^2 V = 0 \quad (3)$$

throughout the sheet. However, the expression:

$$V(X,Y,t) = (\cosh \alpha Y) \cos(\omega t - \alpha X) \quad (4)$$

does satisfy Laplace's equation yet contains the desired phase component. The amplitude coefficient, cosh αY, is symmetrical about a horizontal line through the center of the sheet. It does not affect the phase shift and allows the use of identical current sources at the top and bottom edges of the sheet.

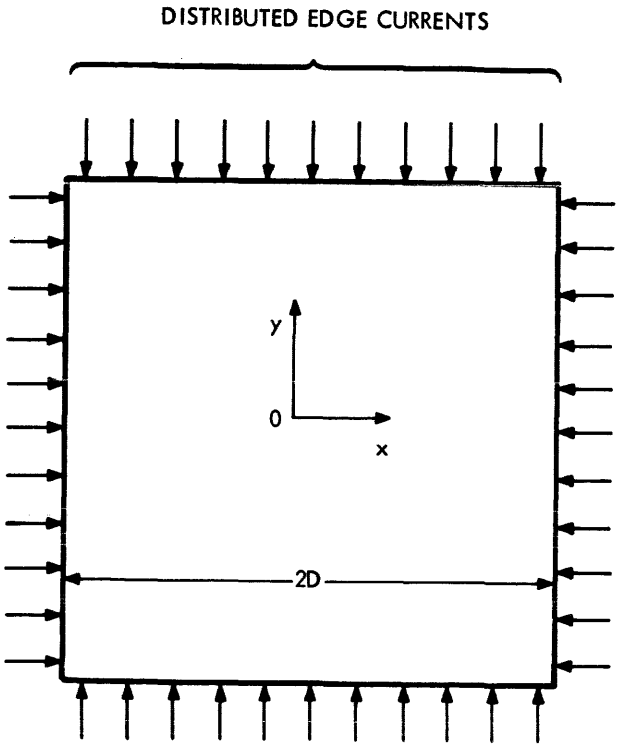


Figure 2 – Idealized model of edge driven conductive film

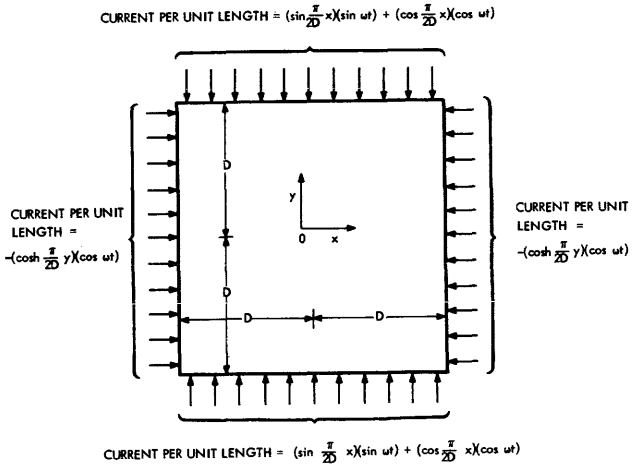


Figure 3 – Drive current distribution

A field of this type is established as illustrated in Fig. 3 for the case where $\alpha = \frac{\pi}{2D}$. Edge currents are injected as shown to satisfy boundary conditions. A vertical line through the center of the sheet has the value X = 0 and only cos ωt components of voltage exist along this line. The left and right edges have the values X = ±D and only sin (ωt) components of voltage are present. The currents applied to the left and right edges of the sheet are needed to satisfy boundary conditions imposed by the lack of surface continuity. The $(\sin \frac{\pi}{2D} X) \sin(\omega t)$ current sources require no left and right boundary correction since the sheet is terminated at points where the derivative of the sine (the amplitude function) is zero and no current attempts to flow across the boundary. This is not true for the cos (ωt) sources, since only "positive" cos(ωt) current is injected along the top and bottom edges and must be balanced by "negative" sources at left and right edges. Boundary correction currents are shown in Fig. 3 along the left and right edges.

Phase shift along the Y axis is produced by simply rotating the structure 90° and applying similar drive signals. As mentioned previously, frequency multiplexing is used to prevent interaction between the X and Y drives.

Practical excitation of the surface

In practice it is difficult to supply continuously distributed current along the circumference of a conductive film; instead discrete drive spots, evenly spaced around the film, are used to approximate this condition. The number of these has been selected to cause less than a one per cent error contribution within the usable portion of the tablet. Each spot is supplied current, through summing resistors, from up to four voltage sources – two for the X axis (sin $\omega_x t$ and cos $\omega_x t$) and two for the Y axis (sin $\omega_y t$ and cos $\omega_y t$). Spot A (Fig. 4), for example, has a current contribution from the X axis sine and cosine generators to set up the X field and a contribution from the Y axis cosine generator to satisfy the boundary conditions for that dimension. The corner spots require current from all four generators.

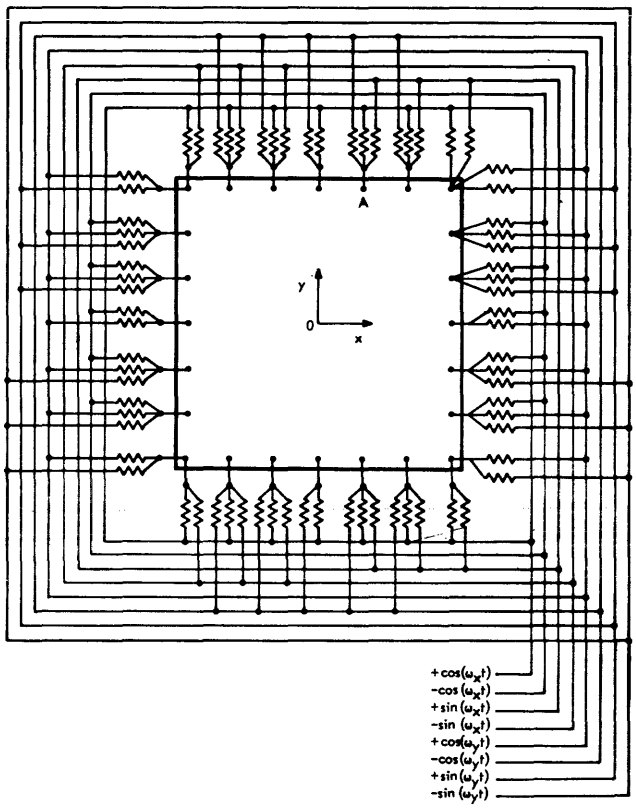


Figure 4 – Practical generation of edge currents

A film of high uniformity is difficult to realize and for this reason we have found it necessary to tailor the drive networks to the individual glass panels. Ideal current values are obtained only through individually computed combinations of resistors and generators. Much work has been done at Sylvania to determine the resistor network required for a particular writing panel. The first step is the selection of a sufficiently uniform sample of coated glass. It has been empirically determined that a film must be uniform within about ±5% to realize the desired ±1% phase linearity. A special in-house measuring instrument is employed in film evaluation.

In the second step the drive spots are attached and a technician temporarily hooks up a set of current sources to the spots. The currents are adjusted to the theoretical values. Voltages induced at each spot by the currents are measured and recorded on punched cards. Finally, the network is designed by our labora-

tory computer, which is programmed to calculate the value of each resistor required in the network and to select the closest standard unit from a table of 1% values.

A printed circuit board which surrounds the writing panel (Fig. 5) holds the resistors and provides the connections to the drive voltage generators and to the spot terminals. Since each generator has a positive and negative terminal, eight generator busses are needed, and to maintain required flexibility in the summing network design, seven resistor values are required for each spot. Twenty-four spots are placed around the circumference of the writing panel making the resistor total 168.

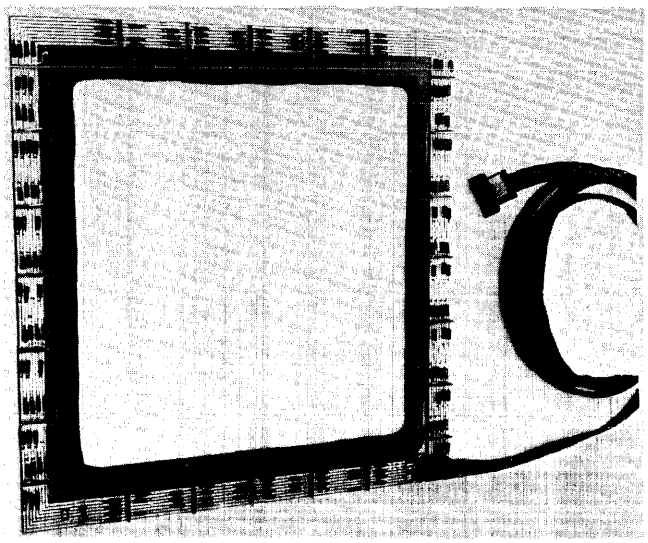


Figure 5 – Exposed tablet drive network

Choice of parameters

During the initial development work on the tablet, it was felt that the frequency of the phase varying signal should be low to ease the electronic circuitry requirements. Since the tablet signal varies from $-\frac{\pi}{2}$ to $\frac{\pi}{2}$ radians across the writing panel, a low operating frequency would exhibit a large time displacement, and since the digital conversion would be made on the basis of time measurements, lower clock frequencies and small fixed time errors could be accepted. Too low a frequency, on the other hand, would introduce filter problems and produce a low data rate. Therefore, a frequency of one kilohertz was chosen as a reasonable compromise.

It was also discovered during the initial development that appreciable phase errors at low operating frequencies could be introduced by surface contamination due to fingerprints on the writing panel or by touching paper to the pen tip. Calculation of the time constant of a sheet of paper when interposed between the conductive film and the very high input impedance of the pen showed a significant phase error at frequencies below 50 kilohertz. Fortunately, the high frequency carriers introduced for axis separation also preserve the phase integrity of the low frequency modulation under these conditions. By using suppressed-carrier modulation, the one kilohertz tablet signal for each axis is converted to two sidebands spaced one kilohertz on either side of the imaginary carrier frequency. Any extraneous phase shift which affects both sidebands equally (which is true of that caused by surface contamination and paper) will not affect the modulation phase. Normal operation of the tablet is not impeded and the amplitude envelopes of the suppressed-carrier signals contain the expected phase information.

Drive electronics

Fig. 6 is a block diagram of the drive electronics for the X direction. The Y axis circuits are similar except for a different (110 kHz) carrier oscillator frequency. The sinusoidal output of the 1 kHz oscillator can be phase shifted between 0° and 10° by the X-axis centering control to allow a precise alignment of the physical and electrical centers of the writing panel. This signal is then divided into two channels, one (sine) containing a gain control to equalize the output amplitudes, and the other (cosine) containing a 90° phase shifter to produce the quadrature component. Shunt modulators operating at 90 kHz produce the double sideband suppressed-carrier signals. The two double sideband signals are each split into two components, 180° out of phase, by the driver stage and are amplified in power and voltage by the four X-axis tablet drive amplifiers.

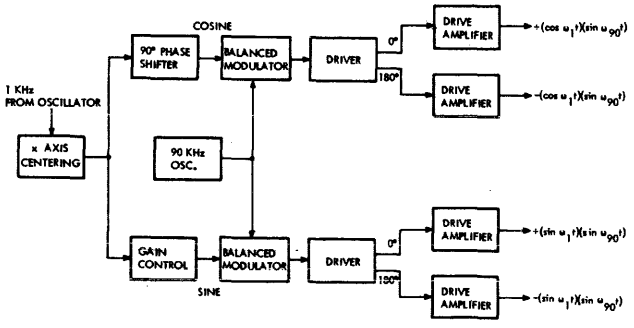


Figure 6 – Drive electronics block diagram

Pickoff electronics

A high input impedance electronic pen capacitively senses the tablet signals and transmits them via a

coaxial cable to a low noise amplifier. A unity gain amplifier (a field effect and bipolar transistor combination) in the pen buffers the input from the cable and supplies a guard voltage to capacitance reducing elements in the pen. Power to the amplifier is supplied through the cable signal path.

Fig. 7 is a block diagram of the X-axis pickoff electronics. The output of the low noise amplifier divides into two channels, one with a 90 kHz bandpass filter for the X-axis and the other with a 110 kHz filter for the Y-axis. These filters offer 25 db suppression of the orthogonal axis components and represent a starting point for axis separation. The output of the 90 kHz filter, composed primarily of two sidebands at 89 and 91 kHz, is demodulated by a detector operating synchronously with the 90 kHz oscillator used in the drive portion. A low pass filter following the demodulator contains wave traps at 20 kHz and 90 kHz to provide over 70 dB attenuation of the orthogonal axis demodulation components and the X-axis carrier. This filter has a relatively high cut-off frequency (7 kHz) in order to minimize its phase shift at 1 kHz. The filter output is a nearly pure 1 kHz sinusoid that is phase shifted by an amount proportional to the pen's position along the X axis. It is 90° out of phase with the reference signal at the center of the writing panel and varies from +10° to +170° between the left and right edges of the writing aperture. A limiter following the filter produces a square wave with transitions occurring at each 1 kHz zero crossing.

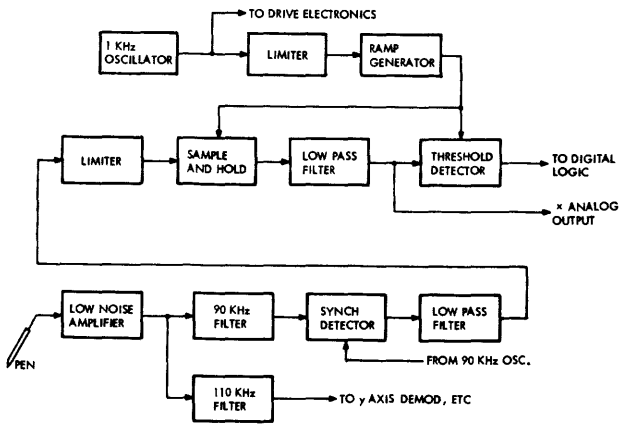


Figure 7 – Pickoff electronics block diagram

The upper portion of Fig. 7 shows a 1 kHz sinewave oscillator whose output is limited to provide a square wave as the phase reference for the digital and analog processing. For each half cycle of the reference square wave, a ramp generator is triggered producing a 2 kHz sawtooth waveform. This becomes the analog input to a sample and hold circuit which is energized at each zero crossing of the X axis limiter. It is possible to sample twice per 1 kHz period since the signal phase is always within 10° to 170°. The output of the sample and hold circuit is a DC voltage proportional to position. Since the information bandwidth required for hand generated graphics is typically less than 20 Hz, a low pass filter is placed after the sample and hold circuit and its output becomes the external analog output.

Digital electronics

Digital signals could be obtained directly from the analog output through the use of conventional analog to digital conversion techniques. However, the cost of performing a monotonic 12 bit conversion is high, and the resulting accuracy would be limited to that of the sawtooth voltage used in the generation of the analog output. A simpler, less expensive, and more accurate method of digital generation utilizes the time displacement inherent in a phase shifted signal. In particular, the output from the pickoff electronics is displaced in time from the reference sinusoid by an amount:

$$\tau_d = \frac{\theta}{\omega} \tag{5}$$

where θ is phase shift in radians and ω is (2π) (1000) radians per second.

The displacement τ_d (in the X or Y channel) is directly proportional in principle to the coordinate position of the pen along the X or Y axis. However, τ_d in practice contains noise components as well, generated in the pen and pickoff electronics and limited only by the 7 kHz bandwidth of the channel demodulators. In order to reduce the noise bandwidth, or in effect, to filter the values of τ_d, a special circuit was developed. This circuit has the virtue that it does not introduce phase errors in the demodulated outputs while it does provide an arbitrarily narrow bandwidth. In addition, the circuit produces τ_d directly from the reference 1 kHz square wave and the demodulator square wave output.

Operation of this circuit is as follows (see Figs. 7 and 8): The DC output from the low pass filter used to provide the analog output is supplied as the slice level in a differential input comparator. The second input to the comparator is the 2 kHz ramp generated by the reference oscillator. As the ramp retraces to its starting voltage (corresponding to a zero crossing of the reference oscillator) the comparator output jumps to the "one" state. When the ramp voltage exceeds the DC voltage at the other comparator input, the output from the comparator drops to the "zero" state. The DC voltage represents a filtered version of the zero

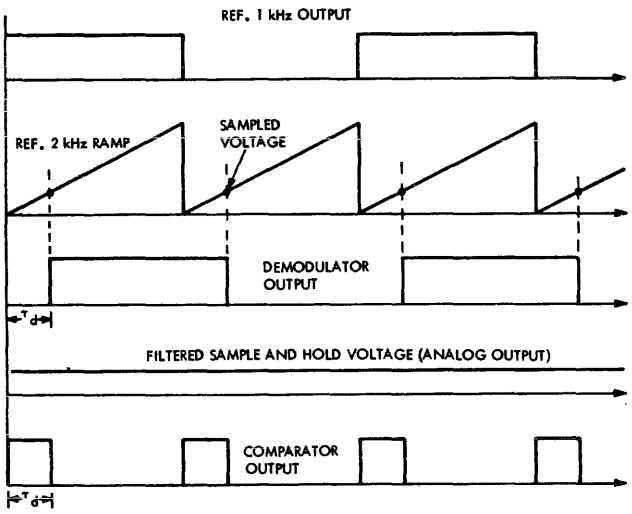


Figure 8 – Waveform of bandwidth limiting circuit

crossing information in the demodulator output. Therefore the comparator output remains in the "one" state for a period equal to τ_d and the noise components of this output are reduced in amplitude to the same value as those of the analog output.

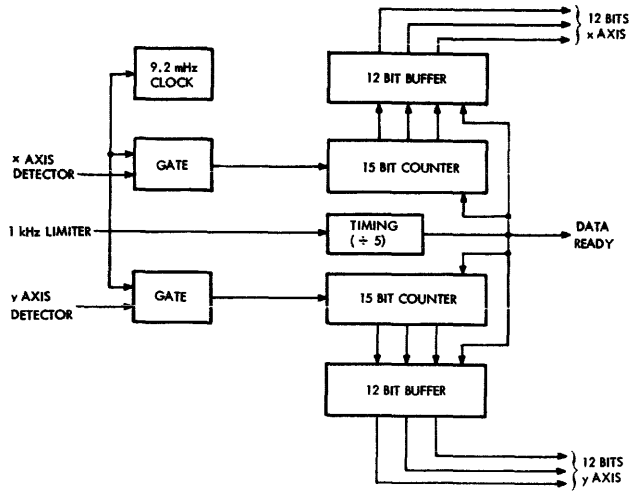


Figure 9 – Digital electronics block diagram

Fig. 9 is a block diagram of the digital logic. The pulses from the threshold detector have zero width with the pen at the left edge of the writing panel and maximum width with the pen at the right edge. Moreover, the pulses occur at a 2 kHz rate. This output is converted to digital form by gating a 9.2 mHz crystal-controlled clock into a 15-bit ripple counter binary register. Eight consecutive pulses are averaged in the counter and only the upper 12 bits are utilized as outputs. These are transferred to a buffer storage register at the end of the 5 millisecond measurement interval (4 milliseconds are used for the eight readings and one millisecond is used for bookkeeping). A "data ready" signal indicates the end of a measurement period. Y axis measurements are performed simultaneously in similar circuits.

Z axis electronics

A measure of pen height above the tablet is obtained by determination of the amplitude of the X channel signal received from the pen. The amplitude is compensated approximately for the cosh αY variation described in Eq. (4). The result is applied to three independently adjustable threshold circuits. Typically, one comparator is set to trigger at the point of pen contact with the writing panel surface, the other two may be, perhaps, at one-half inch and two inches, respectively.

Experience with the Sylvania data tablet

The Sylvania Data Tablet fulfills the goals set forth at the beginning of the program. The basic accuracy of the Data Tablet is ± 1 per cent, determined by measuring the maximum departure from an ideal grid placed over the working aperture. Deviations from the ideal grid tend to be smooth and are largest near the boundaries of the working surface. The resolution of the Data Tablet is a function of the information bandwidth utilized since noise components in the pen and pickoff electronics impose the main limitation to resolution. With an information bandwidth of 20 Hz, ten or eleven bits can be realized; the full twelve bits are obtained with a somewhat decreased bandwidth.

A Data Tablet has been in use at Sylvania for some time as the major input device for our computer-CRT graphics system. An operator enters graphical data by moving a dim spot on the display (with the pen above the surface of the writing panel) until the desired position is reached. He then touches the pen to the tablet surface and the brightened spot or track indicates entered data. In this application high accuracy is not needed since the visual feedback from the display negates any tablet nonlinearities.

Studies were made into the usefulness of various recognition schemes for handprinted alphanumeric characters and ease to which subjects adapted to program-forced constraints. The Sylvania Data Tablet proved extremely useful in this application for several reasons: First, the inking capability allowed the subjects to write on a sheet of blocked paper in a very natural manner. Second, the multiple threshold Z axis feature allowed the programmer to use one level (contact with the surface) to indicate stroke and a second level (about ⅛ inch above the surface) to indicate

completion of a character. In other words, multiple stroke characters are formed with the tip of the pen always below the ⅛ inch level, and once the pen is raised above that level the completion of a character is indicated to the computer. Third, the high resolution provided minute details needed for recognition of small characters. The 1 per cent accuracy of the tablet proved more than adequate for accommodating letter sizes of ¼ inch or more.

REFERENCES

1 T O ELLIS M R DAVIS
The Rand tablet: A man-machine communication device
Proceeding of the 1964 Fall Joint Computer Conference

Computer input of forms

by ALFRED FELDMAN
Walter Reed Army Institute of Research
Washington, D.C.

INTRODUCTION

The original impetus behind the work reported here was a conviction that the storage and retrieval of information, granted computer support, did not really have to be as complex as it now is. The ordinary user, the non-programmer, could and should be provided with the kind of support that would enable him to create his own input, manage his own files, and formulate his own queries. In many cases, the user might willingly exchange efficiency of processing, and perhaps some of the machine's capabilities, for the convenience of obtaining simplicity of input, latitude for a variety of information, a degree of generality in the organization of the stored data, absence of coding incompatibilities, and some facility for unpremeditated retrieval.

To substantiate this conviction, a system was elaborated, which is described following this introduction. It has these characteristics :

1. The decision was made to use forms, such as business forms, questionnaires and data sheets, as the fundamental vehicle for the input of information. Between the extremes of free text, with unconstrained format but problematic organization, and punched cards, with a high degree of organization achieved through constraints, such forms might provide just the right medium having a maximum of organization with a minimum of constraints.
2. These forms are encoded directly, without the user having to consult code books or worry about record formats. The user, furthermore, is free to create his own forms, anytime. The machine will not raise incompatibilities to the introduction of new forms.
3. The internal coding of the input is free form; each item of information, corresponding to an entry in a specific box on a form, is provided by the program with a distinctive label. This arrangement is possibly more readily convertible and flexible than a dependence on fixed fields.
4. The data collected are filed essentially in their source form. The user can gain a good idea of what information is available for retrieval by examining the *blank* forms used for input.
5. Forms also are used for retrieval specification and for instructions to the computer for displaying, modifying or purging specific entries on file. These query-forms (Q-forms) differ from the input forms in that each Q-form must have its own supporting program. Each of these Q-forms serves a special purpose, and as long as one is available for the intended purpose, it allows quick and ready access to the data base. Additional Q-forms with additional capabilities can be created but must be provided with supporting programs.
6. Starting an information system based on the present technique entails no preliminary programming. (However, special programs are needed to deal with such special subject areas as Organic Chemistry.) If started independently, such a system is compatible with other systems to the extent that they may be merged, temporarily or permanently.

Admittedly, the system described here depends on an intensified use of computers. As a result, processing times increase, as well as storage requirements. This may be a fair price to pay for the alleviation of human effort. More critical are possible shortcomings in the areas of more sophisticated retrieval and data manipulation. An evaluation on this score will have to await experience with larger files, which are not as yet available. However this may turn out, the fact that a user, not versed in programming, could be able to go in and out of a fairly versatile system, and without delay, has advantages even in the face of other limitations.

Why forms?

The system described here hinges on the use of forms. Forms, at present, are widely used for gather-

ing and storing information. Bureau of the Census figures indicate an annual volume of around $200 million worth of forms in use in this country.[1] Forms are easy to fill in, and easy to check. This convenience allows the forms to be filled-in directly by oneself or one's secretary, placing the recording of the information close to its source. Forms have been perfected over centuries of use; they function effectively and simply, by means of clearly legible captions, explanations and instructions which are placed directly where needed.

A form is a checklist to prevent omissions, and a screen to eliminate the superfluous. An author has remarked:

> "Ordinary people ... do not set up to be experts; they write neither novels nor treaties; they publish no autobiographies; they seldom write to the press ...; they keep their opinions and experiences to themselves. Inarticulate, they are unheard; unheard, they are forgotten."

But with form questionnaires, "this vast unvocal mass" can be sounded out.[2]

Forms, if not altered in design for accommodation to some machine requirement, are a considerable improvement over many other means of computer input. In particular, forms can accommodate a wide variety of information. For instance, because they make available the use of two-dimensional space, chemical structures can be entered on them as normally represented. This, incidentally, promotes compatibility. If such structures must be linearized in order to enter the computer, a choice must be made among different methods of linearization, resulting in an incompatibility whenever different linearization methods were used on different files. The encoding of chemical structures in their normal representations avoids this problem.[3] Although seemingly insignificant, the same problem arises with the manner in which ordinary subscripts or superscripts are linearly coded on punched cards.

The organization impressed on the input by the forms is carried over to the data base itself. Being able to control the organization of the form, the user is in a position to direct the organization of the files created, and to determine their access points. The technique has a potential of generality, inasmuch as there is no theoretical limit to either the number or the diversity of the forms which may be included.

The encoding of forms.

The work described here was performed with special two-dimensional encoding typewriters.[4] With the variety of input devices available, such as consoles with light-pens, optical scanners, etc., the encoding of forms would seemingly not present much of a problem. No doubt, given a few technical improvements, this will be the case. As it is, however, the special encoding equipment was used because the available methods all have certain deficiencies.

Of the methods currently in use for coding forms, the most prevalent consists in transferring the contents of the forms to punched cards. To facilitate this transfer, the form itself is changed, requiring entry character by character. On such forms, keypunch assignments are indicated on the form itself; abbreviations must be used, even code books. Such a form has lost much of the convenience and versatility of ordinary forms.

In another method for coding forms, tape typewriters are used.[5] To distinguish one entry on the form from another, use is made of external, paper-tape programs—one for each different form. The paper-tape program causes the typewriter to position itself for access, in succession, to all the boxes on the form to which the program applies. At each stop, the typist has the option of filling-in the box, or of by-passing it. The positioning codes, transferred to the tape created in this process, serve to identify the entries. This method is wasteful of the operator's time, especially if a number of stops are by-passed. Dependence on external programs makes it clumsy to code different forms. Finally, making corrections is even more difficult than with ordinary tape typewriters.

A time-sharing system is another alternative for the input of forms. The console displays, originating from the computer, correspond substantially to a dismembered form, each successive display being equivalent to one box of a form. This method has the capability of immediately updating a file with the input furnished. In such an arrangement, furthermore, there is virtually no possibility of a discrepancy occurring between the information, as entered by the user and as received by the machine; and error-correction is simple.

The disadvantage of on-line consoles for the present application is that the user cannot create his own input forms. Each display flashed onto the console is contingent on the user's preceding answer; otherwise, there might be too many irrelevant displays. Devising the proper concatenation of the displays and providing for all the possible answers in return, is a task of the programmer's domain. But on a printed form, the irrelevant boxes, although present, are readily disregarded.

Nor do optical scanners, in the present state of the art, prove suitable for the encoding of forms. Their error-correcting technique, consisting of identifying

an erroneous line and in replacing it at the bottom of the page, becomes complicated when applied to entries inside boxes on a form.

The modified tape-typewriters used for the present work were developed for the encoding of chemical structures. These machines have been described elsewhere.[6] Briefly, they avoid the problem of discrepancies between typescript and punched tape, by avoiding altogether the need to correct the tape. These machines do code not only the identity of the character typed but also the *location* where each character appears on the typescript. An error, corrected on the typescript, is caught when processing the tape, when, by virtue of the encoded locations, the computer will become aware of an overlap. In actual operation, this error-correction technique is very convenient.

Because locations are encoded, these typewriters can be operated virtually as if they were ordinary typewriters. The tabulator key, for instance, can be set at will, and the platen can be twirled manually. Filling-in a form, the typist can skip entries, or go back to correct one. Whatever the sequence of input, the entries on the form retain the distinctions attributable to the locations they occupy. Therefore the typist does not have to identify the entries. The positional information, automatically generated by placing an entry within the confines of the appropriate box on the form, enables the computer subsequently to assign a predetermined label to each entry.

The above typewriters operating off-line, the reliability of the coding operation is of great concern. Should the coded input become garbled unbeknownst to the typist, vast amounts of it can accumulate before this circumstance is discovered at processing time. Reliability, however, is considerably promoted by the absence of restrictions in the use of the typewriters, and by the suppression of paper-tape corrections. Practically speaking, the typist cannot spoil the coded record of a visibly correct typescript. Machine failure, in turn, is detected by means of parity checks and the like, and interlocks prevent the coding of data past the point of failure. Lacking these safeguards, some comparable machines[7] are not as attractive for the purpose at hand.

Creating a new form

To create a new form, the user, first of all, assigns it a specific number. Then the form is laid out on the typewriter (Fig. 1). Subsequently computer processing will generate a master form (Fig. 2), suitable for duplication, from which a supply of blank forms can be prepared. In the course of the processing, a number is assigned to each box on the form (but an override

DATA SHEET FOR COMPOUNDS

WR NR.	Submitter	Key nr.
Shipped	Received	Acknowledged

Name of compound

Structure

Mol Formula	Mol wt	Analyses		
Appearance		Element	Calculate	Found
Quantity	Code nr.	C		
Notebook ref.	Prep by	H		
Test System		N		
		S		
BP / MP	Refr Ind			
IR				
UV				
Chromatography		Literature		
Warning				

Stability	Stable	Unstable
Acid		
Base		
Heat		

Solubility	Sol	Insol
Water		
Acid		
Base		
Methanol		

Hygroscopic

Equations indicating synthetic route

Remarks

Discreet	Synthesized	Gift	Purchased

Figure 1 – New form, as laid out on a typewriter

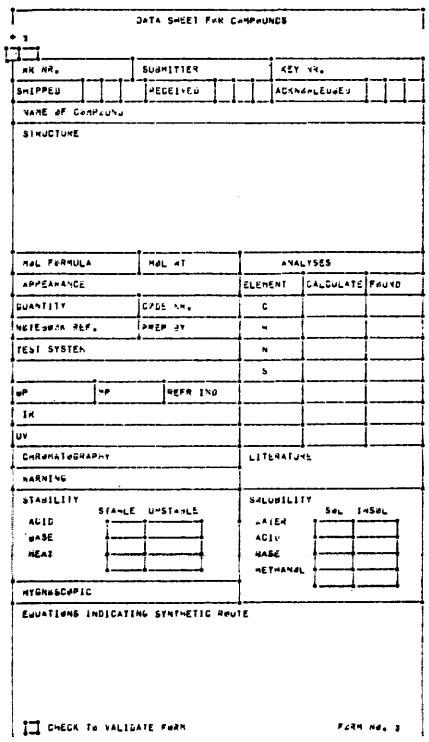


Figure 2 – High-speed printer output. This form corresponds to the one in Fig. 1, but the program has added a reference box (top) and a validation box (bottom)

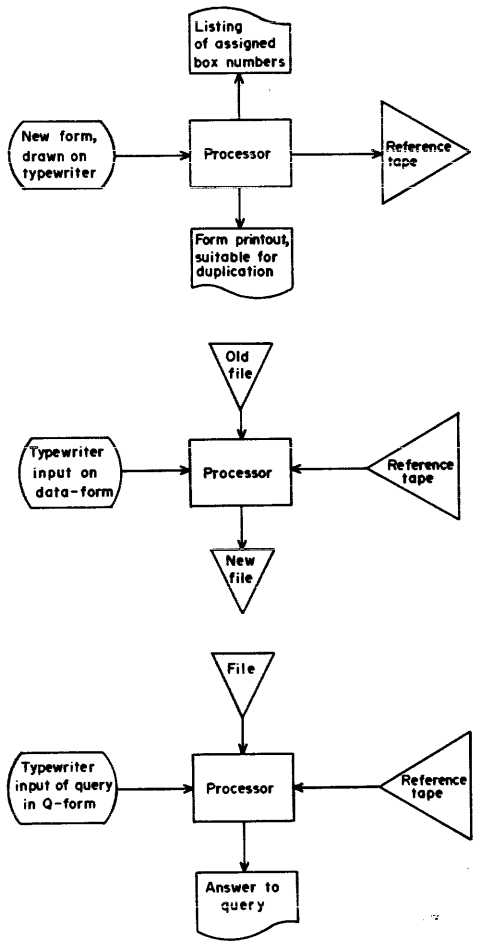


Figure 3—The diagrams show, from top to bottom, the creation of a new form, the updating of a file, and the querying of a file

option is available). These numbers will be used to code the entries these boxes are destined to hold The diagram in Fig. 3 shows these operations.

Such a form differentiates itself from other forms by always providing boxes, not just spaces, for all the information to be entered. Being contained in a particular box defines an entry; and obviously, the contents of a box is not allowed to spill over. Each new form is provided by the program with two additional boxes (compare Fig. 1 with Fig. 2). In the first of these boxes, the typist repeats the form's identification number. During subsequent machine processing, this number will identify the particular form, and its location will serve as a reference point to the location of all the other information on the form. The second of these boxes, placed at the bottom of the form, is to contain a validation check. By not filling it in, the typist can discard an erroneously filled-in form. (It is not possible to otherwise discard the coded counterpart on tape.) By filling it in, the computer is enabled to compensate for any misalignment of the form in the typewriter.

New forms can be created at will, without restrictions except perhaps as to their size, and without requirements for programming. As soon as a form has been created, the system is ready to accept it, and its input will be compatible, in the sense discussed below, with the prior input, or with independently created files. Thus, the user is provided with a flexible input system, accommodating a variety of input, which he himself can tailor to suit his own needs.

Nevertheless, it will be obvious to the user that the design of a new form takes much thought. For instance, a box captioned "temperature" may collect numbers representing either degrees Fahrenheit or Centigrade. He must therefore make the caption specific. Similarly, if the system is to be asked subsequently to alphabetize names, separate boxes ought to be provided for the different parts of a name, so that the last name can be selected unequivocally. The means by which he controls the information to be stored, as well as its organization, are relatively simple —by furnishing adequate space to avoid constricting entries, by being liberal in providing boxes for separating different types of data, and by being explicit with his captions. But although this requires thought on his part, the thought here is directed properly to the prospective data and to its retrieval, and not to the machine and to its requirements.

Should it be necessary to alter the design of a form, care must be taken that the same information is assigned the same box number as before (the number-override option is then exerted); but the shape or the relative location of the boxes on the redesigned form do not affect the retrieval. If the new form is to accept additional data, the boxes destined to receive this data must be assigned numbers not used on the replaced form. If the changes are more drastic than that, it is better to create a new form, having a different form number.

Accessibility of the stored data

Highly constrained information, such as input on fixed-field coded cards, results in highly consistent files. It might therefore be expected that forms, encoded under relatively lax rules, would result in files of commensurately low consistency. But this is not altogether the case.

If the purpose of the constraints was solely to insure consistency, then one might expect such an analogy to hold. But although consistency may profit from the standardizations, abbreviations, etc., which constrain the input, the constraints are there, in good measure, not to insure consistency, but to maximize the utilization of scarce space. The IBM card has solely 80 columns, and ever since the days of the first card

collators, processing has been easier if only one card is used per item of input. So, in order to conform to this processing precept, the input information is chopped up and squeezed to the very limits of recognizability.

In this game, each file fends for itself. Input rules, not to speak of additional space-saving tactics such as superimposed coding, are developed according to a strategy which takes account of the different contents of particular files. Consequently, the same information may be coded differently on different files. This lack of compatibility among files of this kind precludes their being merged into a combined data base, desirable as this may seem. For that matter, how often has not a file become obsolete because a change to a new card format, with the corollary need for updating the backlog, would have proved too costly ? The very constraints work thus often to the detriment of compatibility.

This is not to say that an improved compatibility *among* files, obtained by reducing input constraints, will be without detriment to the consistency *within* each file involved. But in this case, a certain shift of the burden for data correlation from input to output, a shift from rigid rules developed *a priori* to an intelligent examination of virtually original input, is not without merits. This is especially so because it would appear that problems due to inconsistencies are experienced not so much in searching a single file, as in attempting to correlate two independent files.

Different kinds of data are affected differently by reducing input constraints. Assessing the potential precision of retrieval of data entered on forms, the highest score will go to entries entered by means of a check mark. Numerical entries are next, absolute precision in their retrieval being affected only by outright input errors. (A referee wondered whether, using unconstrained input on forms, the similarity between the numbers

1,234.5	1234.5	01234.5
1.2345×10^3	1.2345E3	1234 1/2

would be perceived. In the current state of the program, the first three are accepted at their true value; the last three are rejected, with a message stating that they are not acceptable numbers.)

Next down the line of diminishing precision of retrieval, are entries consisting of single words. Here, synonymy can arise. If only a single file is searched, this problem can be solved at retrieval time, without too much difficulty. If the searcher is looking for the color "red" in a particular box on a form, he must also, in this case, look for "scarlet", "crimson", and for any other words his dictionary might suggest. But suppose that the investigator is attempting to correlate two files. Suppose, for instance that he wishes to determine whether the color of an insecticide detracts from its effectiveness. He has obtained two files; on one are recorded the sensitivities of various insects to various colors, and on the other the actual colors of various pesticides. He intends to match any color given on one file with the same color on the other. Because of synonymy, he will not obtain all the legitimate matches.

Possibly, the problem may be solved by making two passes. In the first, the different terms, used on each file to denote color, are listed. The investigator can then declare which terms he considers synonymous. In the second pass, the desired correlation is then obtained. From the investigator's point of view, this procedure has the advantage that it deals in English, and not in perhaps difficulty reconcilable coding conventions. This area is presently being investigated, but experience is still lacking.

Precision of retrieval diminishes further where entries consist of texts. The program presently used has no text-searching capabilities. But even if these were available, precision of retrieval would regress to the vanishing point, where sundry texts are lumped under captions such as "remarks", never to be retrieved except as adjunct to a response to an independently specified search. Not that they are without value. A referee pointed out that the most useful information in a patient's record is usually the hand-scribbled notation that doesn't fit the confines of the prescribed box. Here, indeed, we arrive at the limits of the system. But then, there are those who maintain that the degradation of information begins the very moment one's thoughts are couched into words!

Interrogation of the data base

For the sake of the present discussion, the accumulated contents of identical forms, stored on tape, represent a file; and a combination of these files, originating from different forms, constitutes a data base.

To query such a data base, the prospective user must ascertain what information it may contain, and devise a strategy for extracting the information of interest to him.

To this end, he may consult the blank input forms. These forms convey a considerable amount of information about the data originally entered. Forms "talk" about the information they contain, and the printed captions, directives and explanations that are present on a form are as pertinent for retrieval as they are for input. Thus, although the system does not code concepts such as temperature, an examination of the

blank forms used will reveal where, and in what contexts, temperatures are recorded. Original entries are retrieved by citing the corresponding form and box numbers.

Blank forms perform this service even after they have become obsolete, and are no longer used for input. The information a form carries about the information it contains remains valid.

Should there be too many blank forms to make such an examination convenient, the captions can be listed in an index. A sample of such a caption-index is shown in Fig. 4. Such an index is different from the conventional indexes accessing text. Because of the nature of the generation of index terms used with the latter, the presence of a given index term can be almost accidental. With the described caption-index, each single term will obviously produce *all* the material contained in the specified box.

Index to Forms

(in a reference such as 3/17, the first number is the form number, the second is the box number)

Accession numbers -- see Reference numbers	
Acids, solubility of compds, in	3/68,3/69
--, stability of compds, in	3/62, 3/63
Bases, solubility of compds, in	3/72,3/73
--, stability of compds, in	3/66,3/67
Boiling point	3/43,4/19,4/20
Carbon, calc'd., in elemental analysis	3/28
--, found in elemental analysis	3/29
Chemical compounds, analysis of,	3/20
--, boiling point of	3/43,4/19,4/20
--, chromatographic data on	3/57
--, crystallographic data on - see Crystallography	
--, dangerous	3/59
--, density, exptl.	4/18
--, elements contained in	3/23
--, explosive	3/59
--, hygroscopic	3/76
--, i.r. spectrum of	3/49
--, lit. on (syntheses, tests)	3/58
--, melting point of	3/44,4/23,4/24
--, mol. formula of	3/18,4/3
--, mol. wht. of	3/19,4/4
--, received at WRAIR, date of,	
day	3/9
month	3/10
year	3/11
--, received at WRAIR, date of acknowledgment of,	
day	3/13
month	3/14
year	3/15
--, refr. index of	3/45
--, shipped by WRAIR, date of,	
day	3/5
month	3/6
year	3/7
--, solubility in acids	3/68,3/69
--, solubility in bases	3/72,3/73
--, solubility in methanol	3/74,3/75
--, solubility in water	3/64,3/65
--, stability of	3/60
--, stability in acids	3/62,3/63
--, stability in bases	3/66,3/67
--, stability to heat	3/70,3/71 4/20,4/24
--, structure of,	3/17,4/5
--, toxic	3/59
--, u.v. spectrum of	
--, X-ray data on	4/22
Chemical elements, calculated for elemental analysis	3/28,3/33,3/37 3/41,3/47,3/51 3/55
--, found in elemental analysis	3/29,3/34,3/38 3/42,3/48,3/52 3/56

Figure 4 – Sample of index referencing to boxes on forms

Although the terms of a caption index can be used in combination, there is a point beyond which these terms are unable to achieve further discrimination. This point is reached when it becomes necessary to discriminate within the confines of a single box. As was discussed earlier, the methods and the precision of retrieval vary from this point on. Even so, however, the preliminary sequestration achieved by means of accessing boxes on forms, is valuable in its own right, and is obtained with comparative ease.

We come now to the actual techniques for querying the files or the data base. This might be done by coding parameters on cards, by devising a special language, etc. In the following, however, the use of forms for querying forms is described.

Forms for querying and altering the data base

Describing the use of forms for accessing files (albeit punched-card files), Postley[8] reported that "operators and management can use these . . . directly, without submitting their problems to programmers and without becoming programmers themselves"; furthermore, he mentioned that problem definition time is decreased.

In Postley's system, a few standard forms describe all retrieval operations. Varying the parameters entered on these forms allows one to vary the format and type of output to be obtained. The system described here, however, allows for the use of any number of query forms (or Q-forms); there are some for querying the file, for specifying output formats, even for correcting items on file, or for applying plausibility checks at input time. These forms can be used in combinations, and new forms, with additional capabilities, can be added. Perusal of existing forms, or of an index thereto, indicates thus not only what data is available, but also what procedures are available to reach it.

The Q-form shown in Fig. 5 is used to obtain listings. Initially, a rough layout of the expected output is sketched on quadrillated paper. The form is then filled-in : the desired items of data are specified by their input-box numbers, and the desired locations by their coordinates. To left-adjust data, the beginning x-coordinate is specified; to right-adjust data, the terminal x-coordinate is specified. Centering is obtained by specifying both. A capability exists for entering headings, and the like. The form can be used in conjunction with other forms, such as the Q-form specifying a search (cf. Fig. 8), in which case it will control the format of the result of the search.

In Fig. 6, the output shown corresponds to the specifications given in Fig. 5. In Fig. 6, 3×5 cards are produced from the same file, by entering different parameters on the same Q-form.

INSTRUCTIØNS FØR THE SELECTIØN AND DISPLAY ØF ITEMS FRØM A FØRM

FØRM FRØM WHICH INFØRMATIØN IS TAKEN NØ. 3

BØX NØ.	CØNTAINS GRAPHICS YES	NØ	TRANSFER TØ LØCATIØN BEGIN X	BEGIN Y	END X	END Y	CAPTIØN (1)	LØCATIØN ØF CAPTIØN (2)
1		X	1	1				
18		X	1	3				
2		X	1	6				
17	X		20	1	55		STRUCTURE	H
16		X	60	1			NAME	H
19		X	7	8			M.WT	L

SPACING BETWEEN CØNSECUTIVE ITEMS 4 HALF LINES.

(1) CAPTIØN SHØULD NØT EXCEED 10 CHARACTERS

(2) IF CAPTIØN GØES AT HEAD ØF A CØLUMN, ENTER H.
IF CAPTIØN GØES AT LEFT ØF AN ENTRY, ENTER L.
IF CAPTIØN GØES AT RIGHT ØF AN ENTRY, ENTER R.

X CHECK TØ VALIDATE FØRM FØRM NØ. 2

Figure 5 – Form for the specification of printout format. The specifications entered here produced the output shown in Fig. 6

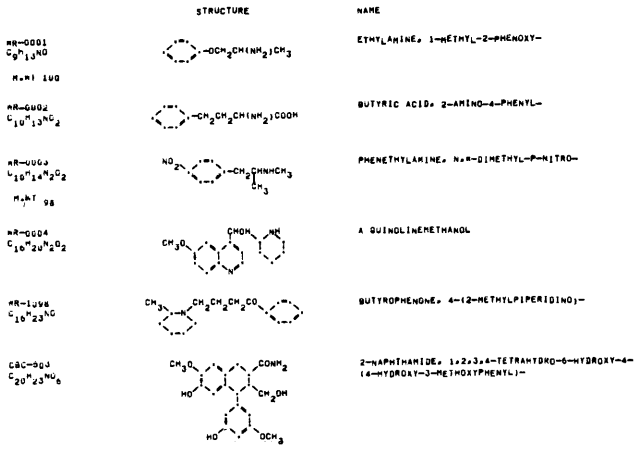


Figure 6 – Output format generated by using the form shown in Fig. 5

By means of the Q-form shown in Fig. 8, retrieval may be specified. If a single file is searched, the criterion for retrieval (or one of its criteria) is specified in the first box of part B on the form. If a correlation is desired, the last two boxes in part B are filled-in instead. This form, although working only with numerical data at present, illustrates the wide scope to which such forms are adaptable.

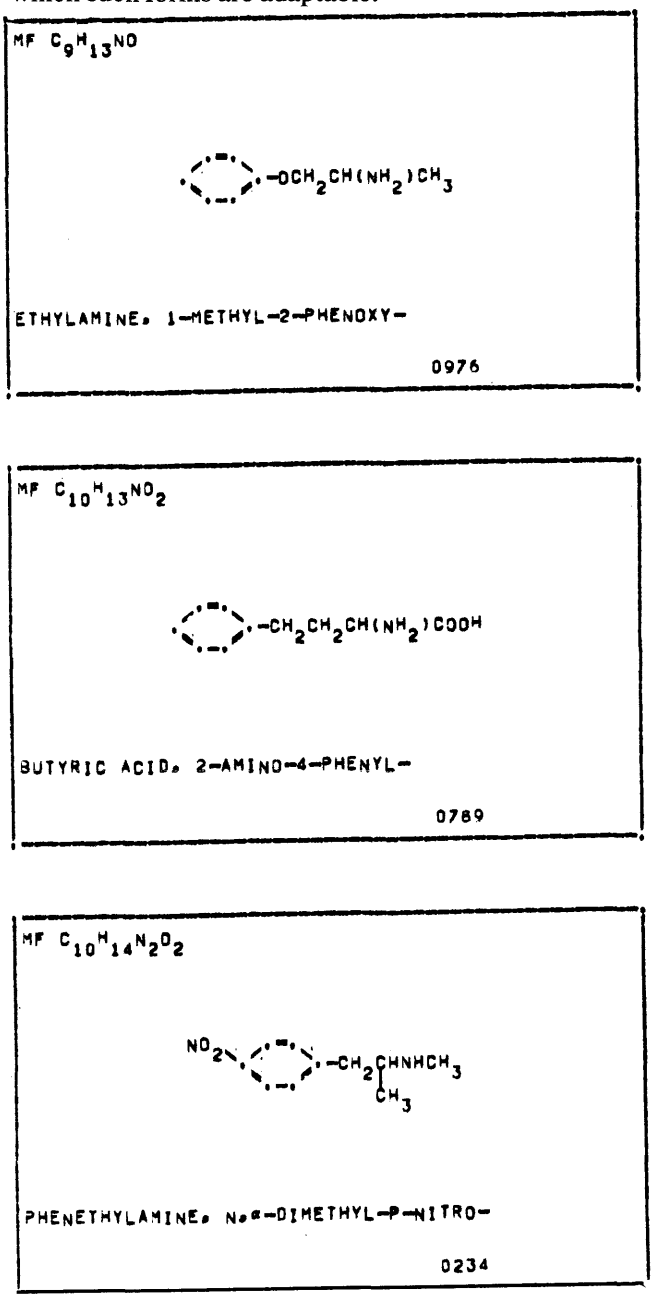


Figure 7 – Another output format generated by using the form in Fig. 5, differently filled-in, however

Fig. 9 shows a Q-form by means of which entries that are on file can be replaced or purged. This form is designed to use either a single or a double criterion for making a change. The double criterion is needed if, for instance, an accession number was mistakenly repeated on another input form.

Fig. 10 shows a Q-form capable of making plausibility checks. This form is normally processed with the

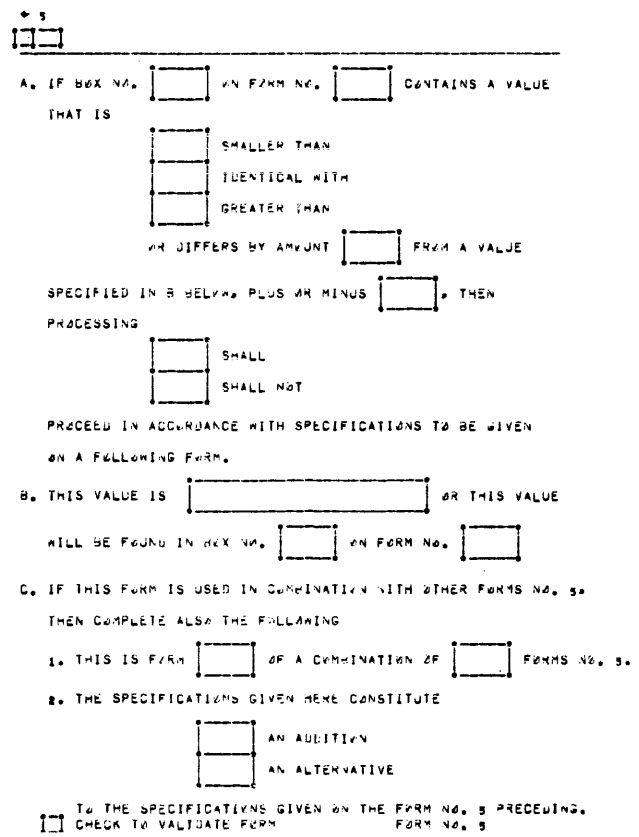


Figure 8—Form for retrieval specification

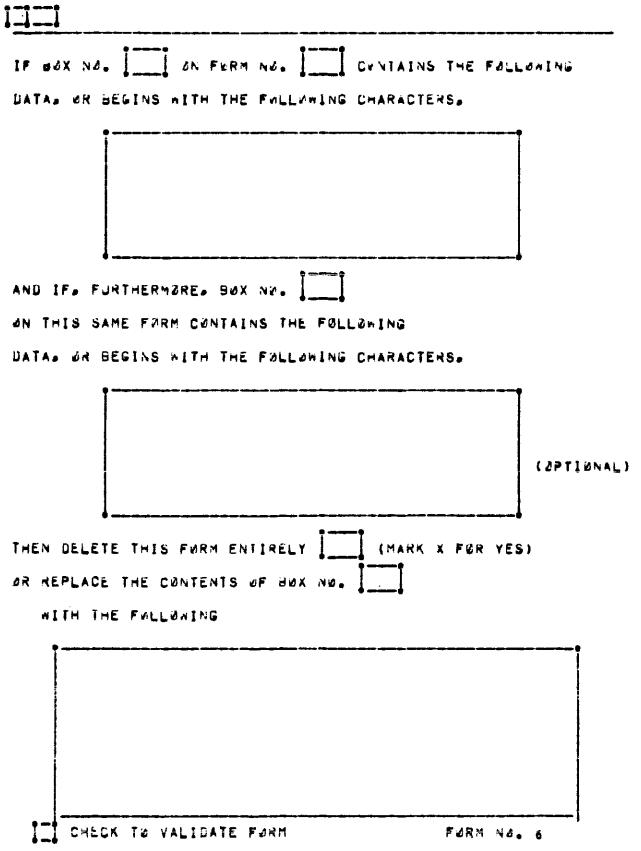


Figure 9—Form for specifying correction of file

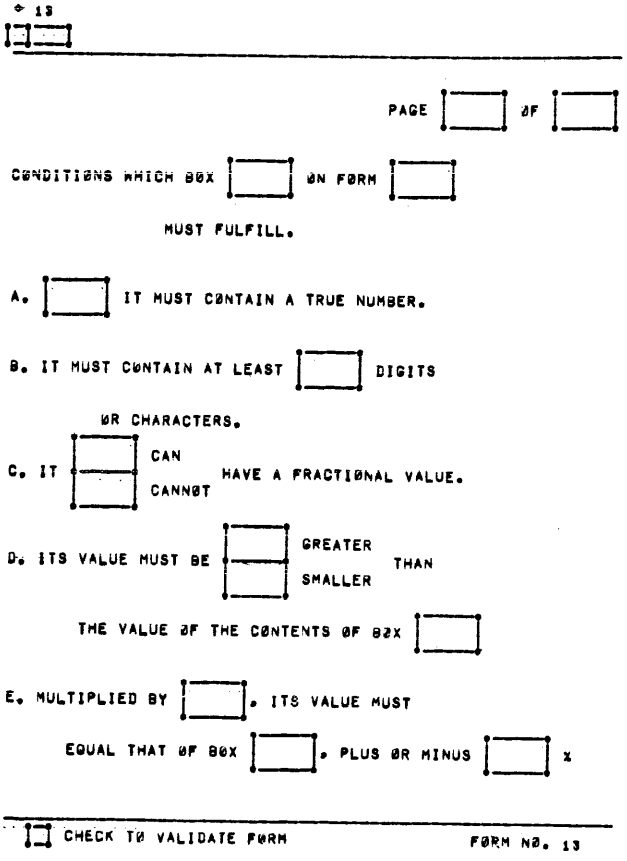


Figure 10—Form for specifying plausibility checks

input. The checks performed by this form might perhaps be more thorough[9]; it is shown here chiefly to illustrate the variety of tasks that may be performed simply by filling-in a form.

Prospects

A new generation of computers is emerging, which provides greater computing capabilities and vastly expanded memories; these computers are therefore much less in need of limitations on the input. Consequently, there is increasing talk of generalized data bases, with capabilities that go beyond the processing of bank accounts or of airline reservations—data bases which could provide answers to requests both varied and unforeseen.

For many types of input, the old, tried and proven methods will, no doubt, continue to persist. But other types of input, or other modes of use, will virtually insist upon new approaches. Through the use of two-dimensional encoding typewriters, which are now becoming available, the power of organization which forms can provide can be fully exploited. The versatility of this combination, the modified typewriters and the forms, will bring closer the day of processing systems that are truly machine independent.[10]

REFERENCES

1 J J BARTOSIEWICZ
The manifold business forms industry—an economic review
BDS A Quarterly Industry Report *5*:11 1964

2 A C PATTERSON
The questionnaire as a means of educational research I The extent and reliability of questionnaire investigation
Scott Educ J *25*:683 1942

3 A FELDMAN D B HOLLAND D P JACOBUS
The automatic encoding of chemical structures
J Chem Document *3*:187 1963

4 The encoding process, as described here, is covered by U S Patent 3,358,804 Other patent(s) are pending

5 D HEARNE
Tape operated writing machines
Automatic Data Proc Feb 1962 pp 32-37

6 D B HOLLAND D P JACOBUS A FELDMAN B R MOBERLY
The coordinate concept—an approach to tape punching
Control Eng *11*:60 1964

7 M KLERER J MAY
Two-dimensional programming
Proc FJCC *27*:63 1965

8 J A POSTLEY
File management application programs
DPMA Quarterly *2*:20 1966

9 R J FREUND H O HARTLEY
A procedure for automatic data editing
J Amer Statist Assoc *62*:341 1967

10 I am indebted to Dr David P Jacobus and to Daniel J Minnick, of this Institute, and to Robert R Puttcamp, of the Harry Diamond Laboratory, for helpful discussions and suggestions.

Machine-to-man communication by speech Part 1: Generation of segmental phonemes from text

by FRANCIS F. LEE
Research Laboratory of Electronics, Massachusetts Institute of Technology
Cambridge, Massachusetts

For many years man has been receiving messages from machines in printed form. Teletypes, computer console typewriters, high-speed printers and, more recently, character display oscilloscopes have become familiar in the role that they play in machine-to-man communication. Since most computers are now capable of receiving instructions from remote locations through ordinary telephone lines, it is natural that we ask whether with all of the sophistication that we have acquired in computer usage, we can communicate with the computer in normal speech. On the input of the computer, there is the automatic speech recognition problem, and at the output, the problem of speech synthesis from messages in text form. The problem of automatic speech recognition is substantially more difficult than the speech synthesis problem. While an automatic speech recognizer capable of recognizing connected speech from many individual speakers with essentially no restriction on the vocabulary is many years away, the generation of connected speech from text with similar restrictions on vocabulary is now well within our reach.

With touch-tone telephone, people can call a computer and, after the initial connection has been made, use the calling buttons as an input keyboard to communicate unambiguously to the computer, thereby temporarily bypassing the very difficult problem of speech recognition.[1] With computer-generated speech, we can foresee the use of touch-tone telephone sets as the only remote terminal device for large and complex information retrieval systems. In this paper and in the companion one by Jonathan Allen, we shall discuss the problems encountered in the computer generation of connected speech from text source, and our solutions.

It should be clear that with the addition of a printed character-recognition unit, the system is useful as a reading machine for the blind. It was originally with this purpose that our research into computer generation of connected speech from text source began.[2]

We should like to point out that while it may be acceptable to call out stored audio signals corresponding to words or phrases in order to form short messages, such as with the automatic time information service over telephone or the currently available computer audio response systems, it is not possible to extend the method to handle even a vocabulary that might be encountered in reading a second-grade level book. A generative scheme, in which a reasonably small amount of stored data are used, is both desirable and necessary.

The elements of speech sound

We use only twenty-six letters of the Roman alphabet and a few additional symbols to express our language in text form. These letters and additional symbols are called "graphemes." A grapheme is the smallest unit of construction of this written language. The smallest segment of speech sound on the level of production and preception is called a "phoneme." The common element between the written words [cat] and [pack] is the grapheme [a], and in sound the common element between the spoken words [great] and [raise] in the phoneme /e/.*

Speech synthesis by rule

In recent years, several workers[3,4,5,6] have demonstrated that it is possible to synthesize continuous speech from phoneme specification. In order to regu-

*Phonemes are shown between slashes, and graphemes are *shown between square brackets.*

late the intonation and rhythm, it is necessary to provide additional specifications beyond the segmental phoneme level. The generation of the segmental phonemes and the suprasegmental or prosodic features from printed text has been the main concern of researchers in the speech synthesis field.

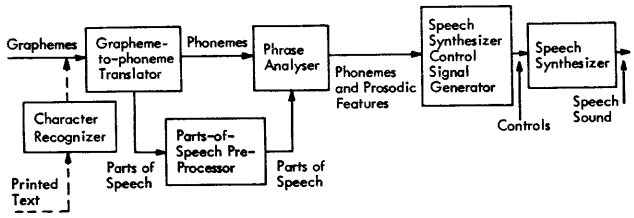


Figure 1 – A text-to-speech conversion system

A text-to-speech conversion system

The approach which was taken here in converting text information to speech is shown in Fig. 1. The entire process is divided into 5 modules. The first module accepts graphemes as input, and translates each written word into its phonemic representation with the proper stress markers. As a result of the translation process, the obtained parts-of-speech information is further condensed, with some ambiguities removed, by the parts-of-speech pre-processor module. The resultant parts-of-speech information and the phoneme strings of the sentence are then used as input to the phrase-analyzer module. For multiple phonemic representations of the same word, this analyzer module makes a selection based on syntactical considerations. The phrase-analyzer module places phrase boundary markers and in the sentence environment re-evaluates the stresses obtained in the grapheme-to-phoneme translator module. The speech-synthesizer control signal generator module converts the phonemic specifications, stress information, and phrase markers into the speech-synthesizer control signals. The speech-synthesizer module is a hardware unit generating the actual speech output. Since our speech synthesizer and its control signal generation is similar to that reported by Holmes, Mattingly, and Shearme,[5] we shall restrict our present report to the first three modules in Fig. 1, namely, the grapheme-to-phoneme translator, the parts-of-speech preprocessor, and the phrase-analyzer.

The rest of this paper deals with the problem of grapheme-to-phoneme translation.

The phonic method

The phonic approach to the grapheme-to-phoneme translation problem was taken first by Higginbottom,[7] in 1962, and later by Bhivani, Dolby, and Resnikoff,[8] Weir and Venesky,[9] Monroe.[10] They maintained that it is possible to derive a letter-to-phoneme translation through the use of letter context. This phonic method has been used by First and Second Grade teachers in the United States in the teaching of English pronunciation. After a detailed and exhaustive study of the phonic method, it became clear to us that while the method works reasonably well with children, it is totally unacceptable for processing by machine. This should not be surprising because a child, six or seven years old, can obtain cues from many sources, such as illustrations in a book or knowledge of the subject before his first encounter with the written words.

In analyzing words for phonic rules, the paradigm forms of the basic words, i.e., the regularly inflected forms, are not usually considered, since it is thought that they can be readily derived. To a machine, however, we must be more precise. For example, for the two words [invited] and [profited], we must prevent the machine from mistaking them as */invitid/* and */profaitid/*. If we consider [ed] as the paradigm suffix, it is possible to look up in a list to see whether [invit] is present or not. If it is not, the mute [e] may be appended. Clearly, such a list would be very long. We must also have an exception list for words ending in [ed] which must not be separated, such as with the word [quadruped] which must not be rendered [quadrupe] + [ed]. Similar problems arise with other suffixes beginning with a vocalic sound.

The second problem with the phonic method is the difficulty in handling compound words. Compound words abound in English. When a mute [e] occurs in a nonfinal position, there is no simple way of ascertaining that it should be mute other than having a list of all words ending in mute [e] and each time performing an exhaustive matching operation. We would also need an exception list to handle those words ending in a nonmute [e], for example, words like [apostrophe, catastrophe, acme, recipe], etc.

With the phonic method there would be no simple way of determining the location of word stresses. While parts-of-speech information can be used to assist in the determination of stress location, the determination of parts-of-speech from spelling and context, as done by Klein and Simmons,[11] involves the use of more exception lists.

To resolve ambiguities such as [refuse], (verb) and [refuse] (noun), a syntactic analysis must be performed by using a larger context. Again, the parts-of-speech information is needed.

Translation through the use of a morph dictionary

While the smallest segment of speech on the level of production and perception is a phoneme, the smallest

unit of speech that has meaning in a given language is called a "morpheme." A morpheme is composed of one or more phonemes. For example, the spoken word for "cat," represented phonemically as /kæt/, is a morpheme consisting of three phonemes /k/, /æ/, and /t/. Clearly, by themselves, the phonemes /k/, /æ/, or /t/ have no meaning, but /kæt/ does.

While the morpheme /kæt/ can exist alone or be combined as in /kæts/, [cats], some morphemes can exist only as adjuncts to other morphemes. They are called "bound morphemes." Bound morphemes serve to impart some quality to the combined form, as the /s/ in /kæts/, which has the meaning of the plural form of the noun. When a morpheme is not a bound morpheme we call it a free morpheme. The /kæt/ in [cats], for example, is a free morpheme.

Since a word can exist in both printed and spoken form to differentiate between the two forms of a given word, we shall call the spoken form the "p-word" and the written form, the "g-word."* We see that a p-word is either a free morpheme or a free morpheme combined with a bound morpheme. We are all familiar with a rather free occurrence of the compound formation of English p-words such as those corresponding to the g-words [greenhouse, whitestone] etc. There are those p-words with a collection of prefixes and suffixes such as those p-words for [prehistorically, loveliest], etc. A more general and recursive definition of a p-word is

> "A p-word is either a free morpheme, or a free morpheme combined with a bound morpheme, or a free morpheme combined with a p-word, or a bound morpheme combined with a p-word."

While this definition has to be supported by additional rules of combination in order to be useful as a generative rule, it is quite suitable for the purpose of identifying a p-word in terms of its constituent morphemes.

In a majority of cases, the formation of a p-word that contains more than one morpheme amounts to little more than a mere concatenation of the constituents, such as those p-words corresponding to [whitestone, greenhouse] etc. There are, however, interesting cases in which the p-words involve a transformation in the combined form. For example, /spesəfai/+ /ik/ gives /spisifik/, [specific], and /gæləksi/ + /ik/ gives /gələktik/ [galactic]. The rules governing these changes are morphophonemic rules.

Morphophonemic rules may depend upon the parts-of-speech classification of the resultant p-word. They may also depend upon phonological considerations. The three representations of the bound morpheme /s/ corresponding to the plural forms of nouns is an example of phonological dependence. We have /kæt/ + /s/ → /kæts/, /dɔg/ + /s/ → /dɔgz/ and /buʃ/ + /s/ → /buʃiz/. The morphophonemic rule in this case may be described as /s/ is changed to /iz/ if the last phoneme in the preceding morpheme is among the set /s/, /ʃ/, /dz/, /z/, /tʃ/, and it is changed into /z/ if the preceding morpheme ends in a voiced phoneme, and remains as /s/ otherwise.

If we now turn our attention to the written form of words, we can find a parallelism in that a g-word, except when it is a root word, can be broken down into smaller elements. Let us call the smallest meaningful units in written form *morphs*. We can define free morph and bound morph similarly as we did for free and bound morphemes. A general and recursive definition of a g-word may be stated:

> "A g-word is either a free morph, or a free morph combined with a bound morph, or a free morph combined with a g-word, or a bound morph combined with a g-word."

In English, the formation of a g-word containing more than one morph is often the simple concatenation of the individual morphs. There are several very commonly encountered variations requiring adjustment at the junctions. These adjustments can be stated in the form of general rules which we shall call morphographemic rules. We are all familiar with the morphographemic rule which doubles the final consonant letter after a stressed vowel in situations like [ca*pp*ing, equi*pp*ed], etc.

The problem of grapheme-to-phoneme translation is the problem of g-word to p-word mapping. Since a direct individual grapheme-phoneme relationship cannot be established except in very simple cases, a general treatment requires us to seek the correspondence at a deeper level.

While the relationship between a g-word and a p-word is very complex, the relationship between a morph and a morpheme is almost one-to-one, barring homographs and homophones. To map a g-word into a p-word, it is only necessary to decompose the g-word into its constituent morphs by using the morphographemic rules in reverse, mapping the morphs into the corresponding morphemes, and applying the proper morphophonemic rules to obtain the p-word. The procedure is illustrated in Fig. 2.

We shall now proceed to identify the morphographemic rules of English, discuss the morph-to-

*p-stands for phonemic and g-stands for graphemic.

morpheme dictionary, and the method for decomposing a g-word into its constituent morphs.

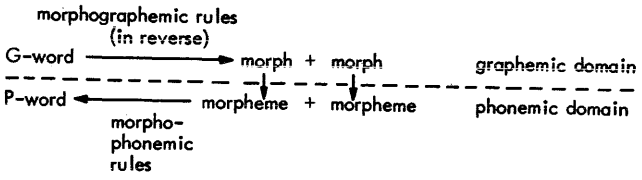


Figure 2 – G-word to P-word mapping

Identification of morphographemic rules of English

In order to describe the morphographemic rules of English, it is necessary to recognize certain properties of the morphs. We shall use the following subscripts to denote classes which possess certain properties.

a: A free morph that never combines with others, but always stands alone such as [me].

r: A free morph that may combine with others such as [house].

v: A bound vocalic suffix morph that has a vocalic beginning such as [able] in [readable].

c: A bound consonantal suffix morph that does not have a vocalic beginning such as [ness] in [kindness].

t: A bound terminal suffix morph that does not permit any other suffix morphs to be added to its right.

The bound morph [s] is both terminal, as well as consonantal, and the bound morph [es] is both terminal, as well as vocalic.

p: A prefix bound morph such as [un] in [unknown].

d: A morph that must have its final letter doubled when followed by a morph in class "v" such as [hit].

o: A morph that may optionally have its final letter doubled when followed by a morph in class "v" such as [model].

Let α,β denote sequences of letters, $\alpha\beta$ denote the concatenation of α and β, C denote a single consonantal letter that is preceded by a vowel letter, and T denote the final representation of the g-word. We can list the most often used morphographemic rules of English:

1. $(\alpha)_x \rightarrow T$; $x \epsilon \{a, r, d, o\}$ (free-morph rule)

2. $(\alpha)_x(\beta)_y \rightarrow (\alpha\beta)_y$; (compound word rule)
   $x, y, \epsilon \{r, d, o\}$

3. $(\alpha)_p(\beta)_x \rightarrow (\alpha\beta)_x$; (prefix rule)
   $x \epsilon \{r, d, o\}$

4. $(\alpha)_{r,o}(\beta)_{v,c} \rightarrow (\alpha\beta)_r$ (general suffix rule)

5. $(\alpha C)_{d,o}(\beta)_v \rightarrow (\alpha CC\beta)_r$ (doubling final consonant letter rule)

6. $(\alpha)_x(\beta)_t \rightarrow (\alpha\beta)_a$ (terminal-suffix rule)

7. $(\alpha' e')_r(\beta)_v \rightarrow (\alpha\beta)_r$ (mute final-e rule)

8. $(\alpha' y')_r(\beta)_x \rightarrow (\alpha' i'\beta)$; (final-y rule).
   $x \epsilon \{r, d, o, v, c\}$

These rules are to be interpreted in the following manner:

1. Free Morph Rule: Any letter sequence in class "a," or "r," or "d," or "o" may appear as a final form in a text. Since subscript "a" occurs on the the left-hand side of only this rule, a letter sequence in class "a" can only appear unchanged.
2. Compound Word Rule: This is the general rule for the formation of nonhyphenated compounds. For example, [snow]+[flake]→[snowflake]. The compound belongs to the same class as the second element, so $[over]+[bid]_d \rightarrow [overbid]_d$; thus, rule 5 can be applied to give $[overbid]_d + [ing]_v \rightarrow [overbidding]_r$.
3. Prefix Rule: This rule constrains the prefix to be attached only to the left, but it permits multiple prefixes. For example,
   [un]+[do]→[undo], and
   [re]+[undo]→[reundo]
4. General Suffix Rule: This is the general case of adding a suffix to the right of a letter sequence. For example,
   $[reason]_r + [able]_v \rightarrow [reasonable]_r$
   $[kind]_r + [ness]_c \rightarrow [kindness]_r$
5. Doubling Final Consonant Letter Rule: This rule gives us
   $[bid]_d + [ing]_v \rightarrow [bidding]_r$, as well as
   $[model]_o + [ing]_v \rightarrow [modelling]_r$. Rule 4 permits alternative construction in the second case as
   $[model]_o + [ing]_v \rightarrow [modeling]_r$.
6. Terminal Suffix Rule: This rule forbids the further appendage of suffixes after a terminal suffix has been used.
7. Mute Final-e Rule: This rule drops the final mute e of a morph when a vocalic suffix is appended. For example,
   $[move]_r + [ing]_v \rightarrow [moving]_r$. The construction
   $[mile]_r + [age]_v \rightarrow [mileage]_r$ is permitted under rule 4.
8. Final-y Rule: This rule changes the final letter y of the leading constituent into "i" when a

compound is formed or a suffix is appended. For example,

$[hand]_r+[y]\rightarrow[handy]_r$ according to rule 4,

$[handy]_r+[work]_r\rightarrow[handiwork]_r$ according to rule 8 and

$[dry]_r+[er]_v\rightarrow[drier]_r$ according to rule 8.

The alternative construction for $[dryer]_r$ is permitted under rule 4.

It must be emphasized here that while these rules are insufficient for the purpose of generating only legitimate g-words, well-formed English g-words encounter no difficulty in being decomposed into their constituent morphs. The decomposition of g-words into their constituent morphs is *parsing* on the word level.

There are many more morphographemic rules in English, but their utilization factors are very low. In the interest of efficiency in processing, only those listed here are actually implemented. As we shall see by restricting the implementation to only these listed rules, there is no sacrifice in the quality of translation if we can be somewhat liberal in the choosing of morphs for the dictionary.

The morph-to-morpheme dictionary

Webster's *New Collegiate Dictionary* (7th Edition) contains approximately 100,000 entry words. The inclusion of all paradigmatic forms not listed in the dictionary would probably double the count. Furthermore, with the freedom existing in the English language in forming compound words, it is difficult to put a theoretical upper bound on the total number of all possible words. By choosing to construct a dictionary on the basis of morph entries instead of g-word entries, the dictionary size can be brought down to a more manageable level. Our estimate is that with a dictionary containing 32,000 morphs, all entry g-words in Webster's Dictionary, plus all their paradigms and all reasonable compounds can be adequately decomposed. The storage for this dictionary is *read-only,* and is estimated to be on the order of 4,000,000 bits. For the reading machine project, we have been operating with a 3000-morph dictionary corresponding roughly to a Fourth Grade level.

An entry in the dictionary consists of four parts, the morph, which acts as the search and compare key in the decomposition process, the subscript marker, which directs the decomposition process, the morpheme, which is the target translation, and the parts-of-speech information. For those cases in which a morph leads to multiple morphemes, such as [refuse], (verb) and [refuse], (noun), the morpheme field and parts-of-speech field are repeated for each different morpheme.

Since the morphographemic rules indicate that when two morphs combine, the changes in the spelling occur only to the left morph, we have decided to scan a printed word from right to left during the decomposition process. The morphs in the dictionary, for this reason, are listed in reverse spelling order. The actual search is performed with an indexed sequential technique. The decomposition procedure is coded as a call to a recursive subroutine, which repeatedly calls upon itself until the g-word is successfully decomposed or when it has been determined that no decomposition is possible. Figure 3 illustrates the various stages of decomposition of the g-word [grasshopper]. It should be pointed out that the collating seq ience of the letters and morph terminating symbols is such that morph [*d*shop] is encountered before [*d*hop].

Given G-word: grasshoppers

First level match:	cs	Remainder: grasshopper
Second level match	ver	Remainder: grasshopp
		Modified to: grasshop
Third level match:	d shop	Remainder: gras
Fourth level match:	a as, rejected on basis of subscript a.	
No further fourth level match possible, return to third level: grasshop		
Third level match:	d hop	Remainder: grass
Fourth level match:	r grass	Remainder: (null)

Complete decomposition: $(grass)_r + (hop)_d + (er)_v + (s)_c$

Note: subscripts are underlined, e.g. cs, ver

Figure 3 – Example of a G-word decomposition

Selection of morphs for the dictionary

We have not established the criteria for the selection of morphs for storing in the dictionary. Let us start by proposing a set of simple criteria and examine them in some detail.

1. All base words are morphs. A base word is not a compound word, and contains no prefix or suffix of any kind. For example, [bake] is a morph.
2. All prefixes and suffixes that do not reflect changes in the pronunciation of the base words to which they may be attached are morphs. For example, [un, ing] are morphs.

From a linguistic point of view, all suffixes, regardless whether they meet the second criterion, ought to be treated as morphs. From an engineering point of view, the given restriction makes it unnecessary to process complex morphophonemic rules.

Between those suffixes that always change the pronunciation of the g-word remainder such as the suffix

[ity] and those that never do such as suffix [ing], there are many that affect it to intermediate degrees. For example, the suffix [able] does not usually affect the pronunciation of the base word, as in [attainable, removable, changeable], etc., but it does affect some base words, as in [inflammable, applicable], etc. In other cases the spelling of the base word itself is changed as in [tolerable, navigable], etc. In our approach, we choose to include as morphs such partially mutating suffixes, as well as those changed g-words. For example, we list [able] as a morph to achieve economy and list words like [inflammable, applicable, tolerable, navigable] as morphs, too. Briefly, whenever exceptions to a general rule have to be made, we can avoid rule complication by the creation of new morph entries.

Performance of the translator and the resultant synthetic speech without phrase analysis

Although the morph dictionary currently in use is derived from the cumulative word list of a fourth grade reader,* the eight morphographemic rules have been tested with representative samples from the Webster's 7th New Collegiate Dictionary. The vocabulary our system is capable of accepting is considerably larger than the reader from which the dictionary was derived. We have not yet made any provision to provide an approximate translation when the decomposition process fails. It seems that with a 32,000 morph dictionary, such occurrences should be very rare, except for proper names and misspellings, which may be handled in terms of letter-by-letter spelling.

After the grapheme-to-phoneme translator was implemented, we connected the output directly to the speech synthesizer control signal generator module, bypassing the phrase-analyzer. The parameters transmitted to the speech synthesizer control signal generator included only the segmental phonemes, primary and secondary stress markings on the morphemes and an indication of whether the sentence ends in a period or a question mark. We were quite encouraged by what we heard at the speech synthesizer output. We realized that for improved comprehension over longer utterances, additional processing of the data is essential. This additional processing is represented by the work on the phrase-analyzer which is presented in part 2 of this paper.

Roads to Everywhere, published by Ginn and Company, 1964.

As an added remark, it should be pointed out that the morph decomposition idea can be used to greatly reduce the size of the dictionary needed by mechanical translation of natural languages.

ACKNOWLEDGMENT

This work was supported principally by the National Institutes of Health (Grant 1 PO1 GM-14940-01), and in part by the Joint Services Electronics Program [Contract DA 28-043-AMC-02536 (E)].

REFERENCES

1 G C PATTON
Touch-tone input and audio reply for on-line calculation
Bell Telephone Laboratories Internal Memorandum March 29 1967

2 D E TROXEL F F LEE S J MASON
A reading machine for the blind
Digest of the 7th International Conference on Medical and Biological Engineering 1967 Stockholm

3 J L KELLY C LOCKBAUM
Speech synthesis
Proceedings of the Speech Communication Seminar 1962 Stockholm

4 L J GERSTMAN J L KELLY
An artificial talker driven from a phonetic input
Journal of Acoustical Society of American vol 33 1961 p 835 (A)

5 J N HOLMES I G MATTINGLY J N SHEARME
Speech synthesis by rule
Language and Speech vol 7 part 3 July-September 1964 pp 127-143

6 I G MATTINGLY
Synthesis by rule of prosodic features
Language and Speech vol 9 Part 1 January-March 1966 pp 1-13

7 E M HIGGINBOTTOM
A study of the representation of English vowel phonemes in the orthography
Language and Speech April-June 1962 pp 67-117

8 B V BHIVANI J L COLBY H L RESNIKOFF
Acoustic phonetic transcription of written English
Paper delivered at the 68th meeting of the Acoustic Society of American October 21 1964

9 R H WEIR and VENEZKY
Formulation of grapheme-phoneme correspondence rules to aid in the reaching of reading
Cooperative Research Project No S-139 Report Stanford Univ 1964

10 G MONROE
Phonemic transcription of graphic post-base suffixes in English A computer problem
PhD Thesis Brown Univ June 1965

11 S KLEIN R F SIMMONS
A computational approach to grammatical coding of English words
Journal of the Association for Computing Machinery vol 10 No 3 July 1963 pp 334-347

Machine-to-man communication by speech Part II: Synthesis of prosodic features of speech by rule

by JONATHAN ALLEN
Research Laboratory of Electronics, Massachusetts Institute of Technology
Cambridge, Massachusetts

For several years, research has gone on in an attempt to develop a reading machine for the blind. Such a machine must be able to scan letters on a normal printed page, then recognize the scanned letters and punctuation, and finally convert the resultant character strings into an encoded form that may be perceived by some nonvisual sensory modality. Within recent years, at the Massachusetts Institute of Technology, an opaque scanner has been developed,[1] and an algorithm for recognizing scanned letters has been devised.[2] The output display can take many forms, but the form that we feel is best suited for acceptably high reading speeds and intelligibility is synthesized speech. Effort has recently been focused on the conversion of orthographic letter strings to synthesized speech.

An algorithm for grapheme-to-phoneme conversion (letter representation to sound representation) has been invented by Lee,[3] which is capable of specifying sufficient phonemic information to a terminal analog speech synthesizer for translation to synthesizer commands. The algorithm uses a dictionary to store the constituent morphs of English words, together with their phonemic representation. Hence each scanned word is transformed into a concatenated string of phonemic symbols that are then interpreted by the synthesizer.

The resulting speech is usually intelligible, but not suitable for long-term use. Several problems remain, apart from those concerned directly with speech synthesis by rule from phonemic specifications. First, many words can be nouns or verbs, depending on context [refuse, incline, survey], and proper stress cannot be specified until the intended syntactic form class is known. Second, punctuation and phrase boundaries may be used to specify pauses that help to make the complete sentence understandable. Third, more complicated stress contours over phrases can be specified which facilitate sentence perception. Finally, intonation contours, or "tunes" are important for designating statements, questions, exclamations, and continuing or terminal juncture. These features (stress, intonation, and pauses) comprise the main prosodic or suprasegmental features of speech.

Several experiments[4,5,6] have shown that we tend to perceive sentences in chunks or phrasal units, and that the grammatical structure of these phrases is important for the correct perception of the sentence. In order to display this required structure to a listener, a speaker makes use of many redundant devices, among them the prosodic features, to convey the syntactic surface structure. When speech is being synthesized in an imperfect way at the phonemic level, the addition of these additional features can be used by listeners to compensate for the lack of other information. The listener may then use these cues to hypothesize the syntactic structure, and hence generate his own phonetic "shape" of the perceived sentence. There is little reason to believe that the perceived stress contour, for example, *must* represent some continuing physical property of the utterance, since the listener uses some form of internalized rules to "hear" the stress contour, whether or not it is physically present in a clear way. Hence, once the syntactic surface structure can be determined, the "stress" can be heard. Alternatively, prosodic features can be used in a limited fashion to help point out the surface structure, which is then used in the perception of the phonetic shape of the sentence.

The present paper describes a procedure for parsing sentences composed of words that are in turn derived from the morphs provided by the grapheme-to-phoneme decomposition, as well as a phonological procedure for specifying prosodic features over the

revealed phrases. As we have indicated, only a limited amount of the sentence is parsed and provided with prosodics, since the listener will "hear" the entire sentence once the structure is clear. We consider first the required parts-of-speech preprocessor, then the parser, and finally the phonological algorithm.

Parts-of-speech preprocessor

After the grapheme-to-phoneme conversion is complete, many words will have been decomposed into their constituent morphs. For example, [grass-hopper] → [grass] + [hop] + [er], and [brow-beat] → [brow] + [beat]. Each of these morphs corresponds to a dictionary entry that contains, in addition to phonemic specifications, parts-of-speech information. In the case of morphs that can exist alone ([grass, hop, brow,] etc.) this information consists in a set of parts of speech for that word, called the grammatical homographs of the word, and this set often has more than one homograph. For prefixes and suffixes ([re-, -s, -er, -ness,] etc.), information is given indicating the resultant part of speech when the prefix or suffix is concatenated with a root morph. Thus [-ness] always forms a noun, as in [goodness] and [madness].

Other researchers[7,8] have used a computational dictionary to compute parts of speech, relying on the prevalence of function words (determiners, prepositions, conjuctions, and auxiliaries), together with suffix rules of the type just described and their accompanying exception lists. This procedure, of course, keeps the lexicon small, but results in arbitrary parts-of-speech classification when the word is not a function word, and does not have a recognizable suffix. Furthermore, ambiguous suffixes such as [-s] (implying plural noun or singular verb) carry over their ambiguity to the entire word, whereas if the root word has a unique part of speech like [cat], our procedure gives a unique result; [cats] (plural noun). Hence the presence of the morph lexicon can often be used to advantage, especially in the prevalent noun/verb ambiguities.

The parts-of-speech algorithm considers each morph of the word and its relation with its left neighbor, starting from the right end of the word. If there are two or more suffixes [commend*ables*, topic*ality*] the suffixes are entered into a last-in first-out push-down stack. Then the top suffix is joined to the root morph, and the additional suffixes are concatenated until the stack is empty. Compounding is done next, and finally any prefixes are attached. Prefixes generally do not affect the part of speech of the root morph, but [em-, en-,] and [be-] all change the part of speech to verb. Compounds can occur in English in any of three ways, and there appears to be no reliable method for distinguishing these classes. There can, of course, be two separate words, as in [bus stop], or two words hyphenated, as in [hand-cuff], or finally, two root words concatenated directly, as in [sandpaper]. The parts-of-speech algorithm treats the last two cases, leaving the two-word case for the parser to handle. The algorithm ignores the presence of a hyphen, except that it "remembers" that the hyphen occurred, and then processes hyphenated and one-word compounds as though they were both single words. The parts of speech of the two elements of the compound are considered as row and column entries to a matrix whose cells yield the resulting part of speech. Thus Adverb·Noun → Noun ([under-world]). In general, since each element may have several parts of speech, the matrix is entered for each possible combination, but the maximum number of resulting parts of speech is three. Combinations of suffixes with compounds ([handwriting]) can be accommodated, as well as one-word compounds containing more than two morphs.

The algorithm has a special routine to handle troublesome suffixes such as [-er, -es, -s], in an attempt to reduce the resulting number of parts of speech to a minimum.

In this way, the algorithm makes use of the parts of speech information of the individual morphs to compute the parts of speech set for the word formed by these morphs. These sets then serve as input to the parser, after having first been ordered to suit the principles of the parser.

Parsing

As we have remarked, if a listener is aware of the surface syntactic structure of a spoken sentence, then he may generate internally the accompanying prosodic features to the extent that they are determinable by linguistic rules forming part of his language competence. Hence we desire to make this structure evident to the listener by providing cues to the syntax in the prosodics of the synthesized speech. To do this, we must first determine the structure, and then implement prosodics corresponding to the structure. Since we are trying to provide only a limited number of such cues (enough to allow the structure to be deduced), we have designed a *limited parser* that reveals the syntax of only a portion of the sentence. We have tried to find the simplest parser consistent with these phonological goals that would also use minimum core storage and run fast enough (in the context of the over-all reading machine) to allow for a realistic speaking rate, say, 150-180 words per minute. Because the absence, or incorrect implementation of prosodics in a small

percentage of the output sentences, is not likely to be catastrophic, we can tolerate occasional mistakes by the parser, but we have tried to achieve 90 per cent accuracy. These requirements, for a limited, phrase-level parser operating in real-time at comfortable speaking rates within restricted core storage, are indeed severe, and many features found in other parsers are absent here. We do not use a large number of parts of speech classifications, nor do we exhaustively cycle through all the homographs of the words of a sentence to find all possible parsings. Inherent syntactic ambiguity ([They are washing machines]) is ignored, the resulting phrase structures being biased toward noun phrases and prepositional phrases. No deep-structure "trees" are obtained, since these are not needed in the phonological algorithm, and only noun phrases and prepositional phrases are detected, so that no sentencehood or clause-level tests are made. We do, however, compute a bracketed structure *within* each detected phrase, such as [the [old house]] and [in [[brightly lighted] windows]], since this structure is required by the phonological algorithm. The result is a context-sensitive parser that avoids time-consuming enumerative procedures, and consults alternative homographs only when some condition is detected (such as [to] used to introduce either an infinitive or a prepositional phrase) which requires such a search.

The parser makes two passes (left-to-right) over a given input sentence. The first pass computes a tentative bracketing of noun phrases and prepositional phrases. Inasmuch as this initial bracketing makes no clause-level checks and does not directly examine the frequently occurring noun/verb ambiguities, it is followed by a special routine designed to resolve these ambiguities by means of local context and grammatical number agreement tests. These last tests are also designed to resolve noun/verb ambiguities that do not occur in bracketed phrases, as [refuse] in [They refuse to leave.]. As a result of these two passes, a limited phrase bracketing of the sentence is obtained, and some ambiguous words have been assigned a unique part of speech, yet several words remain as unbracketed constituents.

The first pass is designed to quickly set up tentative noun phrase and prepositional phrase boundaries. This process may be thought of as operating in three parts. The program scans the sentence from left to right looking for potential phrase openers. For example, determiners, adjectives, participles, and nouns may introduce noun phrases, and prepositional phrases always start with a preposition. In the case of some introducers, such as present participles, words further along in the sentence are examined, as well as previous words, to determine the grammatical function of the participle, as in [*Wiring circuits* is fun.] Once a phrase opener has been found, very quick relational tests between neighboring words are made to determine whether the right phrase boundary has been reached. These checks are possible because English relies heavily on word order in its structure. Having found a tentative right phrase boundary, right context checks are made to determine whether or not this boundary should be accepted. After completion of these checks, the phrase is closed and a new phrase introducer is looked for. This procedure continues until the end of the sentence is reached.

When the bracketing is complete, further tests are made to check for errors in bracketing caused by frequent noun/verb ambiguities. For example, the sentence [That old man lives in the gray house.] would be initially bracketed.

[That old man lives]$_{NP}$ [in the gray house]$_{PREP\ P}$.

Notice that sentencehood tests (although not performed by the parser) would immediately reveal that the sentence lacks a verb, and further routines could deduce that [lives], which can be a noun (plural) or verb (third person singular), is functioning as a verb, although the bracketing routine, since it is biased toward noun homographs, made [lives] part of the noun phrase. We also note the importance of this error for the phonetic shape of the sentence, since [lives] changes its phonemic structure according to its grammatical function in the sentence. An agreement test, however, compares the rightmost "noun" with any determiners that may reflect grammatical number. In this case, [that] is a singular demonstrative pronoun, so we know that [lives] does not agree with it, and hence must be a verb. After the agreement test has been made for each noun phrase, local context checks are used in an attempt to remove noun/verb ambiguities that are important for the phonological implementation, and yet have not been bracketed into phrases containing more than one word. Thus in the sentence [They *produce* and develop many different machines.], the algorithm would note that [produce] is immediately preceded by a personal pronoun in the nominative case, and hence the word is functioning as a verb. Such knowledge can then be used to put stress on the second syllable of the word in accordance with its function.

At the conclusion of the parsing process described above, phrase boundaries for noun phrases and prepositional phrases have been marked, but the structure within the phrase is not known. In order to apply the rules that are used for computing stress patterns within the phrase, however, internal bracketing must be

provided. For this reason, determiner-adjective-noun sequences are given a "progressive" bracketing, as [the [long [red barn]]], whereas noun phrases beginning with adverbials are given "regressive" bracketings, as [[[very brightly] projected] pictures]. A preposition beginning a prepositional phrase always has a progressive relation to the remaining noun phrase, so that we have [in [the [long [red barn]]]] and [of [[[very brightly] projected] pictures]] Furthermore, two nouns together, as in [the local *bus stop*], are marked as a compound for use by the phonological algorithm.

The procedure described above is thus able to detect noun phrases and prepositional phrases and to compute the internal structure of these phrases. The grammar and parsing logic are intertwined in this procedure, so that an explicit statement of the grammar is impossible. Nevertheless, the rules are easily modified, and additions can readily be made. If, for example, we decide to detect verbal constructions, this could easily be done. At present, however, we feel that recognition of noun phrases and prepositional phrases and the provision of prosodics for these phrases is sufficient to allow the listener to deduce the correct syntactic structure for large samples of representative text.

Phonological algorithm

The method for detecting and bracketing noun phrases and prepositional phrases has now been described. We assume that this surface structure is sufficient to allow the specification of stress and intonation within these phrasal units. The basis for this assumption is given in the work of Chomsky and Halle.[9] The phonological algorithm then uses the surface syntactic bracketing, plus punctuation and clause-marker words, to deduce the pattern for stress, pauses, and intonation related to the detected phrases.

In the present implementation, only three acoustic parameters are varied to implement the prosodic features. These are fundamental frequency (f_0), vowel duration, and pauses. It is well known that juncture pauses have acoustic effects on the neighboring phonemes other than vowel lengthening and f_0 changes, but these effects are ignored in the present synthesis. We thus consider f_0, vowel duration, and pauses to constitute an interacting parameter system that serves as a group of acoustic features used to implement the prosodics. The "sharing" of f_0 for use in marking both stress and intonation contours is another example of the interactive nature of these acoustic parameters.

Stress is implemented within the detected phrases by iterative use of the stress cycle rules, described by Chomsky and Halle.[9] These rules operate on the two constituents within the innermost brackets to specify where main stress should be placed. All other stresses are then "pushed down" by one. (Here, "one" is the highest stress.) The innermost brackets are then "erased," and the rules applied to the next pair of constituents. This cycle is then continued until the phrase boundaries are reached. For compounds, the rules specify main stress on the leftmost element (compound rule), whereas for all other syntactic units (e.g., phrases) main stress goes on the rightmost unit (nuclear stress rule). For example, we have

```
[the [long [red barn]]]
                2    1
  4    2    3    1
```

where initially stress is 1 on all units except the article *the*, and two cycles of the phrase rule are used. The parser has, or course, provided the bracketing of the phrase. Also,

```
[in [[[very brightly] lighted] rooms]]
      2    1
      3    2    1
4     4    3    2    1
```

requires three applications of the rules, and

```
[the [new [bus stop]]]
                1    2
  4    2    1    3
```

which contains a compound, requires two iterations. It is clear that for long phrases requiring several iterations, say n, there will be n + 1 stress levels. Most linguists, however, recognize no more than four levels, so the algorithm clips off the lower levels. At present, three levels are being used, but this limit can be easily changed in the program.

In the examples it has been implicitly assumed that each content word started with main stress before the rules were applied. Each word does have *a* main stress initially, but in general each word has its own stress contour, as, for example, in the triple [nation, national, nationality]. (As Lee[3] has pointed out, pairs such as [nation/national] can be handled by placing the two words directly in the morph dictionary, but we have tried to extend the stress algorithm to cover many of these cases. Clearly, there is a compromise between processing time and dictionary size to be determined by experience.) Thus the algorithm must compute the stress for individual words by applying rules for compounds and suffixes. The compound rule is the same as for two separate words that comprise a compound (e.g., [bus stop, browbeat]). Each morph in the lexicon is given lexical stress, so that an initial stress contour is provided. Each suffix is also provided with information about its effect on

stress. Hence [-s, -ed] and [-ing] all leave the root morph stress unaltered, and have the lowest level stress for themselves. Another example is the suffix [-ion], which always places main stress on the immediately preceeding vowel (e.g., [nationalization, distribution]). At present, such changes in stress of the root word are not computed by rule. In this way, stress contours for individual words are first computed, and then these are "placed" in the bracketed phrase structure and the stress cycle is applied until the over-all stress pattern is obtained for the whole phrase. Note that function words receive no stress, so that stress is controlled for these words, even though they do not appear in bracketed phrases.

Pauses are provided in a definite hierarchy throughout each sentence. The following disposition of pauses has been arrived at empirically, and represents a compromise between naturalness and intelligibility. At present, no pauses are used within the word at the juncture between any two morphs. Within a bracketed phrase or between two adjacent unbracketed constituents no pauses are used between words. At phrase boundaries, pauses of 200 to 400 msec have been used to set off the detected phrase. Short pauses of 100 msec are used where commas and semicolons appear, and pauses of 200 msec are inserted before clause-marker words such as [that, since, which] etc., which serve to break up the sentence into clausal units. Finally, terminal pauses of 800 msec are provided for colon, period, question mark, and exclamation point. Thus a hierarchy of pauses is used to help make the grammatical structure of the sentence clear.

The provision of intonational f_0 contours by rule has been described by Mattingly,[10] and our technique is similar to his. The slope of the f_0 contour is controlled by the specific phonemes encountered in the sentence, and by the nature of the pause at the end of the phrasal unit. Rising terminal contours are specified at the end of interrogative clauses just preceding the question mark, except when the clause starts with a [wh-] word, as [where is the station?]. In the absence of a question mark, the intonation f_0 contour is falling with a slope determined by rule as is done by Mattingly.

The starting point for f_0 at the beginning of a sentence is fixed at 110 Hz. The jumps in f_0 for the various stress levels vary with the initial value of f_0, but nominally they are 12, 18, and 30 Hz corresponding to the stress levels 3, 2, and 1 respectively. As noted before, 1 corresponds to the *highest* stress in our system.

Subjective experience

The method of implementing prosodic features on the limited basis described above has been used in connection with the TX-0 computer at M.I.T., driving a terminal analog synthesizer. While the resulting speech is still unnatural in many respects, a substantial improvement in speech quality has been attained. It appears that by using limited phrase level parsing and implementation of prosodics mainly within these phrases, sufficient cues can be provided to the listener to enable him to detect the grammatical structure of the sentence and hence provide his own internal phonetic shape for the sentence. Since this system will become part of a complete computer-controlled reading machine operating in real time, it is encouraging to find that such a limited approach is able to improve the speech quality markedly. We anticipate that further work on both phonemic and prosodic synthesis rules will yield even greater intelligibility and naturalness in the output speech, with little additional computing load placed on the system.

DISCUSSION

The speech synthesis system described here has been developed for research purposes. Hence the implementation of our speech synthesis system has remained very flexible so that further improvements can be easily accommodated. Better rules for phonemic synthesis are being developed, and will be incorporated into the system. Much work remains to be done on the determination of the physiological mechanisms underlying stress, and the resultant observable phonetic patterns which arise from these articulations. Particular attention is being focused on the nature and interaction of f_0 and vowel duration as correlates of stress. There will also undoubtedly be further improvements in the parsing procedure as experience dictates. From the linguistic point of view, the lexicon for a language should contain only the idiosynchrasies of a language, everything derivable by rule being computed as part of the language user's performance. Engineering considerations, however, clearly dictate a compromise with this view, and the cost of memory versus the cost of computing with an extensive set of rules must be examined further. It may, for example, become feasible to compute lexical stress by rule, but any advantages of this procedure must outweigh the cost in time and program storage for these rules.

ACKNOWLEDGMENTS

This work was supported principally by the National Institutes of Health (Grant 1 PO1 GM-1490-01) and in part by the Joint Services Electronics Program (Contract DA28-043-AMC-02536(E)); additional

support was received through a fellowship from Bell Telephone Laboratories, Inc.

REFERENCES

1 C L SEITZ
An opaque scanner for reading machine research
SM Thesis MIT 1967

2 J K CLEMENS
Optical character recognition for reading machine applications
Doctoral Thesis MIT 1965

3 F F LEE
A study of grapheme to phoneme translation of English
PhD Thesis MIT 1965

4 G A MILLER
Decision units in the perception of speech
IRE Transactions on Information Theory Vol IT-8 No 2 P 81 February 1962

5 G A MILLER G A HEISE W LICHTEN
The intelligibility of speech as a function of the contest of the test materials
J Exptl Psychol 41 p 329 1951

6 G A MILLER S ISARD
Some perceptual consequences of linguistic rules
J Verb Learn Verb Behav 2 p 217 1963

7 S KLEIN R F SIMMONS
A computation approach to the grammical coding of English words
J Assoc Computing Machinery 10 334 1963

8 D C CLARKE R E WALL
An economical program for the limited parsing of English
AFIPS Conference Proceedings p 307 Fall Joint Comp Conf 1965

9 N CHOMSKY M HALLE
Sound patterns of English
(in press)

10 I G MATTINGLY
Synthesis by rule of prosodic features
Language and Speech 9 1 1966

An on-line multiprocessing interactive computer system for neurophysiological investigations

by FREDERICK D. ABRAHAM
Brain Research Institute, Neuropsychiatric Institute and Psychiatry, UCLA
Los Angeles, California
and
LASZLO BETYAR and RICHARD JOHNSTON
Data Processing Laboratory, Brain Research Institute, UCLA
Los Angeles, California

INTRODUCTION

The principal dependencies of neurophysiologists upon the computer are for data collection and analysis, experimental control, and the development of theoretical models. One possible system providing these functions is one that allows several investigators to on-line time-share a moderate sized digital computer capable of performing input, output, and computational functions in a simple interpretive language that is easy to understand and use in a fast decision experimental environment. A community of neurophysiologists in the UCLA Brain Research Institute share such a computer in its data processing laboratory (DPL) by means of remote console stations in the investigators' laboratories connected to the DPL by a direct cabling system.[5,15] A larger computer facility, available to a larger community of health scientists, is used for batch processing where problems do not need continuous interaction with the investigator for on-line control or analysis, or do need greater computational capability.[9] The two facilities possess compatible I/O formats, thus making some problems soluable by the combination of both computers, and giving other problems the flexibility of either approach. Essentially the DPL is a multiprocessing system[12,18,21] with an emphasis on I/O functions and an interpretive system appropriate for neurophysiological investigation and with some unique solutions to resource allocation and system integrity in its temporal-spatial (core) algorithm. The economic advantage of such a system is not argued, nor is CPU economy necessarily maximized with present use, though from the standpoint of I/O devices which are so important for such research, a central facility may possess some advantages. Reliability and demands of time-critical users must be realistically estimated for neurophysiological users for whom the on-line aspect may be with reference to the integrity of their experiments.

Computer facilities and functions

A. The basic DPL hardware system

The central processor unit (CPU) is a Scientific Data Systems 9300 general purpose digital computer, with a 32K, 24 bit word memory, three time-multiplexed communication channels (TMCC), and two direct access communication channels (DACC) (Fig. 1). Seven magnetic tape units are on one TMCC. Character devices including printer, card reader, plotters, typewriter, paper tape reader, and paper tape punch are on another TMCC. A 30-channel analog-to-digital (A/D) converter and a 16-channel A/D converter are available on another TMCC. The A/D's have a common multiplexer with a 100 Khz conversion rate and a precision of 10 bits plus sign. An 8M character disc storage is on a DACC. A digital-to-analog converter (D/A) is on the other DACC. There is also a 24 bit parallel input of sense lines capable of detecting 24 channels of discrete laboratory events, and a 24 bit parallel output of relay drivers capable of operating 24 channels of discrete events in the investigators' laboratories. The system interface unit contains the A/D multiplexer, the D/A connector, the 24 relay drivers and their connections, the 24 sense line connections, and a 6 bit character buffer for input from the consoles in the investigators' laboratories. The remote consoles (Fig. 2 & 3) located in the investigators' laboratories include a 64 key lighted keyboard and a storage oscilloscope. The console also

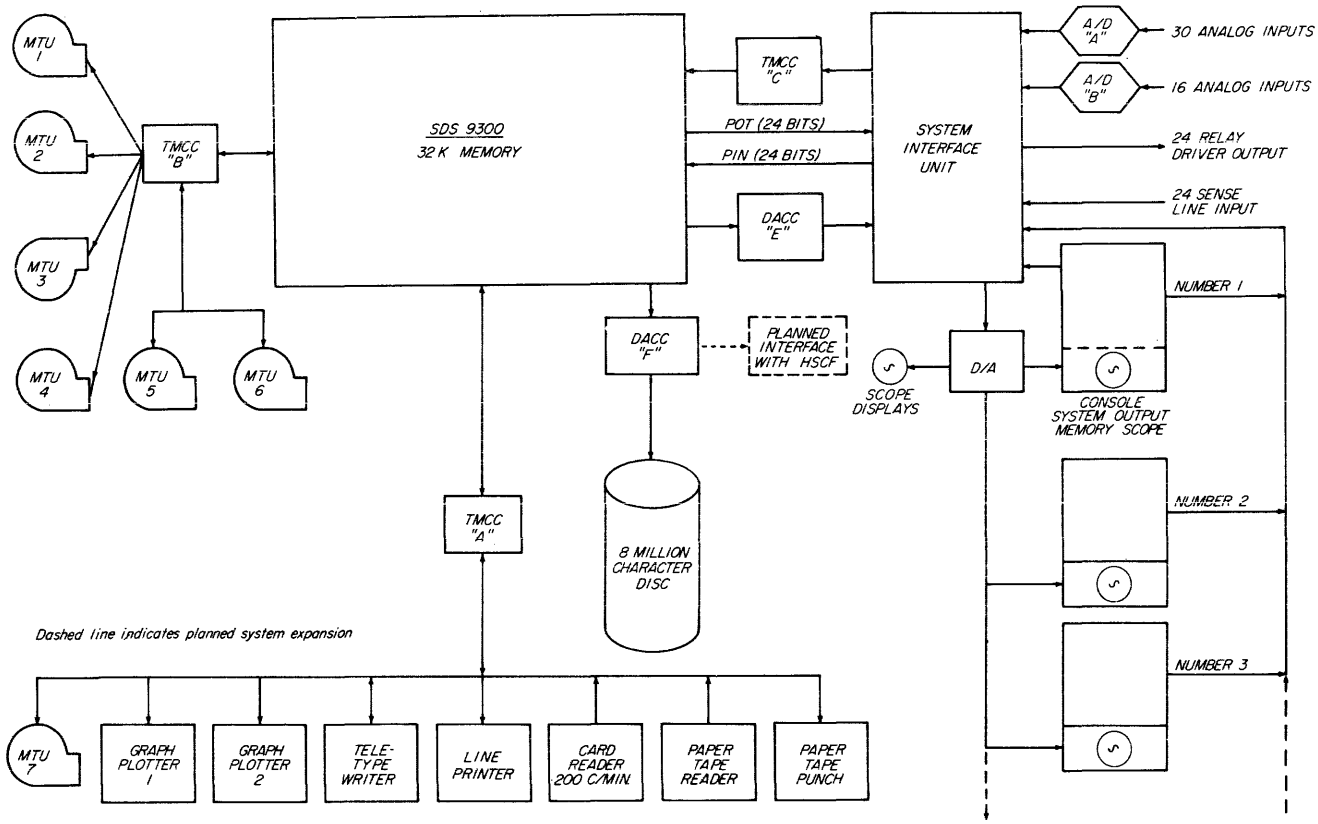


Figure 1 – DPL hardware configuration

contains a few lights to indicate the status of a few computer functions. The console (differing from others[7,8,13,16]) represents two most important features of the system; the on-line time-sharing feature, and the function of a simple communication between computer and researcher. The consoles allow on-line multiprogramming and multiprocessing of programs already entered via the console or other routes. Its keys are mnemonically labelled and lighted. The mnemonic labels indicate the most basic functions available to the user in terms of events and computations familiar to his research. The lights indicate whether the keyboard is in an upper or lower case mode of operation. The storage oscilloscope is capable of both graphic and alphanumeric display.

B. The basic DPL software system

The executive system and the interpreter (Shared Laboratory Interpretive Processor, SLIP) for the users' multiprocessing are resident in core. This system performs the reading, interpreting, and processing of console keypresses, and the generation of scope displays. In addition, the computer operator may execute various background activities which take advantage of idle CPU time. These background activities represent the normal production activity of the DPL and include A/D conversion, "off-line" plotting on two Calcomp digital plotters, "off-line" printing of assembler output, card-to-tape conversion, and routine tests of digital tapes. At present, there is no capability to time-share routine compilations and assemblies. These operations are performed at scheduled times under the standard SDS supplied monitor system.

Console activities may be controlled through a continuous sequence of keypresses, performed in a direct execution mode of available public users' programs. Additional private users' programs may also be written in this mode, and then used in a program execution mode. Any key press may have two types of meanings according to whether the keyboard is in upper or lower case. In upper case the keys have an operator (function) meaning; in lower case they usually have an operand (data) meaning. Each key may have up to 60 operator meanings, the first ten of which are written as machine language subroutines as part of the interpreter, representing a compiler in provid-

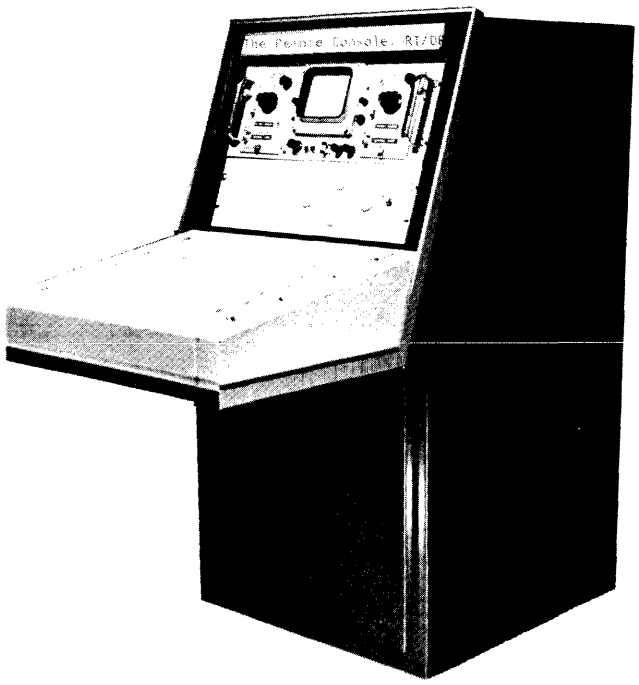


Figure 2 – Console

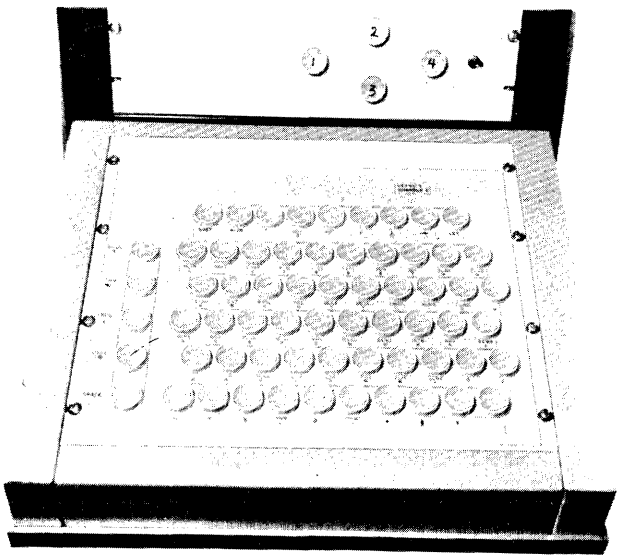


Figure 3 – Console keyboard

ing a language to the user in terms of I/O and computational functions required in this research (differing from other interactive interpretive languages[7,8,11] and medical systems[16]). The remaining 50 operator definitions per key are optional sequences of other operator and operand meanings. The first ten meanings per button are public routines likely to be used by many investigators but modifiable by none (currently, only the first of the ten exists for each key, see Table 1). The remaining 50 meanings are available for private multiprogramming usage. That is, each investigator, by his reference identification, may define 50 more programs per key that are usable and modifiable only by himself (resident on disc or magnetic tape). Similarly, the operand (data) meanings, locatable by a special indexing procedure, are private, with up to 217,088 pieces of data storable on each of 53 keys. All operations are performed in the console "accumulator" (the principal working register of the computer) which may hold a vector of up to 250 real or 125 complex numbers. Operators may be unary, that is operate on the contents of the accumulator leaving the keyboard in operator (upper case) mode. Plotting (graphic scope display of accumlator contents), and Sine (replacing the contents of the register with their sine) are examples of unary operators. Other operators are binary, requiring additional data which must be obtained in the operand mode after execution of the binary operator. A terminator key press in operand mode returns the keyboard to operator mode after the data operand is defined by appropriate key presses. For example, a vector may be added to another, replacing the register with the vector sum. The data may be generated as needed by key presses representing numbers, or may be called from storage by key presses representing its location indexing. Programs and data are placed on disc as they are defined and may remain there safely for short term needs such as a week or so. If they are desired to be kept longer, they may be dumped into magnetic tape or punched cards and reloaded at a later time. Data may be collected from experiments in either a continuous or triggered fashion for spectral transient response, or spike analysis. The data are disc stored with appropriate indexing for further console operations, or stored on magnetic tape in IBM/360 format for subsequent analysis at the other computer facility.

C. Uniqueness and I/O capabilities

An important difference between SLIP and other similar time-sharing remote console systems[7,8] is the availability to the console user of specialized hardware for data collection and experimental control. These capabilities are supplied in addition to a comprehensive set of mathematical and I/O operations. Other major differences between SLIP and other time-sharing systems is in the space (core memory) and time algorithm. Space is allocated dynamically on an as-required basis for each user. Programs and data are saved on disc, and are automatically read in when referenced. More frequently used data and programs remain in core for longer periods of time. With respect to time allocation, each user may execute one function, and then the computer will automatically service others on a cyclic commutator basis. Time consuming operations such as convolution relinquish the CPU several

times before completion. Time-critical (synchronous) activities such as A/D conversion are interrupt driven and are given the highest system priority. I/O channel access is on a first come, first served basis.

D. The Health Science Computer Facility (HSCF).[9]

While the DPL is a moderately sized facility, it is specifically geared for the treatment of analog neurophysiological data as well as the on-line time-shared macrolanguage used by the neurophysiological investigator. For many problems, data analysis need not necessarily be performed on-line or with the investigator's continual badgering. A larger, batch processing 360 system available to a larger community of scientists is used for such problems, taking input prepared by the DPL facility and returning output that is given its final form, e.g., plotted D/A data analysis displays, at the DPL. Separate programs are used at each facility on their respective disc storages. No direct link yet exists between the two facilities, and principal data communication between them is via I/O IBM/360 format compatible magnetic tapes, with additional program control and parameter entry to the HSCF computer with punched cards. Examples showing the various options available with these computers for use with the principal types of neurophysiological research follow. Other medical systems have been reviewed elsewhere.[10,16,17]

Exemplary neurophysiological control and analysis

A. EEG spectral analysis in behavioral experiments[1,3,4,6,14,20]

Experimental control with data collection, analysis, and display for cats in chronic learning situations while measuring EEG activity from several brain sites provides an excellent example of the interaction between the investigator's laboratory and both computer facilities (Fig. 4). The DPL console system is used to control the experimental situation with a relay-driving program, while simultaneously A/D converting several channels of amplified EEG activity and time and stimulus codes with another program that places the data on a digital magnetic tape in IBM/360 Fortran compatible format. The investigator uses prepared programs and uses the console to call them into use, to enter parameters concerning relay driving conditions and digitizing parameters, to enter labels, and to initiate and terminate the relay driving and A/D routines. Spectral analysis of the data is preformed at the HSCF by submitting the data tape together with control cards for both their monitor system and a user's spectral analysis program resident on disc.[4,9] Both listed and magnetic tape or disc outputs are obtained, and subsequent programs at the HSCF further prepare the data, with or without averaging, for various types of data display. The data displays are obtained at the DPL facility with background programs that can read the IBM tapes and control either Cal-comp plotters or oscilloscopic data displays. Computer operators at DPL usually perform this task, as the investigator often does not wish to view the data until it is complete. This mode of data treatment is quite fine where a predetermined exhaustive data analysis is required from a compulsively well designed experiment. In pilot experiments, one may wish to perform partial analyses in an attempt to determine which features are of greatest interest. Spectral analysis programs written for the DPL console system are in progress to do this. Essentially the trade off is initially increased investigator time for savings of subsequent HSCF time by eliminating analysis of unnecessary aspects of the data. Development of new analyses and displays can be quite efficient with such a system as well. Some of the types of data displays from spectral and related time-series analysis include three dimensional maps of spectral power, co-spectral power, coherence, and phase as a function of time (successive, selected samples) (Fig. 5A). Also, coherence and phase as a function of frequency for a given sample; and maps of brain locations depicting phase and coherence for a given EEG frequency band are very useful displays (Fig. 5B). Phase is usually converted to a time measure rather than an angular measure.

B. Evoked potentials in sensory experiments[2,6]

An experiment on auditory electrophysiology provides another example of the various options available with such a computer system in efficiently yielding statistical parameters and sequential analysis when averaging transient neural responses. In this case, experimental control is usually laboratory rather than computer based, while data collection is via the console system, differing only from the EEG in detecting pulse events and digitizing only a brief (usually 50 - 1000 msec) stretch of data after each pulse which includes the transient neural response to auditory stimuli. The data may be stored in disc with the console system indexing and analyzed as it is collected, as such data analyses may be performed much more rapidly than spectral analysis. Thus one could have such an average evoked response with several additional statistical properties displayed on the oscilloscope and photographed (with the computer relay drivers controlling photography if one wishes) within milliseconds after the presentation of the last stimulus in a series to an animal. This could be of use in making consequent experimental decisions, or in

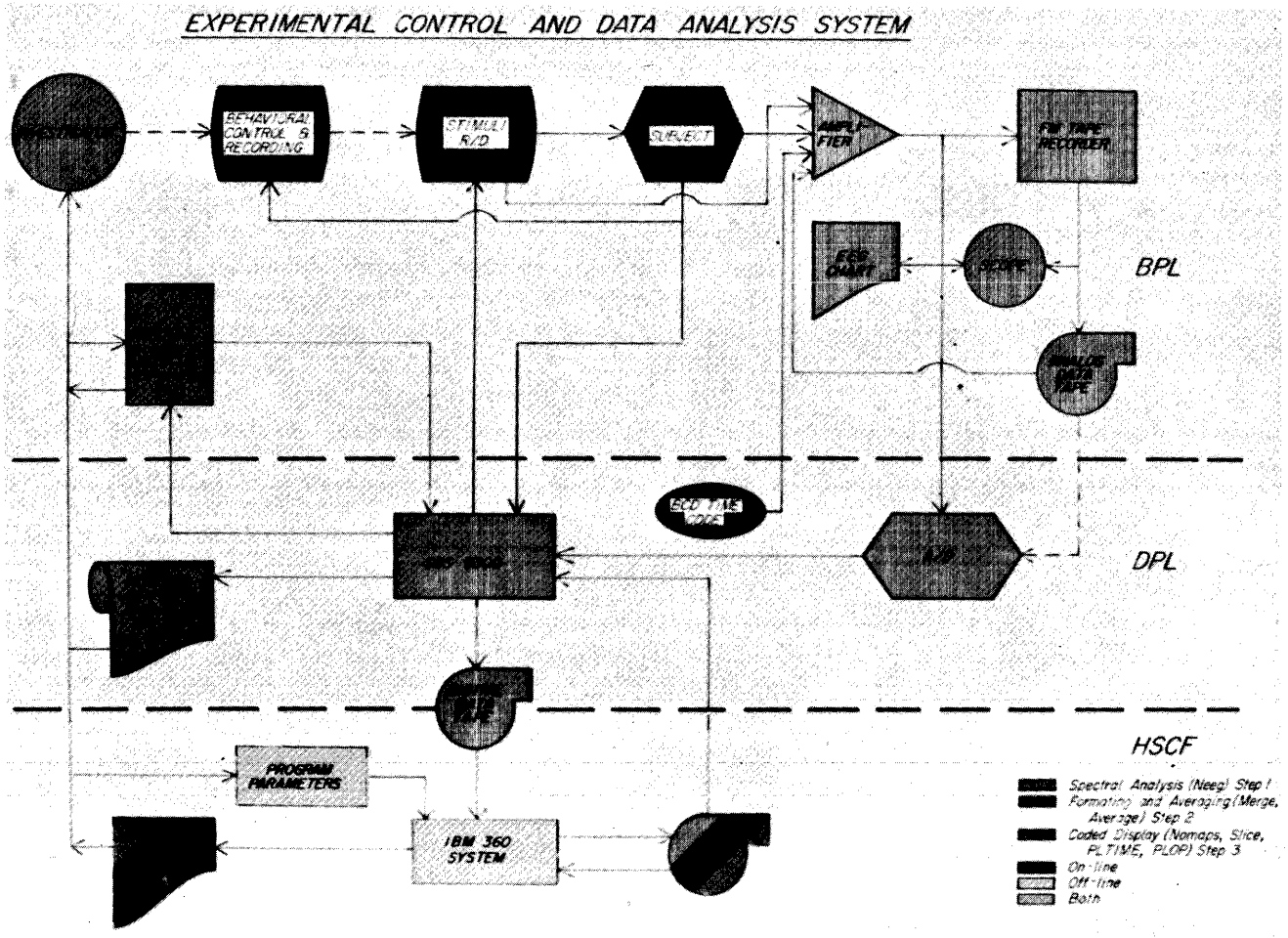


Figure 4 – Experimental control and analysis flow

further data analysis decisions, or simply be convenient in eliminating additional turnover steps that use of the other computer facility would require. However, if the analysis needs were sufficiently predetermined, or required many brain locations to be analyzed, or if the efficiency of immediate solution were not required, then the data analysis could be treated as in the EEG example. That is, data is A/D converted in the triggered fashion onto a magnetic tape, analyzed at the HSCF, and analyses graphically displayed via tape return to the DPL with console control.

Some typical displays include averages of evoked potentials selected from a series with confidence intervals for the entire average evoked potential (Fig. 5C), or histograms for particular temporal parts of the evoked potential. If one were interested in sequential changes during the course of time during which averaging occurred, amplitudes and latencies of various components of the evoked potentials may be displayed graphically as well (Fig. 5D.) An example of the flexibility provided by having both computers available would be say in determining the durations of averaged evoked potentials of interest for each location measured in auditory nervous systems, or in the number of responses or samples over which interesting changes were occurring, and then sending the data from the whole experiment to the other computer for complete analysis based on such determinations.

C. Behavioral experiments with relay-driving contingent upon behavior

Some learning experiments require food delivery or some other events to occur in an animal's environment as a function of statistical characteristics of some

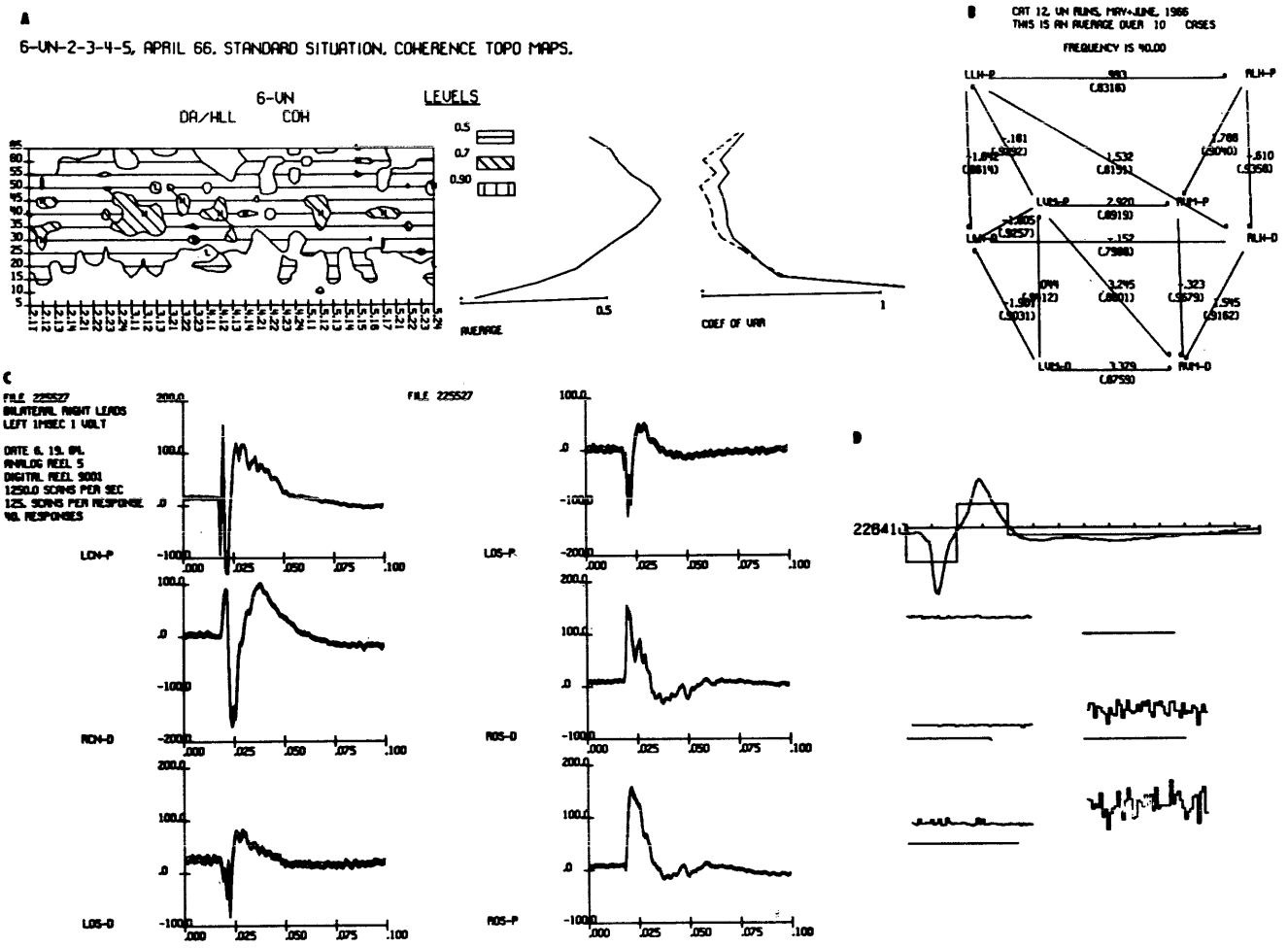


Figure 5 – Exemplary data analysis displays

repeated response it may perform. The sense line and interrupt features of the console DPL system along with its relay driving capability allow great flexibility in such control with parameter entry for such statistical decision making. The DPL computer can simultaneously digitize and analyze physiological data, and compute and display analyses of the behavioral data constantly throughout the experiment.

D. Neurophysiology as an independent variable

Relay driving could also be contigent upon on-line analysis of electrophysiological data as an independent variable, with either behavioral or other neurophysiological data collected as the dependent variable. This is a unique type of experimental design made possible by such on-line neurophysiological computer system. Such experiments are currently planned but not yet operative.

Evaluation

It is, of course, platidudinous to point out the importance of availability and reliability to the investigator.[7,19] With respect to the integrity of his investigations, time-criticality may exist in the microsecond domain for very critical tasks such as A/D conversion, relay driving, sense line reading, and data analysis when used as independent variables. Time-important problems may also exist for I/O and system availability on an hourly basis, for his "on-line" processing of his experiments, or minutes in terms of waiting for his "off-line" data analyses.

The necessity of insulating the system from novice users prompted the decision to develop an interpretive processor. The success of similar systems gave support to this decision. The large amount of system overhead required for SLIP increases considerably the amount of execution time required for a given prob-

TABLE 1

INTERPRETATION OF CONSOLE KEYS IN THE BASIC SYSTEM LEVEL (1).

UC	LC	KEY	FUNCTION OF THE OPERATOR	UC	LC	KEY	FUNCTION OF THE OPERATOR
RETN	0	00	Program terminator	LOG	V	41	Compute the logarithm of AC
WAIT	1	01	Wait requested	EXP	B	42	Each element of AC acts as an exponent of e
CONT	2	02	Continue operation	ABS	N	43	Replace AC with absolute values
BUG	3	03	Debug routine	HIST	M	44	Distribute values in AC with present bin width
LIST	4	04	Alphanumerical display of AC	CONJ	,	45	Set the signs of values in AC
TYPE	5	05	Print on scope	X	.	46	Multiply
RUN	6	06	Starts or stops a previously defined background activity	/	/	47	Divide
				≤	≤	50	Logical operator less than
SET	7	07	Parameter set-up	=	=	51	Logical operator equal
BLNK	8	10	Scope erase	≥	≥	52	Logical operator greater than or equal
PLOT	9	11	Vector display of AC on scope	SUM	(	53	Sum the AC
FIND	Q	12	Search a value in the AC	Δ	Δ	54	Compute the delta between successive elements of AC
EXT	W	13	Extract element(s) from AC				
PRGL	E	14	Load program to AC for correction	PROD	)	55	Compute the product of elements of AC
PRG→	R	15	Store program from AC	*	*	56	Filter the AC
TRNC	T	16	Truncate the values in AC	↑	↑	57	Raise AC to an exponential
HEDL	Y	17	Examine the header of AC	+	+	60	Add
TMOD	U	20	Modify the type of data in AC	-	-	61	Subtract
MODE	I	21	Select scope at plot mode	SPACE	SPACE	62	No operator
CURS	O	22	Extract and extend a part of AC	LOAD	CR	63	Load data into AC
INC	P	23	Increment or decrement continuation index or level	PROG	↓	64	Signify subsequent key-presses as a user generated program
SHFT	A	24	Change the initial index of AC	END	←	65	Program or repeat loop end indicator
ROT	S	25	Rotate AC to the left or right	DATA	o	66	Defines indirect data (UC) Terminator of a binary operand (LC)
FLIP	D	26	Invert the order of values in AC				
MOD	F	27	Execute module n	RSET	RSET	67	Reset the console to wait input status (error correction)
MIN	G	30	Find the smallest element of AC				
RAND	H	31	Generate a random vector in AC	ALTR	α	70	Editing operator
ZERO	J	32	Count zero-crossings in AC	"	"	71	Comments mode definition
INTP	K	33	Linear interpolater	SKIP	$	72	Unconditional branch
AVG	L	34	Compute the average of AC	REP	:	73	Do loop generator (repeat)
SGMA	;	35	Compute the standard deviation of AC	√	√	74	Label sign
SIN	Z	36	Compute the sine of AC	?	?	75	Conditional branch
COS	X	37	Compute the cosine of AC	→	→	76	Store data from AC
ATAN	C	40	Compute the arctangent of AC	LEV	k	77	Operator or data level change request

Note: UC = Operator (upper case) KEY = Character representation of KEY
LC = Operand (lower case) AC = Accumulator

lem (sometimes by as much as a factor of 30 when compared to a Fortran version). This increased time may be unacceptable to users requiring large amounts of computation (e.g., matrix inversion, sorting, etc.). The current answer to this problem is writing an efficient machine language program off-line and then merging it with the SLIP system. The program is then accessed by the SET function (Table I), using a four character name.

Current daily limitation on system availability is being remedied by an assembly program development which will enable time-sharing by SLIP and assemblies. The elimination of this limitation will enable acute neurophysiological experiments, running continuously for 24 or more hours, to have constant access to control and data processing of the system.

From the user's viewpoint, system reliability has been quite good. Most interruptions aremomentary and have an adverse effect on the investigator only for time-critical on-line experimental processes such as A/D data collection, sense line reading, and relay driving. These seldom occur, but, of course, can be very frustrating (costly) to the investigator when they do.

ACKNOWLEDGMENTS

Much of the credit for the initial planning of the DPL system must be given to Mr. Dan Brown.[1,15] Lionel Rovner[15] was principally responsible for hardware developments, while Mrs. Howard, Mr. McGill, and Mr. Wyman assisted in software development. USPHS Grant NB 02511, NASA Grant NS 6505, ONR Grant ONR 233(91), assisted DPL development, while Calif. DMH Grant 2-36 and USPHS Grant 13268 assisted the neurophysiological research. Mr. Brown and Dr. Walter[19,20] largely developed the HSCF data programs with the assitance of Mr. Parmallee and Mrs. Free. The HSCF is heavily supported by USPHS Grant FR 3.

REFERENCES

1 F ABRAHAM D BROWN M GARDINER
Calibrations of EEG power spectra
Communications in Behavioral Biology vol 1 1968

2 F D ABRAHAM J T MARSH
Amplitude of evoked potentials as a function of slow presenting rates of repetitive auditory stimulation
Experimental Neurology vol 14 ,1966

3 F D ABRAHAM N M WEINBERGER
Possible mammilothalamic tract involvement in feeding behavior
The Physiologist vol 10 1967

4 W R ADEY
Spectral analysis and pattern recognition methods for electroencephalographic data
Data Processing Conference Copenhagen 1966

5 L BETYAR
A user-oriented time-shared on-line system
C ACM vol 10 1967

6 M B BRAZIER
The application of computers to electroencephalography
Loc. Cit. Ref. 17

7 G E BRYAN
Joss: 20,000 hours at the console: A statistical summary
AFIPS FJCC vol 32 1967

8 G J CULLER
A start in conversational programming for elementary mathmatical problems
IFIPS Conference Proceedings vol 2 1965

9 W J DIXON
Use of displays with packaged statistical programs
AFIPS FJCC Proceedings vol 32 1967

10 I ETTER
Requirements for a data processing system for hospital laboratories
AFIPS FJCC Proceedings vol 32 1967

11 S L FEINGOLD
PLAINT - A flexible language designed for computer-human interaction
AFIPS FJCC Proceedings vol 32 1967

12 M S FINEBERG O SERLIN
Multiprogramming for hybrid computation
AFIPS FJCC Proceedings vol 32 1967

13 C MACHOVER
Graphic CRT terminals - characteristics of commercially available equipment
AFIPS FJCC Proceedings vol 32 1967

14 W A ROSENBLITH
Processing neuroelectric data
MIT Press Cambridge 1962

15 L D ROVNER D BROWN R T KADO
A time-shared computing system for on-line processing of physiological data
Proceedings Symposium on Biomedical Engineering Milwaukee vol 1 1966

16 W J SANDERS G BREITBARD D CUMMINS R FLEXER K HOLTS J MILLER G WIEDERHOLD
An advanced computer system for medical research
AFIPS FJCC Proceedings vol 32 1967

17 R W STACEY B WAXMAN
Computers in biomedical research
Academic Press New York 1965

18 A TONIK
Development of executive routines, both hardware and software
AFIPS FJCC Proceedings vol 32 1967

19 D O WALTER
Rapid interaction with a digital computer-plusses and minuses
The Physiologist vol 9 1966

20 D O WALTER R T KADO J M RHODES W R ADEY
Electroencephalographic baselines in astronaut candidates estimated by computation and pattern recognition techniques
Aerospace Medicine vol 38 1967

21 H WYLE G J BURNETT
Management of periodic operations in a real-time computation system
AFIPS FJCC Proceedings vol 32 1967

Graphical data management in a time-shared environment

by SALLY BOWMAN and RICHARD A. LICKHALTER
System Development Corporation
Santa Monica, California

INTRODUCTION

At System Development Corporation there is a conviction that one of the most plausible ways to make the cost of software decline is to build general-purpose software that is capable of solving a variety of problems. SDC's most successful effort in this field has been in the area of general-purpose data management. Our initial large-scale, time-shared data management system, TSS-LUCID, enabled the nonprogrammer to describe, load, query, and maintain a data base. In use for over two years, this system provided enough generality to solve such diverse problems as comparison of salary data in different segments of the aerospace industry, analysis of statistical data for a customer in the oil industry, and monitoring of public-supported cancer research projects. Currently being implemented on the IBM 360 family of computers is an improved, more powerful version called TDMS (Time-Shared Data Management System).

The role of cathode ray tube displays as applied to data management systems is the basic concern of this paper. Traditionally, CRT displays have been used for tabular data display, geographical displays associated with command and control systems, and for engineering applications. Little use has been made in the area of graphical display of structured data files as an adjunct to an information retrieval system. In addition, little attempt has been made to use the scope to assist the user in forming his data retrieval request. With the widespread availability of time-sharing and data management systems, the need for an easy-to-use, yet powerful, graphical display system has become more critical. This paper describes the graphical display system currently available on SDC's time-shared Q-32 computer in Santa Monica. The system described (called DISPLAY) is the forerunner of the display component of TDMS.

Design goals

In January 1967 we began work on DISPLAY by limiting the design to that set of display problems which relate to analyzing data from a large data base. Thus, DISPLAY was to be concerned with scatter plots and with regression curves but not with rotating three-dimensional figures.

Our first design goal was to provide satisfactory response within a time-shared computer. We knew from experience that the nonprogrammer user will tolerate long delays while the teletype clatters along reassuringly, but that he will panic very quickly when confronted with an unchanging CRT.

Constructing a display system that will deliver even tolerable response within a general-purpose time-sharing environment, however, requires careful breakdown of the input/output requirements to maintain an interactiveness between system and user and thus avoid lengthy delays waiting for service when placed in a lower priority queue.

Our second goal was to produce a system easy for the nonprogrammer to use. Communication had to be so natural as to appear inevitable. Still, the system had to have enough power to accomplish useful work.

Achieving these first two goals—we hoped—would help us achieve the third: that of gaining users for the system in order to obtain feedback to improve the system. Our ultimate goal was to use this experience in designing the display portion of the Time-Shared Data Management System, a general-purpose data management system being developed at SDC.

Design environment of display

The DISPLAY system was designed and built for the IBM Q-32 computer. This is a powerful machine with 64,000 words of core memory, a 2-μsec cycle time, drum storage, and 4 million words of auxiliary disc storage. The Q-32 is operated in a time-sharing mode at all times. During "prime time," the number of users averages between 25 and 30. SDC's general-purpose time-sharing algorithm provides equal service to all users; thus DISPLAY receives no special advantage over any other program. To the best of the authors' knowledge, there are no other display systems that operate in a time-shared environ-

ment without receiving some special consideration from the executive.

Auxiliary to this computer are 6 CDC DD19 cathode-ray-tube scopes which are drum-refreshed automatically every 22 milliseconds. The 1024 × 1024 scope grid provides good resolution, and the refresh rate provides excellent insurance against flicker. Normal usage of the CRT provides 680 characters and/or vectors to each user, but DISPLAY can operate with up to 1,360 characters. Associated with each CRT is a light pen with a two-position switch, providing an aiming circle and a flicker when fired. Additionally, a teletype is provided for normal communication with the Time-Sharing System and (in some instances) with the DISPLAY system. In previous experiments we had learned that a welter of input equipment is distracting and awkward, if not downright frightening to the user, so we deliberately kept to the minimum of scope, light pen, and teletype.

In addition to the hardware available, the designers had a startling richness of software at their disposal. This included an on-line, interactive JOVIAL compiler; elaborate debugging, editing, and other on-line programming tools; and TSS-LUCID, which provided all the machinery necessary to describe, load, maintain, and interrogate large data bases. Additionally, an on-line interpreter, TINT, with full ALGOL capabilities was available.

The importance of these software assets cannot be overemphasized. In fact, in building DISPLAY, the TSS-LUCID query program and the TINT interpreter were used almost "as is." In order to do this, of course, a sequencer program had to be built, which effectively time-shares core within the time-sharing system.

System description

DISPLAY provides an automatically generated graphical presentation of data stored in a TSS-LUCID data base. The user's entire attention is focused on the scope. All parameters which he may require are listed and he supplies values only by means of his light pen. To make this possible, the program follows the user's light-pen selection and dynamically updates the scope to supply all the choices legal as a result of his last action. If the user changes his mind or makes a mistake, he erases by lightpenning what he wants to delete. His previous selection is erased up to the point that he has indicated, and the scope returns to the set of legal inputs that are appropriate, after considering his erasure. Dynamic scope updating achieves three purposes: it makes it easier for the user; it guarantees error-free inputting; and it allows the program to deliver rapid response within the time-sharing system.

Once the necessary parameters for a graphical display are defined, the user executes the request and receives a standard graphic presentation of his data. The standard presentation implies two things: First, the user receives a data plot rapidly without being troubled with the minute detail associated with laying out a plot; second, extensive capability to override the standard presentation must be available.

Data base X-ray

At any time during parameter specification, the user may "browse" through his data base under light-pen control. A list of the data base elements are presented 26 to a display page with up to five pages possible. Included with the element list is the element number, element type, and the number of distinct values which each element has in the data base. The user may then access the value list associated with each element through light-pen control. Because the number of data values can be quite large for each element, a random-access scheme is applied to make possible rapid display of any value. When the user requests a value list display for a selected element, a sample of values is presented including the first value, the last value, and up to 24 equally spaced values. To obtain additional data values for the selected element, the user has light buttons to expand the displayed value list about any value.

This is an iterative process providing the capability to pinpoint a specific value out of a list of up to 17,576 values by only three light-pen choices. This has particular value in providing the user a rapid listing of the range of data values and their stored coding in both coarse and fine scale.

Another feature of the value list display is the inclusion of a frequency of occurrence count for each value. This is very useful in analyzing the data base contents and in error detection within the data base.

Data retrieval

The structure of the TSS-LUCID data base that is used with DISPLAY provides rapid data retrieval and broad user selectivity in determining the particular data subset of interest. The user pays no penalty for the retrieval of one data element over another, since all elements of a given data base are equally retrievable.

Stored with every data base is a cross-indexed inverted file directory into the actual data. When a data retrieval request is defined, the directories are searched to determine the qualifying data records. These directories are then further used to determine disc addresses of the qualifying records; these are the only records that are examined for retrieving the required data.

To select a new data base while using DISPLAY, the user names the desired data base to be loaded using the teletype. The new data base is found and made part of the DISPLAY environment. There is currently no ability to retrieve from more than one data base simultaneously; that is, one may not retrieve data for the X-axis from data base 1, and data for the Y-axis from data base 2.

Standard graphic presentation

DISPLAY is designed to provide an automatically determined, standard graphic presentation in response to the user's light-pen inputs. Axis scaling, axis labeling, data scaling, and data plotting are all performed by the program. The data determines the X- and Y-axis scales. For numeric information, the scale consists of 10 graduation marks with a graduation increment based upon the most appropriate selection of 1, 2 or 5 times a power of 10, which encloses the range of data values and gives the largest graphic display available. An algorithm determines whether the data values should be plotted from zero, or whether it is more appropriate to display a minimum *scale* value which is somewhat below the minimum *data* value displayed.

Hollerith data are displayed evenly spaced on the axis with a spacing determined by the number of distinct values for the variable. On the X-axis, to accommodate a larger number of distinct labels, two lines are used (when necessary) to minimize label overlay.

Titles for the X- and Y-axis are the data base element names specified in the display request. The overall title of the graph is the user's original data subsetting statement or, if there is none, the data base name. Where the user has specified a succession of curves to appear on the scope, each curve is numbered and a legend supplied to distinguish the multiple curves.

User picture interface

When the requested graphic display appears on the scope, the user has at his disposal a set of "touch-up" overrides for further analyzing the data and modifying the display. Accurate *point readout* values are available through light-pen selection. The DISPLAY program interprets the light-pen position and displays the numeric or Hollerith X and Y meaning for that point.

The user may change the *axis scale* either to better analyze a given display area or to place the data in clearer perspective. In many cases, much of the data are bunched in a small area of the scope because of a few extremely large values. Changing the range of the display allows for an expansion of the bunched data for clearer insight into the data relationships.

For appearance sake, all *titling* can be altered; commentary may be added; data points or data sets may be deleted; Hollerith axis labeling can be changed. Columns or rows may be repositioned or eliminated. At any time during this process, if the user destroys the original display, he may by light-pen action return the scope to the standard graphic display.

Back-up hard copy or auxiliary information from the data base is available through the teletype. While his picture remains on the scope, the user may request the TSS-LUCID query program to retrieve, format, and sort output to explain some quirk of the data revealed on the scope.

In addition, the user may *save* a particular graphic presentation and recall it at a later time. Very importantly, the user may use any of the touch-up options on any picture he recalls. The save and recall capability provides the flexibility to store on a single file graphical displays generated from several different data bases.

Application programming

Users who wish to manipulate data mathematically and logically using more sophisticated operators than those provided by the standard DISPLAY program can use TINT (a higher-order ALGOL-type interpreter) within DISPLAY. Although TINT requires that the user have some programming ability, it is designed to facilitate user interaction and incorporates many user-oriented features.

The capabilities of TINT include full iteration control, arithmetic control, conditionals, indexing, parameter specifications, teletype print routines, code insertions, debugging aids, etc. TINT programs can be saved and recalled at a later time.

To use TINT within DISPLAY, the user first specifies the data to be retrieved, and then indicates that he wants to operate TINT. DISPLAY initiates the data retrieval. When it is complete, DISPLAY turns over control to TINT. Then the user either specifies an already written TINT program, or writes a new TINT program on-line. The TINT program operates on the retrieved data and outputs a graphic data array, which in turn is fed into DISPLAY for scope presentation.

Use of display

Several examples of use of DISPLAY are presented in this section, including actual scope photographs. The first example describes in some detail the interaction between the user and the system in the formulation of a display request. The remainder of the examples illustrate the different capabilities

provided within DISPLAY and indicate the variety of applications to which the system has been applied.

Example 1

The user requests a scatter plot showing the amount of taxes paid versus the assessed valuation of property for cities in the State of California with a population of 50,000 or less. In Figure 1, the user has light-penned the parameter X-variable and is preparing to complete the specification. The right side of the scope contains the data base element names from which the user may select to define the X-variable. Shown with the element names are: (1) the element number, (2) the element type, and (3) the number of distinct values for each element in the data base. Hence for COUNTY, the element number is E2, the element type is H for Hollerith, and the number of different counties is 53. On the lower left hand side of the scope, available statistical operators are displayed which may be selected and applied to a numeric type data base element.

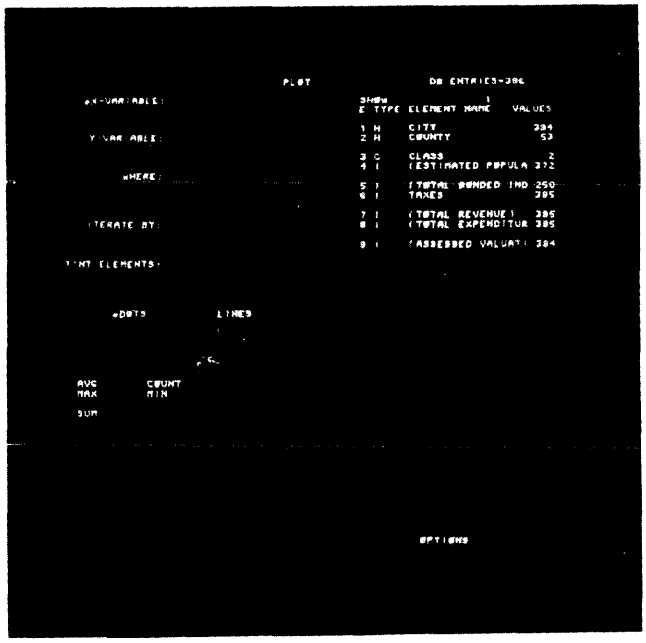


Figure 1 – Initial display showing the data base elements and the available parameters for the graphical display request

The user light-pens the element name TAXES, and the scope changes to that of Figure 2. The element name list has disappeared since it no longer represents a legal input at this point in the input specification. Legal inputs shown are the parameter list and the relational EQ (equal) which appears in the lower left. Selection of EQ allows the user to specify particular data values for the selected variable. In this example, however, the user has satisfied the X-variable

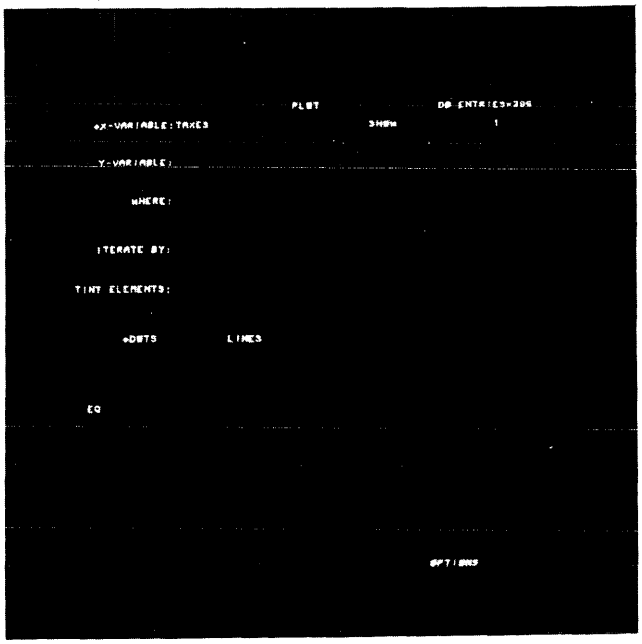


Figure 2 – X-variable specification

specification and is ready to select a different parameter. The user selects Y-variable and light-pens ASSESSED VALUATION OF PROPERTY. In Figure 3, the user has completed specifying the Y-variable and is supplying the data subsetting clause using the WHERE parameter. Having light-penned ESTIMATED POPULATION, the user is pre-

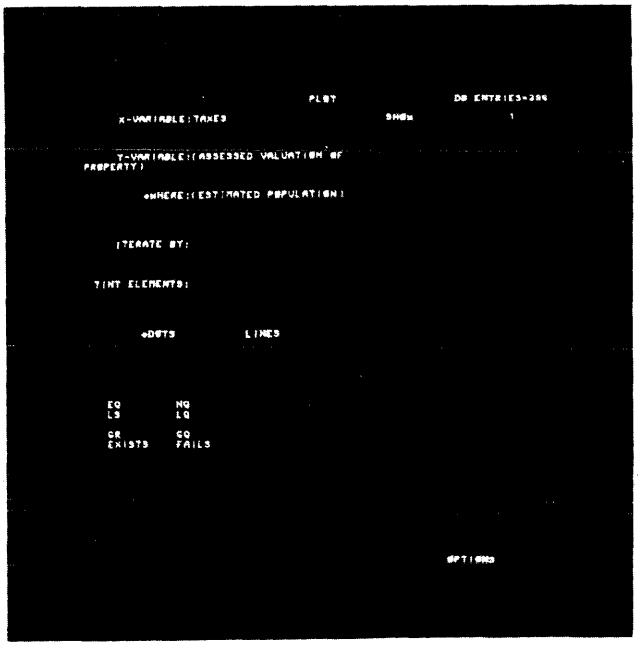


Figure 3 – X-variable, Y-variable specification and start of data subsetting clause

sented the set of legal relationals in the lower left side of the scope. The user light-pens the relational LQ (less than or equal), and in Figure 4 receives a list of values associated with ESTIMATED POPULATION. This list gives him the first value, the last

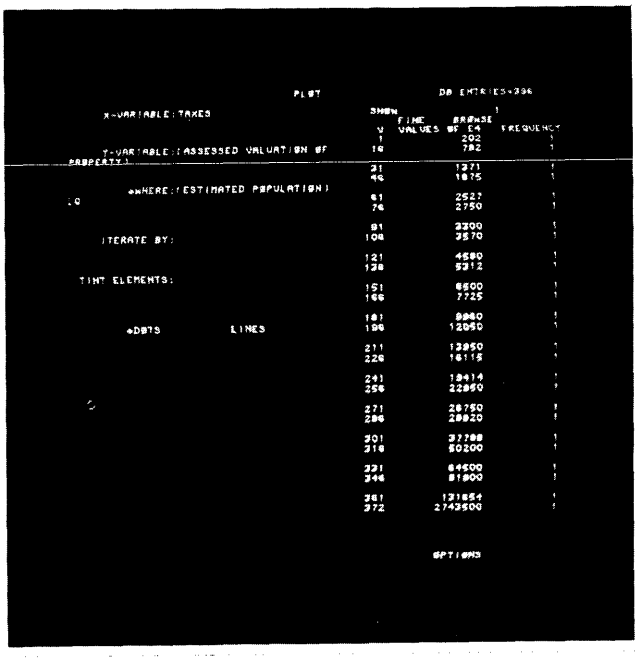


Figure 4 – Display of value list for use in data subsetting clause

value, and a selected sample of intermediate values. The user selects the value of 50,200 (closest value to 50,000) for use in the data subsetting clause. The user could have used the FINE light button to obtain additional data values to select from. After data value selection, the scope changes to Figure 5, where the user may continue specification of the data subset using the Booleans AND and OR, or, as in the example, light-pen the light button EXECUTE to begin generation of the display request.

When the execute action is sensed, user inputs are converted into a data retrieval request, the search of the data base is performed, and an automatic display of selected data occurs (Figure 6). Each axis is automatically scaled and graduated to best reflect the data. Each axis is labeled with the appropriate data element plotted, and the title of the graph is the data subsetting clause provided in the WHERE parameter. To aid the user in data analysis, a set of touch-up options are displayed. In this example, AXIS SCALE is used to expand the area near the origin (Figure 7) to separate the bunching of data points. Light-penning the EXECUTE button results in a rescaling of the plot as shown in Figure 8. Also illustrated in Figure 8 is the use of READOUT. Any data point on the scope may be light-penned, and the actual data values stored

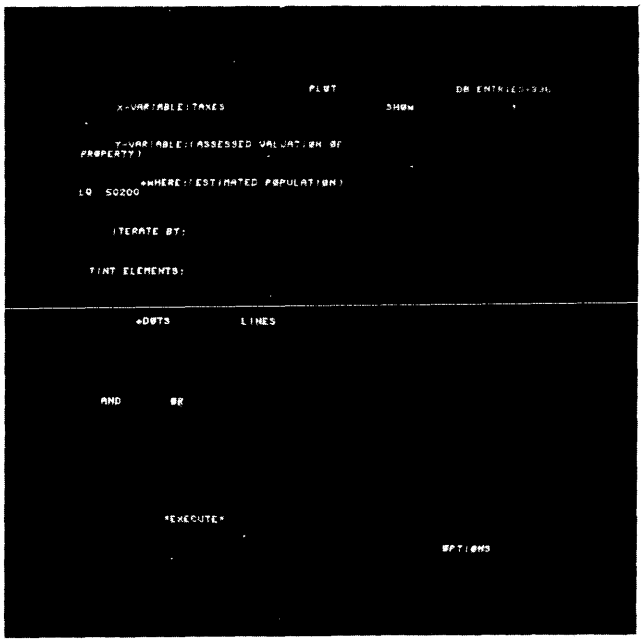


Figure 5 – Complete parameter specification for display request of example 1

in the data base for that point are displayed on the scope, together with a bright plus (+) superimposed on the selected data point.

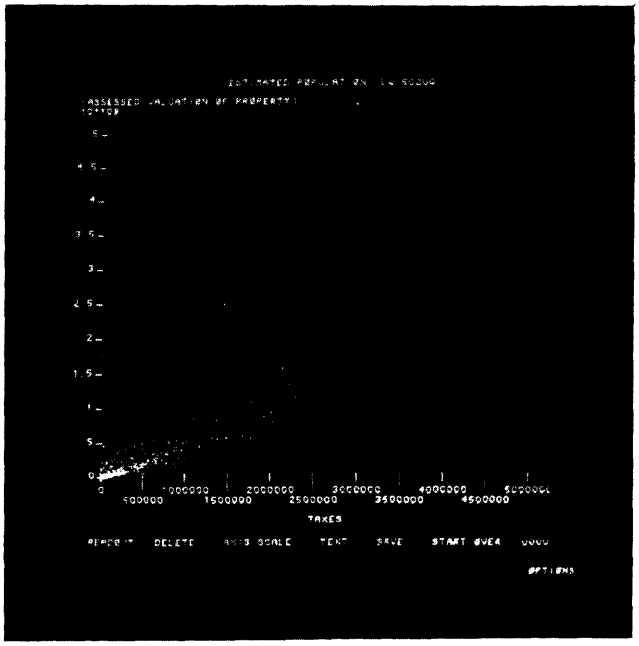


Figure 6 – Standard graphic presentation for example 1

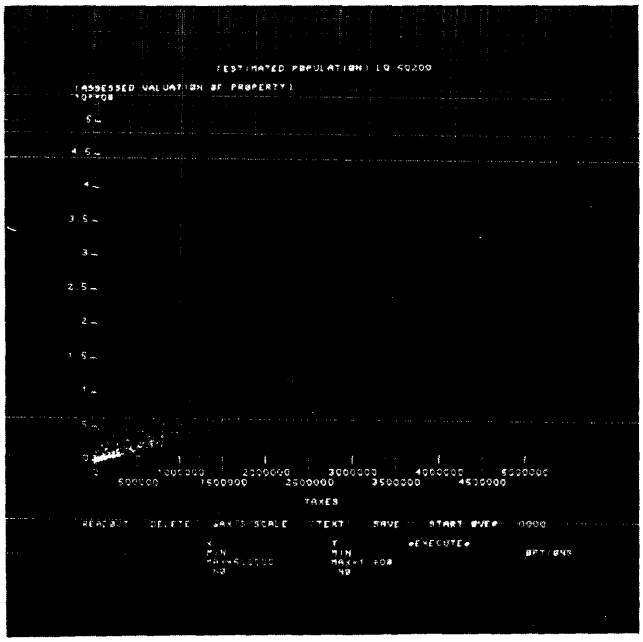


Figure 7 – Use of axis scale

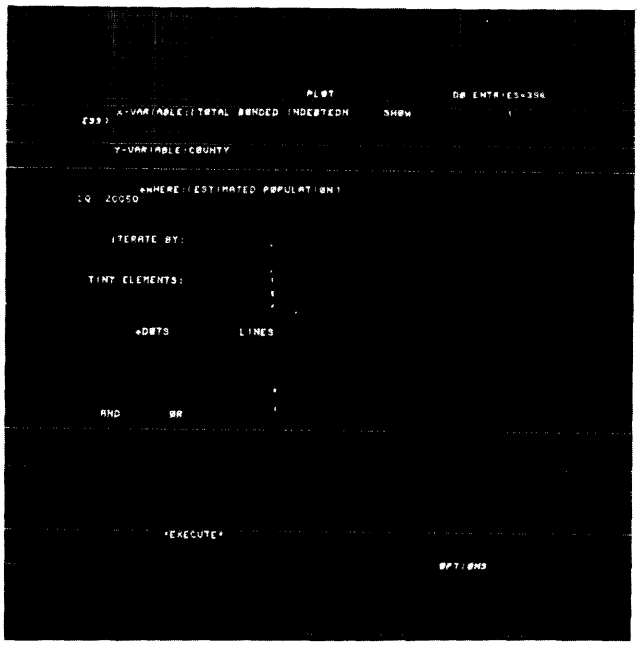


Figure 9 – Parameter specification for example 2

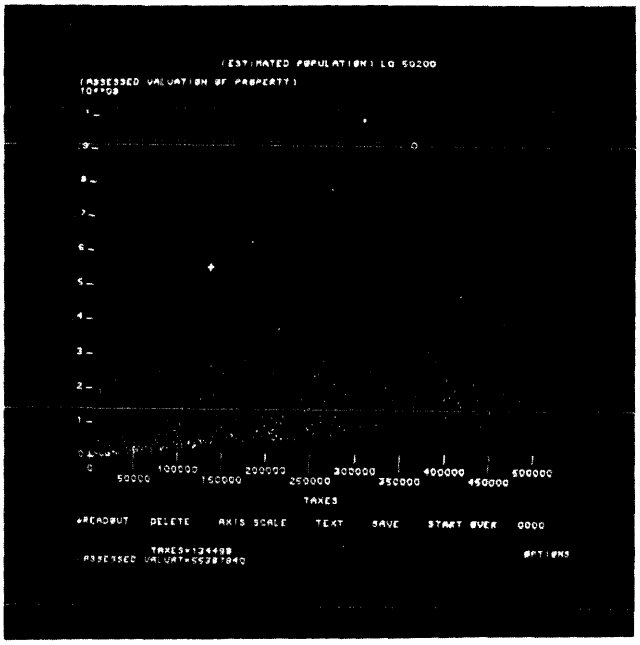


Figure 8 – Rescaled graphic presentation showing point READOUT option

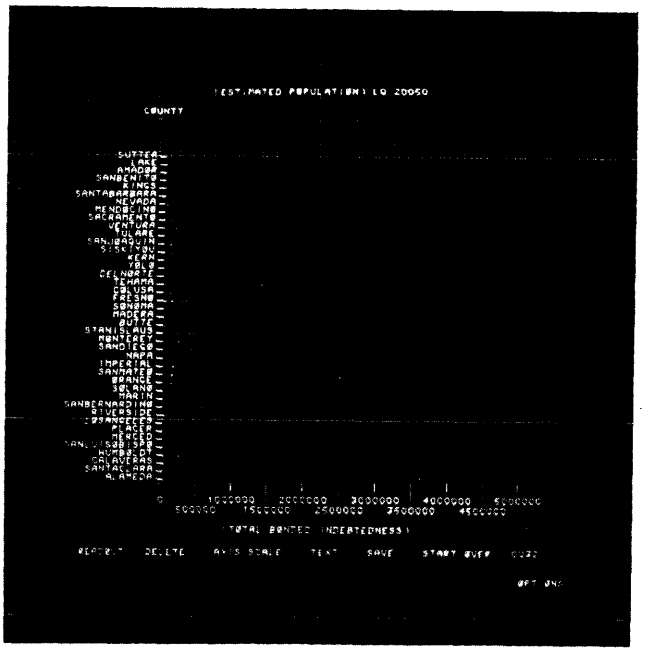


Figure 10 – Distribution by county of total bonded indebtedness for selected cities

Example 2

The user requests a scatter plot, showing the distribution of the total bonded indebtedness per county of cities with a population of 20,000 people or less. Figure 9 shows the complete input specification for this display, and Figure 10 illustrates the resulting scatter plot. Scope capacity considerations occur when the Y-axis contains a large number of distinct Hollerith values. The user is informed of the condition by the counter in the lower right hand corner of the display. In this example the counter reads 0032, meaning 32 possible data points are not shown. The user can get access to these missing points by adroit handling of the DELETE option. For example, the user may delete the X-axis label and the title to cause the counter to go to zero, at which time the missing data points would be displayed.

Example 3

A military data base on status of forces is illustrated next. The display request is for a line plot showing the total amount of troops in training for each assigned readiness level comparing the Army to the Air Force. Parameter specification for this is shown in Figure 11 resulting in the display shown in Figure 12. This display illustrates the use of multiple curve generation which is provided by the use of the ITERATE BY parameter. Each curve is numbered and a legend is provided to distinguish the several curves. READOUT also distinguishes the particular curve that a data point is on.

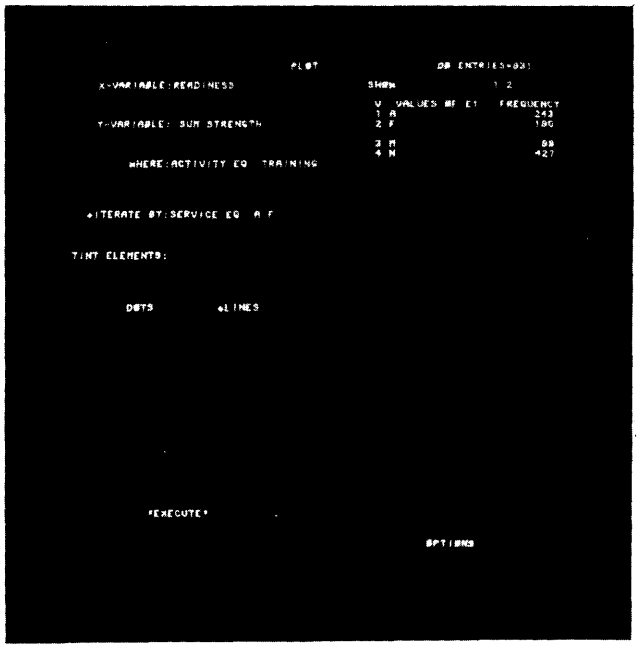


Figure 11 – Parameter specification for example 3 showing the use of the ITERATE BY parameter for multiple curve generation

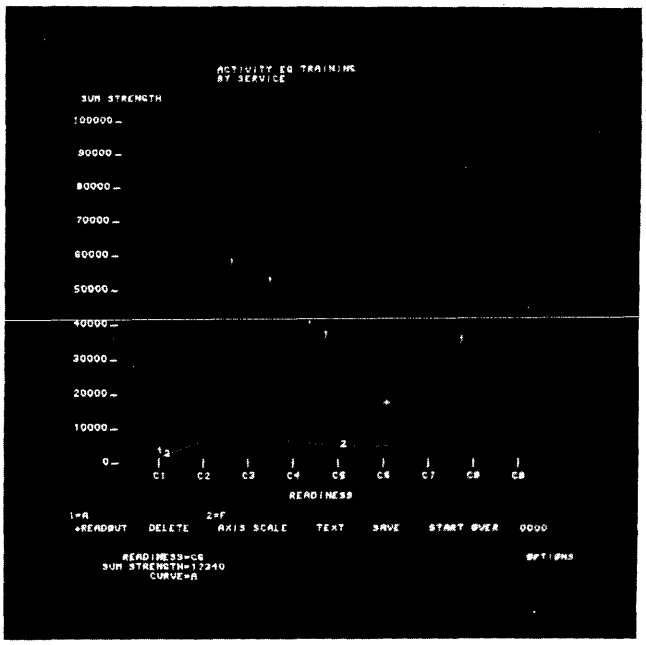


Figure 12 – Line plot display showing total troop strength by readiness level

Example 4

The use of DISPLAY with scientific data is presented. Oceanographic data measurements have been compiled into a data base containing the elements appearing in Figure 13. The illustrated scatter plot (Figure 14) shows a plot of salinity versus depth for a specific data subset. In this example, the automatic scaling algorithm is shown off to good advantage in that the X-axis is not plotted from zero but instead from a meaningful minimum value.

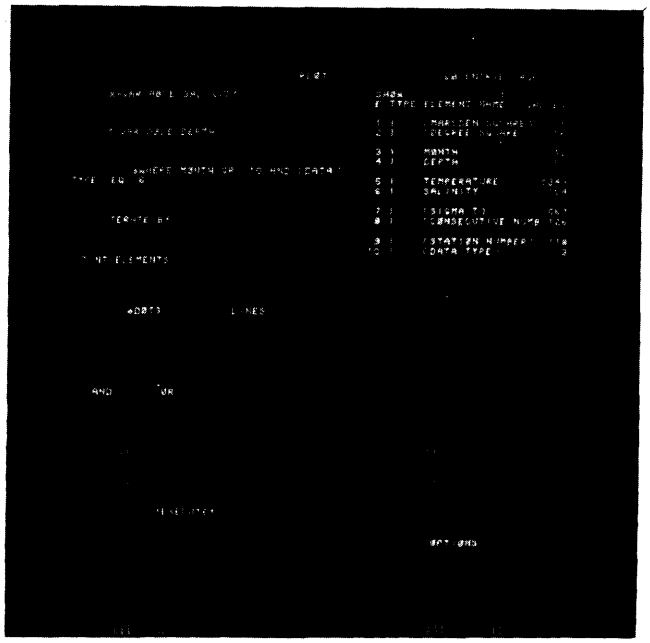


Figure 13 – Parameter specification for example 4 using the oceanographic data base

Example 5

The use of a statistical analysis program is represented in this example. A representative sample of oil credit card purchases was loaded in a data base, and DISPLAY was used to plot a frequency distribution of the individual invoice charges. The elements in the data base and the user's specification to achieve the display are shown in Figure 15. The resulting frequency plot (Figure 16) shows some interesting insights into customer buying with large frequency peaks at the $2, $3, and $5 invoice charge.

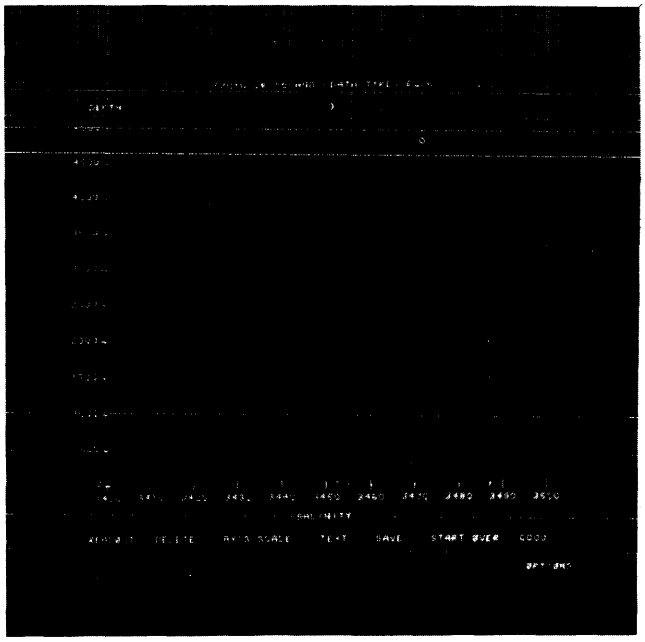


Figure 14 – Scatter plot showing the distribution of salinity by depth from the oceanographic data base

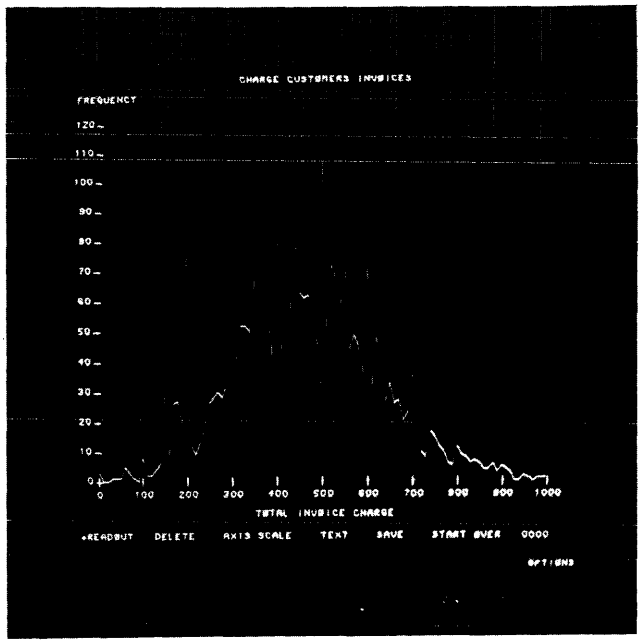


Figure 16 – Frequency distribution of invoice charges for an oil company's charge accounts

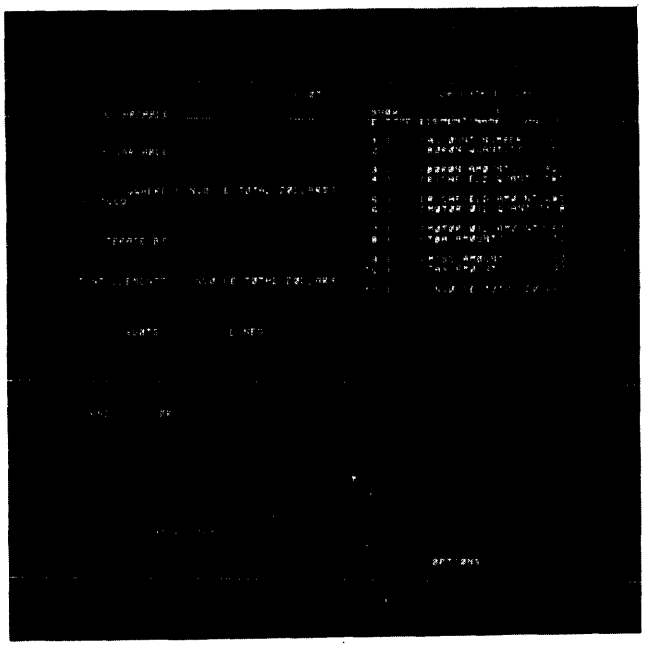


Figure 15 – Parameter specification for example 5 showing the use of TINT

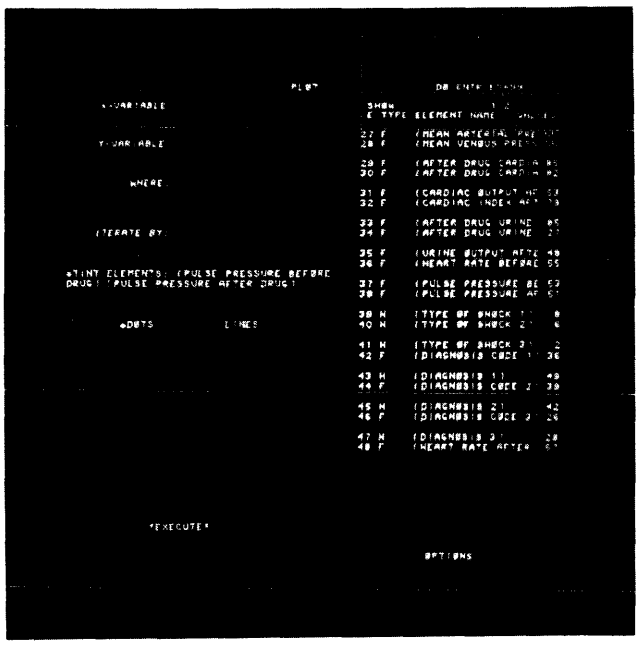


Figure 17 – Parameter specification for example 6 showing partial list of data elements within the digitalis drug data base

Example 6

The final example illustrates the use of DISPLAY with medical data received from the University of Southern California shock ward and the effects of the drug digitalis on the patients. Some of the elements in the data base are illustrated in Figure 17, together with the input request. In Figure 18 the resulting display shows a regression plot of pulse pressure before the drug was used versus pulse pressure after the drug was administered. The discontinuity on the curve is a pictorial representation of the standard deviation of the data.

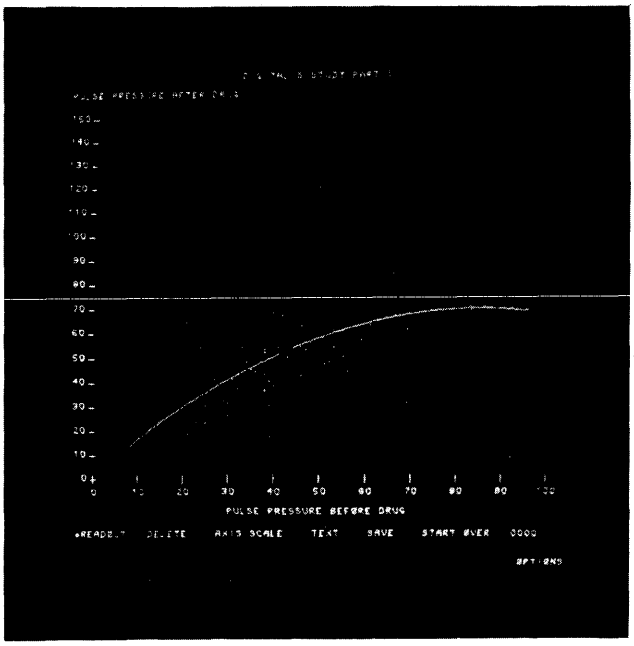


Figure 18—Data plot and regression curve analysis showing the effects of the use of digitalis on pulse pressure

CONCLUSION

In evaluating the success of this project, the designers find that the major goals have been met. One of these was rapid response. In most respects, response is well within the limits of user patience. The inputting of the initial parameters; the scope examination of the structure and content of the data base; the regeneration of the display in response to the users' most complex "touch-up" request—all appear nearly instantaneously. Time for retrieval, however, is long. It varies from 15 seconds to five or six minutes elapsed time as a complex function of the size of the data base, the complexity of retrieval statement, and the number and kind of other time-sharing users. The effect of the lengthy retrieval time, however, is much less disastrous than we had feared, for a user very quickly learns that retrieval takes time and resigns himself to this one wait much more tolerantly than we had anticipated.

Our second goal, that of easing the user's communication with the system, has been fully achieved. With the scope guiding the user at every step, users quickly arrive at the point of data analysis. Originally, we had hoped that a new user could learn to use DISPLAY on at least a basic level with no more than 30 minutes training. Experience has shown this to be true. The ability to specify a complete graphic display through the use of the light-pen and to have the data base elements and data values automatically displayed when needed facilitates user interaction and overcomes the normal anxieties experienced by new users not familiar with display systems. We feel that the communication channel which we have opened with DISPLAY may well be the most appropriate way for a nonprogrammer to communicate with many of the sophisticated computer programs presently being installed on third-generation machines.

Our third goal—that of attracting real live users from whose experience we could learn—has also been achieved. Since July, when it became usable, DISPLAY has proved to be sufficiently general-purpose in nature to solve a variety of problems. A large oil company used DISPLAY to study their distribution of credit card purchases with respect to dollar volume, gallons bought, and number of invoices per customer to better understand the customer population and buying habits. A research project within SDC used DISPLAY to analyze the results of treating shock patients with the drug digitalis. Another application in the medical field involved analysis of cancer research projects by state, by area, by university, and by topic. Still another example was the analysis of the type of sensing equipment used in satellite surveillance. It is important to note that these applications were conducted with only minimal training periods with DISPLAY and in the main by nonprogrammers. These applications were accomplished by the people who had the data problem rather than by data processing personnel.

Currently, DISPLAY is available only for use with the SDC Q-32 Time-Sharing System in Santa Monica. Further research is planned using the man-machine interface techniques evolved from this project in a broader range of data management functions, including data base description, maintenance, report generation, and fact retrieval. Applying the work to the small, tabular scopes without light-pens is also being studied. The most immediate goal, however, is to redesign DISPLAY to operate under the SDC 360 Time-Sharing System in association with SDC's Time-Shared Data Management System.

REFERENCES

1 E BENNETT E HARRIS J SUMMERS
AESOP: A prototype for on-line user control of organizational data storage, retrieval and processing
Proceedings of the Fall Joint Computer Conference vol 27 pp 435-455

2 E L JACKS
A laboratory for the study of graphical man-machine communication
Proceedings of the Fall Joint Computer Conference vol 26 pp 343-350

3 D T ROSS R H STOTZ D E THORNHILL C A LANG
The design and programming of a display interface system inte-

grating multi-access and satellite computers
ESL Memorandum 170429-M-190/MAC-M-353 M I T Electronic System Laboratory

4 S H CAMERON D EWING M LIVERIGHT
DIALOG: A conversational programming system with a graphical orientation
Communications of the ACM June 1967

5 R KAPLOW J BRACHETT S STRONG
Man-machine communication in on-line mathematical analysis
Proceedings of the Fall Joint Computer Conference vol 29 pp 465-477

6 *Digigraphic System 270; system information manual*
Control Data Corporation Digigraphic Laboratories Burlington Massachusetts 1965

7 T R ALLEN J E FOOTE
Input/output software capability for a man-machine communication and image processing system
Proceedings of the Fall Joint Computer Conference vol 26 pp 387-396

8 J MC CARTHY D BRIAN G FELDMAN J ALLEN
THOR—A display based time-sharing system
Proceedings of the Spring Joint Computer Conference vol 30 pp 623-633

9 J I SCHWARTZ E G COFFMAN JR C WEISSMAN
A general-purpose time-sharing system
Proceedings of the Spring Joint Computer Conference vol 25 pp 397-411

10 I E SUTHERLAND
Sketchpad, a man-machine graphical communication system
Proceedings of the Spring Joint Computer Conference, vol 23 pp 329-345

11 B D FRIED
The STL on-line computer
Vol 1

12 G J CULLER
Function oriented on-line analysis
Workshop of computer organization 1962 pp 191-213

13 G BURCK and the editors of *Fortune*
The computer age and its potential for management
New York, Evanston and London: Harper and Row 1965

14 J C R LICKLIDER W F CLARK
On-line man-machine computer communication
Proceedings of the Spring Joint Computer Conference vol 21 p 113 1962

On the formal definition of PL/ I

by K. BANDAT
IBM Laboratory Vienna
Vienna, Austria

INTRODUCTION

This paper describes a formal definition of PL/I which has been produced by a group in the IBM Vienna Laboratory. The paper contains the outlines of the method rather than details of PL/I. The definition currently exists as a Technical Report "Formal Definition of PL/I"[1] which contains the abstract syntax and the semantics for PL/I program text. A second version of this Technical Report is under preparation and will complete the description of PL/I in these areas where the first version omitted language features, or showed major deviations from the current language. The language described in the Report is PL/I as specified in the PL/I Language Specifications of the IBM Systems Reference Library[2], supplemented by additional information.

Needs for language descriptions

The new potential user of a programming language – somebody who is assumed to be familiar with high level programming languages in general – needs information on the language on several levels of precision and completeness.

At the first contact with the language he would like to know the salient features of the language, the parts of the language which are similar to languages he is familiar with, and the new concepts in the language which mark the step forward in programming language development. The user would like to see the potential areas of application for the new language. He should find all this information in an introductory document to the language like a primer. This primer in an intuitive way explains the concrete representation of programs and data, the various data types and data structures of the language, the structuring of programs by blocks and procedures, the operations which can be used in expressions, etc. A primer need neither be a complete description of the language nor need it be precise in all details. For tutorial purposes simplifications and omissions can be appropriate.

A description of the language with an increased level of completeness and precision is required for the actual user who starts to write programs in the language. He needs to have the full information on the concepts of the language and a complete set of rules for writing programs which will be accepted by a compiler. He would also want to know what result the execution of his program will give. Traditionally these needs for most programming languages were served by language manuals which state fairly precisely how a program has to be written. For the meaning of a program manuals frequently supply a natural language description which explains semantics of the language in an informal way, leaving a certain amount of questions open to the intuition of the reader who has to generalize from examples in the manuals and from the interpretation of programs on existing compilers.

These means will not satisfy the more advanced programmer who wants to know properties of the language or of a program in all details, say, e.g., in order to clarify whether two programs written to solve the same problem are in fact equivalent.

The highest level of precision in language description is needed for the implementer of a language. He requires a reference to the language which for every conceivable question can deliver a complete and correct answer. The implementer should neither be forced nor be allowed to answer questions on the language by his personal interpretation of a manual. The reference tool can also serve as the communication tool by which the language designer conveys the complete information on the language to the implementer. This makes it necessary that the method used in producing the reference document allows easy modifications of the description for the incorporation of language changes and language extensions.

We are convinced that the methods developed establish a tool for describing programming languages and PL/I specifically with a degree of precision which could not be achieved up to now by using informal methods.

Formal methods for language definition

If we accept the need for a rigorous, complete and

unambiguous definition of a high level programming language we have to find which methods and metalanguages can be used for the purpose.

It is frequently claimed that it is appropriate and more convenient for the user to describe a programming language with the aid of a natural language as the describing metalanguage. This may be tolerable for a language manual which has to serve a tutorial purpose as well as to give a description of a language. However, for achieving a precise and unambiguous definition the use of natural language is inappropriate. Natural languages are not well defined languages, they are lacking an exact syntax which allows the unambiguous parsing of sentences and clauses, and they employ words with multiple or vaguely defined meaning. Thus it can never be guaranteed that rules formulated in a natural language, unless applying rigorous restrictions, will have a well defined and unambiguous meaning. Current experience with language manuals written in English has shown up this fact very extensively. Only a formal method used in defining a programming language will yield the required precision and unambiguity. Although formalization is a well established method in the foundation of mathematics and logic, it was only recently that attempts have been made to apply similar methods to programming languages.

In defining a language we have to distinguish between the syntax of the language, i.e., the set of formation rules defining all strings which are well-formed programs, and the semantics of a program, i.e., the meaning of a well-formed program.

For the definition of the syntax of programming languages Backus Normal Form or some of its equivalents are commonly accepted tools.

For the definition of the semantics of a programming language two groups of methods are known, translation and interpretation.

The translational approach requires the existence of a completely defined language L. If a translator can be designed, which translates any well-formed program, written in the language L′ to be defined, into a program in the language L, then the definition of the language L together with the translator L′ to L completely define the semantics of L′. In order to be precise and unambiguous, the rules translating the program have to be written in a metalanguage which itself is completely defined.

The methods of semantic definition by interpretation have in common that they specify a function or a process which for any given set of input data and any given program text yields the output data. This can be achieved by designing an abstract mechanism which serves the purpose of a machine for which the language to be defined is the machine language. The complete logical description of the programming language contains the description of the possible states of the abstract machine and the way these states are changed by interpreting pieces of program text.

Language definition by a compiler

Occasionally it is claimed that an implementation of a language is sufficient as the definition of the language and that all questions about the language which cannot be answered by the manual, can be answered by processing sample programs with the compiler. This of course can be claimed only for the time when a language has already been implemented and not for the development phase of the language.

A compiler designed as a tool for converting a program written in a high level language into an object program in machine or assembly language which can be executed on a machine, has several drawbacks when used as the definition of the language. A compiler is defined and operable only in connection with its environment, i.e., the machine or assembly level target language and the actual machine. If a compiler should be used as the reference for a language, it has to be ensured that for the period of time while the compiler is used as the reference both this environment and the compiler itself remain unchanged. Currently this can hardly be guaranteed to its full extent for any implementation. Furthermore an implementation of a language defines many details in the semantics of a program which are not defined by the semantics of the programming language. Thus for PL/I the compiler contains a specific choice for the order of evaluation of operands of an expression, whereas the language leaves this order explicitly undefined. When a question concerning a language is solved by processing a characteristic program it cannot be distinguished how far the result reflects the situation of the language and how far the result reflects specific implementation or hardware properties.

It seems feasible to design a compiler which avoids these problems, for reference purposes only. This compiler—besides requiring a fixed environment—would need information on all points where information in addition to the semantics of the language is needed for interpreting a program. In fact this would be an implementation of an interpretive formal definition.

The system constituting the formal definition of PL/I

In designing the system for defining PL/I the required precision of definition, the presumptive user and the impact on the language itself had to be con-

sidered. It was desirable to isolate the concepts and properties of PL/I from one another avoiding unnecessary interrelations and cross references and allowing separate consideration and evaluation of the concepts. Specifically a clear separation of all problems of program representation and notational conventions from the functional concepts of the language had to be achieved. Furthermore, it had to be shown in which areas the language requires a specific choice and additional definitions for a specific implementation.

The system designed for the definition of PL/I consists of several stages of processing and interpreting program text. The block diagram of Fig. 1 shows the elements of this system. The compile time facilities of PL/I are considered as a separate sublanguage, defining PL/I program text as the result ofprocessing PL/I source text in a compile time preprocessor. A compile time concrete syntax defines well-formed source text. The concrete syntax of PL/I program text defines well-formed programs for semantic interpretation. Before the interpretation of program text all semantically irrelevant representation properties are removed by converting concrete program text to abstract text, a tree representation for a program. The PL/I machine interpreting abstract text is so designed as to reflect the concepts of PL/I by the various constituents of its state.

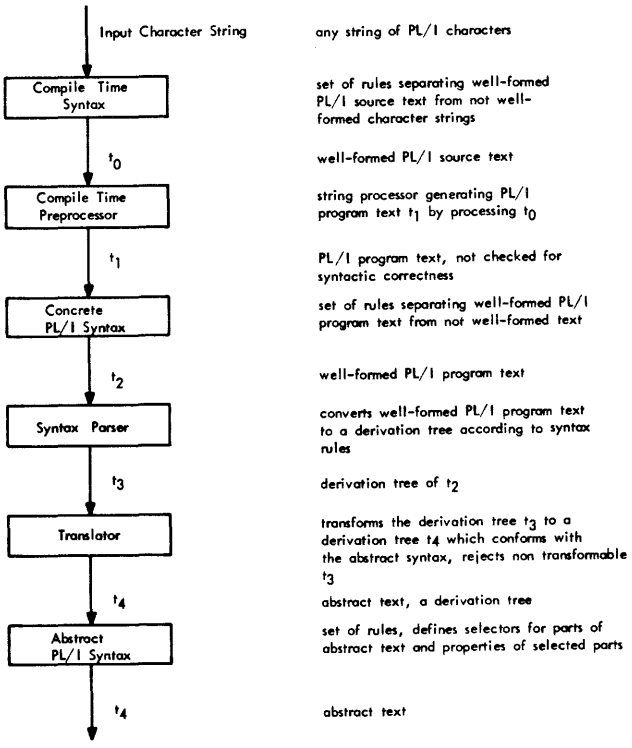


Figure 1a – System for the formal definition of PL/I – definition of program representation

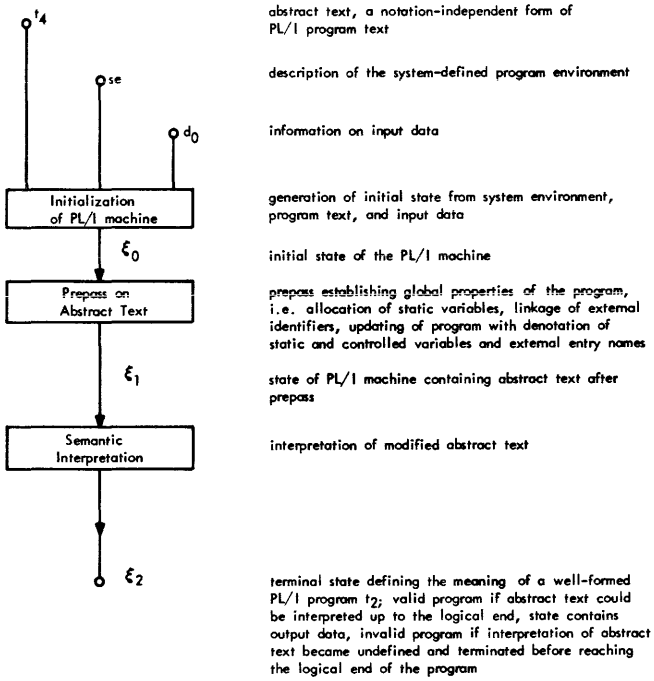


Figure 1b – System for the formal definition of PL/I – definition of program semantics

Trees and operations on trees

For the definition and handling of pieces of abstract text and parts of the state of the PL/I machine a notation has been developed which, free from redundancy, only reflects relevant properties of the considered objects.

A class of abstract objects has been defined which can be represented by trees. For this class of objects functions are given which enable the generation of trees from elementary objects and the modification of trees by deleting or changing subtrees. Properties of classes of abstract objects can be defined by an abstract syntax.

The basic functions will be explained with the help of examples. In these examples the characters enclosed in <> are meta-names used for discussion only and denote an elementary or a composite object. Capital characters denote elementary objects. The abstract objects <x> and <x'> used as examples are shown in their tree representation in Fig. 2 and Fig. 3.

Selectors on trees

In trees – or, more precisely, in the abstract objects they represent – a subtree attached to a node is selected by a selector function.

Thus in the example of Fig. 2 the sub-objects of the composite object <x> are selected as follows:

$$< r > = \text{sel-1} (< x >)$$
$$< s > = \text{sel-2} (< x >)$$
$$< t > = \text{sel-3} (< x >)$$

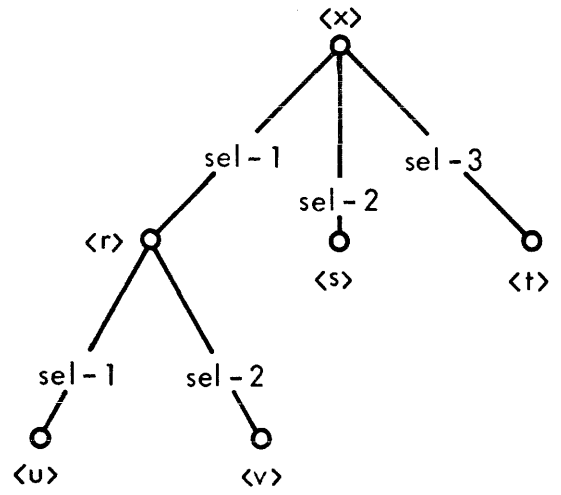


Figure 2 – Composite object <x> represented as a tree

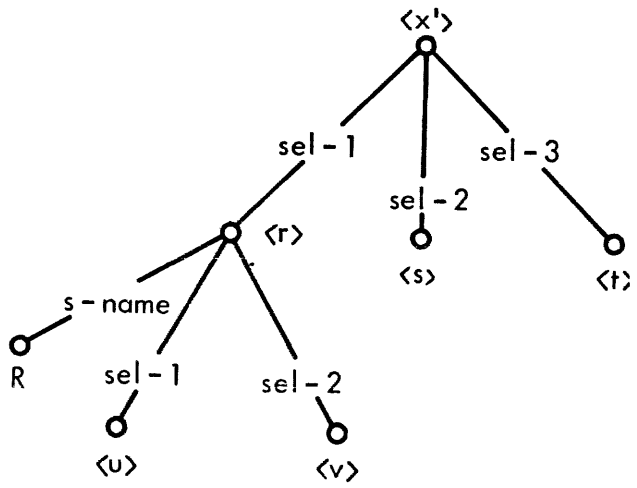


Figure 3 – Composite object <x'> represented as a tree

As <r> again is a composite object, its parts are selected by:

$$< u > = \text{sel-1} (< r >)$$
$$< v > = \text{sel-2} (< r >)$$

The object <u> can be obtained as a sub-object of <x> by a composite selector, where the dot serves for the functional composition:

$$< u > = \text{sel-1} \cdot \text{sel-1} (< x >)$$

The application of a selector to an object which has has not been specified as selecting a proper object, yields the null object Ω.

$$\text{sel-4} (< x >) = \Omega$$

The function 'is-Ω' allows checking whether a selector on an object yields a proper or a null object:

$$\text{is-}\Omega (\text{sel-3} (< x >)) = \text{FALSE and}$$
$$\text{is-}\Omega (\text{sel-4} (< x >)) = \text{TRUE}$$

It is to be noted that selectors are functions that are local to the object for which they are defined, the same selector name sel-1 is used for the object <x> and for its sub-object <r>.

Selectors on the same level do not imply an ordering. Thus a question "which is the first branch attached to the node" has no defined answer.

Nodes of a tree do not according to the definition possess names. The meta-names in < > used here serve only for the explanation. If a node requires to be named, that name has to be attached as a separate branch. In the example of Fig. 3 the object <r> possesses the name R, accessible by the selector s-name.

Generation of a tree

For the generation of an abstract object from its elementary objects, a generation function μ_0 has been designed which establishes the arrangement of selectors and selected objects to form a new composite object. For the object of Fig. 2 the generating function is written as:

$$< x > = \mu_0 (<\text{sel-1: } \mu_0 (< \text{sel-1: } < u >>, < \text{sel-2: } <v>>), < \text{sel-2: } < s >>, < \text{sel-3: } < t >>)$$

The pair selector:selected objected is enclosed in pointed brackets, where the selected object may itself be a composite object, generated by a μ_0-function, as is the case in the example given for the object selected by sel-1 on the first level.

Modification of a tree

For the modification of objects the modification function μ has been introduced. It contains as the first argument the object to be modified, while the other arguments are pairs of selectors and objects as in the generation function μ_0. The function μ can be applied in several ways. If a selector of an argument has already been defined for the object to be modified, the object paired with the selector replaces the old selected sub-object. If the selector has not yet been defined for the object, the μ-function establishes a new branch for the object with the object from the pair <selector:object> as a new sub-object.

The modification of the tree in Fig. 2 to form the tree in Fig. 3 would be written as:

$$< x' > = \mu(< x >; <\text{s-name} \cdot \text{sel-1:R} >)$$

Another possible modification of a tree is the replacement of a proper sub-object by the null object Ω. Selector functions yielding Ω are not represented in the tree representation of objects. Thus replacing a sub-object by Ω means the deletion of the respective selector from the tree. The modification of the tree <x'> from Fig. 3 into the tree <x> from Fig. 2 would be written as:

$$< x > = \mu\ (<x'>;\ <\text{s-name} \cdot \text{sel-1}:\Omega>)$$

Predicates of objects

For the definition of the structure and properties of all objects belonging to a specific class, a notation has been introduced which allows giving an abstract syntax for a class of objects. This notation in the definition of PL/I has been applied for the definition of states of the PL/I machine and for the syntax of PL/I programs in a generalized form. An abstract syntax for a class of objects again can be represented as a tree.

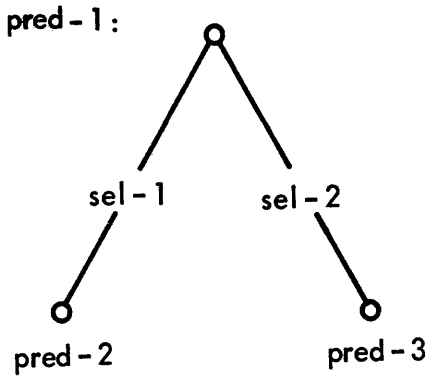


Figure 4 – Abstract syntax of a class of objects

A tree of the form shown in Fig. 4 would define the class of objects for which the predicate pred-1 is true, as the class of all those composite objects for which the application of the selectors sel-1 and sel-2 yield sub-objects for which pred-2 and pred-3, respectively, are TRUE. The application of a selector other than sel-1 or sel-2 on an object for which pred-1 = TRUE, yields Ω

The formula representation of the predicate tree of Fig. 4 in the notation used in the definition of PL/I is written as:

is-pred-1 = (<sel-1: is-pred-2>, <sel-2: is-pred-3>)

Special objects

Several places in the abstract syntax of program text as explained in the next section, required the representation of lists. As the sub-objects of an object are unordered, specific selectors elem (i) where i is integer, have been introduced to achieve an ordering of elements. This is shown as an example in Fig. 5 for an object of type pred-*l*, being a list of three elements.

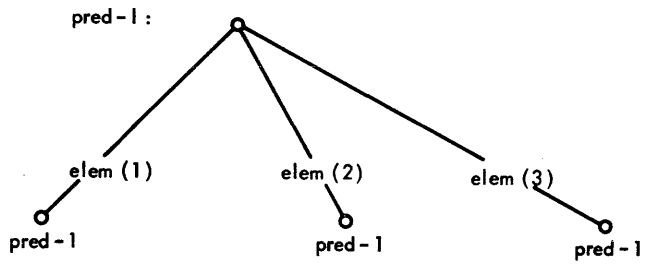


Figure 5 – Abstract syntax of a list of three elements

A specific notation is used for the formula representation of a list, the example given would be represented by:

$$\text{pred-}l = (\{<\text{elem (i): pred-1}> \mid 1 \leq i \leq 3\})$$

For objects whose sub-objects are attached by selectors of the type elem(i) normal functions on lists as length, head and tail have been defined.

The structure of the PL/I machine involves major parts belonging to a class of objects called directories. A directory is an object whose sub-objects are selected by unique names. A directory of type pred containing sub-objects of type pred-1 would be defined as:

$$\text{is-pred} = (\{ < \text{n: pred-1} > \parallel \text{is-unique-name (n)} \})$$

While in the definition of a list the condition following the vertical stroke defines the number of elements of the list, the condition following the two vertical strokes in the definition of a directory leaves the number of elements open but specifies the type of selectors.

Abstract PL/I program text and syntax

For the definition of the meaning of a PL/I program an abstract form of this program is interpreted.

The idea of representing a program in an abstract form has been shown by J. McCarthy. A well-formed concrete program has only one corresponding abstract form, whereas the abstract form of a program may have a multiplicity of concrete representations.

The abstract form for a PL/I program – its abstract text – is the result of processing the derivation of PL/I program text – a PL/I program not containing compile time statements – on the translator as shown in Fig. 1. By a set of rewriting rules the translator eliminates all semantically irrelevant notations like delimiters, and explicitly establishes those declarations which in the concrete text are defined by default rules, factored attributes, and implicit and contextual declarations. The rewriting rules also establish expansions as defined for the LIKE attribute and for not

fully qualified structure references and insert system-defined initial condition enabling prefixes. The rewriting rules eliminate the information on the ordering of branches of the derivation tree of concrete text where the order is semantically irrelevant. The abstract text can be represented as a tree.

For the abstract text a set of rules—the abstract syntax of PL/I—is given which define properties and structure of the tree representing the text. The rules define the branches which build the tree and the selectors which give access to these branches.

As an example the normal form of an assignment statement could be defined by the abstract syntax as:

is-assign-stmt = (<s-st:is-assign>,
<s-lab-list:is-name-list>,
<s-pref:is-cond-name-set>,
<s-left-part:is-reference-list>,
<s-right-part:is-expression>,
<s-name-opt:is-option>),

As explained for the tree notation used in the formal definition, selectors on the same level of an object have no implied ordering. Thus the selectors s-left-part and s-right-part in the definition of the assignment statement no longer reflect any ordering in a string. The mnemonic names only hint at the origin of the branches in the program text. In places where ordering is relevant it is reflected in the abstract text, thus, e.g., the ordering of statement labels is kept in the label list selected by s-lab-list, while the semantically irrelevant ordering of condition prefixes is no longer shown, the prefixes being in a set of names, i.e., in an unordered collection of elements.

The PL/I machine

The semantics of a PL/I program are defined by interpreting the preprocessed version of this program—its abstract text—on a PL/I machine. The PL/I machine is an abstract sequential machine not defined by hardware but as a set of possible states, together with the functions defining the transition from any state to its successor states.

The set Σ of possible states ξ which the PL/I machine can adopt is defined using the notation of abstract syntax in the same way as for abstract text.

Fig. 6 shows a simplified structure of a state of the PL/I machine.

The initial state ξ_0 of the PL/I machine includes the abstract text of the program as an argument of the first instruction in the control part.

A general language function Λ applied to a given state yields a set of successor states. The fact that a given state may have more than one successor state reflects the property of PL/I, that in several cases the order of evaluation of parts of program text is undefined.

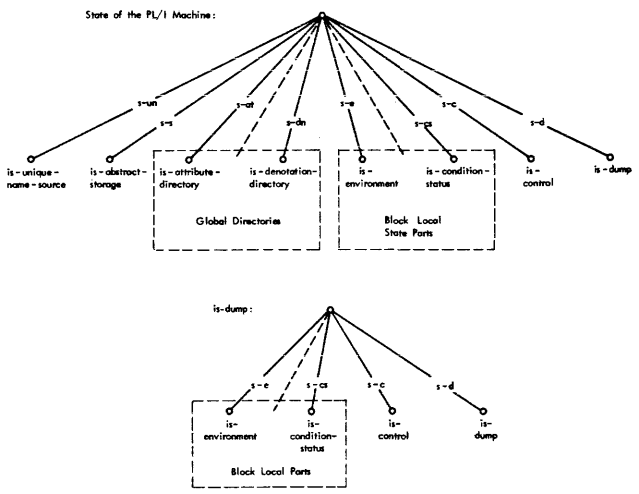


Figure 6—Structure of a state of the PL/I machine

The iterated application of the language function Λ on the initial state yields a computation:

comp (ξ_0): $\xi_0, \xi_1, \ldots, \xi_n$

where $\xi_{i+1} \in \Lambda(\xi_i)$, and $\xi_n \in \Sigma_e$

provided that one successor state ξ_{i+1} is selected from the set of states defined by $\Lambda(\xi_i)$. Making all possible selections the application of Λ to an initial state yields a set of computations. The terminal state ξ_n is element of a set Σ_e of states for which $\Lambda(\xi_n)$ is undefined, i.e., no successor state exists.

A computation is successful if it terminates on interpreting the logical end of the program. If due to a semantical error in the program the interpretation does not continue until the logical end is reached, the computation terminates unsuccessfully. On the successful termination of a program the control part of the state becomes empty. The set of computations defined for a given initial state may contain both successful and unsuccessful computations.

Unique name source N

One of the parts of the PL/I machine is a source for unique names. These names are used to denote specific elements of the abstract storage and to serve as selectors in the directories of the machine. Unique names are also used to distinguish multiple uses of the same identifier for different declarations in a program. Within the scope of one declaration in the environment part E of the PL/I machine one unique name is associated with the declared identifier.

The storage part S

The storage part S of the PL/I machine has been created to reflect the properties of storage as they are relevant in PL/I. The language requires the definition of contiguity in storage and the definition of pointers which permit reference to parts of storage not using the name for which the storage had originally been allocated. The storage part as described here is a simplified model which does not provide means for describing PL/I cells, areas and offsets and the notion of contiguity.

The storage part S is a directory of the form:

is-abstract-storage =

({<a: (<s-value: is-value-representation>)> ||
is-unique-name (a)})

The unique name a is an elementary address of storage. The value of a storage element is not directly accessible by the address, but via a second level selector s-value. This property serves the purpose of separating for a variable the cases of non-allocated storage from allocated but not initialized storage. For allocated but not initialized storage it holds that:

is-value-representation (s-value · a (S)) = FALSE.

Block activation and block local state parts

One of the salient features of PL/I is the possibility of controlling the scope of names, the scope of condition enabling prefixes, and storage allocation by the procedure and begin-block structure of a program.

The control part C, the dump D, and the block local state parts E, CS, and EI are those parts of the PL/I machine which serve specifically the purpose of interpreting the effects of block activation.

An identifier is declared in a block by linking the identifier with a set of attributes in a declaration. The same identifier can be redeclared with a new set of attributes denoting a new entity in an inner block and denotes the new entity in the scope of this block. Thus an identifier may have various uses throughout a program.

For resolving the problem of multiple use of an identifier the environment part E is used. In the interpretation of a program on the PL/I machine each identifier in its scope of declaration is associated with a unique name n from the unique name source. The pairs <identifier:n> for each block activation are collected in the environment part E.

On establishing a block activation, E is generated from the environment part of a second block, updating it with the identifiers declared in the block being activated. If the activated block is a begin block, the second block is the dynamically preceding block activation. If the activated block is a procedure block, the second block is that block in which the procedure has been declared. In the united set for a redeclared identifier the association with the new unique name replaces the old one.

On termination of a block all identifiers declared as variables of storage class AUTOMATIC loose their meaning. Storage which has been allocated for these variables has to be freed on block termination. A set of identifiers in EI serves the purpose of preserving, for use in the freeing of storage, all locally declared identifiers until block termination.

The condition status CS for a block activation contains all information on the condition enabling status as established by condition prefixes, and on the action to be performed when a condition is raised as defined by the system action or by executed ON-statements.

The control C in any state of the machine contains the instructions to be executed next. Each block activation has its own level of control, which is established on block activation and deleted on block termination, and is transferred to the corresponding part of the dump if a new block is activated. The last instruction executed before the control for the current block becomes empty, is the instruction performing the block termination.

The dump D of a state of the PL/I machine reflects the nested structure of block activations of the program being interpreted. Whenever a new block is activated, a new dump D is established, where the contents of the local directories, the dump and the control of the activating block, are stacked on top of the old dump D. On normal termination of this block the old contents as stored in the dump D are re-established in the local directories and in the control, and the top level of the dump D is deleted.

Parallel task and event part PA

In the interpretation of a program containing parallel tasks and I/O events these parallel actions are sequentialized. All active tasks and events have entries in one of the global directories, the parallel task and event part PA, containing all information necessary for their interpretation. At appropriate points in the program interpretation a priority evaluation decides which task will execute the next instruction or instructions. Several directories of the PL/I machine shown in Fig. 6 are global for all tasks and I/O events. Others are local to a specific task of I/O event. Thus, e.g., the control part has to contain only instructions belonging to one task. These task-local state parts are established for the selected after the priority evaluation using the information kept for this task in PA. When the task looses control, the contents of the task-local directories are saved by transferring them back to PA.

Meaning of names and global directories

An identifier which has been declared as the name of an entity is associated with a complete set of attributes.

In the abstract text – the normal form of the program to be interpreted in the PL/I machine – all declarations of one block are collected in the declaration part of this block. These declarations are interpreted in the prologue action for each activation of a given block. It is necessary to ensure that declarations can be retrieved during the execution of the program for various purposes like matching of attributes in parameter passing, evaluation of attributes on allocation of controlled storage or finding the block denoted by an entry name. This can be achieved either by searching through the abstract text whenever information on the text has to be retrieved or by transferring information on the text to a specific part of the PL/I machine where it can be accessed without employing a text searching mechanism.

The definition system for PL/I applies the second method. Global directories contain the meaning of names, i.e., what a name denotes, which attributes are associated with it and, if relevant, which storage has been allocated to it.

As already shown in the environment for a block activation each identifier which can be used in a reference is associated with a unique name n. This unique name serves as a selector in the attribute directory AT and in the denotation directory DN, both of which are global directories of the PL/I machine.

The *Attribute Directory* AT associates the unique name of an identifier with the attributes declared with the identifier. The entry in AT is part of the prologue action which has to be performed in the interpretation of a block activation. The attributes declared with an identifier are transferred to the respective element in AT denoted by the unique name, without performing any transformation or evaluation of the declaration. The declaraction may contain expressions which for their later evaluation need the meaning of all names appearing in the expression. For this purpose the elements in AT contain an environment part in addition to an attribute part.

The abstract syntax of AT is formally defined as:

```
is-at=({<n: (<s-attr:is-attr>, <s-e:is-e>)>||
is-unique-name (n)})
```

The attribute part with the predicate is-attr, identical to an attribute part in abstract text, has the abstract syntax:

```
is-attr=(is-prop-variable ∨ is-data-param ∨
         is-entry ∨ is-file ∨ is-based ∨...)
```

A scalar proper variable of type arithmetic would have the following sub-structure of attributes:

```
is-prop-scal-variable=

    (<s-stg-cl: (is-static ∨ is-automatic ∨ is-ctrl)>,
     <s-da  : (<s-mode: (is-real ∨ is-compl)>,
               <s-base:  (is-bin ∨ is-dec)>,
               <s-scale: (is-fix ∨ is-float)>,
               <s-prec:  (is-integer ∨
                          is-integer-pair)>)>,
    <s-init:is-initial-value>)
```

The *Denotation Directory* DN associates the unique name of an identifier with the entity it denotes. An entry name denotes the body of a procedure and the environment of the declaration, a label denotes a statement list and a block activation identification, variable names denote generations which basically are collections of storage addresses.

The abstract syntax of DN is:

```
is-dn=({<n:
    ((<s-body:is-body>, <s-e:is-environment>)
   (<s-st-list:is-statement-list>,
     <s-bl-idf:is-block-identification>)∨
   is-unique-name) > || is-unique-name (n)})
```

The *Aggregate Directory* AG has the form:

```
is-ag=({<n:is-generation-list>||is-unique-name (n)})
```

An element of a generation-list, i.e., one generation has the form:

```
is-gen=(<s-da:is-da>,
        <s-addr:is-a>)
```

Aggregates are denoted by unique names which are selectors to the aggregates in the aggregate directory.

The generation serves the purpose of collecting the storage element allocated for a variable and information on the data attributes. The respective parts of the generation are selected by s-addr, and s-da. The notion of generation reflects a basic concept derived for the current formal definition of PL/I. In PL/I variables do not always possess private storage but may be based on already existent and allocated variables. This sharing pattern is reflected by the way in which two generations comprise the same elementary storage items a (S).

Example for the interpretation of a PL/I variable

A simple example will show how the various directories are involved in the interpretation of a PL/I variable.

Let us assume a block B as

```
BEGIN;...DECLARE X FLOAT (8) AUTO-
MATIC, Y, Z;...X = expression;...END;
```

The example of Fig. 7 shows the same block transformed into abstract text, where all declarations are collected and completed in a declaration part.

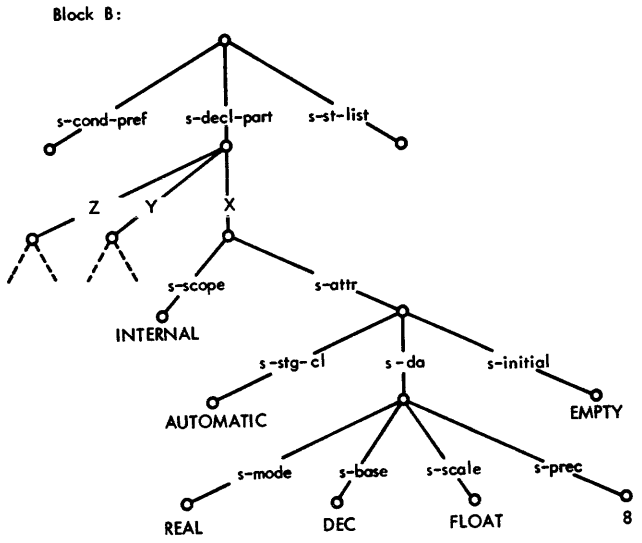


Figure 7 – Example for abstract text of a block B, containing the declaration of variable X

When the block B is activated during the interpretation of the abstract text, its declaration part in the prologue action is evaluated and parts are transferred to the local and global directories.

All locally declared identifiers are linked to unique names. The new environment is made up by selecting the unique name of an identifier using the identifier itself as selector.

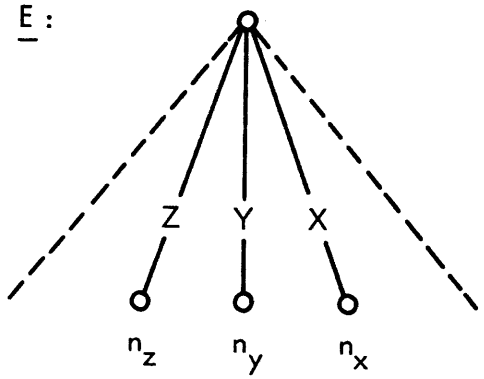


Figure 8 – Structure of an environment part containing unique names selected by identifiers

Continuing in the prologue action an entry is made for each newly declared identifier in the attribute directory AT.

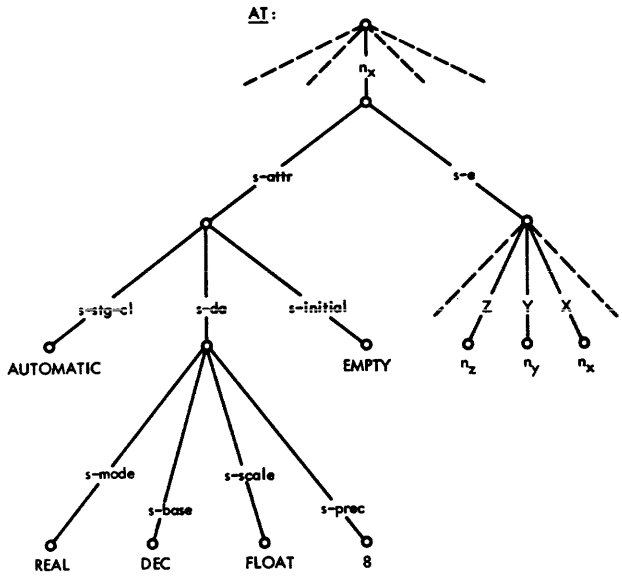


Figure 9 – Attributes and environment associated with n_x in AT

Using the same unique names n_x, n_y, n_z as selectors, the prologue action establishes entries in the denotation directory DN, taking unique names b from the set of aggregate names.

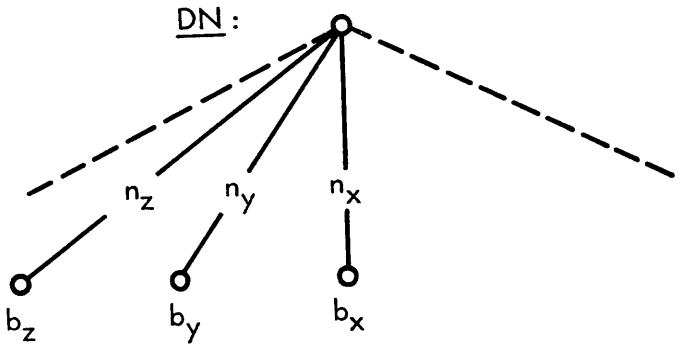


Figure 10 – Denotations for variables X, Y, Z in DN

When the denotation for the variable X has been established a generation is created. The later is established as element of AG selected by b_x.

The generation-list of b_x contains only one element, the variable X having been declared AUTOMATIC. Only in the case of CONTROLLED storage class the aggregate would contain a list of generations. The generation contained in the aggregate for X denotes one storage itme a (S) and the data attributes for a value which can be assigned to a (S).

In interpreting the statement-list of the block B there occurs a reference to X in an assignment statement. The value of the expression has to be assigned

to the proper location in storage, which is found by the access chain.

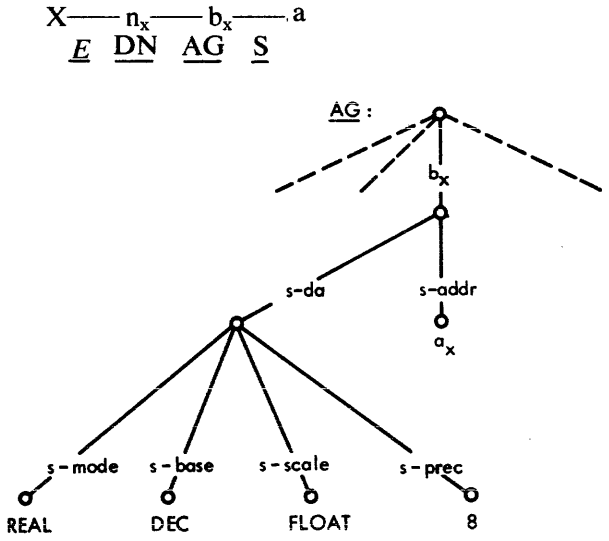


Figure 11 – Generation for a variable X

Control part and state transitions

The control part C of the PL/I machine is significant for the changes of the state of the machine in the precess of interpreting abstract text. Changes of any subpart of the PL/I machine – as in the example in previous section – can only be performed by executing instructions which are elements of the control part.

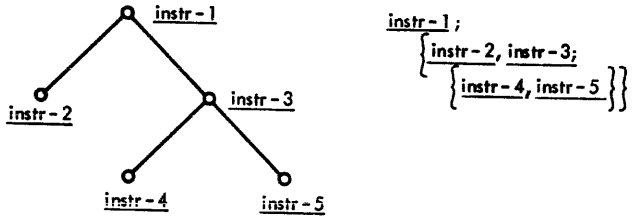


Figure 12 – Control tree and formula representation

The control part has the form of a tree where the nodes contain instructions. The formula representation equivalent to the control tree in Fig. 12 is used for representing a control tree in the formal definition.

The PL/I machine is a sequential machine, i.e., no two actions can be performed in parallel. Thus only language properties can be described which in their logical significance can be sequentialized.

All terminals of the control tree designate instructions which are candidates for execution, but only one instruction at a time is executed. This choice of one of several instructions for execution reflects those situations in PL/I where the order of execution for some program parts is undefined, as, e.g., in operand evaluation. Each instruction in the control tree can have arguments and a set of successor control trees. Only those instructions for which the set of successor control trees is empty, are the proper instructions on the terminal nodes of the control tree which are candidates for execution. Thus instr-3 in Fig. 12 is no candidate for immediate execution. Each instruction is deleted from the control after it has been executed.

A form of the control tree using specific selectors is used to allow the evaluation of arguments of an instruction.

In the control tree of Fig. 13 the arguments f_1 and f_2 are evaluated by the execution of instructions instr-2 and instr-3.

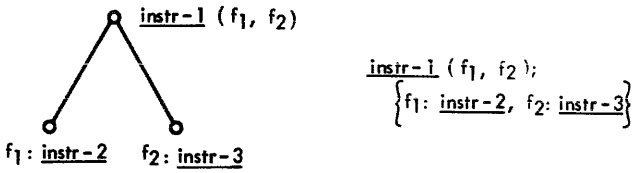


Figure 13 – Control tree, passing of argument values

A proper instruction in the Formal Definition of PL/I is defined in the following format:

instr-name $(x_1, \ldots, x_2)$ = RETURNS:e_0
s-part$_1$:e_1
.
.
.
s-part$_n$:e_n

Such an instruction on execution returns the value obtained in evaluating the expression e_0, e.g. to an argument place if used as in Fig. 12, and changes parts of the state of the PL/I machine selected by s-part$_n$.

The arguments $x_1, \ldots, x_n$ may contain pieces of the abstract text to be interpreted. The expressions $e_0, \ldots, e_n$ are expressions in the metalanguage of the formal definition, i.e. functions on abstract objects involving μ-functions and selectors. The arguments $x_1, \ldots, x_n$ can appear in these expressions.

As an example the process of updating DN with a new denotation for a declared variable with the unique name n_x as shown in the example in the previous section would require a control tree for execution:

update-dn (n_x, b_x),
b_x:un-n

The instruction would be defined as

Def.: un-n = RETURNS:head (N)
s-n:tail (N)

Def.: update-dn (t_1, t_2) =
s-dn: μ (DN;<t_1:t_2>)

The instruction update-dn (n_x, b_x) can be executed, if the execution of un-n has brought the unique name from the list N to the argument b_x. In taking off the head element of N, un-n changes the part N of the PL/I machine to the tail of its previous state. Finally the execution of update-dn with the arguments (n_x, b_x) adds a new element b to DN, selected by n_x.

The control tree is initialized in the initial state ξ_0 of the PL/I machine with initial-instruction (t_0) where t_0 is the abstract text of the program. If the program is interpreted successfully the interpretation process stops if the control C becomes empty. For an unsuccessful computation the interpretation terminates before the text is completely interpreted by arriving at a state of the machine for which no successor state is defined by the language function Λ.

ACKNOWLEDGMENT

The methods for the formal definition of PL/I as presented in this paper have been developed by P. LUCAS and K. WALK in the IBM Vienna Laboratory. Vital sources of information for the methods involved were J. MCCARTHY,[3] C. C. ELGOT,[4] and P. J. LANDIN.[5]

REFERENCES

1 *PL/I definition group of the Vienna laboratory: Formal definition of PL/I*
IBM Laboratory Vienna Technical Report TR 25.071 1966

2 IBM Systems Reference Library:
Operating system/360 - PL/I language specifications
Form-No. C28-6571-4 1966 IBM Data Processing Div White Plains N Y

3 J McCARTHY
A formal description of a subset of ALGOL—in: Formal language description languages for computer programming
T B Steel ed 1966 pp 1-12 North Holland Publ Company Amsterdam

4 C C ELGOT A ROBINSON
Random-access stored-program machines. An approach to programming languages
J ACM 11 1964 no 4 pp 365-399

5 P J LANDIN
A correspondence between ALGOL 60 and Church's lambda-notation
Comm ACM 8 1965 no 2 pp 89-101 and no 3 pp 158-165

LISP A: A lisp-like system for incremental computing

by ERIK J. SANDEWALL
Uppsala University
Uppsala, Sweden

BACKGROUND: THE LISP LANGUAGE

The work reported here is based on the LISP 1.5 programming language. Let us therefore begin with a short review of LISP 1.5. Those who have used LISP would be able to proceed directly to the section OUTLINE OF THE SYSTEM.

All *data* in LISP are in the form of S-*expressions*, i.e., correctly parenthesized expressions, whose ultimate components (*atoms*) are similar to FORTRAN variables or constants. Thus

((A 3.14 BETA) ((GAMMA) (T345)))

is an example of an S-expression. Its sub-expressions, e.g.,

(A 3.14 BETA)

(GAMMA)

etc., down to and including the atoms A, 3.14, etc. are also S-expressions.

Inside the computer, each S-expression is represented as a list structure, i.e., as a system of address references between cells in core. This need not worry us here.

For each atom, there is a *property list*, on which characteristic facts about the atom may be stored. One atom may have several properties, and they are therefore distinguished through the use of *attributes*. A property-list is essentially a collection of attribute-property pairs. Any atom may be used as an attribute, and any S-expression as property.

So much for data. The *program* in a LISP job is also written as an S-expression. This makes it possible for a program to modify itself. The program is usually interpreted at run time, rather than compiled.

The format for writing expressions in the program is exemplified through

(PLUS A 3.14 (MINUS C) (TIMES F (PLUS G H)))

which would mean the same as the FORTRAN expression

A+3.14+(–C)+F*(G+H)

Thus functions in LISP are always prefixed to their arguments, and always stand immediately after a left parenthesis. Usually, functions for manipulation of S-expressions dominate in a LISP program, and arithmetic functions like the ones in the example are only sporadically used.

There are two ways to write the program, either as a FORTRAN-like, step by step program, or as one single expression. In the former case, we may use atoms like statement numbers, and write GOTO statements in the format of (GO POSITION). In the latter case (which is considered more "pure"), conditional expressions and recursivity are used, and virtually eliminate the need for assignment statements, goto statements, etc. The statement structure collapses, and the "program" degenerates into one single expression, that is to be evaluated.

The present work is based on the latter type of "program".

As a subroutine-like feature, it is possible to define in LISP new functions in terms of elementary or previously defined ones. Suppose F1, F2, and F3 are defined as functions, and we want to define G as a function in such a way that the value of

(G X Y)

is identical to the value of

(F1 (F2 X) (F3 Y))

for any X and Y. Through the use of a certain elementary function, we put the S-expression

(LAMBDA (X Y) (F1 (F2 X)(F3 Y)))

on the property-list of the atom G and (usually) under the attribute EXPR. The atom LAMBDA plays a special role in the system, and must always be there.

It is important to distinguish between the *S-expression*

(F1 (F2 X)(F3 Y))

on one hand, and *the value* of this expression, when it is used in a (or as a) program, on the other hand. In the former case, we are referring to a piece of data, which is passive, and which is only potentially a program. In the latter case, we are referring to the output of a program. In this paper, we shall always let e.g.,

(F 1 (F2 X)(F3 Y))

refer to the S-expression itself, not to its value. If we want to refer to the value, we shall write either

"the value of the S-expression (F1 (F2 X)(F3 Y))"

or, conforming with mathematical notation,

f1(f2(x), f3(y)).

Note that the function symbols are now outside the parentheses that enclose the argument(s).

For further information about LISP 1.5, the reader is referred to John McCarthy's original article,[1] to the LISP 1.5 manual,[2] or to a recent textbook.[3] It may also be worthwhile to study some programs that have been written in LISP, as reported in e.g.[4,5,6] What has been said here should however be sufficient to get some grasp of the present paper.

Outline of the system

LISP A is an extended and modified version of LISP 1.5. Its main characteristics are:

1. Atoms for sets of objects

If an atom has certain properties on its property-list, the system recognizes it as a symbol for a set of objects. Such an atom is called an *ambject*. For example, we might have an ambject BOY for "the set of all boys".

As ambjects stand for sets, their property-lists may e.g., contain references to other ambjects that stand for subsets or supersets of the given set.

2. New: symbolic functions

Besides the types of functions used in LISP 1.5 (i.e., machine-coded SUBR's and FSUBR's, and LAMBDA-expression EXPR's and FEXPR's), LISP A recognizes *symbolic functions*, which are evaluated in a special way. Let FATHER be a symbolic function. When LISP A evaluates the form

(FATHER JOHN)

it assumes the atom JOHN to have an ambject as value,(*) and returns as value a new ambject which has the "meaning" list (FATHER JOHN) on its property-list under the attribute MEANING.

Symbolic function are used for building up data structures. They can be combined freely with other types of functions. Thus it could be sensible to evaluate the expression

(DESCRIBE (FATHER JOHN)).

The symbolic function FATHER creates an ambject for "the father of John" (unless such an ambject is already there, in which case it is retrieved, rather than re-created). The ordinary function DESCRIBE is defined through ϵ LAMBDA-expression, and it digs into the data structure to create a description of its argument.

Likewise, it might be sensible to evaluate the S-expression

(SETTRUE (ISFATHER DICK JOHN)).

Here, ISFATHER is a symbolic predicate, which is a kind of symbolic function. An ambject generated by a symbolic predicate can have an indication about truth or falsehood on its property-list. The ordinary function SETTRUE sets an indication that its argument is true, and it may also be defined to perform some inference from this fact.

3. New:RHO-expressions

There is still another type of functions, namely RHO-expressions. They have in principle the same form as LAMBDA-expressions, but they work quite differently. If

$$fn = \rho((x,y)\ g(x,y))$$

and (FN A B) is evaluated with A and B ambjects, then the system will essentially apply the function G to each combination of members of A and members of B. In other words, (FN A B) performs: "for each X in A and each Y in B, evaluate (G X Y)". The facts "X is in A" etc. must of course be retrieved from the property-lists of the ambjects A and B.

---

* It is convenient to make the convention that all ambjects have themselves as values. Thus when the system evaluates
(FATHER JOHN)
it first evaluates the atom JOHN, which is an ambject and has itself as value. This means that
father(john)
and
father(JOHN)
are identical.

4. RHO-expressions are triggered automatically.

The system takes two actions when evaluating an expression like (FN A B) above:

41. It evaluates (G X Y) for each X that is presently known to be in A, and each Y that is presently known to be in B.

4.2 It stores away the expression (FN A B) in the data base. Subsequently, each time a new Xi is added to the cases of A, the expression (G X_i Y) is evaluated for each Y in B. The symmetric action is taken when a new Y_j is added to the cases of the B.

For example, we might evaluate

ρ((x) print(describe(x))) (cousin(Peter))

where DESCRIBE has been defined before;

COUSIN is a symbolic function. cousin (Peter) is an ambject that stands for the set of all Peter's cousins;

and the RHO-operator will cause a description to be printed for each one of Peter's cousins that the system has recognized, and each cousin that it later recognizes.

Another example is

ρ((x) settrue(admire(x,father(x))))
(boy $\cap$ discussed-obj)

where FATHER and BOY have been defined before;

ADMIRE is a symbolic predicate;

SETTRUE has been defined before;

DISCUSSED-OBJ is the set of all objects that are currently interesting to the system. This is a kind of heuristic information.

For each boy who is also a discussed object, this operator will assert that the boy admires his father. The situation might be that every boy admires his father, but that we only want to execute the inference for those boys that attract our interest in particular.

In these examples, the RHO-expressions were used for their side-effects. After we have given an interpretation of rho-expressions in terms of the sets they take as arguments, we shall be able to evaluate them for their value as well. This is discussed in section SEMANTICS OF RHO-EXPRESSIONS.

5. Expressions can be treated as objects

The symbolic function SYMQUOTE makes it possible to form sets of expressions, and to manipulate those sets. SYMQUOTE creates an ambject, that stands for another ambject. (Does this sound confusing?) Remember that the value of e.g. the S-expression

(FATHER JOHN)

is an ambject that stands for John's father. The value of the expression

(SYMQUOTE (FATHER JOHN))

is an ambject, that stands for the ambject, that stands for John's father. Iterated quoting is permitted (but probably rarely useful).

The function SYMQUOTE should be distinguished from the function QUOTE in ordinary LISP. (QUOTE is also defined in LISP A). To take an example, the value of the expression

(QUOTE(FATHER JOHN))

is the S-expression

(FATHER JOHN).

This S-expression then has not been evaluated: The purpose of QUOTE is to prevent its argument from being evaluated, and to return it as it is. On the other hand, if we evaluate

(SYMQUOTE(FATHER JOHN))

we do first evaluate

(FATHER JOHN)

and obtain an ambject as value. After that, SYMQUOTE takes the argument and creates a new ambject just like any other symbolic function.

There is one important difference between SYMQUOTE and other symbolic functions, however. This has a philosophical parallel. If we know that Dick and John's father are identical persons, we can conclude that the wife of Dick and the wife of John's father are also identical. However, we can not claim that the *linguistic expression* "Dick" and "John's father" are identical. One consists of four letters, the other of eleven.

The function SYMQUOTE performs exactly that kind of quoting. Thus for any symbolic function fn *except symquote*, we would have

$a = b \supset fn(a) = fn(b)$

In conclusion, the value of the expression

(SYMQUOTE DICK)

is an ambject that stands for the ambject Dick; the value of the expression

(SYMQUOTE (FATHER JOHN))

is another ambject and stands for the ambject father (John).

SYMQUOTE can be used to speak about expressions that are already in the data base. For example, it makes sense to form the set of all ambjects obtained from the evaluation of

(FATHER JOHN)
(FATHER LUCIA)
(FATHER PETER)

etc. If this set is called FATHEREXPRESSION, it would be correct to evaluate e.g.

(SETMEMBER (SYMQUOTE (FATHER JOHN))
FATHEREXPRESSION).

Here, SETMEMBER is an ordinary function that declares its first argument to be a member of its second argument. It can be compared to SETTRUE which was used above.

6. Thanks to SYMQUOTE, LISP A "knows what it is doing", each time it uses a symbolic function.

If FATHER is a symbolic function like before, and the system evaluates the form

(FATHER JOHN),

it will create (or retrieve) two ambjects:

6.1 An ambject fj with the meaning "John's father",

6.2 An ambject qfj with the meaning "the ambject fj".

The ambject qfj is identical to what would have been obtained as the value of

(SYMQUOTE (FATHER JOHN)).

The LISP A system returns fj as the value of

(FATHER JOHN),

but before that, it has automatically performed the operation

(SETMEMBER (SYMQUOTE (FATHER JOHN))
FATHEREXPRESSION).

In this way, each RHO-expression that has previously been evaluated with FATHEREXPRESSION as one of its arguments, will now be given a chance to operate on qfj. In other words, the system is able to take care of operators that say "whenever you see an expression where FATHER occurs as the leading function symbol, do the following: –". The action done may be some logical inferences, or some external action like a printout.

7. LISP A is an incremental computer

In ordinary use, the behavior of the LISP A system is governed by the RHO operators. The evaluation of one operator may trigger another, which in its turn triggers several other operators. (The question whether the resulting trees shall be handled on a depth-first or a parallel basis, is discussed in section IMPLEMENTATION). All operators are given to the LISP A system independently, and they communicate through the changes they make in the data base (= the system of ambjects). There is no single, deterministic program. For these reasons, we characterize the LISP A system as an *incremental computer.* (This term was introduced in an article by Lombardi and Raphael.[7] For a discussion of their system, see section OTHER INCREMENTAL COMPUTERS, (towards the end of the paper).

8. LISP 1.5 is (almost) a subset of LISP A

Some small and unimportant changes have been made to LISP 1.5, but in general, everything that can be done in LISP 1.5 can be done on almost identically the same form in LISP A. The differences are described in section DIFFERENCES FROM LISP 1.5.

Examples of ambjects and symbolic functions

Before we proceed, let us specify the interpretation of ambjects and symbolic functions more in detail.

Rule 1. All ambjects in the LISP A system stand for sets that consist immediately of objects. There is no obvious way to represent the objects themselves as ambjects, nor to represent sets of sets.

Example. The ambject ESKIMOO would stand for the set of all individual eskimoos.

Example. The ambject JOHN would stand for the set whose only member the person John is.

Rule 2. All symbolic functions take sets as arguments and yield new sets as values.

Example. The ambject

father(JOHN)

stands for the set whose only member John's father is.

Example. The ambject

father(JOHN ∪ NILS)

stands for the set that has John's father and Nils' father as members. If John and Nils happen to be brothers, the set has of course only one member.

Example. Let the symbolic function DRIVERS be defined in such way that, if CAR4 stands for the set of one particular car, then

drivers(CAR4)

is the set of all persons that ever drove this car. If X-CO-CAR is the set of all cars owned by X Co., then

drivers(X-CO-CAR)

is the set of all persons that ever drove some of their cars.

Example. Let the symbolic function BELONGINGS be defined such that e.g.

belongings (JOHN)

stands for the set of all objects that belong to John. If CAR stands for the set of all cars in the world, and X-CO for the set whose only member X Co. is, then we have

$$\text{X-CO-CAR} = \text{CAR} \cap \text{belongings(X-CO)}$$

and therefore of course

$$\text{drivers (X-CO-CAR)} = \text{drivers(CAR} \cap \text{belongings(X-CO))}$$

Remark. The functions FATHER, DRIVERS, and BELONGINGS all satisfy

$$\text{fn}(a \cup b) = \text{fn}(a) \cup \text{fn}(b)\ ;$$

this rule is characteristic of symbolic functions.

One predicate(*) is important. It takes one argument and says that its argument is a set of *exactly* one member. Clearly, we have

$$*a \supset *\text{father(a)}$$

but not

$$*a \supset *\text{drivers(a)}$$

nor either

$$*a \supset *\text{belongings(a)}.$$

Remark: The fact that the LISP A system only "thinks about" sets of objects, never objects themselves, means that we have to reinterpret some statements made in section OUTLINE OF THE SYSTEM. Thus the function SETMEMBER must be defined in such a way that evaluation of

(SETMEMBER A B)

asserts that $*a \wedge a \subseteq b$.

Semantics of RHO-expressions

Like before, let

$$f = \rho((x,y)\ g(x,y)).$$

We shall interpret

$$f(a,b) \text{ as } \cup_{r \in a} \cup_{s \in b} g(\{r\}, \{s\})$$

In plain words: Let a and b be sets. Select an arbitrary r that is a member of a, and an arbitrary s that is a member of b. Form x = the set whose only member r is; y = the set whose only member s it. Construct the object g(x,y). This is a new set. Form the union of all such sets. The result is f(a,b).

From this, it immediately follows:

$$*a \wedge *b \supset \rho((x,y)g(x,y))(a,b) = g(a,b) \qquad \text{(R1)}$$

$$f(a_1 \cup a_2, b) = f(a_1,b) \cup f(a_2,b) \qquad \text{(R2)}$$

if f is a RHO-expression, and similarly for the second argument.

Consequently,

$$a_1 \subseteq a_2 \supset f(a_1,b) \subseteq f(a_2,b) \qquad \text{(R 3)}$$

if f a RHO-expression

This interpretation immediately generalizes to other rho-expressions with an arbitrary number of arguments. With this background, it is clear that the following ways of handling set-inclusion, RHO-expressions, etc., are sound:

1. Each ambject A has properties on its property-list under the following attributes:

SUBSET	The corresponding property is a list of all subsets of A(*). This includes A itself.
SUPERSET	The corresponding property is a list of all sets that have A as a subset. This includes A itself.
STAR	A flag which, if present, indicates that A has exactly one member.
OPERS	The corresponding property is essentially (see below) a list of all expressions on the form (FN ... A ...) where the leading function, FN, is a RHO-expression, and A occurs as one argument.

2. To evaluate a form whose leading function is a RHO-expression, we do as follows:
   2.1 Evaluate the arguments. They should all yield ambjects A1, A2, ... Ak.
   2.2 Create an ambject that is later to be returned as the value of the whole form. Give it a suitable MEANING property. (2.1-2.2 is the same treatment as is given forms with symbolic functions).
   2.3 Add the ambject created in (2.1) to the OPERS property of each argument A1, A2, ... Ak. (To be precise, the OPERS property of an ambject A is therefore a list of ambjects, each of which has a MEANING property which is a form with a RHO-expression as leading function and A as one of the arguments.)
   2.4 Consider each argument Ai. Its SUBSET property is a list of ambjects Ai1, Ai2, ... Aiki. Each such Aij stands for a subset of Ai. By checking for the occurrence of a STAR flag, select those Aij which have exactly one member, i.e., which satisfy
   STAR flag, select those Aij which have exactly one member, i.e., which satisfy *Aij.

By forming various combinations of such Aij, we now notice that from law A1 above, i. e.

$$*a \wedge *b \supset \rho((x,y)g(x,y))\,(a,b) = g(a,b)$$

follows

$$*a \wedge *b \supset \rho((x,y)g(x,y))\,(a,b) = \lambda((x,y)g(x,y))\,(a,b)$$

We generalize this from two arguments a,b, to k arguments A1, A2 ... Ak, and decide on the following method:

2.5 Evaluate all expressions

$$(FL\ A1m_1\ A2m_2 \ldots Akm_k)$$

where FL is the original RHO-expression, except that the RHO has been changed into a LAMBDA, and each Aim_i is a one-member subset of the argument Ai.

2.6 In step 2.5, we obtain one ambject as value for each possible combination of one-member subset. By virtue of rule R2 above, each such ambject is a subset of the ambject created in (2.2). Mutual references are therefore put on the SUPERSET viz. SUBSET properties.

3.1 Checks whether the fact a ⊆ b is already known (i.e., whether the ambject be is already on the SUPERSET property of a). If so, it returns; otherwise, it continues.

3.2 Adds each member b′ of the SUPERSET property of b (including b itself) to the SUPERSET property of a. If we know *a (i.e. if a has the flag STAR), we also apply the operators of b′ to a; see step 3.4.

3.3 Adds each member a′ of the SUBSET property of a to the SUBSET property of b. If we know *a′, we also apply the operators of b to a′, see step 3.4. Then return.

3.4 To apply the operators of b″ to a″, first retrieve the OPERS property of b″. It is a list of ambjects whose MEANING properties are forms

(F1 ... ... B″ ...)

F1 is a RHO-expression. It would be possible and legal to re-evaluate these RHO-expressions as described in step 2.4-2.6.* In step 2.5, where we form all combinations of one-member subsets, we would then obtain combinations where a″, the one-member subset of b″, is included, which is what we desire. However, we would *also* obtain combinations where other subsets than a″ are used, and these combinations have already been considered. To avoid this inefficiency, we first put the SUBSET property of b″ on the push-down-list, and replace it with another property that *only contains a″*. After that, all forms on the OPERS property of b″ are evaluated, and finally the old SUBSET property is restored from the push-down-list.

4. There is a function SETSTAR, which is similar to SETSUBSET. It puts the flag STAR on its sole argument, and triggers the necessary operators. The details are analogous to those for SETSUBSET.

The above algorithm can be considered as an encodement of the rules R1 -R3 for RHO-expressions and the * predicate, as given at the beginning of this section. As each occurrence of a symbolic function is reported via SYMQUOTE and may trigger operators, axioms for symbolic functions and predicates may be encoded as RHO operators.

Logic with four truth-values

We agree that a symbolic predicate should be a special kind of symbolic function. It is desirable that

* More precisely, is a list of all ambjects, that stand for subsets of the set that A stands for.

(*)Step 26 must be slightly modified; the ambject created in step 25 is now to be a subset of the ambject that carries the RHO-expression i.e. the ambject on b″:s OPERS property.

the difference between predicates and other functions should be kept as small as possible. In particular, to make it possible to use predicates inside RHO-expressions, we should let the value of a symbolic predicate be a *set of truth-values*.

Let t and f be the original truth-values, defined on relations between objects. Introduce the sets

$$T = \{t\}$$
$$F = \{f\}$$
$$S = \{t,f\} \text{ (stands for "sometimes")}$$
$$= \{\ \}$$

It easily follows

$$*p \wedge p \subseteq T \supset p = T \qquad \text{(R4)}$$

This rule is important and will be used below.

We now extend the ordinary logical connectives to this four-valued logic. The connectives shall satisfy the general axiom fn(a ∪ b) = fn(a) ∪ fn(b), so we have e.g.

$$T \vee T = \{t\} \vee \{t\} = \{t\} = T.$$
$$T \vee S = \{t\} \vee \{t,f\} =$$
$$\{t\} \vee (\{t\} \cup \{f\}) =$$
$$(\{t\} \vee \{t\}) \cup (\{t\} \vee \{f\}) = \ldots$$
$$T \cup T = T$$

Let us not here delve further into this logic. One of our early examples of the use of RHO-expression was

$$\rho((x)\ \text{settrue(admire(x,father(x)))})$$
$$(\text{boy} \cap \text{discussed-obj}).$$

If we assume the very reasonable axioms

$$*a \supset *\text{father(a)}$$

and

$$*b \wedge *c \supset *\text{admire(b,c)},$$

we can re-write the operator on the equivalent form

$$\text{settrue}(\rho((x)\ \text{admire(x,father(x))})$$
$$(\text{boy} \cap \text{discussed-obj})).$$

If we evaluate the operator in this latter version, the following will happen (although not necessarily in this order):

1. The ambject boy ∩ discussed-obj is evaluated and assigned some properties by operators that are triggered by the use of the function ∩.
2. The RHO-expression is applied to its argument. A new ambject, af, is introduced.
3. The function SETTRUE is applied to af, which is then set equal to the set T through a property.
4. Let r be an ambject which has obtained the flag STAR, and which has the ambjects BOY and DISCUSSED-OBJ on its SUPERSET property. When the last of these three conditions becomes satisfied, the above RHO-expression will be triggered. The system evaluates father(r) and admire(r,father(r)). These will of course be new ambjects. If the above axioms are properly encoded, the system will put the flag STAR on them. Moreover, by the specification of how to handle RHO-expressions, the system will evaluate

   setsubset(admire(r,father(r)), af),

   where af is the ambject introduced in step (2). By rule (R4) above, the ambject admire(r,father (r)) will be set EQUAL to af and therefore to T.

This was one example of how RHO-expressions can sometimes be evaluated for their value, rather than for their side-effect.

Differences from LISP 1.5

Through the introduction of new types of functions, the traditional means of handling functions in LISP become inconvenient. We found an alternative system of conventions, which may have some interest in itself.

The association-list in ordinary LISP assigns values to atoms. The value may be a FUNARG - expression (in which case the atom can be used as a function symbol) or an arbitrary S-expression (in which case the atom can only be used as the argument of some function).

On the association-list, it does not matter whether an atom stands for a function or something else, but in other parts of the LISP system it does. On property-lists, functions are defined as EXPR-properties or FEXPR-properties; other values as APVAL-properties. If the atom G has the EXPR-property gg, then the two expressions

(FUNCTION G) and
(FUNCTION gg)

are equivalent, but if the atom A has the APVAL-property (aa), the two expressions

(QUOTE A) and
(QUOTE aa)

are not at all equivalent.

In LISP A, the distinction between functional and other values is abolished. This leads to the following consequences:

1. The pseudo-function FUNCTION is superflous. We use QUOTE instead.
2. When LAMBDA-expressions are used directly in forms, they must be quoted. To avoid con-

fusion, we introduce the symbol ETA to be used instead of LAMBDA. Thus the following would be a correct form:

((QUOTE (ETA (X Y) (.....)))

(..... .....)

(.....))

3. When function definitions (e.g. LAMBDA-expressions) are named by atoms, they are put as APVAL-properties (viz. as VALUE-properties in PDP-6-LISP-type implementations).
4. Different kinds of functions, which were previously distinguished through their attributes (EXPR, FEXPR, SUBR, etc.) must now be distinguished some other way. We use the following transformations:
   - 4.1 A previous EXPR on the form
     (LAMBDA a b)
     is re-written on the form
     (ETA a b)
   - 4.2 A previous FEXPR on the form
     (LAMBDA (a1 a2) b)
     is re-written on the form
     (PHI a1 b).
   - 4.3 A previous SUBR is re-written on the form
     (ECODE . a)
     where a is the starting address for the machine code routine.
   - 4.4 A previous FSUBR is re-written on the form
     (FCODE . a)
     with a as for ECODE.
5. The general evaluation function for LISP A, evala, evaluates the leading function of a form just like any of the arguments. It therefore becomes possible to evaluate the expression for a function immediately before it is used. For example, we can write

   ((DERIVATIVE SINE) X)

   where (DERIVATIVE SINE) evaluates into the value of COSINE(*) which is then applied to the value of X.

The above changes have the obvious advantages of preparing the ground for the two types of functions in LISP A: symbolic functions and RHO-expressions.

(*) The value of the atom SINE is most likely an expression (ECODE . a). The function DERIVATIVE takes this as argument and uses it in an expression of the type

derivative (f) =
if ..... *else*
if f=sine *then* cosine *else* .....

The value of (DERIVATIVE SINE) is therefore an expression (ECODE . b), which also happens to the value of the atom COSINE.

To increase compatibility with LISP 1.5, it is possible to define functions LAMBDA and LABEL as PHI-expressions. If we let the value of the atom LAMBDA be

(PHI R (CONS (QUOTE ETA) R))

we can use LAMBDA-expressions as leading functions in forms, just like in LISP 1.5 (LAMBDA-expressions that were EXPR's or FEXPR's must of course still be transformed into ETA- or PHI-expressions). A similar definition of LABEL becomes a little bit more involved.

Implementation

A preliminary version of evala, the evaluation function in LISP A, was coded in ordinary LISP for the PDP-6 computer. That version lacked some facilities that have been described here. Most important, it only permitted RHO-expressions to be evaluated for their side-effects, not for their value. On the other hand, there were some additional facilities in that system.

A modernized version of evala according to the specifications in this report is currently (March, 1968) available in LISP for the CD 3600 computer.

The crucial feature in these implementations was the use of a *queue* for expressions-to-be-evaluated. The need for this arises e.g. when we handle RHO-expressions that contain one or more occurrences of symbolic functions. Such a RHO-expression *is triggered* by the evaluation of an expression

(SETSUBSET .....);

it *itself triggers* evaluation of similar expressions (e.g., when SYMQUOTEd expressions are set as subsets of larger sets). Obviously this process may continue in a chain or tree. At each step, several new RHO-expressions may be triggered.

The order of evaluation of such expressions can be chosen in several ways:

1. *Depth-first.* If evaluation of expression e triggers expressions f1, f2, . . . , fk, we first evaluate f1 and all its consequences to the end of the tree; and only then start to evaluate f2.
2. *Queueing.* We keep a queue of expressions that are to be evaluated. If e triggers f1, f2, . . . , fk, these operations are put at the end of the queue, and the evaluation of all is postponed until the expressions before them in the queue have been handled. When we reach f1, we put the expressions that it triggers at the end of the queue; proceed to f2, etc.
3. *Queueing with priority.* This is like (2), except that expressions which are deemed particularly

significant, are permitted to step into the queue at some point other than its end.

Other alternatives, and more sophisticated ones, are also possible. In our implementation, we have preferred the "queueing with priority" scheme.

Other incremental computers

The System by Lombardi and Raphael.

The idea to use LISP as the basis for an incremental computer is not new; it was originally put forward by L. A. Lombardi and B. Raphael.[7] They describe a modified LISP system which can (in some sense) evaluate expressions, even when the values of some variables have not been specified.

Lombardi and Raphael specify three key requirements for an incremental computer. We shall relate the present work to theirs by discussing whether LISP A satisfies those requirements. The first of them is:

"The extent to which an expression is evaluated is controlled by the currently-available information context. The result of the evaluation is a new expression, open to accommodate new increments of pertinent information by simply evaluating again with a new information".

In LISP A, we can write expressions which satisfy this by using RHO-expressions. (We can also avoid paying the cost for it by using LAMBDA-expressions.) The "current information context" for forms with a RHO-expression as their leading function is information about subsets of the arguments in this form. We would say that the form has been completely evaluated when all one-member subsets of all arguments are explicitly known, and all combinations of them have been considered by the system. Usually, this is not the case, and evaluation is performed to an extent "controlled by the currently-available information context".

As we have seen, forms with RHO-expressions are stored away in such a way that they are "open to accommodate new increments of pertinent information" about subsets.

The LISP A system does not satisfy Lombardi and Raphael's second requirement ("algorithms, data, and the operation of the computer (!) should be specified by a common language") or their third condition (this language should be understandable by untrained humans). But neither does their incremental LISP, nor any other system we have heard of.

Our system is extremely inefficient in terms of computer time, and it can be assumed that theirs is less wasteful. On the other hand, LISP A seems to have the following two advantages over their system:

(1) When an expression is evaluated incompletely for lack of information, the system remembers this and resumes evaluation when further, pertinent information increments become available.

(2) Through the introduction of ambjects and symbolic functions, our system comes closer to having "a large, continuous, on-going, evolutinary data base", which should be the characteristic environment of an incremental computer. The data base of Raphael's program is identical to that of LISP 1.5, i.e. it is restricted to property-lists of atoms.

Lombardi has later published a more extensive treatise of incremental computers.[8] He there concentrates on the basic representation of data in core, and introduces his own system with three references in each cell. These seem to be quite different problems from the ones tackled in this paper, and a comparison is therefore not attempted.

Future Developments of LISP A.

The following developments seem natural:

A. Write a machine-coded version of evala.

B. Introduce a notation through which pseudo-parallell execution of several expressions can be performed. This is very natural, since e.g., the order of evaluation of the specializations of a RHO-expression is immaterial.

C. Attack storage problems by making use of backing storage, drum or disk. Facilities for parallel execution of expressions then become very important, because they help us to use the time when we are waiting for information from backing storage.

SUMMARY

LISP A is a modification and extension of LISP 1.5. Besides minor modifications, two new types of functions have been added to the language. One type *(symbolic functions)* is used to create and extend the data base. If ISFATHER is a symbolic function, evaluation of (ISFATHER JOHN DICK) will create a representation for the relation in the data base, without asserting its truth. This representation can then be used with conventional LISP functions, which set it true, ask whether it is true, etc. The other type (*RHO-expressions*) can be used to write a kind of rules of inference, which are automatically triggered in desired situations. The LISP A system is governed by such RHO-expression operators, which trigger each other. There is no coherent program, just a set of operators which communicate through the changes they cause in the data base. The paper gives a general description of the LISP A system.

ACKNOWLEDGMENTS

The author is indebted to professor John McCarthy at Stanford University for his kind guidance during the year 1966/67, when the work reported here was started. He would also like to thank fil.lic. Jacob Palme of the Research Institute of National Defense, Stockholm, Sweden, for many valuable discussions during the last year.

REFERENCES

1 J McCARTHY
Recursive functions of symbolic expressions and their evaluation by machine, part I
Communications of the ACM *3* (April 1960) p 184

2 J McCARTHY et al
Lisp 1.5 programmer's manual
The M.I.T. Press 1962

3 C WEISSMAN
LISP 1.5 primer
Dickenson Publ Co Belmont Cal 1967

4 E C BERKELEY et al
The programming language LISP: Its operation and applications
The M.I.T. Press 1966

5 D G BOBROW
Natural language input for a computer problem solving system
Doctoral Thesis M.I.T.

6 B RAPHAEL
SIR: A computer program for semantic information retrieval
Doctoral Thesis, M.I.T.

7 L A LOMBARDI B RAPHAEL
LISP as the language for an incremental computer in Ref. 4

8 L A LOMBARDI
Incremental computation
In Frank L Alt ed
Advances in Computers Vol 8
Academic Press New York 1967

TGT: Transformational grammar tester*

by DAVE L. LONDE and WILLIAM J. SCHOENE
System Development Corporation
Santa Monica, California

INTRODUCTION

Chomsky defines a generative grammar as one that "attempts to characterize in the most neutral possible terms the knowledge of the language that provides the basis for actual use of language by a speaker-hearer."[1] It is "a system of rules that in some explicit and well-defined way assigns structural descriptions to sentences."[1] The syntactic component of such a grammar specifies the well-formed strings of formatives (minimal syntactically functioning elements) in the language and assigns structures to them.

Transformational grammars are built on the concept of the logical separation of two types of structure, the deep and the surface structure. Accordingly, there are two systems of rules in the syntactic component of a transformational grammar: phrase structure rules generate deep structure, taking the form of a labeled bracketing or tree; transformational rules map trees to other trees and determine the ultimate surface structure of a sentence. The deep structure is operated on by the semantic component of the grammar to determine meaning; the surface structure is interpreted by the phonological component.

The emphasis on explicitness is a distinct advantage of generative grammars. However, it imposes a great burden on the linguist. Given a deep structure, the determination of the applicable rules in the derivation of the surface structure of a particular sentence is an extremely tedious and time-consuming task, difficult to perform with accuracy. For example, verifying by hand the correctness of the derivation of typical sentences in the IBM Core Grammar[2] took us on the average two hours per sentence. And, as the grammar becomes large (as the linguist attempts to account for more phenomena in the language he is describing) it becomes more difficult to provide for all of the possible interrelations of the rules.

*The work reported herein was supported in part by contract F1962867C0004, Information Processing Techniques, with the Electronic Systems Division, Air Force Systems Command, for the Advanced Research Projects Agency Information Processing Techniques Office.

TGT is a system of computer programs that can provide assistance to linguists in building and validating transformational grammars. The name TGT, "Transformational Grammar Tester," connotes a more ambitious project than was intended or undertaken. Not enough is known about some of the components (e.g., the semantic component)[3] to warrant testing. And although there has been a considerable amount of work done on phonological rules, TGT has been addressed almost exclusively to the debugging and maintenance of the syntactic component.

For more than a decade, computers have been used experimentally to process segments of natural language text according to syntactic rules. The experiments have included both synthetic syntactic processing (generating sentences, together with their structural descriptions according to some previously specified grammar) and analytic syntactic processing (recognizing for a given sequence of formatives considered to be a sentence, each structural description, if any, that the grammar can assign to the sequence).[4,5]

Unlike TGT, projects engaged in syntactic processing of natural language have seldom had as their sole objective the refining and extending of the grammar initially specified; often this has not been the primary objective.

Nevertheless, important contributions to grammar validation have been made by Petrick,[6] although the technique used, that of recognition, is appropriate only for grammars that are relatively complete.

The system for syntactic analysis at The MITRE Corporation, described in Zwicky, *et al.* [1965],[7] and modified as described in Gross [1967],[8] is currently being used to perform some of the same functions included in TGT.

An off-line system for compiling, updating and testing the IBM Core Grammar was written in LISP 1.5 by Blair [1966][9] and is currently in use at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York.

Further work on an off-line computer system for grammar testing by synthesis and by constrained

"random" generation is being currently performed by Dr. Joyce Friedman at the Department of Computer Sciences, Stanford University, with whom we have probitably exchanged information.

TGT was designed with the goal of being as general as possible while still accommodating its immediate and primary user, the Air Force UCLA English Syntax Project (AFESP)* which is currently reviewing and attempting to integrate the work that has been done on transformational analyses of English. While it was impossible for AFESP at the outset to make definite decisions regarding its needs, since principled decisions on matters such as rule conventions could only be made after the data were gathered and analyzed, it seemed apparent to us that its needs could better be served by a new system, that (among other things) would be user-oriented and interactive, would handle subcategorization and selectional features, and would facilitate extension and modification. (However, for comments on factors limiting these goals, see the Discussion section at the end of this paper.)

System overview

In constructing a grammar with TGT, the linguist will be asking the same questions and employing many of the same procedures that he would have used were the tester not available. With the tester, many of those amenable to programming can now be done by the computer. The current version of TGT is programmed in the JTS version of the JOVIAL language and occupies 38,800 words of 48 bit memory, operating under the time-sharing system for the AN/FS Q-32 computer. An ITT model 33 teletype is used as the standard input/output device. A CRT with a capability of displaying 680 characters on a 1024 × 1024 matrix is an optional device for output. CRT output is faster and more readable; but its use, because of hardware requirements, dictates proximity to the Q-32. Teletype output is not subject to such distance limitations and is frequently used, even by those who have a CRT, as a means of obtaining hard copy output.

The most important tasks to be performed by TGT center around its ability to execute transformations. Using TGT the linguist can determine the applicability of transformations, execute them and display their results. Many ancillary functions are provided for specification and manipulation of rules and test structures. Combinations of these functions are employed to aid the user in determining the implications of changes in the rule set. For example, the linguist can save entire derivations of sentences that he considers correct. He may then insert a new rule, or change or delete an existing one, and then have the computer apply the rule set to the base phrase markers of the saved sentences. Changes in the derivations can then be immediately determined.

*Capabilities**

A. Displaying and saving trees

Most TGT operations apply to the most recent or "current" tree in memory. Trees may be created by the primitive DISPLAY (abbreviated as D). The tree in Figure 1 could have been initially input by typing:

```
D (S(# PRE(Q) NP(DET(ART(WH DEF))
    N [THING <+N> <+PRØ> <-HUMAN> <-SG>])
    AUX(T(PAST)) VP(V[DISAPPEAR <+V>] (#))        (1)
```

The components of the tree (categories, features, and complex symbols) are automatically numbered.

The "current" tree may be named and saved if desired by typing SAVE or S followed by an alphanumeric string. If there is already a tree in the system with that name, TGT informs the user, who is then given the option of changing the name of the new tree or replacing the old tree. The most recently used trees are saved in core memory. The others are stored on disc. Saved trees may be restored at any time by using the DISPLAY primitive with the name of the saved tree. This then qualifies as the most recently displayed or "current" tree.

Trees are initially left-justified on the scope. If the entire tree cannot be displayed, a message is output indicating the numbers of the top-most nodes of the major subtrees that are not displayed. Normally a right-justify command (RJ) is sufficient to display the missing subtree(s). However, an additional command, CENTER, is also useful in these cases. Accompanied by a number, it causes the subtree with that number to be displayed exclusively. CENTER also enables the user to position the current tree anywhere on the scope by specifying the direction in which the tree is to be moved (left, right, up, or down) and the number of units it is to be moved.

The tree in Figure 1 may be listed on the teletype by typing the command LIST, resulting in the TTY printout shown in Figure 2.

It is seldom necessary to input an entire tree parenthetically as in (1) above. Most desired trees can be produced by altering a few basic trees with the following primitives:

* The official title of the project is, Integration of Transformational Studies on English Syntax, Principal Investagator—Robert P. Stockwell, Co-Principal Investigators—Paul M. Schachter and Barbara Hall Partee.

*A more detailed account of system capabilities can be found in "TGT User's Manual," by W. J. Schoene, System Development Corporation document (in press).[10]

	*Input**			*Explanation*
(2)	ERASE		N1	Erases subtree whose number is N1
	E		N1	(that is, N1 and all that it dominates).
(3)	AD	A	N1	Add A as a daughter to N1.
(4)	ALS	A	N1	Add A as a left sister to N1.
(5)	ARS	A	N1	Add A as a right sister to N1.
(6)	SUB	A	N1	Substitute A for N1.
(7)	EAD	N1	N2	Erase N1 from its original position and add it as a daughter to N2.
(8)	EALS	N1	N2	Erase N1 and add it as a left sister to N2.
(9)	EARS	N1	N2	Erase N1 and add it as a right sister to N2.
(10)	ESUB	N1	N2	Erase N1 and substitute it for N2.

*Where N1 and N2 represent the numbers of nodes of the current tree and A may be one of the following: the number of a node in the current tree, the name of a saved tree, or a structure (subtree, feature or complex symbol) input in the parenthetic notation of (1) above.

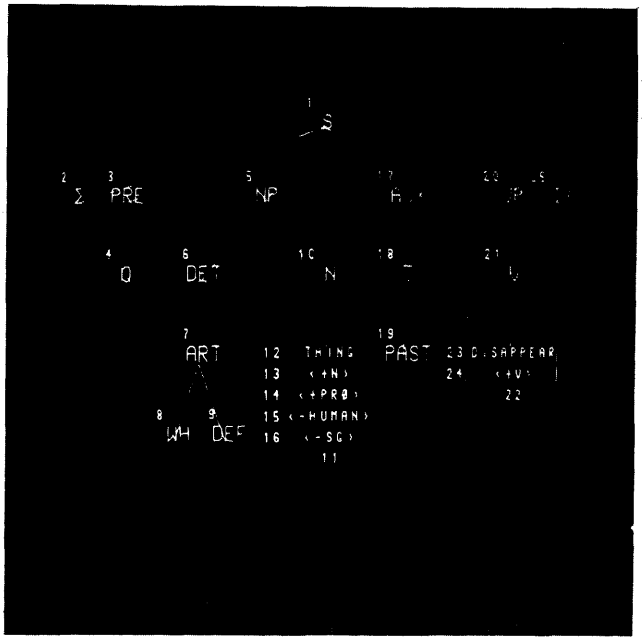


Figure 1—CRT display of a tree input from teletype and representing the structure of the sentence, "What things disappeared?"

```
(S 1)
  (# 2)
  (PRE 3)
    (Q 4)
  (NP 5)
    (DET 6)
      (ART 7)
        (WH 8)
        (DEF 9)
    (N 10)
      (*CS* 11) [ (THING 12) (<+N> 13) (<+PRO> 14) (<-HUMAN> 15)
                  (<-SG> 16)]
  (AUX 17)
    (T 18)
      (PAST 19)
  (VP 20)
    (V 21)
      (*CS* 22) [ ( DISAPPEAR 23) (<+V> 24)]
  (# 25)
```

Figure 2—Teletype representation of the tree in Figure 1

B. Phrase structure rules

Phrase structure rules are not essential to the operation of TGT, but their presence permits a legality check on the trees. The command VERIFY, input from the teletype, will return the message YES or NO indicating that the current tree could or could not have been produced by the phrase structure fules. In the present system, the complex symbols (e.g., those symbols enclosed in straight line brackets in Figure 1 and numbered 11 and 22), are ignored by the VERIFY subroutine.

As in Chomsky [1965][1] phrase structure rules are expected to be context free. They may be entered into the system by typing the identifier PS followed by the symbol to be expanded, an arrow (minus sign and greater-than sign) and the string indicating the legal expansion.

$$\text{PS VP} -> \text{AUX (MV (NP, NIL),}$$
$$\text{COP (NP, ADJ, NIL))} \qquad (11)$$

Example (11) above indicated that VP may be expanded as any one of the following strings:

AUX	MV	
AUX	MV	NP
AUX	COP	
AUX	COP	NP
AUX	COP	ADJ

Parentheses indicate that of the strings within that are separated by commas, just one is to be chosen. A NIL within the parentheses indicates that none need be chosen.

$$\text{PS S} -> \text{(S CONJ S (CONJ S, NIL)*, NP VP)} \qquad (12)$$

Example (12) indicates that S may be rewritten as

```
NP   VP
S    CONJ  S
S    CONJ  S  CONJ  S
S    CONJ  S  CONJ  S  CONJ  S
etc.
```

Thus, an asterisk following parentheses indicates reapplicability of the set within.

The expansion of a symbol can be changed merely be reinputting the symbol and its new expansion. TGT saves only the most recent.

C. Transformations*

Transformations are input via the teletype using the operator RULE. The canonical form is:

RULE rule-name structural-description => structure-change · conditions

(conditions: optional)

RULE HYPOTHET1 X A *1B *2(C,D E, *3F) X G *4B X =>

1 = 2 + 1; 2 = Ø; 3 = Ø. 4 EQ 1. (13)

In our format, only structures to which a structure change or a condition applies need be numbered. Any structure, including features and dominating and dominated symbols, may be numbered. (However, the variable X is subject to certain limitations.) A number immediately preceding a choice set applies to each of the left-most members of the set that are not already numbered. Thus, if a certain proper analysis** of the "current" tree is found, the change 1 = 2 + 1 will add the structure headed by C or D as a left sister to the structure headed by B and then erase the original C - or D-dominated structure. If, however, the proper analysis was found with *3F following *1B, the structure headed by F is erased. This rule applied to the tree in Figure 3 yields the tree in Figure 4.

A special condition on this transformation, which must be satisfied before any structural changes can be performed, is that the structure headed by *4B be the same as the structure of *1B. TGT is designed to facilitate the addition of conditions on transformations. There is no limit to the number of conditions that may be imposed on a structure.

Rule H2 in (14) below introduces a notation for expressing proper analyses of subtrees within the current tree, and for specifying fixed length variables and features.

RULE H2 X [B *1C D]A / < + F >< + G> /E *2N *3P [X / <+H – I> /Z X]J X => 1 = 1 + *4M; 4 < L 2; 4 < L3 (14)

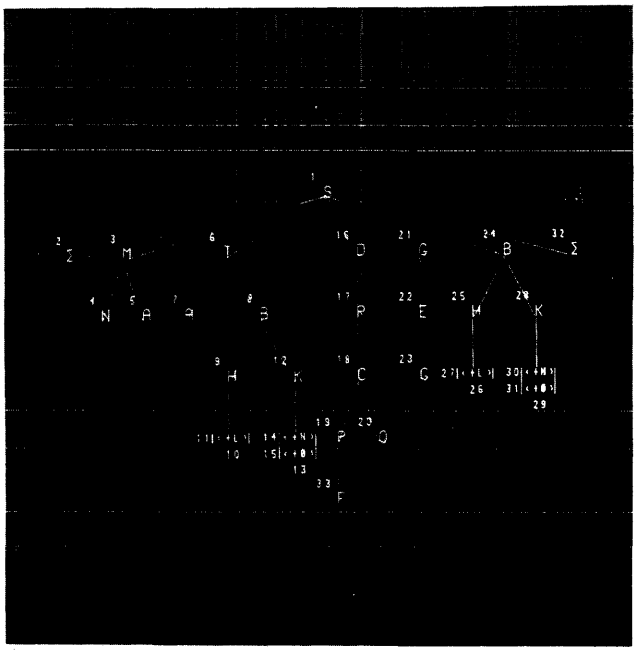


Figure 3 – Sample tree to which transformational rule HYPOTHET1 is applied

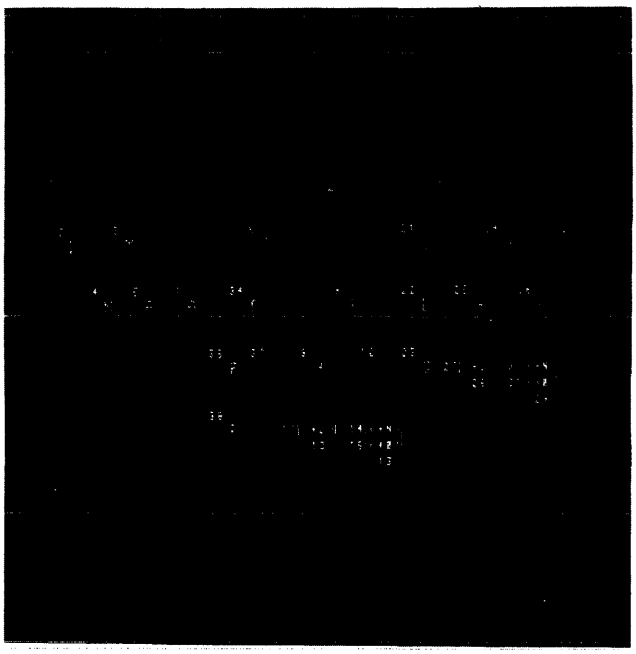


Figure 4 – Results of applying rule HYPOTHETI to the tree in Figure 3

*A detailed description of the algorithm used to determine proper analyses and to perform structure changes can be found in "An Algorithm for Pattern Matching and Manipulation with Strings and Trees" by D. Londe and W. Schoene, System Development Corporation document (in press).[11]

**A formal description of proper analysis can be found in Petrick [1965].[6]

Square brackets indicate that the first symbol following the right bracket (]) must dominate the sequence of symbols inside the brackets and, furthermore, that this sequence must constitute a proper analysis of the dominating symbol.

TGT permits bracket nesting to any level. If the linguist is concerned only that a symbol A dominate a symbol B regardless of whatever else it may dominate, he can surround the B with general variables, [X B X]A, thus preserving the convention for interpretation requiring a proper analysis. *Immediate* dominance can be expressed as a special condition on the dominated symbol, i.e.,
X[X *1B X] A X => structure changes. 1 COND5. Where condition 5 requires that the node to which it applies (in this case *1B) be immediately dominated by the symbol immediately following the right bracket (A).

Because of the comparatively recent acceptance of features in transformational grammars and the variety of their function in different writings, TGT handles them internally as if they were special conditions on the symbols immediately dominating them, thus facilitating modification. Rule H2 specifies that the E following [B C D] A must *immediately* dominate the features < + F > and < + G >. Where the linguist is concerned that a symbol dominate a particular feature at some unspecified distance, he may make use of the notation indicated by [X/ <+H – I> /Z X]J in rule H2, where Z is a variable of fixed length, one node. We thus preserve our convention of specifying the immediate dominator of features.

A very convenient innovation is illustrated in the structure change portion of this rule. Generally structure changes manipulate structures that are present in the tree before execution of the rule. Thus, when we specify that some structure numbered 2 is to be added as a daughter to some structure 1, and that 1 is to be adjoined as a right sister to 3, we intend to adjoin 1 in its original state, i.e., before 2 was added.

In TGT we have provided a notation that allows numbers to be assigned to structures that are created by the structural change portion of the transformation, whether these be newly introduced constants or structures that were already in the tree and moved to new positions. These new structures may subsequently be moved or have structures adjoined to them. In H2 we are adding a new symbol M as a right sister to *1C, and to this new symbol we are adding *3P and *2N as left daughters. Rule H2 applied to the tree in Figure 5 yields the tree in Figure 6.

RULE is used only to define transformations and assign them names. (As with trees, TGT informs the user when the name is not unique.) Transformations are normally assigned a position, an order of operation,* in a cycle or post-cycle. TGT readily allows the user to define and redefine the order of operation (see discussion of INSERT, Section D).

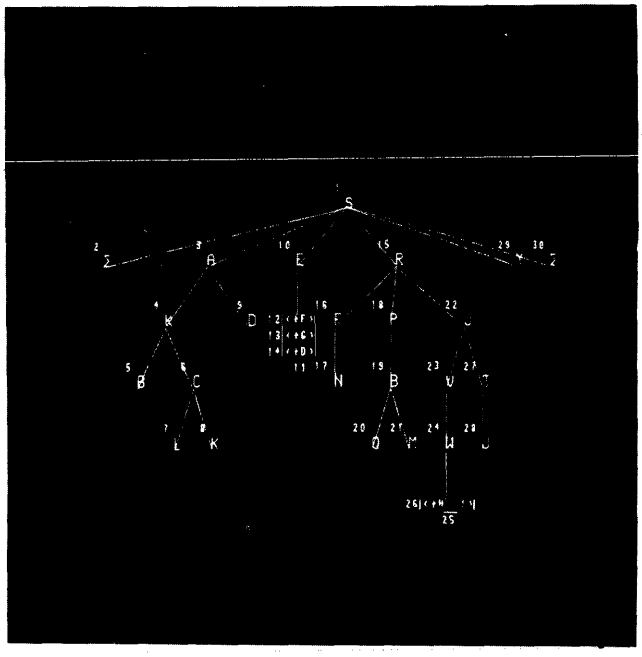


Figure 5 – Sample tree to which transformational rule H2 is applied

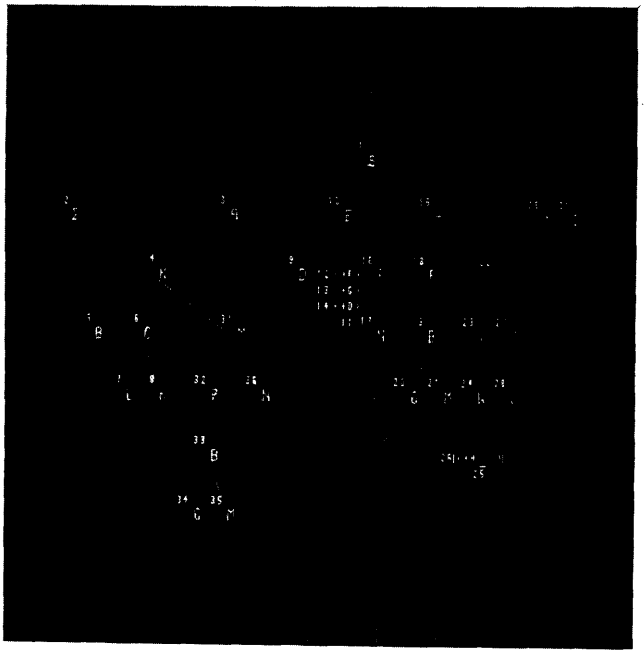


Figure 6 – Results of applying rule H2 to the tree in Figure 5

*Chomsky [1965][1] mentions the possibility of intrinsically ordered rules. Although we could have accommodated this concept in TGT by arbitrarily assigning numbers to rules and using a random number generator, we deferred this until someone attempts to write a grammar where the rules are not extrinsically ordered.

TEST, EXECUTE and DERIVE actually apply the tranformations to the current tree. In general, TEST is used to determine whether the structural description of a transformation successfully applies to the current tree.

The TEST command is accompanied by the name of a rule or the names of the first and last rule of a rule sequence, or the word ALL, in which last case every transformation in the cycle and post-cycle is matched against the current tree. The name of the rule and the message YES or NIL are output for each rule as it is tested. For each transformation that successfully applies, the user is given the option to execute the structural changes of the transformation and continue testing, to execute and wait for another command, to continue testing without changing the current tree, or to wait for a new command.

EXECUTE may specify one rule or the first and last of a sequence of rules. No message, other than the rule names and YES or NIL is output. This command is used in preference to TEST when the user knows that he wants the structural changes of each successful transformation to be performed.

Traditionally, transformations are applied to structures headed by the symbol S.* Where the "current" tree has more than one S, the user may wish to specify the number of the particular subtree to which he intends TEST or EXECUTE to apply.

DERIVE applies each transformation in the cycle to the "lowest" S in the tree. It recomputes "lowest" S after each cycle until all subtrees have been processed. It then applies all of the postcyclic rules. DERIVE records the S node number associated with each cycle and the transformations that were executed during the cycle.

It is not meaningful to talk of the execution times of transformations because they will vary with the tree, the rule, the conventions of interpretation, and, of course, the computer. However, to give some rough idea of the speed of TGT on the Q-32, we input rules 1-13 of IBM Core Grammar I (except rule 4, which has complex special conditions on its application) and applied them to test trees 28 and 29 of that grammar. The average execution time per rule was .04 seconds.

D. Other capacilities

The command MATCH enables the linguist to determine easily how a successful transformation applies to a tree, by printing on the TTY those symbols of the structural description that were successfully matched by a node in the tree and by printing that node number. After HYPOTHETI has applied to the tree in Figure 3, the command MATCH would output:*

5	X
7	A
8	B
18	C
21	G
24	B

(15)

Any successful transformation may be reapplied to determine whether it could have applied to the current tree in more than one way. The command AGAIN causes the search for a proper analysis to continue by successively backing up from the last node matched as if there had been no match. Thus, in (15) above, the rule interpreter would look below node 24 for another B; finding none it retreats to G 21, ignores the original match and eventually finds the analysis below:**

5	X
7	A
8	B
18	C
23	G
24	B

(16)

Applied once more, AGAIN would eventually look below 18 for a C, a D followed by an E, or an F, and it would find the analysis indicated by (17) below:

5	X
7	A
8	B
33	F
20	X
21	G
24	B

(17)

The analysis that includes 33F and 23G can be attained by invoking a special condition, condition 6. This suspends the A over A convention* allowing the interpreter to go below a symbol, A, to find another A. This condition is symbol specific in TGT and thus may be selectively applied among the symbols of a structure description.

Where a successful match is made on an optional symbol immediately preceded or immediately followed by an X, there is always at least one alternate analysis, that where the NIL is chosen.

*Suggestions have been made recently, however, (e.g., UCLA Conference on English Syntax, summer 1967) that each recursive symbol in the phrase structure rules be so treated.

*Variables are not printed when they encompass no nodes.

**The reference to the A over A convention under (17) is pertinent to this analysis.

*A discussion of this convention and exceptions can be found in Chomsky [1962][12] and Chomsky [1964][13]

For each transformation, the command INSERT allows the user to specify information concerning its operation in the rule set: its order of operation; whether it is cyclic or postcyclic; whether it is obligatory or optional; whether (if successful) it may be reapplied to the tree before the next sequential transformation; and whether the interpreter may look below an S in trying to find a proper analysis.

The command EDIT facilitates the correction of faulty transformation input by removing the necessity to retype the entire rule. This command is accompanied by two strings of symbols, X and Y. EDIT searches the erroneous rule for string X and replaces it with string Y.

A printout of the names of all the trees in the system at any time can be obtained with the command TREES. Similarly, all rule names are printed in response to the command RULES. The primitive DISPLAY and LIST also accept rule names as input and will either print on the TTY or display on the CRT scope the appropriate transformation.

Trees and rules are erased from the system's memory by means of the DELETE command accompanied by the rule or tree name.

Upon termination of a run, TGT presents the user with the opportunity of saving on magnetic tape the trees and rules he may have created during the run. Each initiation of TGT allows the user to input a file of trees and rules from tape, or to start up with neither rules nor trees in the system. Thus, many versions of a grammar may be extant and may be undergoing modification and testing simultaneously. Files of trees, rules, and derivations may thus be accumulated from run to run and may be subsequently used by others and merged with the files created by other linguists.

The program

The organization of the TGT program is straightforward, consisting of an executive routine and an indefinitely extensible set of service routines, which perform the basic system tasks such as tree creation and manipulation, rule testing, and generation of displays and teletype output. In TGT's basic interactive cycle, the executive routine interprets each teletype request to determine which service routine is needed, reads and legality checks the parameters associated with the request, and transfers control to the approriate service routine. At the completion of its task, the service routine returns control to the executive routine and the cycle is ready to begin again.

Of more particular interest are the functions associated with the interpretation of transformations. When the transformation is input each symbol name is looked up in the dictionary and is replaced by its dictionary number. If the symbol is not present it is added to the dictionary.

The rule is then converted to a form convenient for interpretation and resides in a table whose capacity is, at present, 1100 entries. Each symbol in the transformation occupies an entry.

Each entry contains the following information: dictionary entry number of this symbol; location in this table of the next sequential symbol; location of previous sequential symbol; location of next optional symbol (in the case of a symbol within choice parentheses); location of the first special condition on this symbol.

X *1A ([C D] * 2Z, F G) T => 1 = 0. Z NQ 1. (18)

The transformation (18) above would be represented in the rule table as indicated below.

Number	Dictionary Entry of Symbol*	Next Sequential	Previous Sequential	Optional Next	Special Condition Pointer/ Type
1	X	2	0	0	0
2	A	3	1	0	0
3	Z	10	2	4	6
4	F	5	2	0	0
5	G	10	4	0	0
6		7		2	4
7		8			1
8	C	9	0	0	0
9	D	0	8	0	0
10	T	0	3	0	0

Entry 6 above represents the special conditon imposed on the symbol Z. The number 4 indicates which condition is imposed (nonequality), and the number 2 under column "Optional Next" of entry 6 is a parameter pointing to the structure headed by the symbol A in entry 2.

We handle recursion as a special condition. Thus the symbol Z has two conditions 4 and 1. Condition 1 is always the last condition to operate since it essentially effects a re-entry of the procedure, at which point a proper analysis consisting of the symbols C and D is attempted for the substructure headed by this Z.

The numbers of the tree nodes corresponding to rule symbols are saved in this table so that the tree can be changed in case the pattern match is successful, and so that the pattern matching algorithm can try all possibilities.

*For convenience the actual symbol is used here.

If, for example, the node to the right of A in the tree does not dominate a C and D and is not the symbol F, the pattern matching algorithm will back up to the tree node recorded in entry 2 as matching rule symbol A. The variable X is extended to encompass this A and the algorithm attempts to find another A.

A flow diagram and more details of the program can be found elsewhere.[10,11]

DISCUSSION

We feel that TGT is a tool that is flexible in its capacity to perform operations and is convenient with regard to the manner in which commands may be expressed. We found that within the present framework we could often handle conditions attached to the *structure change* portion of a transformation, although we had not intentionally designed the system to do this. Other difficulties have been resolved by breaking complex rules into several simpler rules. We anticipate that many future problems will be solved by expanding the existing set of conditions, which can be expressed directly in the rule. Some other linguistic capabilities that the user may need are less tractable.

The advisability of using an entirely different rule schema for handling conjunction has been demonstrated in Schane [1966][14] and Schachter [1967].[15] Plans to program the Schachter schema and integrate it into the current model of TGT are under way.

Several different conventions regarding the movement of tree branches by structure changes have been provided internally, but have not yet been made easily accessible to the user. More programming would be necessary, however, were a set of tree-pruning conventions, such as suggested in Ross [1965],[16] to be adopted.

A computer system such as TGT can make significant contributions in testing a lexicon and integrating it into a transformational grammar. We are at present familiarizing ourselves with various proposals regarding the form of the lexicon and conventions for lexical insertion.

The authors have no illusions regarding the comprehensiveness or generality of TGT. Transformational grammar is in a dynamic state of development. By the time a system of programs is written and checked out, it is in danger of being obsolete. However, a linguist who sets out to write a grammar faces the same problems. Once he has written a number of rules, he may not readily change rule conventions unless he is willing to play the Red Queen, who must run as fast as she can just to keep from falling behind.

ACKNOWLEDGMENTS

We are indebted to Dr. Barbara Hall Partee of AFESP for her criticisms and suggestions regarding the design and use of TGT, to John Olney of System Development Corporation for his contributions to the initial specifications, and to Dr. Joyce Friedman of Stanford University for suggesting the utility of a single node variable.

REFERENCES

1 N CHOMSKY
Aspects of the theory of syntax
THE MIT Press Cambridge Massachusetts 1965

2 P S ROSENBAUM D LOCHAK
The IBM core grammar of English
In Specification and Utilization of Transformational Grammar Scientific Report no 1 IBM Corporation Yorktown Heights New York 1966

3 N CHOMSKY
The formal nature of language
Appendix A
In E H Lenneberg Biological Foundations of Language John Wiley & Sons New York 1967

4 D BOBROW
Syntactic theories in computer implementations
In H Borko Automated Language Processing: The State of the Art John Wiley & Sons New York

5 BUNKER RAMO CORPORATION
A syntactic analyzer study
Final Report under Contract AF 30(602)-3506 Canoga Park California 1965

6 S R PETRICK
A recognition procedure for transformational grammars
Unpublished Doctoral Dissertation MIT Cambridge Massachusetts 1965

7 A M ZWICKY J FRIEDMAN B C HALL D E WALKER
The MITRE syntactic analysis procedure for transformational grammars
MTP-9 The MITRE Corporation Bedford Massachusetts August 1965

8 L N GROSS
On-line programming system user's manual
MTP-59 The MITRE Corporation Bedford Massachusetts March 1967

9 F BLAIR
Programming of the grammar tester in specification and utilization of a transformational grammar
Scientific Report no 1 IBM Corporation Yorktown Heights New York 1966

10 W J SCHOENE
TGT user's manual
System Development Corporation document in press

11 D L LONDE W J SCHOENE
An algorithm for pattern matching and manipulation with strings and trees
System Development Corporation document in press

12 N CHOMSKY
The logical basis of linguistic theory
Ninth International Congress of Linguistics
Cambridge Massachusetts 1962

13 N CHOMSKY
Current issues in linguistic theory
In Fodor and Katz The Structure of Language Prentice-Hall Englewood Cliffs New York 1964

14 S A SCHANE
A schema for sentence coordination
MTP-10 The MITRE Corporation Bedford Massachusetts April 1966

15 P SCHACHTER
A schema for derived constituent conjunction
Unpublished Working Paper UCLA Air Force English Syntax Conference September 1967

16 J R ROSS
A proposed rule of tree pruning
Unpublished paper presented to the Linguistic Society of American December 1965

DATAPLUS–A language for real time information retrieval from hierarchical data bases

by NORMAN R. SINOWITZ
Bell Telephone Laboratories
Holmdel, New Jersey

INTRODUCTION

To the average person, a computer user is synonymous to a programmer. In fact, to the average programmer, a user is synonymous to a programmer. Information retrieval is an area in which the computer user is not–or, at least, should not be–required to be a programmer. This paper describes a real time information retrieval system which is at once highly accessible, relatively inexpensive, and simple enough to be used by nonprogrammers.

The system was implemented early in 1967 on a GE-265 computer operating under the time-sharing monitor developed by the Missile and Space Division of the General Electric Company (Valley Forge, Pennsylvania). Access by users is thus from remote teletypewriters communicating over standard telephone lines.

By implementing a system on a commercially available time-shared computer service, the effort of developing the time-sharing software is eliminated, and the cost of the development is spread over the many users of the service. The retrieval system was field tested without any investment in expensive hardware. By providing the system with a powerful information processing ability, highly efficient use can be made of the potential information content of the data, with a consequent high performance/cost ratio of the system.

What makes the system easy to use is its query language. Called DATAPLUS, the name derives from the system's ability to access data, plus the ability to process data. The DATAPLUS "compiler," which can be more accurately described as an incremental interpreter, was implemented in FORTRAN, and was designed for real time information retrieval from a hierarchical data base.

At his remote terminal, the DATAPLUS user types his information request in the form of a message consisting of a continuous flow of statements resembling English. Erroneous or unintended statements can be altered by merely retyping them. Providing the user with these facilities has in most instances overcome the inertia people have against learning to "program." Furthermore, these facilities enable the user to spend more time thinking about *his* problem, because he is, to a large extent, thinking in the way he is accustomed.

Since a data item is often meaningful at a level of the hierarchical structure other than that at which it appears in the data base, flexibility is provided for addressing and manipulating the data at various levels. The language is also "open-ended" in the sense that the user is given the ability to create functions of items appearing in the data base, and to instruct the computer to operate on these functions in the same way that it operates on items already in the data base.

DATAPLUS was implemented in order to retrieve information useful to engineers at Bell Telephone Laboratories. So as not to burden the reader with telephone jargon, we shall exemplify our discussion by referring to a familiar (but hypothetical) data base. In addition to being intelligible to a wide audience, the hypothetical data base was chosen because it is structurally isomorphic to the data base for which the system was implemented.

The data base

Consider a department store chain which has branches in a number of the larger cities in the conterminous United States. The company operates throughout most of the country, with a corporate headquarters, and separate divisions in the various states. The state organizations are further broken down by cities. Figure 1 shows the hierarchical structure for this data base.

At the top is the corporate level. The second and third levels represent, respectively, the state and

city divisions. The focal level is the fourth level, that is, the STORE level. Each store has one or more departments, and each DEPARTMENT sells at least one ITEM.

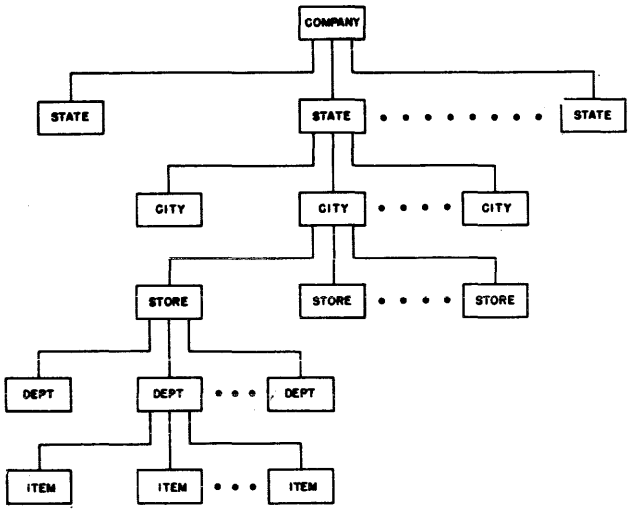


Figure 1 – Hierarchy in the department store data base

Data are collected and stored at the STORE, DEPARTMENT, and ITEM levels. Each datum – which we henceforth call *variable* – has a name and a value. A partial list of representative variable names and their levels is shown in Table 1.

TABLE I – Partial list of representative variable names and their levels

Variable Name	*Level*
STORES	Store
STORE NAME	Store
ADDRESS	Store
COUNTY	Store
EARNINGS	Store
DEPRECIATION	Store
DEPARTMENTS	Dept
DEPARTMENT NAME	Dept
SALES FORCE	Dept
DOLLAR SALES	Dept
ITEMS	Item
ITEM NAME	Item
INVENTORY NUMBER	Item
IN STOCK	Item
ON ORDER	Item
PURCHASE PRICE	Item
SELLING PRICE	Item

A variable whose value is intrinsically numeric will be called a *range* variable; a variable whose value is intrinsically alphameric will be called a nonrange variable. Thus EARNINGS and IN STOCK are range variables, while STORE NAME and INVENTORY NUMBER are nonrange variables.

General description of the syntax

Like any language, natural or artificial, DATAPLUS has syntax, semantics, and vocabulary. Syntactically and semantically, DATAPLUS resembles English. The basic vocabulary consists of a few key words, such as "total," "distribute," "and," "or," "when," and a set of nouns representing items appearing in the data base. This section gives an overall picture of the syntax. Vocabulary and semantics are considered only when necessary to the discussion.

We begin by citing three examples of messages written in the DATAPLUS language.

Example 1 – Suppose we wish to find the total number of stores in Michigan and New Jersey which have any HI FI departments. The message could be...

TOTAL STORES: IN MICHIGAN; NEW JERSEY: WHEN ANY DEPARTMENT NAME = HI FI: GO:

Example 2 – Suppose we wish to obtain a frequency distribution of stores versus dollar sales per store in Miami and Tampa. The message could be...

DISTRIBUTE STORES: BY DOLLAR SALES PER STORE: BETWEEN 0 AND 2000000 IN STEPS OF 20000: IN FLORIDA, MIAMI, TAMPA: GO:

Example 3 – Suppose we want to extract the store name, the names of the departments, and the earnings/sales ratio for stores in Cook County, Illinois. The message could be...

LET RATIO = EARNINGS/DOLLAR SALES PER STORE: EXTRACT STORE NAME, RATIO, DEPARTMENT NAME: FROM STORES: IN ILLINOIS: WHEN COUNTY = COOK: GO:

A message in DATAPLUS thus consists of a continuous flow of statements,* each beginning with a key word, such as "TOTAL," "WHEN," "IN," "LET," and ending with a colon. The final statement in the message must contain the single word "GO."

Every message must have an "IN statement" i.e., a statement beginning with "IN." This informs the program *in* which states and cities the user is interested.

*A statement in DATAPLUS is either a complete sentence, a prepositional phrase, or an adverbial clause.

There are three basic operations in DATAPLUS as implemented to date: TOTAL, DISTRIBUTE, and EXTRACT. Every message must contain a statement that begins with one of the basic operation words TOTAL, DISTRIBUTE, or EXTRACT.

Any range variable may be summed by using the TOTAL statement. DATAPLUS automatically supplies an average along with the total. Thus, "TOTAL EARNINGS:" will cause the system to furnish the average earnings/store.

If there is a DISTRIBUTE statement in the message, there must also be a BY statement and a BETWEEN statement. If we visualize a typical histogram, the DISTRIBUTE statement contains information pertaining to the Y axis variable; the BY statement contains information pertaining to the X axis variable; the BETWEEN statement contains information pertaining to the limits of the X axis and the class interval size.

A WHEN statement is optional in every message. Its purpose is to specify the search. In Example 2, the search is implicitly specified as being over every store in Miami and Tampa; in Example 3, the search is explicitly specified for stores in Cook County.

The LET statement can be used to define a created function of variables in the data base. A LET statement may appear anywhere in the message, provided that the variable that it creates has not been referred to earlier in the message.

Blanks are allowed as they are in English: A blank may not be placed in the middle of a word, but as many blanks as the user wishes may be placed between words. Any special symbol, such as

$$;+-*/,=:()$$

may be surrounded by as many blanks as desired.

The order of the statements—other than the LET and GO statements—is arbitrary. For example, the user may type the IN statement before the TOTAL statement, or vice versa.

Any statement may be overridden by merely retyping it. Thus, the user can change his mind about the content of any portion of the message, provided he has not typed GO.

If he has typed GO and there are errors in his message, appropriate diagnostic comments are printed. He need then only retype the erroneous statements in the message, followed by the GO statement.

Running the program

Once the program is loaded, it communicates its sole signal for user information requests by typing "?:=" after which the user starts typing. The user must be careful not to end a line in the middle of a word or number. Unlike English, DATAPLUS does not allow hyphenation. (However, we have allowed the typesetters to hyphenate DATAPLUS messages in the printing of this paper.)

If a message has syntactic or semantic errors, or if not enough user information is supplied, appropriate diagnostics are typed, and the system again returns with its perennial ?:= symbol.

Once a valid message is given, the system searches the data, types out the processed results, again types ?:= and the cycle begins anew. Thus, a request for a distribution may be followed with a request for extracting data—without reloading the program.

When the cycle begins anew, the program assumes that a new message will follow. This assumption can be suppressed by typing "EDIT": followed by those portions of the message that the user wishes to edit. Thus, after the computer had printed its results for the message in Example 3, the user could continue with...

EDIT: WHEN COUNTY=DU PAGE: GO:

and have the computer supply the same results for Du Page County.

The ability to shift from one operation to another combines with the EDIT feature to provide an extremely useful capability of DATAPLUS—the capability for browsing and "hunch pursuit." Upon examining the computer's reply to one message, the user is often stimulated to formulate another message. Experience with the system has demonstrated that a user frequently finds himself as part of this "feedback loop"—coming away from the system with information much more valuable to him than the information he originally intended to seek.

The vocabulary

The basic vocabulary has two portions—the key words, and the nouns. We have already been introduced to most of the key words. This section is concerned with nouns, which include the state and city names and the names of variables such as those given in Table I.

Although a variable appears on a given level in the data base, it can be used on a higher level in a message. An example of this "raising the level" of a variable is the noun DOLLAR SALES which appears as a DEPARTMENT level variable in Table 1, and was used as a STORE level variable in Example 2. In that example the variable of interest was the DOLLAR SALES PER STORE, viz, the dollar sales summed over all departments in the store.

In DATAPLUS, *the level of any range variable may be raised.* This is accomplished by following the variable name (as it appears in Table I) by the word PER and then by the word DEPARTMENT or STORE.

The variable DEPARTMENTS is a range variable which has the value unity for every department, and the variable ITEMS is a unity-valued range variable for every item in the data base. Since these are both range variables, their levels may be raised. Thus, to obtain a picture of the number of different types of items sold in stores in Dallas, one might say...
DISTRIBUTE STORES: BY ITEMS PER STORE: BETWEEN 0 and 700 IN STEPS OF 7: IN TEXAS, DALLAS: GO:

This will produce a distribution with stores as the Y axis variable, items per store on the X axis, with 100 class intervals each of size 7 items. In this example, the level was raised from item level to store level—i.e., two levels.

If the user tries to "lower the level" of a variable, he will get a diagnostic message. Clearly, the level cannot be lowered since this is a piece of information unobtainable from the data base. It is perfectly valid to follow the word PER by the name of the level on which the variable was collected. In fact, if the user is unsure about the level, he can always play safe by following the variable name with PER and then the level.

The vocabulary can be obtained from the system at run time. If the user types an invalid state name, the computer will type out a list of valid state names. If the user types a valid state name but an invalid city name, the computer will type a list of valid city names for that state. A listing of valid variable names may be obtained by putting the statement "CATALOG": anywhere in the message. Furthermore, the user can obtain definitions of any of the variables by using the DEFINE statement. For example, the statement

DEFINE IN STOCK, ON ORDER:

may be placed anywhere in the message, and the computer will furnish concise definitions of the variables IN STOCK and ON ORDER.

Creating variables

DATAPLUS has provision for extending its vocabulary, that is, for adding to its nouns. This is accomplished by the LET statement, which we mentioned briefly in Section III.

A created variable in DATAPLUS is *always a range variable.* In the previous section we saw how to create certain variables that are not in the data base. Those "raised level" variables (since they had to be summed) were range variables. In Part A of the present section we describe how to create algebraic functions of variables. Since we are performing algebraic operations, the operands, as well as the function, must be range variables. Part B of the present section shows how to create special variables, called *qualified variables.*

A. Algebraic functions

An algebraic function is formed by algebraic manipulations on range variables appearing in the data base, on raised level variables, or on variables created in earlier LET statements.

Some examples are...

(a) LET PERCENT = (SELLING PRICE-PURCHASE PRICE)/SELLING PRICE:
(b) LET RATIO = EARNINGS/DOLLAR SALES PER STORE:
(c) LET CASHFLOW = EARNINGS + DEPRECIATION:

The algebraic operators are +, −, *, /. Exponentiation is not allowed. The hierarchy of operations is the same as in FORTRAN. Redundant parentheses are ignored. (Unary operations are not allowed; thus it is not valid to say LET X = − EARNINGS:)

The level of all variables appearing on the right hand side of the equal sign must be the same. (The construction would be semantic nonsense otherwise.) The created variable is assigned this level. Once a variable is defined in a LET statement, it may be used in any subsequent statement where a range variable of that level is permitted, including another LET statement.

Suppose we wish to obtain a distribution, for stores throughout the United States, of cash flow versus earnings. The message could be...
LET CASHFLOW = EARNINGS + DEPRECIATION: DISTRIBUTE CASHFLOW: BY EARNINGS: BETWEEN 0 AND 500000 IN STEPS OF 50000: IN COMPANY: GO:

B. Qualified variables

Qualification is accomplished in DATAPLUS by enclosure in parentheses. (This construction is a familiar one in English.) Some examples are:

(a) LET X = IN STOCK(INVENTORY NUMBER = A0578):

(b) LET Y = DOLLAR SALES (DEPARTMENT NAME = HABERDASHERY):

(c) LET Z = ON ORDER (INVENTORY NUMBER = B6724):
LET T = Z PER STORE:

The variable X is an item level variable. Its value is the number of units in stock that have an inventory number of A0578. The variable Y is a department level variable. Its value is the dollar sales for the department if the department is a haberdashery department and zero otherwise. The variable Z is an item level variable. The variable T is a store level variable, and represents the number of units on order in the store which have inventory number B6724.

In general, the value of the created variable is the value of the qualified variable if the equality enclosed in parentheses is true, and zero otherwise. The level of the created variable is assigned the level of the qualified variable. Once a variable is created by qualification, it may be used in any subsequent statement where a variable of that level is permitted, including another LET statement.

Suppose we wish to determine, for Chicago, all stores which have more than 30 TV sets in stock. The message could be . . .

LET TVSUPPLY = IN STOCK(ITEM NAME = TV SET): EXTRACT STORE NAME, ADDRESS: FROM STORES: IN ILLINOIS, CHICAGO: WHEN TVSUPPLY PER STORE=30 to 9999: GO:

Boolean operations

Most information retrieval systems allow for Boolean operations on index terms. DATAPLUS allows for Boolean conjunction and (inclusive) disjunction by use of the words "AND" and "OR" in the WHEN statement.

For example, suppose we wish to find the total number of furniture departments in stores in Denver that have a sales force of at most 14 people and dollar sales of more than $100,000. The message

TOTAL DEPARTMENTS: IN COLORADO, DENVER: WHEN DEPARTMENT NAME= FURNITURE, AND SALES FORCE=1 TO 14, AND DOLLAR SALES=100000 TO 9999999: GO:

If we are interested in finding the number of departments whose name is either "furniture" or "childrens furniture," the message could be . . .

TOTAL DEPARTMENTS: IN COLORADO, DENVER: WHEN DEPARTMENT NAME = FURNITURE, OR DEPARTMENT NAME = CHILDRENS FURNITURE: GO:

Intersection and union may be used in the same WHEN statement, with the AND taking precedence over the OR. Thus the message. . .

TOTAL DEPARTMENTS: IN COLORADO, DENVER: WHEN DEPARTMENT NAME = FURNITURE AND SALES FORCE = 1 TO 14, OR DEPARTMENT NAME = CHILDRENS FURNITURE, AND SALES FORCE = 1 TO 14: GO:

will total those departments whose name is furniture and which have a sales force of at most 14 people, or whose name is childrens furniture and which have a sales force of at most 14 people.

It is also possible to use the qualified variable as a means for specifying conjunction. Thus, the messages . . .

TOTAL STORES: IN TENNESSEE: WHEN COUNTY=KNOX, AND EARNINGS=0 TO 70000: GO:

and

LET X=STORES (COUNTY=KNOX): TOTAL X: IN TENNESSEE: WHEN EARNINGS=0 TO 70000: GO:

are equivalent. Both will determine the number of stores in Knox county, Tennessee, which earn less than $70,000.

Two operations that prove extremely useful in retrieving from hierarchical data bases are the ANY and ALL operations. These can be used for variables mentioned in the WHEN statement which are on a lower level than the level specified by the TOTAL, DISTRIBUTE, or EXTRACT functions. (The level specified in the EXTRACT function is given in the FROM statement.) We have already seen a use of the ANY operation in the example . . .

TOTAL STORES: IN MICHIGAN; NEW JERSEY: WHEN ANY DEPARTMENT NAME= HI FI: GO:

where the variable DEPARTMENT NAME is on a lower level than the variable STORES specified by the TOTAL statement. As an example of the ALL operation, suppose we wish to find those departments in New Jersey, all of whose items sell for $8.00 or more. The message could be . . .

EXTRACT STORE NAME, ADDRESS, DEPARTMENT NAME: FROM DEPARTMENTS: IN NEW JERSEY: WHEN ALL SELLING PRICE PER ITEM=8.00 TO 999999: GO:

In this case, the variable SELLING PRICE is on a lower level than the variable DEPARTMENTS.

Although DATAPLUS does not presently support the Boolean negation operation—due primarily to core size limitations—it is often possible to perform negations by appropriate use of qualified variables. Suppose we wish to find the number of stores in Illinois that are *not* in Cook County, we could say...

LET X = STORES (COUNTY = COOK): LET Y = STORES (X = 0): TOTAL Y: IN ILLINOIS: GO:

The variable X will have the value 1 if the store is in Cook County, and 0 otherwise. The variable Y will have the value 1 if and only if X=0, that is, if the store is not in Cook County.

Application to other data bases

Although DATAPLUS was designed specifically to process data files useful to telephone engineers, the language is capable of wider application. The data file described in this paper can use a simple cognate of the language. Specifically, this means that only the nouns need to be changed and this can be accomplished by a trivial modification of a few program tables.

Furthermore, the hierarchical structure of the department store chain is representative of a tree structure common to many data bases. Suitably extended, the techniques employed in DATAPLUS can be applied to other tree-structured data bases, with the complexity of the extension depending on the number of levels in the hierarchy.

Although this paper has been primarily concerned with the DATAPLUS language, it should be clearly understood that the language is the "front end" of an information processing system, the "back end" of which performs such actions as fetching the data from disk to core, calculating distributions, and printing results. Therefore, if one were to use DATAPLUS as the interrogation language for an information system built upon another data file, it would be profitable to borrow as much of the back end of the present system as possible. A large measure of borrowing can in fact be accomplished because of the system's modular design.

There is one subroutine (the Read routine) whose sole function is to fetch data from disk. The other programs are independent of the actual physical layout of data on the disk. Hence, application to other data bases could be performed by appropriately changing the Read routine with minor revisions of the other subroutines.

The fact that the data layout is not an inherent part of the total system suggests three immediate advantages: The data in any application can be structured on disk to take advantage of any idiosyncrasies of the particular data base. The data can be structured to take advantage of the particular computer system's executive program and scheduling policy. The relative efficiencies of different layouts can be experimentally tested by inserting different Read routines.

CONCLUSION

In the literature on query languages comparatively little consideration has been given to the manipulation of data at the various levels of the multilevel file.[1-8] The necessity of such hierarchical operations as the ANY, ALL, and PER operations in DATAPLUS has been recognized in a recent paper[9] describing a proposed general purpose data management system. Such operations, combined with the DATAPLUS capabilities of totalling, distributing, and creating algebraic functions, make it possible to more fully utilize the information potential of the data file, and consequently increase the performance/cost ratio of the information system.

DATAPLUS was implemented on the GE-265 computer operating under the GE time-sharing monitor, which allows only 5000 words of user core available for compiled FORTRAN code. (A number of program overlays were obviously required.) The system described in this paper has thus demonstrated the feasibility of implementing—on a small machine—a highly accessible, relatively inexpensive, and easy to use information retrieval system with substantial processing capability.

REFERENCES

1 C W BACHMAN S B WILLIAMS
The integrated data store—A general purpose programming system for random access memories
AFIPS Conf Proc 1964 pp 411-422

2 *Introduction to integrated data store*
General Electric Computer Department CPB 1048 April 1965

3 J H BRYANT P SEMPLE JR
GIS and file management
Proc of ACM September 1966 pp 97-107

4 *Generalized information system*
IBM Manual E20-0179 Reference No A 01 8773

5 W D CLIMENSON
Annual review of information science and technology
John Wiley & Sons
New York 1966 Volume I Chapter 5
File organization and search techniques

6 J MINKER J SABLE
Annual review of information science and technology
John Wiley & Sons
New York 1967 Volume II Chapter 5
File organization, maintenance and search of machine files

7 C T MEADOW
The analysis of information systems
John Wiley & Sons
New York 1967 Chapter 2
The languages of information retrieval
8 N S PRYWES
Man–computer problem solving with multilist
Proc of IEEE Volume 54 No 12 December 1966
9 R E BLEIER
Treating hierarchical data structures in the SDC time-shared data management system (TDMS)
Proc of ACM August 1967 pp 41-49

A language design for concurrent processes

by L. G. TESLER
Computer Consultant
Palo Alto, California
and
H. J. ENEA
Stanford University
Stanford, California

INTRODUCTION

In conventional programming languages, the sequence of execution is specified by rules such as:

(1) The statement "GO TO L" is followed by the statement labelled "L" (Branching rule).
(2) The last statement in the range of an iteration is followed, under certain conditions, by the first statement in the range (Looping rule).
(3) The last statement of a subroutine is followed by the statement immediately after its CALL (Out-of-line code rule).
... (Other rules)
(n) In other cases, each statement is followed by the statement immediately after it (Order rule).

This paper will define a class of general-purpose languages which does not need these rules. The power of these languages is equivalent to that of Algol, PL-1, or LISP. Other languages which do not need these rules appear in the literature.[1,2,3]

Concurrent processing

The advent of multi-processing systems makes it possible for a computer to execute more than one instruction at a time from the same program without resorting to complicated look-ahead logic. There are many ways in which this capability can be utilized; one way is to find several statements that could be executed simultaneously without conflict, and to delegate their execution to different available processors. This technique is sometimes called "concurrent processing". To employ it there must be a means for determining, during compilation, which statements can be processed concurrently.

Many proposals for programming languages have suggested that more rules like 1, 2, 3, ..., n should be added for explicit indication of concurrence.[4,5,6,7] Examples of such rules are:

(n + 1) The statement "FORK M, N, ..." is followed simultaneously by the several statements labeled "M", "N", ...; the statement "JOIN M, N, ..." terminates the fork.
(n + 2) The range of statements following "DO SIMULTANEOUSLY" can be executed for all values of the iterated variable at once.

The programmer using these facilities must take care that the statements performed simultaneously do not conflict, e.g., do not assign different values to the same variables.[8]

Other proposals have suggested an analysis of potential conflicts, during compilation, to isolate all concurrence that does not depend on run-time data values ("program concurrence").[8,9,10,11,12] This approach is adopted here because its automatic elimination of all possible conflicts guarantees determinacy.

Once it is possible to isolate all program *concurrence* using implicit information, it is tempting to examine the possibility of determining all program *sequence* (i.e., non-concurrence) using implicit information. If that were possible, then not only rules n + 1 and n + 2 but also rules 1, 2, 3, ..., n could be eliminated. In the following section, the class of "single assignment" languages will be defined such that all program sequence and program concurrence are determinable during compilation without explicit indication.

Single assignment languages

A program written in a single assignment language has the following essential characteristic:

No variable is assigned values by more than one statement.

The only effect of executing any statement is to as-

sign values to certain variables named in that statement (no side effects).

Every statement is an assignment statement. The variables names in each statement belong to two exclusive groups:

(1) *Outputs:* Those which are assigned values by the statement.

(2) *Inputs:* Those whose values are used by the statement.

The output variables are said to be *dependent* on the input variables. The abbreviation "AdB" stands for "A is dependent on B". Dependence is:

(a) Transitive: $AdB \wedge BdC \rightarrow AdC$.

(b) Antisymmetric: $AdB \rightarrow \neg BdA$.

(c) Irreflexive: $\neg AdA$.

Circular dependence is not allowed; for example, A can't be dependent on B if BdC and CdA. If neither AdB nor BdA, then A and B are *independent.*

All required program sequence can be determined during compilation by a straighforward tracing of dependence. As the symbol table is built, two-way pointers are constructed between the input and the output variables of each statement. The final symbol table, which is both a data-flow and a program-flow diagram, elucidates the required sequence in the program. Statements that are not found to require sequential execution can be performed concurrently. The rules $1, \ldots, n+2$ are replaced by the single rule:

> *The statement that outputs variable A must be executed before every statement that either inputs A or inputs some B such that BdA.*

The order of appearance of statements is immaterial to this analysis; consequently, an incremental compiler can be employed which accepts statements typed in any order. This property is especially useful when adding a statement to a previously written program because neither unforeseen side effects nor mislocation of the statement can occur.

Optimization

One way of optimizing a program is to reduce the amount of redundant computation by combining "common expressions". In a single assignment program there are no side effects; therefore, common expressions throughout the program can be combined during compilation. Another way of optimizing a program is to allocate storage efficiently. For example, in the program:

```
var: x+y;
a: var-w;
b: 2×var;
c: x-y;
d: -var;.
```

storage for the variable "var" need not be reserved until just before any of the statements "a", "b", or "d" demand the value of "var", and may be released as soon as all of "a", "b", and "d" no longer require the value of "var". These requirements can be detected easily during compilation. As a result, storage is never allocated for a variable except when necessary to guarantee the availability of its value for further computation.

Compel

Compel (*Com*pute *P*aral*l*el), a single assignment language, will be partially defined so that examples of single assignment programs can be given. This language has not been implemented.

Compel programs process three types of quantities: numbers, lists, and functions. A *number* is a floating-point approximation to a complex number, e.g.,

$$2+i4.6$$
$$-17.3$$
$$5$$

A *list* is an ordered set of zero or more quantities, e.g.,

$$[1,5,9] = \begin{cases} [\text{factorial}, [6.2,7], [\,]\,] \\ [\textbf{1 by 4 to 9}\ ] \\ [\textbf{1 by 4 for 3}] \\ [1, \textbf{while preceding} < 9\ \textbf{use preceding} + 4] \\ [\textbf{while index} < 3\ \textbf{use}\ 4 \times \textbf{index} - 3] \end{cases}$$

A *function* has one argument which may be of any type and is frequently a list ("parameter list"), e.g.,

$$\varphi x\,(x \uparrow 2)$$
$$\varphi[a,b]\ (a \times x + b \times y)$$

Blocks may be created for local naming—they have nothing to do with storage allocation or with program sequence. The statements within a block may appear in any order, and the blocks within a program may nest and appear in any order. Each block begins with the line:

input v1, v2, . . . , vn;

where $v1, \ldots, vn$ $(n \geq 0)$ are the names of those variables defined outside the block and used inside the block. Similarly, each block ends with the line:

output w1, w2, . . . , wm;

where $w1, \ldots, wm$ $(m \geq 1)$ are the names of those variables defined inside the block and used outside. Every *statement* is of the form:

VARIABLES:EXPRESSION;

For example, the statement:

a:[1,2,4];

assigns to "a" the single quantity "[1,2,4]".

The statement:

a:**each** [1,2,4]

is analogous to the Algol statement:

for a: = 1,2,4 **do**. . .

where the range of the **do** includes all statements which are dependent on "a". On a parallel computer, all values of "a" can be produced concurrently; thus, a variable (in this case "a") can have several values at the same time. The function **each** *splits* a list of values so that its elements can be operated on individually.

After a list is split and its elements have been operated on individually the results of the operations are *collected* back into a list. For example:

a:**each** [1,2,4];
b:a ↑ 2;
c:*list* **of** b;

Statement "a" splits the list "[1,2,4]" into three quantities; statement "b" squares each quantity; statement "c" collects the results into a single list, i.e., "[1,4,16]". Splitting and collecting are analogous to forking and joining, except that splitting operates on the data flow, but forking operates on the program flow.

Every statement that does not assign values to the output variables of its block can be eliminated from that block by systematic substitution. For example, the three statements in the last example can be reduced to:

c:*list* **of** (**each**[1,2,4]) ↑ 2;

This property is the converse of common expression combination.

The two statements:

a:**each** [1,2,4];
b:a ↑ 2+a ↑ 3;

where "a" is used in no statement but "b", can be reduced to one statement in another way by use of the following construction:

b:a ↑ 2 + a ↑ 3 **with** a **each** [1,2,4];

Its advantage over the two statements it replaces is that, by omission of the colon after "a", "a" becomes a name local to the statement. A local variable is distinct from all other variables of the same name throughout the program.

In the examples that follow, subscripting is specified by a downward arrow, e.g., $a_{i,j}$ is written:

a ↓ [i,j]

Examples

Two simple Compel examples are given below. For each are given:

(a) a program/data flow diagram displaying concurrence and storage release;
(b) an Algol program using *fork, join,* and *do simultaneously;*
(c) a Compel program.

Figure 1 illustrates concurrent execution of statement and optimization of storage.

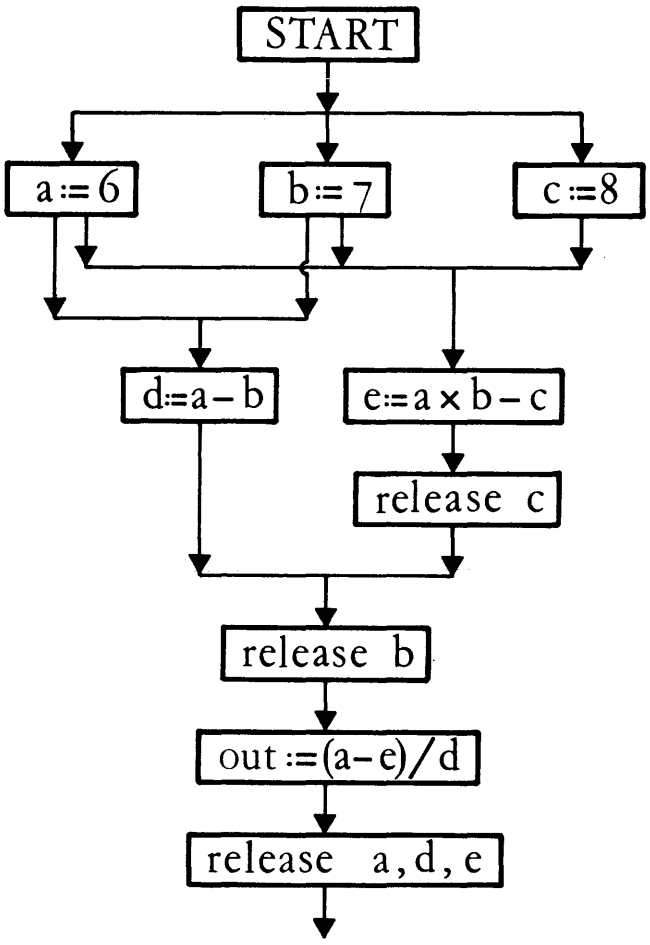


Figure 1 – Concurrent execution of statements and optimization of storage

Algol (extended):

```
begin
real a,b,c,d,e;
comment storage for these variables is reserved
   immediately upon block entry, and is not
   released until block exit;

fork r1, r2, r3 ;
r1:    a:=6 ; fork r4, r5 ;
r2:    b:=7 ; fork r4, r5;
r3:    c:=8 ; go to r5 ;
r4:    join r1, r2 ; d :=a-b; go to r6;
r5:    join r1, r2, r3 ; e:=a×b-c;
r6:    join r4, r5; out:= (a-e)/d;
end
```

```
Compel:
  input;
  out:(a-e)/d;
  a:6;
  e:a×b-c;
  d:a-b;
  b:7;
  c:8;
  output out;
```

Figure 2 demonstrates simultaneous assignment of all the elements of a list.

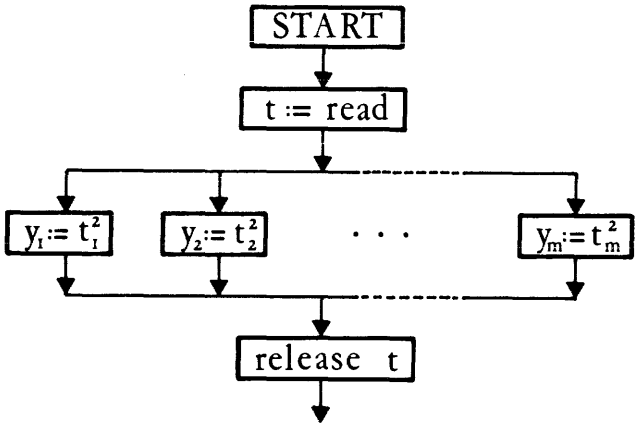


Figure 2 – Simultaneous assignment of all the elements of a list

```
Algol (extended):
  begin
  array t[1:m];
  integer i;
  inarray (t);
  for i :=1 step 1 until m do simultaneously
                  y[i]:= t[i] ↑ 2;
  outarray (y);
  end
Compel:
  input t;
  y: list of (each t) ↑· 2;
  output y;
```

Programming methods

When programming in Compel, some traditional techniques cannot be employed and new methods must be substituted. Some of these methods will be discussed below.

To avoid circular dependence, the input variables of a statement must be different from the output variables; therefore, it is impossible to write the equivalent of Algol's:

```
i:=i+1;
```

However, the incrementation of "i" is never a major step in a program, but merely one small step in a larger process. In Compel, notation is provided to incorporate such a step into an algorithm.

Example 1: to compute the sum of the elements of list "m" one can write:

```
sum:+m;
```

where "+" is a function which returns the sum of the elements of its argument.

Example 2: in an Algol program, a variable may be assigned values in several statements, some of which increment the variable:

```
r:=r0;
for i:=1 step 1 until n do
  begin
  a [i] :=b×r;
  r:=r+2;
  end;
rl:=r;
```

In Compel each variable is assigned values by only one statement:

```
r:each [r0 by 2 for n];
a:list of (b× r);
rl:last (list of r);
```

Conditional statements are not available in Compel; therefore, conditional expressions must be employed to achieve the effect of the Algol statements:

```
y: = y0;
b: = b0;
t: = t0;
if a < 0 then
  begin
  b:=c + 1;
  y:=a;
  end
else
  begin
  t:=r-a;
  y:=v;
  end;
```

A corresponding Compel program:

```
b:  [b0, if a < 0 then c+1 else preceding] ↓ [2];
t:  [t0, if a > 0 then r-a else preceding] ↓ [2];
y:  [y0, if a < 0 then a else v] ↓ [2];
```

The word **preceding** in the generation of a list denotes the immediately preceding element in the list. If it is necessary to refer to several preceding elements, this can be achieved as in the following example, which generate the first 1000 Fibonacci numbers:

```
fibonacci:list of (pair ↓ [2]);
pair:each [[0,1]
  while index < 1000 use
  [preceding ↓ [2],
  preceding ↓[1] + preceding ↓ [2]]
  ];
```

Problems which are solved in Fortran or in Algol by repeatedly changing the values of various matrix elements might seem to be difficult to solve in Compel. Therefore, a matrix reduction by Gaussian elimination will be given as an example of an iterative algorithm. The matrix is stored as a list of rows, and the rows are each lists of elements.

The following block defines "gauss", a function of three parameters: "a", "b", and "eps", where "a" and "b" are matrices:

$$a = \begin{bmatrix} [a_{11} \ldots a_{1n}] \\ \cdots \quad \cdots \\ [a_{n1} \quad a_{nn}] \end{bmatrix} \qquad b = \begin{bmatrix} [a_{11} \ldots a_{1m}] \\ \cdots \quad \cdots \\ [b_{n1} \quad b_{nm}] \end{bmatrix}$$

$n \times n$ $\qquad$ $n \times m$

Gauss returns:

$$\begin{bmatrix} [1\ a'_{12} \ldots a'_{1n}\ b'_{11} \ldots b'_{1m}] \\ [0\ a'_{22} \ldots a'_{2n}\ b'_{21} \ldots b'_{2m}] \\ \cdots \quad \cdots \quad \cdots \quad \cdots \quad \cdots \\ [0\ 0 \ldots a'_{nn}\ b'_{n1} \ldots b'_{nm}] \end{bmatrix}$$

In other words "a" and "b" are placed side-by-side, and row operations are performed which reduce the "a"-part to an upper triangular matrix.

If "a" is singular, Gauss returns *undefined*. The criterion for singularity is that the pivot in some step of the Gaussian elimination becomes $0 \pm$ eps.

The following ten-statement block defines Gauss. Each iteration generates one row, "pivr" of the result matrix. These rows are collected, unless the matrix was singular:

```
input;
gauss: φ [a, b, eps]:
  if length (a) < 1 then [ ]
else if last (list of abs pivot)> eps then
  list of pivr
else undefined;
```

Before iteration begins, scaling is performed. First, "b" is put next to "a" to form the n by n+m matrix "ab". Every row of "ab" is divided by the largest element ("maxv") of its "a"-part:

```
ab: list of (each a) concatenate (each b) ;
scale: list of
    list of (each abrow)/maxv (arow)
  with abrow in ab and arow in a;
```

The program generates n iterations. In the first, "iter" is the n by n+m matrix "scale"; in the i'th, "iter" is collected into a list of rows from the $n - i + 1$ rows "reduce" generated by the (i - 1)st iteration. If singularity is discovered, iteration ceases.

```
iter: each [scale,
    while abs(pivot)> eps ∧ index <n use
      list of reduce];
i: each [1 to n];
n:length(a) ;
```

In the i'th iteration, column i is searched for one of its largest elements, "pivot", located at $iter_{pivi,i}$. We use "max(x)" which produces a list of positions in "x" where "maxv(x)" occurs.

```
pivi: max(i column iter) ↓ [1] ;
pivr: iter ↓ [pivi];
pivot: pivr ↓ [i] ;
```

Every "row" in the matrix, excluding "pivr", is reduced to "reduce" by subtracting from it that multiple of "pivr" which makes reduce = 0:

```
row: iter ↓ [each [1 to pivi-1, pivi+1 to
  n-i+1]];
reduce: list of (each row) - (row ↓ [1]/pivot)
                              x (each pivr) ;
output gauss ;
```

On a sufficiently parallel machine, the execution time for this function is proportional to n ; on a sequential machine, Gaussian elimination time is roughly proportional to n^3.

Programmer education

The language Compel has been taught to several members of Midpeninsula Free University (Menlo Park, California) as part of a course in "Background for Computer Programming". Practically the entire language was covered in two hours, and the members were able to solve simple problems like those in this paper. Among the reasons that it is easy to learn Compel are:

(1) There is no "i = i + 1" paradox to excuse.
(2) There is no need to explain "side effects".
(3) Substitution is completely unrestricted.
(4) The data types are few, simple, and non-overlapping.
(5) Each variable is defined exactly once.
(6) Input and output files, considered lists of numbers, are in the same format as internal quantities, and are accessed in the same manner as

global variables. Therefore, there are no read, write, or format statements to confuse learning.

(7) There is only one kind of statement.

The following concepts, new to a non-programmer, needed to be taught:

(1) Church's lambda notation ("φ" in Compel);
(2) input and output to statements and to blocks;
(3) the need to rewrite equations so that the unknowns are on the left side;
(4) split and collect.

In order to teach Fortran, it is necessary to discuss many machine-oriented restrictions, to introduce more new notations, and to concern the student with much detail which is of little relevance to the problems being solved.

SUMMARY

In single assignment languages, no variable is assigned values in more than one statement, and no statement has side effects.

Explicit indication of program sequence and concurrence is not required; these are determined implicitly by tracing dependence during compilation. Among the advantages of this class of languages are: no side effects; immaterial order of statements; common expression combination; compiler detection of all program concurrence; efficient compiler-determined release and management of storage.

The simple examples provide evidence that languages in this class are useful for general-purpose programming. A complete description of Compel and more practical examples are available from the authors.

ACKNOWLEDGMENTS

This research was supported (in part) by Grant MH 06645-07 from the National Institute of Mental Health. We are indebted to Kenneth Mark Colby for his encouragement during the course of this work, to James Lyon for drawing the figures, to William McKeeman for suggesting the Gaussian elimination algorithm, and to the referees for many valuable suggestions.

REFERENCES

1 J MCCARTHY P W ABRAHAMS D J EDWARDS T P HART M I LEVIN
Lisp 1.5 programmers manual
MIT Press Cambridge Massachusetts 1962

2 E D HOMER
An algorithm for selecting and sequencing statements as a basis for a problem-oriented programming system
Proceedings of the 21st National ACM Conference 1964

3 S SCHLESINGER L SASHKIN
POSE: A language for posing problems to a computer
CACM, vol 10 May 1967 pp 279-285

4 M E CONWAY
A multi-processor system design
Proceedings of the 1963 Fall Joint Computer Conference.

5 J A GOSDEN
Explicit parallel processing description and control in programs for multi- and uni-processor computers
Proceedings of the 1966 Fall Joint Computer Conference.

6 J P ANDERSON
Program structures for parallel processing
CACM Vol 8 Dec 1965 pp 786-788

7 N WIRTH
Letter, A note on program structures for parallel processing CACM, Vol 9 May 1966 p 320

8 J P ANDERSON
Better processing through better architecture
Datamation Vol 13 (August 1967) pp 37-41

9 A J BERNSTEIN
Analysis of programs for parallel processing
IEEE Transactions on Electronic Computers Vol EC-15 Oct 1966 pp 757-763

10 H W BINGHAM D A FISHER W L SEMON
Detection of implicit computational parallelism from input-output sets
Burroughs Corporation TR-66-4 23 pages December 1966

11 D A FISHER
Program analysis for multiprocessing
Burroughs Corporation TR-67-2 71 pages May 1967

12 H HELLERMAN
Parallel processing of algebraic expressions
IEEE Transactions on Electronic Computers Vol EC-15 Feb 1966 pp 82-91

Control of sequence and parallelism in modular programs

by LARRY L. CONSTANTINE
Information & Systems Institute, Inc.
Cambridge, Massachusetts

INTRODUCTION

A variety of schemes for the specification and operation of parallel computation have been described recently [1,2,3,4] (see also references in Gosden[3]). For the most part, the proposed systems are concerned with parallelism on a detailed or statement level. Furthermore, as argued by Wirth,[5] the proposals are not at a truly procedure- or problem-oriented language level but represent transliterations of machine or process-oriented operations. Notable exceptions to this are the **and** and **shared** statements of Wirth,[5] and the **parallel for** Gosden[3] (similarly the DO TOGETHER in Opler[4]).

The purpose of this paper is not to add merely another minor personal variation or extension to the now hackneyed *fork* and *join* statements. Both the *raison d'etre* and the form of the language extensions to follow are found in basic modular programming and program design concepts.[6,7] The proposed features are intended to provide a macro-level capability for the natural expression of *inter*-module sequence relationships requiring a minimum of added analysis by the user. Without a concern for simplicity of expression and minimal analysis in the use of parallel features, there is a considerable danger of escalating already spiralling programming costs.

Technical considerations

A number of issues are very central to the problem of parallel program control. One much confused in the literature is the question of distinguishing necessary, sufficient, and desirable features. The answers vary considerably depending on the language level at which the question is formulated. It appears that a small number of well chosen high level features are *sufficient* and *desirable* but that these probably must be supported by a larger number of *necessary* primitives for completeness on the low levels.

Moreover, it is not merely sufficient to provide facilities that permit or declare parallelism. Some vital technical constraints must be addressed. Among these are: the "critical section" problem, the queuing problem, and the determinacy or reproducibility problem. The critical section problem is that of assuring that simultaneously executed programs which share sections of code (or other system resources) do not simultaneously execute their "critical sections," that is the sections relating to the shared resources. This problem has at least one guaranteed solution.[8,9] The queuing problem is related to (and found to be tied up with) the critical section problem. Having solved the critical section problem as given above, it is desirable to assure that no program can be indefinitely delayed in attempting to make use of some system resource and that some reasonable service discipline is followed. The former is at least assured in the critical section solution of Knuth.[9]

The determinacy question is broader and may even involve, in certain forms, the two problems above. We want the output of a program to be a deterministic function of only its input and initial state, and not depend upon external factors such as the timing of other programs or subprocesses sharing the same facilities. Such a program will give the same output each time it is started from the same initial state and given identical input streams. Consequently, deterministic programs are also called asynchronously reproducible and output-functional. A program model which illuminates sufficient conditions for determinacy has been developed[10] and a detailed mechanism for guaranteeing reproducibility has been designed.[11] It is proven by Luconi[12] that the schemes are, in a certain sense, equivalent. Both of these schemes address the problem of parallel process description at the machine or machine language level and in detail are very complex.

Programming considerations

One problem with most of the schemes for specifying and controlling parallelism is that they are either inherently complex, impose considerable additional

constraints on the programmer, or require additional analysis from him. Some require additional analysis merely to reap *any* benefits of parallelism; others require it to avoid erroneous parallelism, that is, ambiguous or non-deterministic programs.

Unfortunately, programming is sufficiently complicated and error prone without the problems of parallel processing. We would like to extend the power of our languages and processors while not adding to this burden. Moreover, errors in asynchronously communicating programs are likely to be the most serious and difficult to discover and correct.[11] Therefore, in addition to guarantees of determinacy, we want a mechanism for specifying parallelism which requires a minimum of additional analysis, is simple and easy to use, logically transparent, and which, where it permits program-level errors, permits them in ways which are more amenable to discovery and correction.

Control, sequence, and hierarchy

It is very evident that, in terms of process description, sequence is largely artificial in programs. The separation of the flows in a program into control and data streams is, for the most part, an artifact of the machines we use and the attendant algorithmic view of programs. The real statements of our problems are only partially ordered systems, and the sequentialness which becomes the "execution stream" is either inherent in the problem description or must be rather arbitrarily introduced in coding. For some cases, the inherent sequencing can be analyzed automatically and an unordered problem description used to produce a sequential one. Some languages[13,14] have incorporated this concept.

The program graphs used in attacking the determinacy problem in Rodriguez[10] and a somewhat more elegant model of the same name in Martin and Estrin[15] provide a method for specifying sequence constraints, including parallelism, on the basis of the data streams. The constituents of the latter model are summarized in Figure 1; a full description is found in Martin and Estrin.[15]

From the standpoint of our problem, the important aspect of a program graph is that there is no separate "flow of control" or execution stream. A node (subprocess) becomes active when the required input or inputs are present. Now it may be that one item of data that passes through the system is a "unique baton of control," but this needs no separate or independent feature to handle it.

A program may be modeled by a program graph to many different degrees of detail. One degree is of particular interest, that is, where each node of

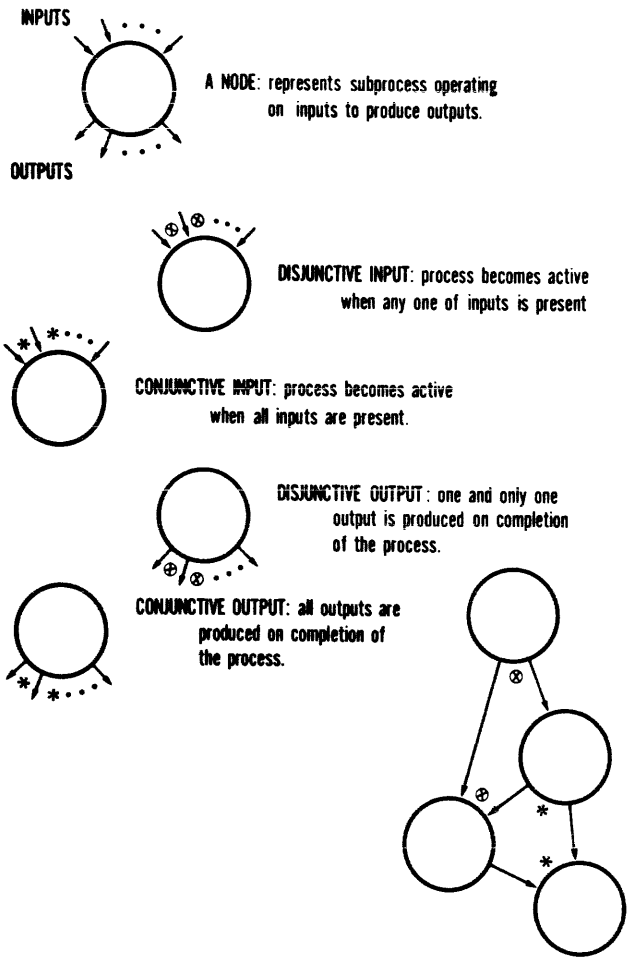


Figure 1 – The program graph, a model of computational processes, including parallel processes

a program graph corresponds to a whole program module (e.g., a subroutine). Parallelism at this macro-level is likely to be fairly easy for the programmer to deal with conceptually. Furthermore, the gains from parallelism relative to the cost of task switching, that is, the system overhead required to initiate and complete each task, are likely to be quite high. *Forks* and *joins* at the statement level are very likely to cost more in executive overhead than the savings possible from parallel operation.

Program hierarchy, that is, the control hierarchy of modules in a system, is an important issue in program design.[7] Essentially, this is the hierarchy determined by normal subroutine calls. As an example, consider the problem of generating lists of "key words" from sentences in some input text. The basis for choice will not be discussed here, but at least three fundamentally different hierarchies are possible, all performing the same overall process.

These are given in Figure 2. The hierarchy of control is, in this case, an artifact, like control sequence, and in concept could be eliminated. The co-routines of Conway[16] are addressed precisely to this matter. In co-routines, the input-output relationships in fact determine control, avoiding some messy programming details that arise solely from the artifact of control hierarchy.

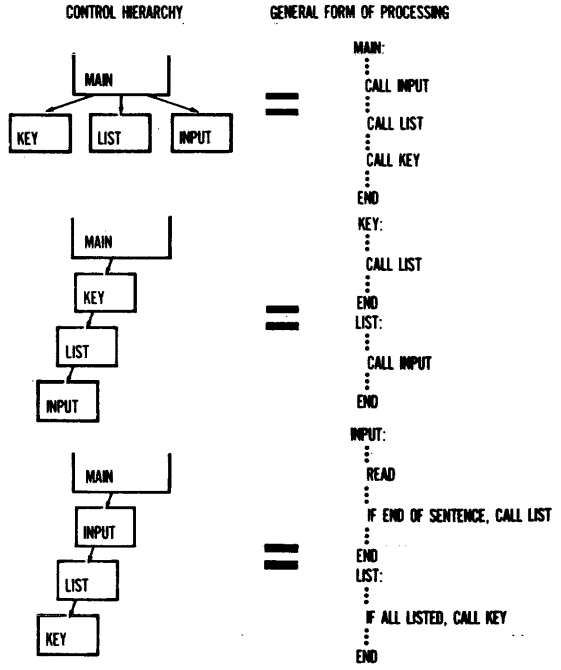


Figure 2—Alternate hierarchies for identical processing. KEY is a rourine to generate key words from a list of words, LIST generates a list of words from a sentence as a text string, and INPUT inputs a sentence, that is, the three modules contain the processing for the functions as given

The idea has been extended in the program strings of Morenoff and McClean.[17,18] The development of program strings was directly motivated by the desire to enable the simple, natural structuring of whole programs into larger units. A program string is illustrated in Figure 3. Each block in the program string is a generalization of the conjunctive node in a program graph. Some part of the block becomes active when some combination of inputs becomes available. Outputs are produced not all in one event, but at various times. Obviously, a block could be decomposed into an equivalent program graph, but to require this of the programmer in the design process is undesirable. It should be pointed out that the buffer files between blocks cause the determinacy problem, in an sense, to disappear. The buffer files completely isolate units in the system. Data, once outputted to a buffer, cannot be accessed or changed by the generating module. Thus the possibility of attempted simultaneous access to the same cell is eliminated. It is still possible to create indeterminate string structures if the behavior of a program is controlled by the sequence of appearance of data from different streams. However, a restriction to one active control stream per module in the string structure prevents this provided that an attempted access on an input stream either inhibits the control stream from continuing if the buffer is empty or results in transmission of the data and continuation.*

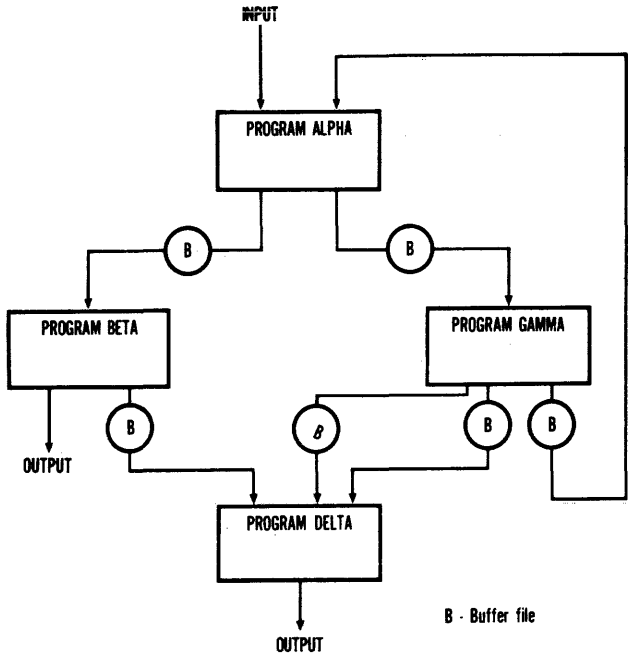


Figure 3—A program string stucture. Each output stream is buffer

It is inconvenient to restrict the control of sequence to those determined by simple input-output interrelationships alone. The fact is that some parts of processes are more naturally expressed in terms of statemental succession, others in a control hierarchy, and still others as co-routine or program string structures. What we now do is to present a set of language features which imbed the concept of program strings into the language and extend its applicability down-

*The proof of determinacy depends on the inability of a module in the structure to make a decision on the presence or non-presence of data on any input stream. With complete isolation and a single control stream, each module is in itself determinate except for possible effects due to its inputs. Consider the first input attempt by a module. Execution of the next sequential instruction guarantees that the data are available and have been accessed independent of the timing of other modules. The control sequence is thus independent of the order of appearance of inputs from different streams. All modules must then be output-functional with respect to inputs from the outside, hence the order of appearance of data on any purely internal stream is fixed by outside input and initial state. Since each internal input stream is identified with one and only one internal output stream, that is, no merges are permitted, the order of appearance of data on these streams cannot depend on timing. Therefore the structure is output-functional with respect to inputs from within the structure as well.

ward to the subroutine level. The structures so defined are also analogous to the computation graphs of Karp and Miller[19] by virtue of imposing a strict pairing of an output set with an input set through the use of the *explicit to - explicit from* pair.†

The data control mechanism

The "normal" subroutine call is the prototypical point of departure since it is the sole structuring and sequencing mechanism for modules in most languages. Consider the call in routine BETA to subroutine ALPHA with four parameters

ALPHA (A,B,X,Y)

It is impossible to tell whether the parameters are merely arguments being transmitted to the module ALPHA or whether some are specifying return locations for the output of ALPHA. Let us assume the general case where some, say X and Y, are output parameters. Then conceptually the effect of such a call is to pass the input parameters A and B to ALPHA, to hand control to ALPHA, which retains it until the output values for X and Y are developed, and then to return the output values and control to the calling module. Control must leave BETA and not return until the output has been completely generated because, in the subroutine to have been completed.

What we would like to do is separate the transmission of information *to* a module from the transmission *from* a module. We would also like to make the passing of control an optional matter which is governed solely by input-output constraints.

Four language constructs are required for this. BETA must be able to transmit data explicitly to ALPHA and to receive it explicitly from ALPHA. ALPHA must be able to accept data from any module calling it, for example, implicitly from BETA, and return data implicitly to BETA as its caller. This separation of the input linkage function from the output linkage for subroutine calls is somewhat inimical to current languages. Some violence to syntax is thus required.

The four proposed statements are shown in Table I. A "normal" call, that is, one which imposes full subordination, is the simple combination of the *explicit to* and the *explicit from*. The *explicit from* is useable in the evaluation stream (as an expression) since the value of a procedure is an output. The execution of an *explicit to* statement assumes only that the parameters are transmitted to the called module, and following the *explicit from* statement it can be assumed that the corresponding parameters have been set by the called module. Control, either the same stream as the calling module or from an independent (parallel) stream, is given to the called module some time between the execution of the *explicit to* and the *explicit from*. The decision of when and how to give control to the called module,

TABLE I – Proposed Language Extensions for Data Control of Sequence and Parallelism Among Program Modules.

NAME	ALGOL FORM	DATA SOURCE	DATA TARGET	SEMANTICS
Explicit to	**to**f(x,...,z)	calling (this) module	entry queue of called module.	transmit the parameters to the called module.
Explicit from	**from** f(x,...,z)*	return queue of calling (this) module.	calling (this) module.	get the value of the called procedure and output parameters.
Implicit to	**return** val,(x,...,z)	called (this) module.	return queue of calling module.	return value of procedure and output parameters as specified by an explicit from.
Implicit from	**receive** (x,...z)	entry queue of called (this) module.	called (this) module.	set values for dummy parameters as obtained from an explicit to.
Full call	f(a,...d)*	calling (this) module.	calling (this) module.	transmit parameters to called procedure, activate, return outputs to calling procedure, and restore control in calling procedure.

*Alternatively to permit the natural use of *explicit from* in expressions, f(x, . . . , z) could be used for that purpose and **call** f(a, . . . , d) for the full call.

†In review it was noted that the structures defined by the proposed mechanism appear to be isomorphic to the computation graphs of Karp and Miller.[19] A proof of this would provide an independent proof of determinacy, as computation graphs are determinate.

within the bracketing mentioned, is that of a supervisory system. Within a called module, the *implicit from* indicates when a module expects its input parameters, and an *implicit to* returns output to the module which generated the inputs of a particular activation.

Now it is only necessary for a programmer to get into the habit of making to-calls at the earliest possible point, that is, as soon as all the parameters are available, and from-calls at the latest point, only when the output values are immediately needed. Thus the use of data control of module activation does not require added analysis for possibilities for parallel processing, only different habits. It should be observed that the use of fully parameterized calls, independence of modules except for explicit task relationships, and the control of intermodule sequencing by programmer specified data constraints permit fairly large and complex systems to be structured as the secondary effect of simply fully analyzing and specifying each subpart of the system.

Asynchronism is permitted by association buffer files in each data stream. A FIFO queue is associated with the entry interface (by which a module is called) and with each linkage return interface (to which a module returns). Note that the latter are required dynamically and will only need to exceed length one when more than one to-call on the same module from the same (or other) module are allowed to queue up.

The modules specified according to the features as described so far are analogous to conjunctive input/conjunctive output nodes of a program graph. This mechanism can be generalized by permitting some parameters to be omitted in any of the statements. Thus, a program might include

```
to F (,,p,q)
   .
   .
   .
to F (r)
   .
   .
   .
to F (,s)
```

which in total would behave like

```
to F (r,s,p,q,)
```

but would permit additional overlap if F were properly written to take advantage of this. For example, p and q may be setup parameters to F, and F would include

```
receive (,,a,b)
   .
   .
   .
receive (c,d)
```

The generalized mechanism is more complicated to implement and involves potentially more processing on each *to* or *from* statement. Input values transmitted by an *explicit to* must be identified by source (i.e., return linkage) to associate portions of the parameter sequence transmitted at various times and prevent mixing of parameters originating from different modules. Each addition to an entry queue must be checked to see if it satisfies a waiting *implicit from*. Once an *implicit from* becomes associated with part of one particular parameter string, all further *implicit from* calls must be satisfied from the same string until this activation is complete. The additional cost may be offset by increased possibilities for parallelism. However, it should be noted that a sophisticated algorithm may be required to select partial parameter strings in satisfaction of an initial *implicit from*. An *implicit from* may be satisfied by several entries in the queue, some of which may hang the module for some time in waiting for other needed parts of the string.

Several interesting possibilities for parallelism are nevertheless opened up by the generalized form. By never requesting parameters which are not used on a particular activation, it is possible that execution could begin earlier. In addition, parameters are frequently used merely to be passed on down in the task hierarchy. Accepting these parameters can be delayed until just prior to transmitting them downward.

Implementation

It should be re-emphasized that the proposed statements are at a very high level and hence the programmer need not be concerned with their implementation. Each statement can be associated with a fairly complex sequence of processing. The whole is assumed to be superimposed on a micro-level system with "conventional" parallel processing instructions. It is at the micro-level that the problems of implementation discussed earlier are to be attacked. They can be solved using present solutions or potentially, in new ways which depend upon the constrained forms of parallelism possible under the data control mechanism.

CONCLUSIONS

What has been presented is a generalization of the normal task subordination mechanism to isolate the input and output portions of the call operation. This permits the simple and natural specification of data-presence constraints in such a way that sequence and parallelism are the by-product of a rather straightforward discipline. It is proposed that this generalization be used as a source language level

method of specifying process parallelism. The data control method proposed is limited to intermodule parallelism. From the standpoint of overhead and supervisor functions, this is probably beneficial. However, nothing prevents data control from being combined effectively with source language specifiers at the statement level such as the **and** and **parallel for.**

ACKNOWLEDGMENTS

Appreciation is expressed to Professor David Martin, now at UCLA, for early criticism and suggestions and for encouraging the publication of this proposal.

REFERENCES

1 J P ANDERSON
Program structures for parallel processing
Communications of the ACM
Volume 8 Number 12 December 1965 p 786

2 J B DENNIS E C VAN HORN
Programming semantics for multiprogrammed computations
Communications of the ACM
Volume 9 Number 3 March 1966 p 143

3 J A GOSDEN
Explicit parallel processing description and control in programs for multi- and uni-processor computers
AFIPS Proceedings of the 1966 Fall Joint Computer Conference
Volume 29 p 651

4 A OPLER
Procedure oriented language statements to facilitate parallel processing
Communications of the ACM
Volume 8 Number 12 December 1965 p 786

5 N WIRTH
A note on program structures for parallel processing
Communications of the ACM
Volume 9 Number 5 May 1966 p 320

6 L L CONSTANTINE
Toward a theory of program design
Data Processing Magazine
Volume 7 Number 12 December 1965 p 18

7 L L CONSTANTINE
Concepts in program design
Information & Systems Press Cambridge Massachusetts
Second Edition 1967

8 E W DIJKSTRA
Solution of a problem in concurrent programming
Communications of the ACM
Volume 8 Number 9 September 1965 p 569

9 D E KNUTH
Additional comments on a problem in concurrent programming control
Communications of the ACM
Volume 9 Number 5 May 1966 p 321

10 J E RODRIGUEZ
Analysis and transformation of computational processes
Massachusetts Institute of Technology Project MAC
Memorandum MAC M 301 March 7 1966

11 E C VAN HORN
Computer design for asynchronously reproducible multiprocessing
Massachusetts Institute of Technology Project MAC
Technical Report MAC TR 34 November 1966
November 1966

12 F L LUCONI
On the equivalence of two asynchronous computation description schemes
Massachusetts Institute of Technology Project MAC
Computation Structures Group Memorandum Number 25

13 A L PUGH III
Dynamo users manual
MIT Press Cambridge Massachusetts 1961

14 S SCHLESINGER L SASHKIN
POSE: A language for posing problems to a computer
Communications of the ACM
Volume 10 Number 5 May 1967 p 279

15 D MARTIN G ESTRIN
Models of computations and systems—Evaluations of vertex probabilities in graphical models of computations
Journal of the ACM
Volume 14 Number 2 April 1967 p 281

16 M E CONWAY
Design of a separable transition diagram compiler
Communications of the ACM
Volume 6 Number 7 July 1963 p 396

17 E MORENOFF J B MCLEAN
Job linkages and program strings
Rome Air Development Center
Technical Report RADC TR 66 71

18 E MORENOFF J E MCLEAN
Interprogram communications program string structures and buffer files
AFIPS Conference Proceedings 1967 Spring Joint Computer Conference Volume 30 1967 p 175

19 R M KARP R E MILLER
Properties of a model for parallel computations Determinacy termination queueing
IBM Research Paper RC 1285 September 1964

Anatomy of a real-time trial–Bell Telephone's centralized records business office

by ALAN B. KAMMAN and DONALD R. SAXTON
Bell Telephone Company of Pennsylvania
Philadelphia, Pennsylvania

INTRODUCTION

In the spring of 1965, The Bell Telephone Company of Pennsylvania undertook a trial designed to eliminate most of the paper records used for negotiations in a business office. The objectives were to computerize these files, recall the records in real-time with video display devices and direct the customers' incoming calls with an Automatic Call Distributor. August 28, 1967, a Service Representative successfully handled the first customer contact. Currently an average of 3,000 contacts weekly are handled at twenty display terminal positions.

This experiment encompasses the 88,000 accounts of residence customers in Upper Darby, Pennsylvania. All business accounts are excluded at this time. Prior to the Centralized Records Business Office (CRBO) trial, 24 girls handled the residence subscribers. They sat in pairs with a tub file connecting their desks. Each file contained bills, toll statements, pending orders, credit information and contact memoranda for approximately 10,000 subscribers. Figure 1 depicts a typical installation.

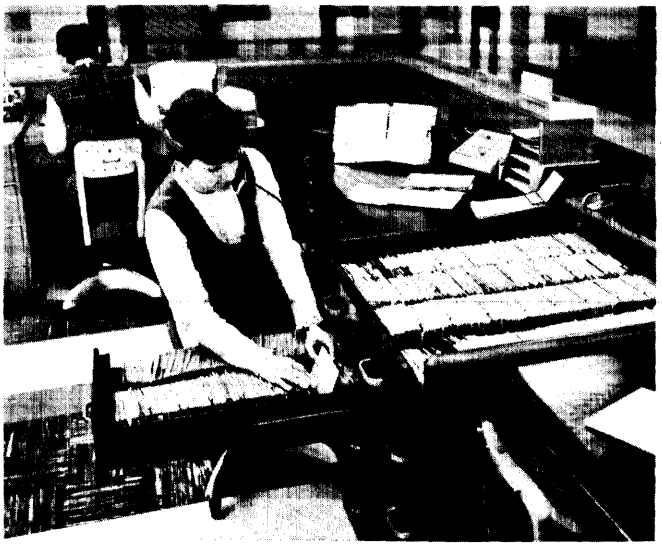


Figure 1 – Service representative using the paper records system

Customer calls to the Business Office were directed through an operator who had access to each Service Representative. During slack periods, calls could be directed to the proper file position after the operator asked the subscriber for his telephone number. During busy hours, calls rarely could be placed to the file location, and Service Representatives excused themselves to leave their position and search for the records in other tub files.

CRBO system overview

Under the CRBO concept, each Service Representative has a cathode-ray tube (CRT) device as illustrated in Figure 2, calls are directed to "open" positions by an Automatic Call Distributor, and the Service Representative types in the customer's telephone number to receive the account information on her screen. Today, the Service Representative is no longer limited by her paper file of 10,000 accounts. Now she can retrieve any record from the computer file. The computer file is located at the Conshohocken, Pennsylvania Accounting Center approximately fifteen miles from the Business Office in Upper Darby.

Figure 2 – Service representative at CRBO position

To serve Upper Darby alone, hardware consists of 28 Raytheon 401 Display Terminals divided between two Raytheon 425 control units. Attached to each unit is a printer-adapter and Bell Telephone's Model 35 teletype for reproducing CRT displays. Each unit is equipped to handle all terminals on either of two four-wire fully duplexed voice-grade circuits, transmitting data at a speed of 2400 bps. Twenty of the CRT's are used by Service Representatives, four by their Supervisors, one by the Public Office Unit and three are located in a training room.

At Conshohocken, IBM's 360/40 computer with 262,144 bytes of core storage handles the messages, using a 2701 high-speed adapter for each circuit. Programs and a specialized customer file are stored on four IBM 2311 Disc Drives, while the majority of the customer files are placed on the 400,000,000 byte-capacity IBM 2321 Data Cell. A 425 Control Unit and two 401 CRT's complete the installation at the Computer Center. Figure 3 illustrates the CRBO hardware configuration.

In case of a system failure, four defensive strategies have been devised; (a) transfer to a single resource such as one communication line or control unit; (b) substitution of a resource such as a Dataphone Subset or terminal device; (c) utilization of paper backup records as long as they are retained and (d) handling customer contacts without records, thus usually necessitating call-backs after the system is restored.

The selection of backup alternatives is dependent upon a fault isolation and recovery procedure executed via a combination of machine and man diagnostics. This procedure assigns to IBM, Raytheon, Bell's Plant, Business Office and Accounting Departments specific responsibilities, tests and controls.

The task force organization

After Bell of Pennsylvania decided to investigate the possibility of real-time retrieval, a feasibility study was conducted by eight managers representing various Headquarters Staff departments. When the study and its projected costs were approved by the President, a Project Director was appointed.

The Project Director then selected four direct subordinates, as indicated in Figure 4. One was in charge of System Design, the second in charge of Applications Programming, another in charge of Business Office Methods, Practices and Training, and the fourth in charge of Area Coordination. These, in turn, added the necessary personnel so the Task Force consisted of 45 people at its greatest point.

In addition, approximately 17 people in the Accounting Department's Standards group were dedicated to the trial although they continued to report within their own organization. These programmers developed the machine-oriented software necessary to mesh the 360/40 with its peripherals and its application programs. The Task Force also received direction from the Planning Department which had spearheaded the trial and was concerned with its integration into the overall mechanization plans of the Company.

System capabilities

The file organization of each account supports the majority of tasks performed by a Service Representative. A request for a billing display will furnish message unit usage, payment arrangements, balance due, local service and any special bill negotiations. To satisfy toll inquiries, the computer system is designed to provide a list of those calls, current negotiations on disputed tolls and, on request, a toll investigation status. The latter display will show, after searching three months' records, identical tolls and calls to numbers similar to the one in question. Additional displays are arranged to provide families of related information on the screen at one time, formatted consistent with different transaction codes.

A capability is provided for each Service Representative to "treat" customers delinquent in paying their bills. The Company lists a series of dates for each account, stating when treatment steps should be taken ranging from an educational call to an interruption of service for non-payment. Upon entering the proper transaction code with a range of telephone numbers, a Service Representative will receive, for the accounts for which she is responsible, a summary of all pending treatment (collection) work for the day. From that point on, she need only press a function key labeled "NEXT" to bring up account after account in decreasing order of collection importance. Once she contacts the subscriber and makes payment arrangements, she has the ability to type notations into the file. This data automatically reschedules the account for display at a subsequent date if the outstanding balance is not reduced below the treatable limits.

"Page Ahead" and "Page Back" function keys are available to access displays with a great deal of data. The large 1040-character Raytheon CRT was selected to reduce the need for page-flipping. At present, the function need be used only ten percent of the time.

The Service Representative can also type in the telephone number and function code "SRB" to access the master file for each subscriber. This display provides the listed name and address, billing name and address, record of all equipment, additional directory listings, cross-reference notations and permanent remarks notations as of the last bill date.

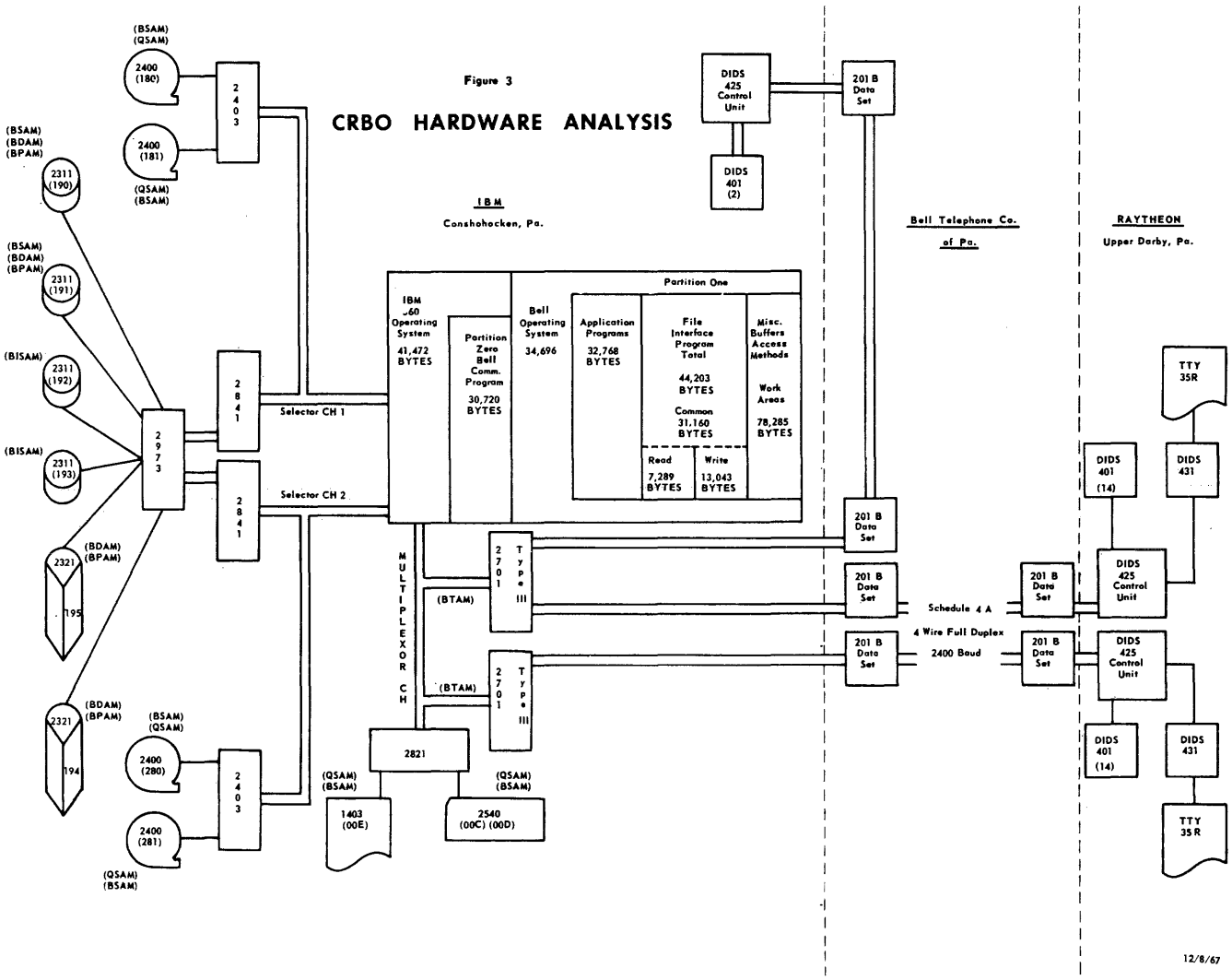


Figure 3 – CRBO hardware analysis

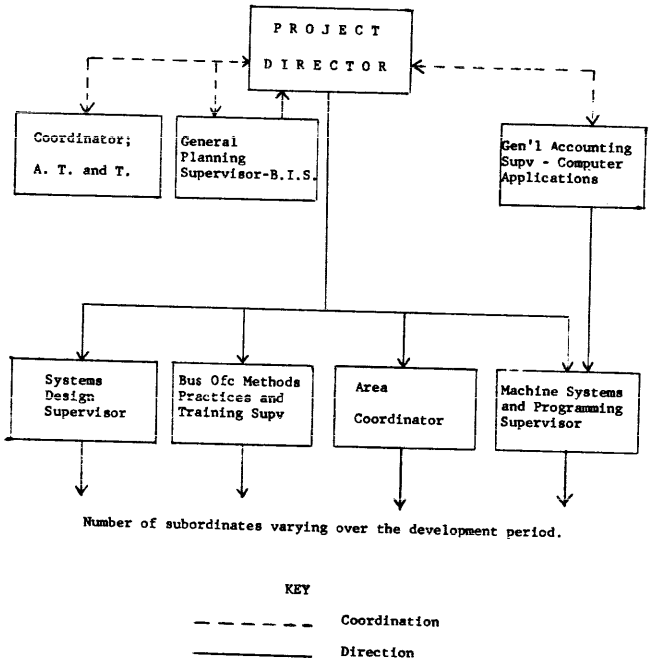


Figure 4—Centralized records business office task force organization

File design

Customer records, with the exception of treatment and notational information, are stored in the Data Cell. The Data Cell Drive contains ten cells with twenty subcells each, and ten magnetic strips for every subcell. The strips have one hundred 2000-character capacity tracks. Three basic customer records are stored on each track, with the capability of overflowing to other tracks addressable from the main record.

When the system calls for data, the basic customer record is read into the CPU and the strip returned to its subcell. Likewise, if information from overflow records and history is required, it is read from the Data Cell into core. The data requested by the Service Representative (i.e., Toll) is formatted page by page to completion. The application program is temporarily interrupted at the end of the third page to transmit the first page. When the application program relinquishes control at the conclusion of formatting, core storage areas are released. If another function is subsequently introduced, the Data Cell is reaccessed.

The Centralized Records Business Office Master File is a distinct entity, separated from the tape files used for the Company's billing application. Ultimately the Company is striving for a single file hierarchy, but for purposes of the trial, new CRBO files were created.

The Centralized Records Business Office Master File is updated daily for cash payments, treatment referrals, number changes, permanent disconnections, final bills and credits. It is updated monthly for service and equipment changes. The updating process consists of three phases; interface, merge and update. An interface program is necessary because the data produced from the five output runs used daily from the Billing operation are not compatible with the CRBO Computer. The tapes must be read through a data conversion program so that each record is reformatted to conform with the CRBO configuration. Five daily billing outputs are regrouped into two "Change Files," then introduced respectively to master and billing update runs.

The billing updates occur several days prior to the date the new bill is received by the subscriber. During the time from the beginning of update to the actual release to the Post Office of the new customer bills, the computer manages the interim period. The Service Representative desiring information from the latest bill the customer has received, will access the file for current records. She does not need to know that a new billing update has taken place. Through internal controls, her request for current data will be routed to the previous month's data. After the mailing of the new customer bill, any requests from the Service Representative for current data will be routed to the latest file. The files are designed to store three months' data; current and the previous two billing periods.

Programming

All real-time application programs used by the Service Representatives, were planned by the System Design group using the SAPTAD process. SAPTAD organizes design logic into six levels of detail, starting with *S*ystem and proceeding through *A*pplication, *P*roject, *T*ransaction, *A*ction and *D*etail. "System" represents the total universe; in this case, a Business Information System. "Application" is the functional, major subdivision such as CRBO. "Project" represents a breakdown of the application into logical components, e.g., "Account treatment." "Transaction" is a further breakdown of the component, such as "payment entry," while "Action" represent the action to be performed as a result of the transaction, e.g., "place data for display." Finally, "Detail" is logical entity of work necessary to perform an action: in this case, "place 'deny-non-payment.'"

All logic conditions, actions and sequences were then placed on Decision Tables. This is a technique of portraying details required in large computer processes. The Tables eliminated machine logic flow dia-

gramming, improved understanding of multiple decisions and provided the ability to prove the table through a simple "Yes - No" process. The tables were turned over to the Application Programming Group who coded them in COBOL E. Subsequently, these were translated into COBOL F to be compatible with certain features in IBM's full Operating System – release eleven.

In CRBO's real-time system, application programs form one of two programming categories. The second is concerned with computer functions and consists of machine-oriented programs. These Utility Programs include the Supervisor, Communications Interface Program (CIP), Transaction Analyzer (TA), File Interface Program (FIP), and IBM's Operating System.

The Supervisor is the control center of the real-time system. This program receives communications from the various parts, controls and coordinates actions, and is responsible for general computer functioning.

The Communication Interface Program (CIP) controls the visual display stations and teletypewriters. It is subordinate to, and under the control of the Supervisor program element. Its major function is to receive and transmit information in the form of coded characters to and from the communication devices.

The Transaction Analyzer (TA) is an application based, machine-oriented program, which takes the incoming message and interrogates the function code associated with it. Then it develops the routing and selection of the application programs necessary for its processing.

The File Interface Program (FIP) is a common interface for linking the CRBO system files to application programs for reading and writing purposes. It insulates the CRBO programs from machine-oriented programs required for controlling the various direct access hardware devices containing the files. It procures the requested information from the peripheral files on a segment basis, expands the data and makes it available to the CRBO application programs. Since all information from a particular record is seldom required, only those portions or segments of the record which are needed are passed to the CRBO program. If other segments are found to be necessary, another call is made to the File Interface Program.

The Operating System for the 360 Computer is a highly complex series of modules and options assembled from a library of routines which can be tailored to the particular needs of the user. One major element of the Operating System contains the processing programs consisting of language translators, service programs and user-written processing programs. The second element, or control program, supervises the execution of the processing programs, controls the location, storage and retrieval of data and schedules jobs for continuous processing.

These Utility Programs were written in the 360 Basic Assembly Language for maximum flexibility and core conservation. Still, however, they occupy so much residency that only 32,768 bytes of core are allocated for the application programs. The Model H40 with its 262,144 byte CPU is, therefore, the minimal size machine in the 360 catalogue which could be used for CRBO.

Testing

Application program unit tests were accomplished in several stages. First, each program was tested in a controlled environment at the Transaction, Action and Detail level of the SAPTAD process. As each Detail coding passed its testing requirements, it was combined with an Action program and retested. This was then repeated at the higher Transaction level and then the program was released for single-thread testing.

Since the application programs were designed to be independent of files or communication devices, and since the CIP, FIP and other utility programs were not completed until late in the development period, simulator programs were designed. These delivered and received data and, in general, issued the characteristics expected of the system when all terminals and files were operative. In addition, the simulators replaced unavailable hardware, and gave absolute control over all conditions so that only one application program was under test.

For example, there was a simulator program to read card test data, simulate an incoming contact and hence call up the proper application programs. The application program under test then called for data from the files. The data was supplied by another simulator reading from card inputs. Following that, the application program transmitted a display which was intercepted by a third simulator, which transferred this "display" to a teletype printer for review.

Single-thread testing took place when the application programs were brought together with the utility programs and most of the hardware. (An exception; Model 35 teletypewriters were substituted for the video terminals.) The tests were conducted using low-volume, controlled input consisting of one transaction at a time.

The same transactions used in the unit test stage were supplied via teletype tape to the single-thread stage, then the resulting output was returned to a teletype printer for verification. When a program failed,

it was returned to unit test for further development.

Then a program was designed to place a load test on the system giving the effect of multiple lines, random transactions and maximum arrival rates. This multi-thread program was written so transactions were fed to the system as fast as the C.P.U. and application programs could handle them. Incoming data was identical to that used in the previous test stages so control of output could be maintained. The data was stacked directly into the communication input queue, and the resultant transaction received directly from the communications output queue, thus eliminating line transmission delay.

Finally, on-line live operation provided the overall systems test. Service Representatives and accounts were added using a controlled schedule, starting with one girl having access to 10,000 accounts for only two hours per day. Within ten weeks this increased to the intended objective of 20 girls accessing 88,000 accounts for the standard working day.

In the interim, debugging took place using dual software packages. One package supported the system for a week, while all changes were made to the off-line software. Each Monday the packages were switched, and hence the system was improved weekly. This technique permitted an accelerated pace in identifying failures which occurred primarily during busy periods, and focused attention on critical areas requiring immediate work.

In concluding the descriptions of software and testing, it should be noted that the Project Implementation Schedule was underestimated. This was caused primarily by incomplete definitions in some cases and pioneering programming techniques in others. It became apparent, however, that the number of activities which could be performed concurrently was also underestimated. The balancing effect caused a total on-line date slippage of only twenty-five per cent of the original estimated elapse-time set two years earlier. A CRBO Event Chronology is listed in Table I.

TABLE I – Centralized records business office - event chronology

Item	*Start Date*	*Complete Date*
Feasibility Study	6-65	7-65
Management meetings, culminating in project approval	7-65	8-65
Task Force appointments	9-65	10-65
General specifications discussions	9-65	11-65
Data collection; existing operation	11-65	2-66
Design documentation techniques investigated	11-65	2-66
Data analysis	12-65	3-66
Completion of design activities	12-65	10-66
Program activities	1-66	11-67
Equipment selection	4-66	7-66
Preparation of Business Office practices	11-66	12-66
Training package development	1-67	10-67
Testing of software	1-67	11-67
Computerization of paper records	8-67	10-67
On-line	August 28, 1967	
Entire office converted	November 20, 1967	

Consultants advice

Early in the design stage of the trial, the human element was considered. The American Institutes for Research (AIR), based in Pittsburgh, Pennsylvania, made valuable contributions in three fields while working closely with Task Force members.

First, they helped to design the floor plan. The new Garrett Road Business Office was given partitioned sections for each of six groups. Each area contains desks for six Service Representatives and their Supervisor. The partitions give the effect of a small office while taking advantage of large-office team-size efficiency. To combat claustrophobia and permit free circulation of air and light, the upper portion consists of open, pastel-colored lattice work. The lower three feet are enclosed, and contain the wiring necessary for the telephone equipment and Raytheon CRT's. A view of the office is shown in Figure 5.

The floor tile is white with dark green striping to give a solid stability in contrast to the pastels, while the brown paneled columns in the office have gold-colored inserts to eliminate any massive effect. The ceiling is a complete series of white light panels, each with internal polarized sheets to reduce glare on the display devices. It is a tribute to the Raytheon CRT's that the green characters on the dark screen are clearly visible under the excellent lighting conditions.

Second, the American Institutes for Research helped to design the desks for Service Representa-

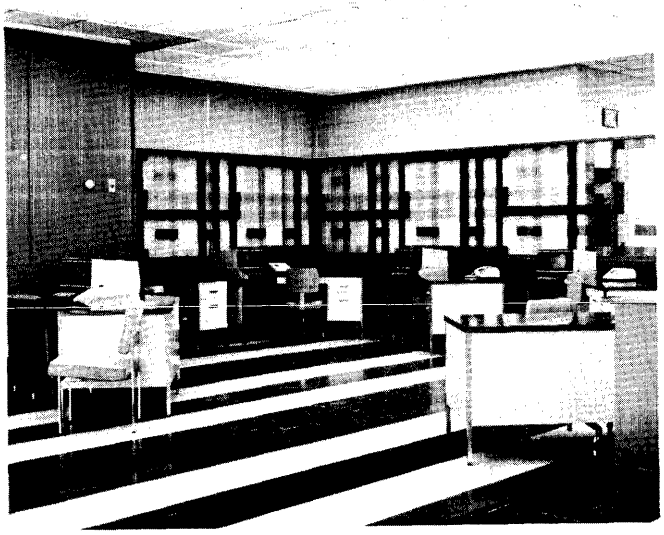


Figure 5 – Centralized records business office

tives. By performing numerous simulation tests on the Representatives who would actually use the positions, a desk with a 45-degree offset for the CRT was designed. The offset was lowered four inches so that the keyboard would be on the same level as the writing surface. The top is a non-glare, brown formica resting on a cream base with silvered legs.

Third, AIR helped the Task Force build training packages for the new applications based on a process designed for the Air Force. The technique, using a flow-chart approach, clarifies material, stresses interaction of the various tasks and clearly defines individual accountability.

The future of CRBO

A major evaluation effort is now under way. Its purpose is to glean information to enable the Bell System to produce an optimal design for the retrieval stage of its contemplated Business Information System. Twenty-four representatives, divided among personnel from the American Telephone and Telegraph Company, the Bell Telephone Laboratories and the Bell Telephone Company of Pennsylvania, have prepared an evaluation program consisting of eleven major areas.

These areas range from testing the effect on subscriber service, to determining the effect on the Service Representative who must deal with a CRT screen all day; from the cost of incorporating CRBO's best features into a major system, to the cost of maintaining it on a stand-alone basis. Outside consultants will be used whenever the expertise is not readily available in the Bell System, and the entire evaluation should take approximately nine months to complete.

Part of the team's functions will be to isolate items which can help the manual Business Office operation. For example, after the first residence groups moved to their new location, they operated with an Automatic Call Distributor and centralized paper record files. Although they had to leave their position every time they needed a record, delays decreased. Citing another case, detailed flow charts and task definitions were prepared for the use of the Task Force System group to design a training package for the mechanized operation. One District now desires to use that preliminary information to review procedures with new supervisors, and to serve as a guide when analyzing contacts during informal training sessions with Service Representatives.

Several advantages are apparent in the mechanized operation. The "Records out of file" condition occasionally encountered in a manual office is non-existent in CRBO. The Service Representatives no longer need to leave their positions to check the centrally located computer printouts which list the latest payments received from subscribers. On-position filing is eliminated. The need to leave the desk to get records in other locations is greatly reduced, and with an average access time of approximately ten seconds, superior service is rendered to the customer.

The treatment functions permitting the Service Representative to receive a summary of her daily collection obligations, followed by the accounts being displayed in descending order of importance, is of value to the Service Representative. Her typed notations concerning payment arrangements automatically reschedule the account for treatment, eliminating the need for written memos, or calendar jottings. The system incorporates a combined training and live file, thus permitting maximum hands-on training for Service Representatives. It has substantially reduced training time by eliminating the need for blackboard and chart work, since test cases available for experimentation can be displayed on the screen by the students.

Since CRBO is specifically a retrieval trial, much paperwork still exists. Communications with the Service Order Typing Room, and treatment or credit notifications to be posted to the billing operation are still done manually. Since the CRBO file parallels the billing file, it has all the problems associated with duplication and reconciliation. In addition, evening update runs take a considerable amount of time and core, thus precluding the use of the computer for other applications.

After the evaluation, the recommendations of the team will strongly influence the future of the Pennsylvania experiment. The true justification of the trial

will come with the use of the evaluation group's data and results to design the optimum system to handle with accuracy and individuality Bell Telephone subscribers throughout the United States of America.

Fourth generation computer systems

by CLOY J. WALTER* and ARLINE BOHL WALTER*
Autonetics Division
North American Rockwell Corporation
Anaheim, California
and
MARILYN JEAN BOHL
Honeywell Inc.
Waltham, Massachusetts

INTRODUCTION

This paper is presented as a discussion of fourth generation computer systems. To predict future developments in the computer industry is to speculate —to theorize on the basis of observable trends and anticipated needs. Numerous questions arise. We do not know the answers to all questions nor do we know how to obtain all the answers. The intent of this paper is to suggest reasonable approaches to developments and to offer a solution to a fundamental EDP problem. How can computers and applications be integrated within a communication and control system?

Computers of prior generations emphasized computation. Fourth generation computers, as envisioned in this paper, will emphasize a communication and control system. The characteristics of fourth generation systems are outlined in the first part of this paper and discussed in detail later. Prior to this discussion, the computer evolution, the software situation, the effects of large scale integration, and fourth generation programming systems are considered.

While one cannot predict characteristics of fourth generation systems with certainty, one can confidently assume that many changes in computing will occur. This paper contains speculation concerning the possible changes. Opinions and suggestions within the paper represent a consensus among the authors but are not representative of the company by which the authors are employed.

Characteristics

We believe that a computer system which possesses the following characteristics will be a fourth generation computer system.

1. The major design criteria will be optimal use of available communication interfaces. The system will be classified as a communication and control system and will be capable of widely diversified processor applications.
2. The system will be controlled primarily by data rather than by programs as were previous machines.
3. Use of hardware to govern communication and control procedures will be emphasized; extensive use of control programs will be substantially reduced or eliminated.
4. Most processing will be executed in real time; operations will be performed on input at a rate that permits output to be available within the response time required by recipients.
5. The system will be readily expandable. Hardware and software will be modular in design. Computing power will be modified without redesign of the system. Hardware malfunctions will be corrected by immediate replacement of disabled modules.
6. The hardware design will permit component parts to be updated; systems need not become obsolete.
7. The system will be designed to operate efficiently, and this efficiency will not be significantly affected by distances between connected elements.
8. Most data will be collected at its source. Cards and attendant keypunching operations will be a secondary source of input.
9. Repetitive entry of input will be reduced or eliminated, and the generation of reports will be on an exception basis.

*Formerly with Honeywell Inc.

10. The system will have an efficient, low-cost program generator.
11. The design will emphasize reduction of total system cost.
12. New software will be simpler—simpler in terms of user convenience, rather than in terms of function.
13. The system will be designed to function without device-specific software routines.
14. Hardware diagnostic routines will be compatible with I/O routines so that on-line diagnostics can be performed simultaneously with normal system operations.

Computer evolution

First generation computer systems were developed primarily for computational purposes. The concept of storing a program to control the operations of a computer and the ability of a computer to cycle repetitively through a sequence of instructions on different data pointed toward use of the computer for computational purposes. Later, the fact was recognized that machines which could be programmed to perform electrical accounting machine (EAM) operations could be marketed. Thus, EDP was born.

Overemphasis on programmed control of system elements led to development of and preoccupation with general purpose computers and a concomitant failure to understand the nature of the applications. How many of us learned to program with little understanding of the computer or of the applications to which computers can be applied? How many of us, after learning to program the 650 machine, really thought we understood data processing?

To provide modularity and flexibility in general purpose machines, computer designers delegated obvious hardware functions to software. A primary design objective was to provide a means by which the user could readily specialize the computer for his particular application. Sorts and other software routines were developed to perform common functions but these elements were designed to make the general purpose computer fit the area or application. The same application had to be modified or reprogrammed as details within the area or application changed.

The development of large data management systems and/or operating systems has resulted primarily from a lack of understanding of the nature of the applications to which computers can be applied. Operating systems have tried to blend hardware, applications, and software. It is the application area which is exploding and which will become dominant.

What has the user seen with regard to hardware? First generation hardware was characterized by vacuum tubes, second generation by transistors, and third generation by integrated circuits. Of much more importance to the user, however, was the reduction in cost of main memory (from approximately one dollar per bit to approximately five cents per bit) and the increase in reliability of the machine. Major advances were made in the reliability of both logic and memory when the change from tubes to transistors occurred. More memory available at reduced cost led to the development of more and better software and an attendant increase in the complexity of applications to which computers could be applied. As main memories became larger, more programs could be resident in memory. Throughput was increased by reducing time loss due to execution of program load and unload routines and relocation functions.

Today's user is interested in total system performance and in the total cost of the system rather than in only the cost of the central processing (CP) unit (currently 15-20% of the hardware cost). He is not as impressed by advanced hardware as he is by efficient operation and ease of programming. The user desires a variety of complex applications, but he wants to tell a computer what to do—not how to do it.

The software situation

Past and present programming can be reviewed briefly as follows.

First generation software was characterized by machine language, subroutines, and assemblers.

Second generation software added higher-level languages, monitors, and macro assemblers.

Third generation software includes operating systems, on-line real-time systems, multiprogramming, and data management systems.

Today's computer user sees:

Sophisticated hardware,

Complex applications,

Increases in application programming costs, and

User defensive programming, i.e., programming around instead of with the operating system.

No new debugging tools have been developed to complement the increased complexity of applications. The percentage of users who know how their operating systems function is decreasing. Processor time is consumed by the operating system for internal scheduling, accounting, and job handling rather than for job execution.

The following questions arise.

Why have we neglected to define software in terms of interfaces, functions, and modules?

Why have we failed to develop more helpful debugging features, more acceptable programming standards, and more useable documentation?

Is the development of operating systems a brilliant solution to the wrong problem?

Do we in the computing field really understand computing?

Today, programming has no theoretical basis and no theories. Why? Programming need not remain a handicraft industry. The designers of fourth generation computers must regard programming as a science rather than as an art. Optimally, scientific theories for programming can be developed. Most assuredly, software systems can be designed and utilized to satisfy clearly defined systems requirements. The problems of programming are not inextricable. Solutions to many programming problems are intermingled with and inseparable from the design of hardware. Engineering personnel must cooperate with software artists to develop a theory of programming based upon an understanding of hardware operations, an understanding of data handling and data control (communication), and an appreciation of software techniques. Only in this manner can redundant programming (repetitive development of programming techniques) be significantly reduced. Parallel developments must be replaced by sequential advancements so that achievements of one individual or group can provide a basis for extended or subsequent advancements by others.

Effects of LSI

The effects of large scale integration (LSI) on fourth generation computer systems can be examined from the viewpoints of both the manufacturer and the user. Major effects of LSI are (1) computer manufacturers will be forced to fabricate LSI chips, and (2) integrated circuit (IC) manufacturers will enter the computer manufacturing field. Competition will increase. Hardware rentals will be reduced, and software will be easier to use.

Effects on the manufacturer

Computer manufacturers will be forced to fabricate LSI chips. Some of the reasons for this action follow.

1. Intimate knowledge of fabrication techniques and corresponding characteristics is essential to circuit, cell and/or chip design.
2. Purchase of LSI chips reveals significant proprietary information about new developments, particularly in the area of LSI design. Vendors who supply LSI chips to computer manufacturers will have access to complete computer designs. Present legal safeguards of designs appear inadequate. Minor changes to chip fabrication without modification of the function of the chip can be introduced to circumvent legal restrictions. A manufacturer will be dependent upon a selected vendor's ability or willingness to continue to supply required components. Second sourcing will be difficult if determination of the internal manufacturing processes is primarily a function of the original supplier.
3. In-house facilities may be required to provide chips which are not available from external suppliers according to a schedule which conforms to the manufacturer's needs and priorities.
4. Computer manufacturers who desire to sell to the military will be expected to demonstrate LSI fabrication capability. Today, military customers demand that systems companies possess microelectronic capabilities even if circuit designs compatible with application requirements can be secured from component vendors.
5. Computer manufacturers commonly desire to lead the development of some aspect of hardware. Such leadership will be difficult if research and development of microelectronic circuitry is relinquished to suppliers of components. Manufacturing operations may be reduced to fabrication of interconnection boards and to simple assembly operations.

All computer manufacturers must develop design automation capabilities to optimize tradeoffs of performance, function, reliability, cost, and size for integrated semiconductor circuits, thick/thin film circuits, LSI circuits, or any combination of these.

Capabilities will be developed in order to meet the following general objectives:

1. To develop and utilize optimum circuit fabrication techniques in order to meet requirements for the manufacturer's computer family.
2. To build microelectronic–LSI circuits on a pilot basis and coordinate efforts of circuit designers and fabricators during the shift to the use of LSI designs.
3. To fabricate LSI chips for which vendors cannot meet delivery, performance, reliability, or price criteria.
4. To develop techniques leading to computer control of design, deposition, mask generation, and testing, and to computer-generated documentation of specifications.

To formulate exact plans for LSI activities is impossible. Materials and fabrication techniques for LSI designs are in various phases of exploratory

research, development, or pilot production. Plans must be flexible and selected equipment must be adaptable to new techniques.

We suggest that LSI activities of computer manufacturers will include at least the following groups:

1. Cost Relationship
2. Liaison
3. Component Test and Evaluation
4. Circuit Test and Analysis
5. Test Equipment and Instrumentation
6. System Organization

The responsibilities within each group are described in the following paragraphs.

Cost relationship

This group will perform cost analyses and determine cost ratios. The cost of fabricating individual LSI cells will be minor when compared to the cost of allocating, partitioning, simulation, routing, and testing. The manufacturer who delegates allocation, partitioning, simulation, and routing functions to a vendor may be forced to forego much of the profit that otherwise might accrue from a computer sale. The number of basic chips to be used and the cost per pattern must be examined. Cost formulas and cost ratios must be developed.

Absolute cost figures are not as important as cost comparisons or cost ratios. Fundamentals must be separated from details. If fundamental costs are identified and organized, details can be viewed in proper perspective. Relationships must be understood. What costs should be compared? Costs which should be calculated are:

1. Silicon costs,
2. Design aid costs,
3. Engineering development costs,
4. Factory production costs, and
5. Actual cost per computer.

A general approach to cost calculation follows:

Let:

- X = the cost in dollars to develop a generalized allocation program.
- Y = the cost in dollars for allocation runs.
- a = the number of different types of chips per computer.
- b = the average number of chips of each type per computer.
- Z = the average cost in dollars for chip fabrication.
- n = the number of computers to be produced.
- T_0 = cost of setting up to test a particular chip type.

The following statements can be made.

The costs for allocation aids (C) is equal to the cost to develop a basic allocation system program (X_0) plus the number of different types of chips (a) times the product obtained by multiplying the estimated cost of specializing the allocation for a particular type of chip (ΔX) by the probability (p) that specialization of the allocation for a new chip type is necessary.

p should be as small as possible. That is,

$$C = X_0 + \Delta X \cdot p \cdot a \tag{1}$$

represents the costs for allocation aids, where

$$X \leq C \leq X \cdot a.$$

Engineering costs can be considered as

$$C_E = X_0 + \Delta X \cdot p \cdot a + Y \cdot a \tag{2}$$

Testing costs can be represented as

$$T \;!\; bT_0 + \Delta T \cdot a \cdot b \cdot n \tag{3}$$

or

$$T = a\,[T_0 + \Delta T \cdot b \cdot n] \tag{4}$$

Factory production cost is $a \cdot b \cdot Z \cdot m \cdot T$. (5)

Actual cost per computer is

$$a \cdot b \cdot Z \cdot T + \frac{X_0 + \Delta X \cdot p \cdot a + Y \cdot a}{n} \tag{6}$$

X will probably be large and Z will probably be small, but their values are significant only when used in comparisons; that is, the ratio between $a \cdot b \cdot Z$ and $\frac{X_0 + \Delta X \cdot p \cdot a + Y \cdot a}{n}$ is very important. The cost ratios must be optimized.

Additional formulas can be generated easily.

If $a \cdot b \cdot Z$ (which represents the silicon cost) is small when compared to $\frac{X_0 + \Delta X \cdot p \cdot a + Y \cdot a}{n}$ major inferences can be drawn. Cost mentioned here can then be compared to total computer system cost (which includes programming, support, training, maintenance, and peripheral equipment).

Liaison

LSI liaison will be very similar to collateral effort

between engineering and manufacturing. Liaison personnel will:

1. Assist the logic design engineer with the application of LSI.
2. Assist in optimizing gate per chip ratios.
3. Provide information regarding scheduling within the pilot line.
4. Assist in expediting miniature components or information needed for the fabrication process.

Component test and evaluation

Component test and evaluation will determine the characteristics of all fabricated component parts and assist with process control and evaluation. Life tests and environmental and mechanical evaluations of elements can be performed. Failure studies and process evaluations can be conducted to detect process variations affecting component part reliability and quality. To provide a well-organized program, the test functions should be coordinated with component applications and reliability engineering.

Circuit test and analysis

Circuit test and analysis will determine whether circuits can be fabricated in LSI form. Activities include construction of breadboards, preparation of specifications, selection of component parts and processes, and design of circuits and chips. Evaluation tests must be performed to determine if chips meet required performance and quality specifications. Other chips considered to be proprietary and extensively used in company equipment should be designed and fabricated on a speculative basis.

This group can prepare an LSI design manual describing available components and fabrication techniques. Guidelines and rules for the preparation of chip layouts according to the various processes can be indicated. New designs can be evaluated and the design manual revised accordingly. The manual should contain a glossary of technical terms and a brief description of the design procedures followed in the development of each type of chip.

Test equipment and instrumentation

This group will conduct electrical inspections and performance evaluation of in-process circuits. They will monitor test equipment requirements and assist as necessary to provide in-process instrumentation.

For test of completed circuits, a test console can be constructed which supports standard test equipment in a convenient manner. This test console should utilize a standard test breadboard adapter which contains any special test circuits unique to particular LSI chips. LSI chips should be mounted on some type of a standard test board so that the complete assembly can be inserted into the test breadboard adapter without damaging the leads.

Computer-assisted testing is currently limited. Efforts to perform automatic testing should be applied to the design and construction of test probes and fixtures which will be initially operated manually and later integrated with computer-controlled adapters.

System organization

This goup will be responsible for interface specifications, tradeoff and utilization analyses, resource allocation, user-oriented systems analysis, and determination of required numbers and varieties of basic chips.

Interfaces arising during the application of new technology must be understood, characterized, and implemented. Creative developments should be stimulated and utilized. Leadership and coordination for defining the interfaces among various design groups implementing LSI should come from a group responsible for system organization.

Tradeoffs can be analyzed in the following areas:

1. Performance, function, and reliability;
2. Sizes of chip production runs;
3. LSI application in functions such as emulation, interpretation, compilation, and control;
4. LSI application in logic-in-memory arrays such as sorting, searching, and signal switching arrays for parallel processing;
5. LSI determination of paging, table lookups, and other processes in the operating system; and
6. The utilization of each type of chip per family member.

Utilization analysis also includes developing methods of partitioning large areas of logic and designing logic structures which can be readily partitioned and interconnected.

Resource allocation includes development of methods to provide LSI techniques to replace resource allocation algorithms. The two major resources to be allocated are subsystems (CP, memory, programs, buses, communication lines, and peripherals) and functional capabilities (logic, arithmetic, and control). User-oriented systems analysis must be conducted to evaluate tradeoffs resulting from maximum or optimum use of LSI from the manufacturer's point of view and from the user's point of view, to insure maximum benefits to the user. Many hardware and software concepts can be viewed simultaneously in an LSI analysis.

Determining the number and variety of basic chips to be used will be an important consideration of sys-

tems designers. Only fifteen or twenty different chips, and a total of only a few hundred chips, may be required for a computer. Repetitive use of chip designs is mandatory.

Determining the optimum attributes and sizes of various LSI memories, logic, and functional arrays will be another responsibility of this group.

We expect IC manufacturers to examine functions currently performed by software and develop LSI designs to perform many of these functions. In fact, major advancements in computer software development may come from IC manufacturers who do not currently market computers.

Effects on the user

The effect of LSI on cost, size, and speed of fourth generation computer systems has been discussed in other articles and consequently is not emphasized in this paper. We can confidently predict that CP cost and size will decrease while speed will increase.

The major advantage available to the user through LSI will be that many of the operating system functions which are currently performed by software can be performed by hardware. Operating systems cannot be eliminated, but operating efficiency can be significantly improved. For example, one task of a current operating system is to allocate resources in the computer system. The task of resource allocation can be simplified if some resources allocate themselves. Hardware advancements which can be achieved through LSI include self-allocating input/output channels and auxiliary memories which do not require main memory for control.

Additional advantages which can be obtained through LSI include:

1. Microprogramming through use of LSI chips (thus allowing a computer to be reorganized according to its work and its workload),
2. Control memory structures which vary in the course of the operation of the machine,
3. Improved fault isolation and self-reconfiguration techniques,
4. Increased use of logic to maintain data integrity,
5. Reduced maintenance costs,
6. Less downtime, and
7. Graceful performance degradation through use of majority voting logic.

LSI will allow a single logical element to be replaced by several logical elements in a manner such that the several elements will be used to determine the state or condition of a situation. The state or condition of the situation indicated by a majority of the elements will be accepted as valid—hence, majority voting logic.

In any event, it appears that, to the computer user, LSI means lower hardware costs and simpler programming languages.

Fourth generation programming systems

To predict languages, degrees of complexity, common techniques, primary considerations, etc., of fourth generation programming systems is both venturesome and difficult. Indeed, to predict in detail or with a high detree of accuracy may be impossible. The effects of LSI on software were discussed in the preceding sections of this paper. This section comprises discussions of design approach, the significance of programming, family planning, user communication with the computer system, a suggested organization for development of fourth generation programming systems, program generation, and charts which depict data and control flow.

First, second, and third generation hardware systems have been designed. Independently, in unrelated efforts, first, second, and third generation software systems have been developed. However, the significance of the total system concept has been disregarded; little, if any, consideration has been given to the formation of first, second, and third generation computing theories. The nature of computing must be re-evaluated, and efforts must be modified accordingly. Cannot creative ideas be applied to integrate software and hardware within effective, useable systems?

Through past generations, computer designers attempted to maximize hardware capabilities (primarily speed). Insufficient thought was given to the user's point of view. For example, he needed a machine which was easy to program, but, in fact, designers seldom, if ever, checked to determine whether their machines would be easy to program or whether programs could be written to maximize utilization of capabilities of system hardware. Major design objectives were to minimize hardware costs, to increase speed, and to plan for batch processing in order to maximize machine throughput. Clearly, insufficient consideration was given to maximizing effectiveness of programming effort. Today, central processor costs are insignificant if compared to total system costs. Programming costs are often several times greater than hardware rental costs. Designing total systems which not only are based upon reliable, efficient hardware but also can be easily programmed is a practical manufacturing objective.

Designers of fourth generation computers should be familiar with hardware, software, and system constructs. Both software and system theories must be developed in cognizance of hardware practicalities. Indeed, fourth generation computing theories must be developed. Design disciplines, which have been significantly lacking (particularly in software efforts), must be established and followed. Hardware personnel must learn software techniques and contribute to total system design.

Before developing a new computer family, a manufacturer must answer the question, "What fundamental EDP problems do I wish to solve?" Efforts and resources can then be channeled accordingly.

Certainly, one fundamental EDP problem is the difficulty of programming. The typical user has neither the desire nor the resources to secure knowledgeable personnel primarily to program his computerized applications. This fact can be a significant deterrent to initial installation or to subsequent upgrading of a computing facility. Fourth generation computer manufacturers must design systems with users in mind.

Proper family (system) planning by the manufacturer is extremely important to the user. To design a family of computers requires discipline; effort must be preceded by forethought. If family members differ only in execution speed and storage capacity, all members present the same logical appearance to the programmer. One instruction set is useable with all models. One specification describes the logical functions of all members of the family. Upward compatibility is easily achieved. Thus, processing under different family members is possible without reprogramming. The user can readily modify his system if he desires.

The difficulty of programming is alleviated in part by the current trend to provide application packages. This trend will continue because small users cannot afford to employ experienced systems analysts. Manufacturers of fourth generation systems will say to the small user, "Submit your data and leave the driving to us."

To the medium-scale user, the manufacturer of fourth generation systems can offer application packages and/or a program generator. If the latter option is selected, the user (commonly, someone not trained in programming or knowledgeable of computers—for example, an accountant) will specialize the system to produce the reports or information that he desires in a form which he specifies. The information flow is shown in Figure 1.

To initiate this information flow, the user operates a desk top input/output device (CRT, TTY, ETY, or other small terminal device) and selects a general program available through the industry-oriented non-resident program generator. He specifies or selects parameter values. Generalized subroutines are fetched from the library by the program generator, and program formation and specialization is completed in the language processor. Special test data supplied by the manufacturer are introduced to test the application program. Processing of the test data produces a representative sample of output which can be expected. If the user is not satisfied, he can enter different parameters and execute another test run or return to the language processor to modify the program which has been created.

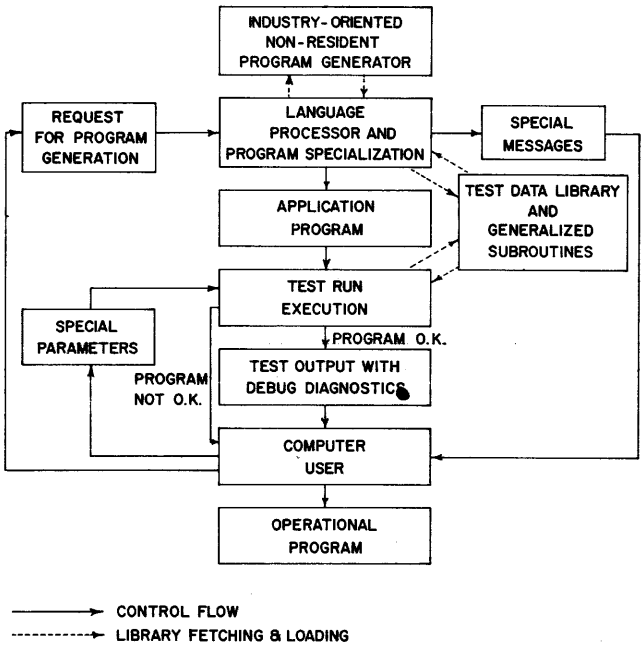


Figure 1—Program generation (Information flow)

A non-resident program generator that is designed to serve a particular industry will control major constructs of system organization for that industry. Specific requirements of each industry will be recognized; for example, a filing technique will be designed for each major industry.

Program execution with user-supplied data is depicted in Figure 2. Inputs and outputs are effected by means of communication lines. The number of lines is not restrictive. The reader is referred to the latter part of this paper for a more complete discussion of the execution.

The large-scale user will commonly design at least a portion of the programs which must be created to perform specialized functions. The language which he is required to use must be readily understood and easily applied. This language should be used not only

to configure the system to perform selected applications but also for inputs and outputs. It must be conversational, thus permitting the user to interface and communicate readily with the system at his disposal. Furthermore, this one language should suffice for all applications—irrespective of the size or complexity of the task to be performed.

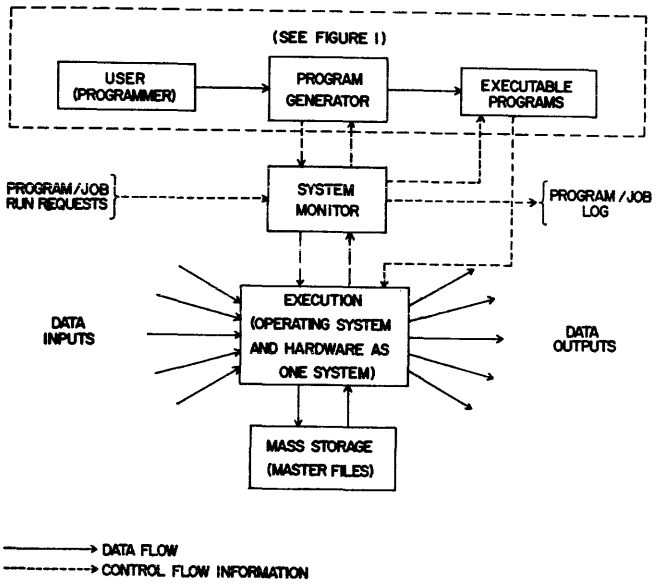


Figure 2—Data and control flow

Software within the system must be comprehensive. A suggested organization for software development and specific responsibilities of each area are shown in Figure 3.

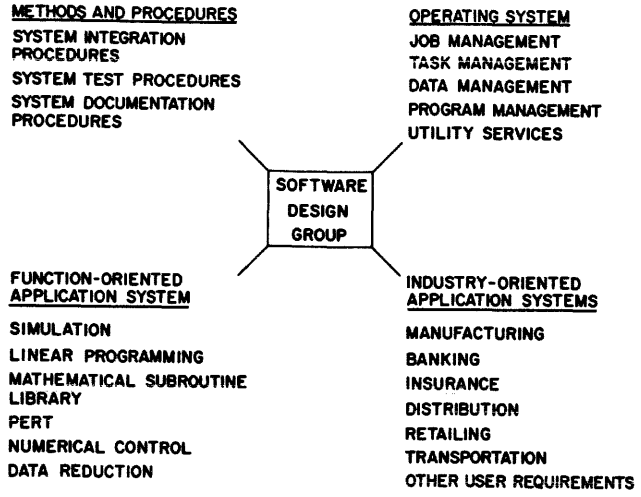


Figure 3—Software design responsibilities

The input to the software design group by the operating system design group is shown in the upper right portion of Figure 3. A suggested organization to provide input to the operating system design group is shown in Figure 4.

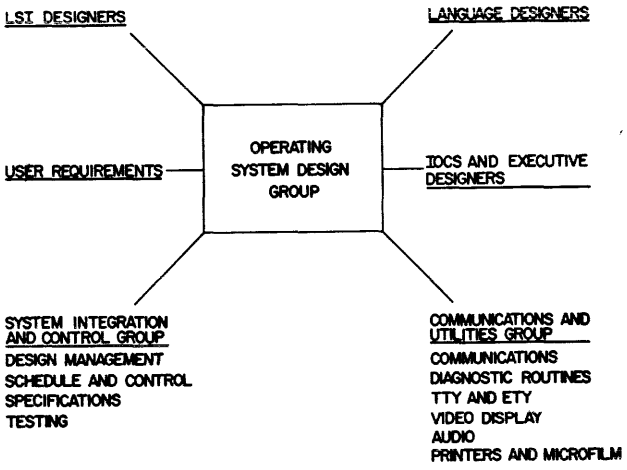


Figure 4—Operating system design responsibilities

Fourth generation computer designers should consider memory levels and interfaces when planning a computer family. The major levels of memory and interfaces are shown in Figure 5. The memory levels depicted in this illustration differ to some extent from the memory levels common in previous generations.

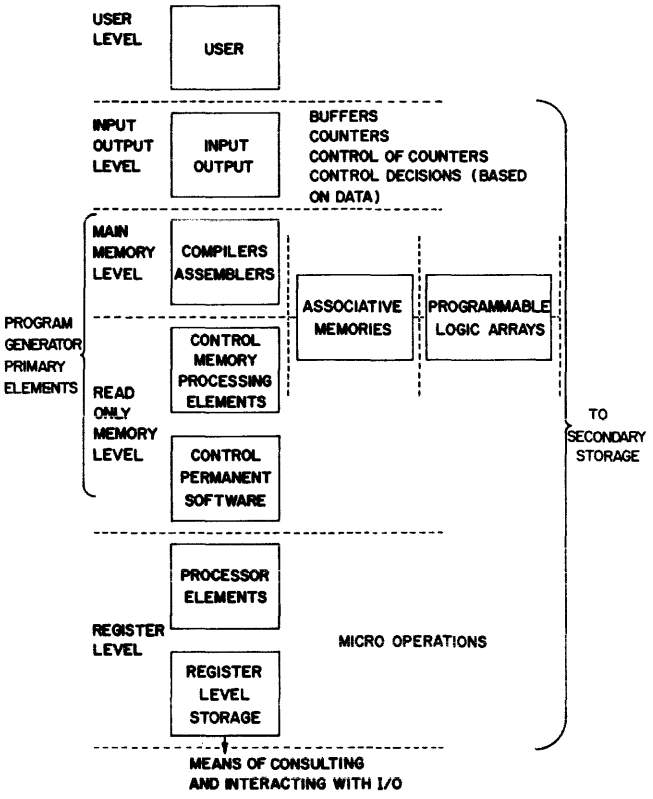


Figure 5—Major memory level interfaces

The disconnected lines in Figure 5 indicate major interfaces which shift as required for particular family members. The roles of associative memories and of LSI programmable logic arrays also vary.

Read only memories and associative memories can be used extensively in medium and large family members—particularly in establishment of program generators.

Programmable logic arrays can perform many of the executive processes currently performed by software and can be used to tailor the system to meet particular user needs.

Progammable logic arrays and associative memories can replace operating system programs and be used to establish logical system organization.

Associative memories can be used for compiling, job assignment, parallel processing, search operations, handling of priorities and interrupts, and recognition of I/O commands. Concurrent operation of high-speed peripheral devices will be facilitated.

Interfaces between software segments and equipment with regard to facility assignment, protection, release, accounting, relative priority, scheduling, and interrupt procedures should be consistent throughout a computer family.

Register-level designers can correlate software modular designs and physical modular designs (functions, translations, data formats, instruction formats, etc.). One group of system/LSI/software designers working at the register level can establish characteristics of the total system—register size, instruction set, multiprogramming, multiprocessing, etc. This group must also answer questions such as whether register logic, counters, comparisons, and control logic can be optimally handled by LSI or by IC's.

Commendable system design utilizes common majorboards, common memory features, common read only memory units, and common software, thereby reducing the cost of design effort. One set of circuits operating at a uniform clock speed can be designed for the entire family. Systems can be carefully designed to maximize cost effectiveness for the manufacturer and, concomitantly, to maximize potential benefits for the user.

Discussion of the characteristics

It is appropriate to suggest methods or approaches by which the characteristics of fourth generation computer systems can be implemented. Some methods have been suggested in preceding sections of this paper. The implementation of other characteristics is discussed in this section. Implementation of the remaining characteristics and integration of characteristics to form a system are discussed in the next section of this paper.

We have stated that the major design criteria will be optimal use of available communication interfaces. Intrasystem and intersystem communication interfaces are required for both hardware and software. Computer professionals are acutely aware that communication capabilities are an important requirement of the next generation of computers.

Fourth generation systems will be controlled primarily by data rather than by programs as were previous machines; i.e., overall system control will be established primarily by input rather than by stored information. Development of this characteristic is dependent upon submission of information in real time. Feedback is a key consideration. Proper interaction between intersystem and intrasystem interfaces is vital. The interrelationships between data (communication bits) and programs (information bits) must be carefully defined.

Use of hardware to govern communication and control procedures will be emphasized; extensive use of control programs will be substantially reduced or eliminated. This characteristic is closely related to the preceding one. Focalizing system design by application of communication networks eliminates much of the need for software and facilitates system control. Again, consideration of both intersystem and intrasystem elements is important. System allocation of its resources and use of LSI for control have been discussed in previous sections. When such techniques are applied, control program requirements will be minimized.

To write that most processing will be executed in real time is to express an opinion. However, a definite trend within EDP toward more processing of data in real time is readily observable. Real time, as discussed in this paper, does not imply the interleaving of programs or the man-machine interaction of time sharing. It does imply that the system will accept inputs as they are made available and process those inputs within the constraints imposed by desired response times.

The system will be readily expandable in terms of both hardware and software. A variable instruction set is not implied. However, nested subsets of software will be available to complement nested subsets of hardware. In fact, this nesting of software is currently practiced. The user's software commonly includes both action macros and system macros. System macros commonly contain nested macros which perform communication functions for specific terminal devices. Such macros can be removed or

specialized. Thus, system modularity results and impetus is given to applying the family concept in terminal design.

An example of functional modularity is a multiplex control device which consists of front-removable elements such as a channel unit, speed/code format decoder, data control unit, and power unit. Desired speed/code combinations in the format decoder can be implemented by replaceable majorboards. Character-rate regulation features for a variety of remote terminals can be established by means of plug-in majorboards.

To construct special purpose computers by specialization or combination of generalized hardware and software modules should be possible. Tailoring hardware to the user's particular needs and/or applications appears straightforward.

Hardware modularity can also be applied to interconnected elements. Interconnect designs will include inter-junction, inter-flat pack, inter-majorboard, inter-unit, inter-shelf, and inter-backboard. Disciplined interfaces can be established between the unit interconnect system, the structural system, and the cooling system, each of which will be constructed as a separate and virtually independent element. A complete enclosure of the unit interconnect system can be designed. All modules can be constructed as entities which are front-removable. A significant objective will be to design systems such that all installation and normal service activities can be performed by means of front access. Hardware malfunctions will be corrected by immediate replacement of disabled modules. Malfunctions in real time systems will be corrected by replacemen of disabled modules within a time span of less than one minute.

Functional modularity will not only help to alleviate interconnection problems within the module but also permit the interconnection of modules such as processors, I/O channel handlers, memory elements, and peripheral devices. Dynamic system reconfiguration will be possible.

Modular design of system hardware is a basic determinant of the degree to which a system can be updated and of the ease with which such updating can be performed. Functional plug-in elements permit the system to be updated. Advancements resulting from technical developments can be readily incorporated in systems currently in operation. However, modular design should not be regarded as a permanent deterrent to obsolescence of fourth generation equipment.

The design of fourth generation systems to permit efficient operation regardless of distances between connected elements is discussed in the last section of this paper

Collection of data at its source is a trend in the computer industry. On-line collection of data will be the standard rather than the exception in fourth generation systems. Translation of data from a medium understandable by the user to a medium understandable by the computer will be an accepted function of the computer.

Most of the data flowing into and out of computers today is unnecessary. Low-cost mass memory will provide a common data base and reduce or eliminate repetitive entry of data. The generation of reports on an exception basis is a technique of system design rather than a problem of hardware or software. The user must recognize that voluminous reports in themselves do not provide answers and that identification of key factors and organization of pertinent reports accordingly is a preferable approach. On-line submission of data or interrogation of the system from remote terminals will be another technique by which desired information can be entered or secured. An overall system approach is needed to determine answers to questions of storage media, I/O devices, types of input and output, frequency of output, etc.

The development of an efficient low-cost program generator has been previously discussed.

Increased emphasis on reduction of total system cost is an obvious trend and needs no explanation.

Software must be designed to facilitate user application. Several methods to ease programming difficulties have been discussed in earlier sections.

Device-specific software routines will be eliminated because the required functions can be performed by a general software routine and interchangeable functional hardware modlues. (See discussion of fith characteristic.)

Hardware diagnostic routines will be performed during normal system operation. Indication of malfunction will be detected so that corrective procedures can be initiated, thus avoiding costly delays that would otherwise occur. For example, suppose that a diagnostic routine to check multiplex operation is run periodically. If the speed/code format decoder in multiplex unit #3 begins to fail, the operating system is instructed to power up multiplex unit #4 and to switch operations being performed by multiplex unit #3 to multiplex unit #4. A message is typed on the typewriter console that the speed/code format decoder on multiplex unit #3 has failed. Maintenance personnel can remove the defective speed/code format decoder and insert a new functional unit.

Compatability of diagnostic routines and I/O routines produces several beneficial results.

1. Minimizes system downtime due to malfunction of hardware elements.
2. Permits graceful degradation.
3. Eliminates the necessity to interrupt normal processing in order to detect and correct minor hardware malfunctions.

Fourth generation computer systems

What is the fundamental nature of computing? We believe that the basis for computing is data handling (data communication) and data control. Data must be communicated to the system, among system elements, and to external recipients. Data are accepted by the system, stored, and processed. Since the system requires I/O, storage, and processing capabilities, why not develop separate processors to perform these functions in an optimal manner? We suggest that multiprocessing systems similar to the configuration shown in Figure 6 will be widely used.

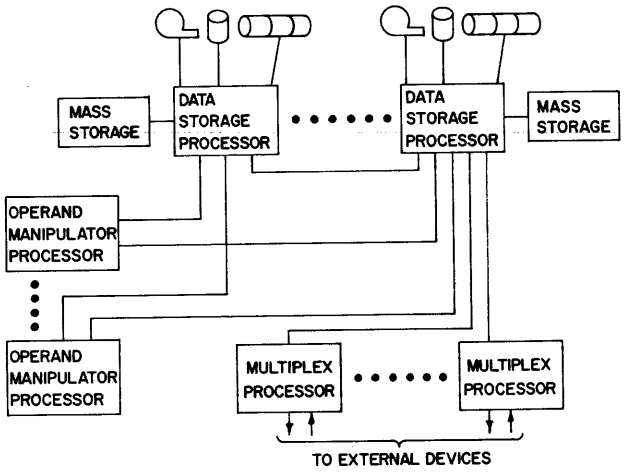


Figure 6—Fourth generation computer systems

The three functional processors can be contained within the same hardware unit. The dots indicate that additional processors can be added to the system. The communication architecture of this system will enhance and encourage modularity by assigning to hardware many of the functions currently performed by software. If several of these small processors are in the system, the failure of one of them will decrease system performance only to the extent that the remaining processors of the same type cannot handle the workload.

The operand manipulator processor will perform the application programming function. All logical processes outside of the system control functions will be executed by this processor. A single communication control system program will reside in this processor.

The data storage processor will handle the requirements for associative memory, secondary storage, mass memory, and communications between processors. The logical structure of the system will be centered around this processor. The data storage element will be divided into zones based upon retention times of stored data. All logical communications between processors or processes will be handled by this processor. This arrangement will permit asynchronous communications.

The multiplex processor will include high-speed record channels with interrupt capability and a multiplex channel designed to service a large number of low-speed devices on a time division multiplex basis. These low-speed devices will include badge readers, teletypewriters, process control stations, bank teller window devices, on-line factory test devices, touch tones, keyboards, and CRT's. A full duplex multiplex channel which can send and receive serial data in either a time division multiplex mode or a record mode will be available in the multiplex processor. Automatic poll and call functions for devices requiring such services will be generated by hardware. Input lines will be scanned automatically by the multiplex channel unit, and data will be brought into main core storage where they can be easily accessed by the data storage processor. The multiplex processor will be capable of continuous operation. Its functions will include accepting data into the system, making queue entries to provide the proper data processing functions, receiving the results of processing, and distributing these results to the external system.

Significant modifications to current mass storage units are needed. These modifications should be introduced to develop mass storage units which will be capable of storing up to a billion characters, have no moving parts, and operate at electronic switching speeds.

Current multiprocessing systems are frequently characterized by identical processors used symmetrically. Such an arrangement reduces a multiprocessor system to a multiprogramming system if interlocks and inter-processor communications are ignored. Techniques of multiprogramming are known and give some insight into multiprocessor systems. If processors are allotted for specific functions as proposed in this paper, hardware can control multiplexing, switching between programs, channel allocations, and several of the storage functions. Operating systems will be easier to design and simpler to understand. Multiprocessing offers potential benefits of speed, because execution is in parallel instead of serial; flexibility, because processing modules can be added without redesign of the system; and increased reliability, because redundant processors allow the

system to "degrade gracefully." Such systems are highly adaptable to potential processor applications.

Fourth generation multiprocessor systems will be characterized by record channel units for communicating among processors and a multiplex processor for communicating to external elements. Record channel units will be used to terminate devices which operate on a record-by-record basis and communicate asynchronously with the processor, e.g., drums, disks, and magnetic tapes. The multiplex channel unit will terminate character-oriented devices, i.e., a large number of independent low-speed devices, each operating on a character-by-character basis. The channel will be a serial time division multiplex loop divided into a number of time division slots. The time slots will be detected by adapters. Each adapter will be connected to at least one control unit which provides hardware interface logic between the loop and an addressed device. The multiplex channel will obtain data for the loop from tables in core and return data from devices to these tables on a data replacement basis. Direct digital control loops can be attached to the multiplex channel.

Fourth generation computer designers will be cognizant of the importance of tradeoffs and of design interfaces and critical paths between physical modules, physical and software modules, and software modules. Designers of current software are more concerned with software-human interfaces than with intra-software interfaces structured to maximize applicability. Tradeoff and interface analyses must answer the questions "How will each change affect the user?" and "How much will each change affect the user?" System tradeoffs in fourth generation computers within communication and control systems will be expressed in terms of response times, communication channel bandwidths, equipment complexities, and numbers of channels.

Facets of interface design that are being established include the following elements:

1. Procedures and standards,
2. Combinations of procedures and standards which function as control elements,
3. Interfaces suitable for use with memory (associative, data only, control only, multiple segment read only, and multiple segment write only),
4. Interfaces between I/O devices, and
5. Interrupt, identification, and other real time and quick time intermodular control functions.

The address structure of a communication-oriented system will permit comprehensive element identification. In addition, use of truncated addresses within any given environment will assure efficient addressing capability. Proper design of communication modes of operation will remove the responsibility for timing considerations from the application program.

In the multiprocessing system discussed in this paper, standardization of interfaces and specifications of standard response times and bandwidths will permit the relocation of application devices in the system.

When the basic nature of applications is considered from a communication and control point of view, the following functions can be identified: data acquisition and reduction, algorithm computation, monitoring, and process optimization and control. These functions can be structured as a horizontal unification of computing elements, I/O and communication elements, and user devices, or data-generating elements. Within all applications, there is also a vertical structuring determined by the specific assignment which the system is initialized to perform. Fourth generation systems must be flexible to permit easy and constant reconfiguration and reoptimization.

SUMMARY

The authors of this paper have attempted to show how computers and applications can be integrated to form a communication and control system. Computer capabilities, tradeoffs, role of LSI, new software systems, examples of design based on interfaces, and overall system configuration have been discussed. Since the primary element that users have in common is data, the development of techniques to achieve data communication and data control is, at the same time, the development of a sound basis for data processing.

One way to stimulate the development of technology is to identify situations in which the results of such development can or must be applied, i.e., to identify and present current or portending needs. The authors have attempted to point out such needs in the computing field. Suggestions and comments in this paper can be considered, studied, and discussed. This discussion can lead to the development of new technology before designs of fourth generation computer systems are finalized. Fourth generation computer systems will be characterized by many of the features advocated herein.

A fourth-generation computer organization

by STANLEY E. LASS
Scientific Data Systems, Inc.
Santa Monica, California

INTRODUCTION

A single processor's performance is limited by its organizational efficiency and the technology available. Paralleling of processors and/or improving the organizational efficiency are the ways of obtaining greater performance with a given technology. Much research has been done on multiple processors and single processors which perform operations on vectors in parallel.

Howerer, significant portions of problems are sequential, and performance in the sequential portions is limited to that of a single processor. This paper describes a proposed new medium- to large-scale computer organization designed to improve single-processor organizational efficiency. The basis of this approach is the separation of memory operations (fetching, storing) control from the arithmetic unit control. Each control unit executes its own programs. Memory operations programs fetch instructions, fetch operands, and store results for the arithmetic unit. Buffering allows a maximum of asynchronism between the arithmetic operations and the memory operations. To perform a given computation, each control unit executes fewer and less complex instructions than a third-generation computer control unit. The less complex instructions require less time to execute and, since fewer instructions per control unit are required, the computer can operate much faster.

Cost-performance of logic

A logic circuit delay of approximately 0.2 nsec has been achieved on an integrated circuit chip. High-speed logic circuit delay of 1.8 nsec has been achieved in the third generation. Low-cost bipolar logic with 250 gates on a chip at 5 cents/gate has been predicted for 1970. Per-gate costs are presently about 50 cents. Cost-performance of logic will thus be about two orders of magnitude better than in the third generation.

Arithmetic operation times

As a result of this cheaper and faster logic, it will be reasonable to minimize operation times by extensive use of combinational logic and separate functional units (e.g., an add/subtract/logical unit and a multiply/divide unit). The implications of this procedure can be emphasized by estimating the arithmetic operation speeds that will result.

These estimates are based on extrapolations from published papers[1,2,3] and include an allowance for the additional logic levels required. Also, the estimates assume a 1-nsec delay in the environment for one level of AND/OR logic along the critical path. The critical-path distance will be minimized by a combination of staying on the integrated circuit chip and keeping the path distance between chips short.

Estimated arithmetic operation speeds are:

Operation	Elapsed Time	Pipelined Time/ Operation
32-bit fixed-point add/subtract	8 nsec	4 nsec
32-bit fixed-point multiply	16 nsec	8 nsec
32-bit fixed-point divide	56 nsec	--
32-bit logic functions	8 nsec	4 nsec
32-64 bit floating-point add/subtract	16 nsec	8 nsec
32-64 bit floating-point multiply	20 nsec	10 nsec
32-64 bit floating-point divide	70 nsec	--

With separate functional units, time can also be saved by using functional unit outputs directly as inputs without intervening storage. Pipelining can also be used to increase the throughput. For pipelined operation, the execution of a function is divided into two or more stages, and a set of inputs can be in execution in each stage. The time between successive inputs can be much less than the elapsed time for the execution of a function.

Cost-performance of memory

Memory costs will be roughly halved by batch-processed fabrication. Access times on the order of 100 nsec and cycle times on the order of 200 nsec will be achieved. This represents nearly an order of magnitude improvement in cost-performance over third-generation memories.

Implications of memory technology

Logic speed is increasing relatively faster than memory speed. Cheaper logic makes it reasonable to perform the arithmetic operations in fewer logic levels. As a result, the disparity between arithmetic operation times and memory access times will increase by a factor of roughly two to three. This implies greater instruction lookahead to efficiently utilize the arithmetic unit's capacity—and increased instruction lookahead is difficult to achieve.[4]

However, a partial solution to this disparity exists and is described in the sections that follow.

Associative buffer and block-organized main memory

A scratchpad memory buffers the processor and main memory. Blocks of words are transferred between the scratchpad memory and main memory. The scratchpad memory and the associative memory together comprise the associative buffer. The operation proceeds as follows:

The virtual address of a requested word is associatively checked with the virtual addresses of the blocks in the scratchpad. If the word is in a block in the scratchpad memory, it is output to the processor. If not, the block containing the word is obtained from main memory and stored in the scratchpad memory, and the word is output to the processor. Similarly, when storing a word, the block must be in the scratchpad memory.

This is similar to paging in third-generation timesharing systems and it involves the same problems (e.g., which block to delete or store when room is needed for a new block). The net result is a substantial reduction in access time when the word is in the scratchpad memory.[5,6]

To provide a basis for comparison, assume a block-organized main memory with each block consisting of 16 consecutive 32-bit words. Eight interleaved block-organized memories of 100-nsec access time and 200-nsec cycle time provide a combined memory bandwidth of over 2×10^{10} bits/second.

Access times from processor to memory are approximately 30 nsec for words in the scratchpad, and 150 nsec for words in main memory. Pipelining through the associative memory and parallel scratchpads is used to achieve a high associative buffer bandwidth.

Assume a fetch or store every 10 nsec, where six percent of these require accessing main memory. The six percent is based on data[5] modified to reflect the differences in computer organization. This corresponds to an instruction rate of approximately 80 million per second. This also corresponds to six blocks per microsecond from main memory or 15 percent of bandwidth. With bandwidth usage this low, another processor could be added without severe degradation in performance due to interference. It also allows high input-output transfer rates with modest interference.

It is desirable, with this design, to group operands and sequence the addressing to minimize the number of block transfers. This lowers the average access time and lowers the main memory bandwidth usage.

Programming implications

Most programming will be in higher-level languages. The computer cost will be a smaller and smaller portion of the total costs of solving a given problem. The main goal of the designer is to maximize the system throughput with programs written in higher-level languages. The user sees a system that executes programs written in higher-level languages.

The average job execution time does not decrease significantly when the computer speed is increased significantly. The explanation for this seems to be that the number of programmers and the number of jobs they submit each day do not change appreciably, but the jobs they do submit are longer in terms of number of instructions executed; e.g., they try more cases or parameterize in finer increments. The number of instructions executed per job by the operating system (including compilers) will probably not increase by more than a factor of five, even if increased optimization of compiled code and decreased efficiency due to use of table-driven compiler techniques (for lower software cost) are factored in.

Operating systems, compiling, and input conversions (e.g., decimal-binary) are essentially input-output functions and their volume is proportional to the number of programmers and people preparing input and reading output. If the computer speed is increased by a factor of twenty-five, then the operating system (including compilers) *time* will decrease by a factor of more than five; and the computer will be executing jobs more of the time. Similarly, the *proportion* of time devoted to byte manipulation, binary-to-decimal conversions, etc., will decrease.

Byte, halfword, and shifting operations may not be included in the hardware for the above reasons. Shifting would be accomplished by multiplying by a power of two.

The equivalence of logic design and programming

Both the logic designer and the programmer implement algorithms. Each has to choose a representation of the data involved. Whereas the programmer uses instructions to implement algorithms, the logic designer uses combinations of logic elements (AND,

OR, NOT, and storage). In addition to verifying that the logic is correct, the designer must observe the electrical limitations of the logic elements and their connections (i.e., circuit delay, fan-in, fan-out, and wire propagation delay) in order to execute the logic function correctly within the time allotted.

Hardware instruction lookahead is, in effect, a recoding of several instructions to obtain the instantaneous control actions. The hardware recoding and the resulting asynchronism depend on conditions within the computer (e.g., variations in instantaneous memory access time due to interrerence). Hardware recoding operates in real time at execution time and is strictly limited in complexity by time and economic considerations.

The recoding can also be performed by software at compile time if execution-time asynchronism is sacrificed. All concurrency is planned at compile time. If an instruction or operand were not available when needed (due to memory interference), the control would halt until it became available. The recoded program, containing control timing and sequencing information, would require several times as many bits as the unrecoded program. It would resemble a microprogram with groups of microinstructions to be executed in parallel. The computer time required for recoding at compile time is proportional to the length of the program, not the number of instruction executions required to complete the program. Also, software recoding is not limited by the real-time constraint. As a result, the software recoding can economically be much more complex and more effective. In the recoded form, operand fetches are initiated several instruction cycles before they are used.

For example, the recoded form of the inner loop of a matrix multiply would be several operand fetches, followed by concurrent operand fetches and arithmetic operations and finally by the last arithmetic operations. The same result could be obtained by an independent operand fetch loop which starts several instruction cycles before the arithmetic operation loop is started. Two separate centers of control are implied. Fewer bits are required to represent the program by specifying the two loops separately, but the number of bits is still more than third-generation instructions require.

The proposed computer organization has a separate control unit for fetching and storing (the data channel control unit) and an arithmetic control unit.

For comparison, note that the CDC 6600 and the LIMAC[7] have separate instructions (but not separate programs) for arithmetic operations and memory operations.

Data channels and their control

Figure 1 shows the data channels which are the information-flow paths in the computer.

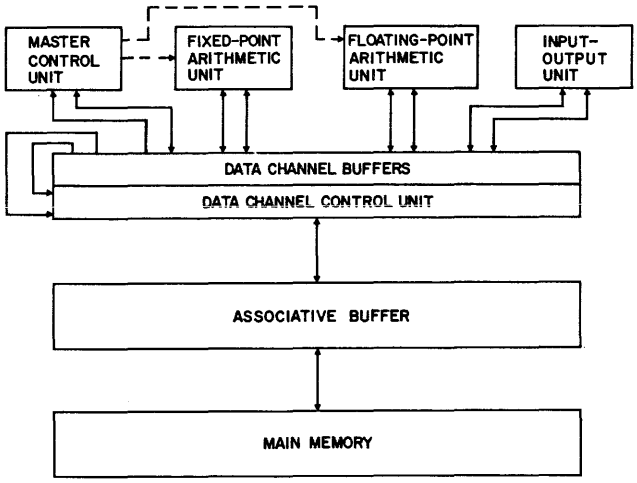


Figure 1 – Information flow diagram. Arrows indicate data paths in the computer. Instructions are transmitted to the arithmetic units over paths indicated by dashed arrows. Double arrows are the data paths for each set of eight data channels. Two data paths suffice for eight data channels, since two data items at most are transferred at a time

Channel commands for multiple-word transfers consist of a virtual memory address, the channel number, a flag indicating load or store pushdown stack, address increment, and count.

For loop control during arithmetic operations, an end-of-record marker follows the last operand. An attempt to read the end-of-record marker as data will terminate the loop.

A channel can be cleared of previous contents by flagging the first command of a new channel program. All store commands of the old channel program for which data was stored in the channel are properly executed. A channel must be cleared or sufficient time must elapse to store the data before subsequent commands reference that data.

Another channel capability is the capability to load a variable number of words (limited by the buffer size) in a circular register. Its use is primarily for storing instructions and constants within loops. It can be entered by flagging the channel command which specifies the last word in the loop. The first word will then follow the last word until the channel is cleared. This usage of the channel will be later referred to as circular mode.

The input-output register of the channel has a data-presence bit to indicate data availability. The register functions in four ways:

1. Nondestructive read:
   The presence bit is left on and the register contents remain the same.
2. Destructive read:
   The presence bit is turned off, the register is filled with the next data word in the channel, and the presence bit turned back on.
3. Nondestructive store:
   The current contents of the register are pushed down one and the presence bit is left on.
4. Destructive store:
   The current contents of the register are replaced.

Provision is made for saving channel status, using the channel for another purpose, and later restoring the channel to its original status.

The master control unit, fixed-point arithmetic unit, and input-output unit each have a data channel reserved for commands. Any data stored in these data channels are transmitted to the data channel control unit for immediate execution as a command.

The second source of commands is the input-output registers of specified data channels. Commands present in the specified data channels (as indicated by the presence bit) are read destructively from the input-output register and executed. The commands are executed by small, fast, special-purpose computers in the data channel control unit.

As an example, the execution of a single command (received from the master control unit) loads a data channel with commands, the first command loads another data channel with commands for fetching instructions, the second command loads still another data channel with commands for storing data, and the remaining commands fetch operands.

Channel command programs loading data can fetch ahead of arithmetic execution a number of words limited by the size of the data channel buffer. Channel command programs end by running out of commands.

Data channel buffering

Channel buffers would be implemented as circular buffers using integrated scratchpad memories. Channel action when used for loading operands is as follows:

Initially, both input and output pointers are set to 0 (see Figure 2). The first input requested goes into word 0, and a data-presence bit is set when the input arrives. Each successive input requested goes to the next higher word (modulo 7). The fetch-ahead depth of our example is limited to 8. Output can only occur if the data-presence bit is set. If the instruction turns off the presence bit, the next output comes from the next higher word. While inputs are requested in the command order, they may arrive at the buffer out of sequence, but they will go to the correct word in the scratchpad.

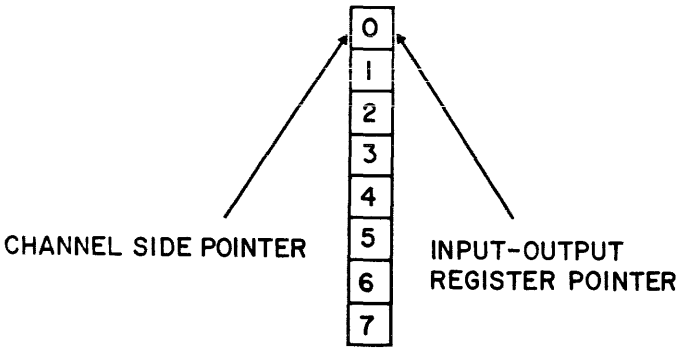


Figure 2 – Data channel buffer

The master control unit.

Eight double-word data channels supply the master control unit with instructions. The control is selected to one of the eight data channels. Instructions present in the selected data channel (indicated by the presence bit) are read destructively from the input-output register of the data channel and executed. There are five types of instructions:

1. Arithmetic instructions are transmitted to the appropriate arithmetic unit.
2. Channel commands appearing in the instruction stream are transmitted to the channel control unit through a data channel.
3. An all-zero instruction is a no-operation.
4. An instruction is provided to conditionally switch between the instruction unit data channels.
5. An instruction is provided to conditionally skip a specified number of instructions.

Arithmetic unit control

An arithmetic instruction specifies the two inputs, destructive or nondestructive read, and the operation to be performed. The inputs are from data channels and functional unit outputs (see Figure 3).

A store instruction transmits the data on an output bus or a data channel to a data channel. One data channel leads to the channel control unit for computed channel commands.

The first stage of instruction execution is testing whether the specified inputs are present. If they are not, the control hangs up until they arrive. During the last stage, the inputs are latched in the functional unit while the operation proceeds. The output-presence bit is set when the operation is completed. The presence bit is turned off by a destructive read of the functional unit output (by an instruction), or by test-

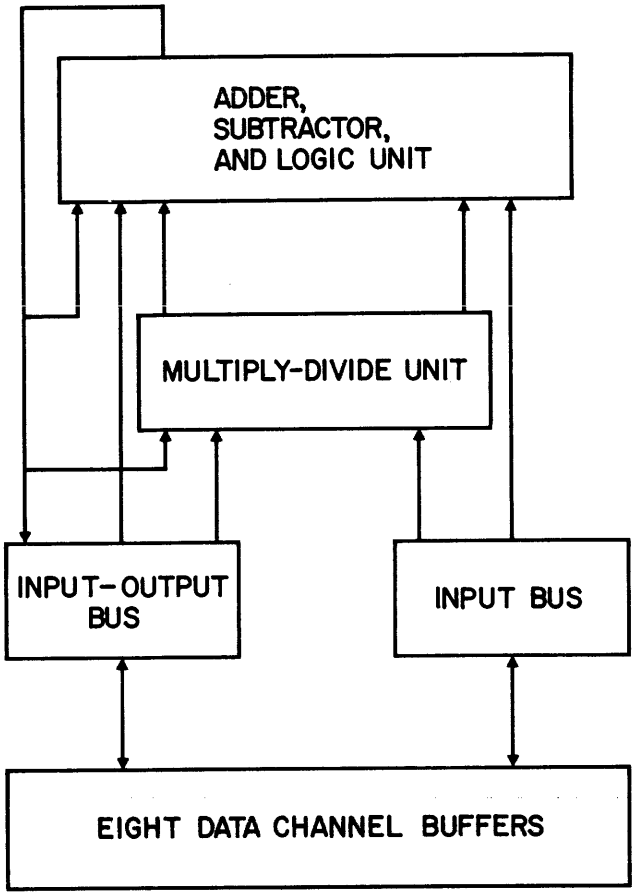


Figure 3 – Arithmetic unit organization. Arrows indicate data paths and direction.

ing the condition code generated by the operation. For example, a compare is accomplished by testing the condition code of a subtract operation.

To pipeline, two pairs of inputs must be latched in before the result of the first pair is read. Trying to read an end-of-record marker results in switching control to the next data channel in the control unit. This is used mainly for terminating loops.

To facilitate data exchange between the separate fixed- and floating-point arithmetic units, two data channels are common to both.

Channel commands for instruction and data sequencing

The channel commands for a set of instructions and their data are normally located together in memory.

Sequential instructions are executed by transmitting a channel command to the data channel control unit specifying the instructions and their data.

Small loops are the same as sequential instructions, except that circular mode is specified in the channel command. Arithmetic data may also use circular modes. Leaving the loop is accomplished by switching to another data channel (by conditional branches) or by trying to read data and getting an end-of-record marker. If this data channel is itself in circular mode, we have a loop within a loop.

Conditional branches are handled by anticipatory loading of a channel with the successful branch instruction stream and switching to the channel if the branch is successful.

Unconditional branches are handled at the channel command level by loading the channel with the branched instruction stream.

Subroutines are handled by loading a channel with the subroutine (or at least with the beginning of it) and then switching control to that channel. Returning is accomplished by switching back to the original channel. Some subroutining can be specified at the channel command level by channel commands for an unconditional branch to the subroutine and an unconditional branch back.

However, sooner or later all channels will be in use, in which case a channel status is saved, the channel is used, and later the channel status is restored. This is analogous to saving a program location counter, executing a subroutine, and then restoring the location counter.

Hardware design and packaging

The computer is naturally partitioned into nearly autonomous units. Repetition of parts is found in the multiple uses of data channels (which are mostly memory), the special-purpose computers in the data channel control unit, and the associative buffer (which is mostly associative and addressable memory).

The hardware complexity of control needed to achieve a high level of concurrency is minimized by separate control of memory operations and of the arithmetic unit. Complex instruction-lookahead hardware is not required.

Input-output

Input-output would be controlled by a small computer with a scratchpad memory for input-output commands and buffering.

The small computer is the interface between the peripherals and the data channels. It also generates commands to control input-output data transfers in the data channels.

Time-sharing and multiprogramming

Paging from a rotating memory is the currently popular solution to managing main memory in a time-sharing environment. If, in a typical third-generation system, a 50×10^6 instructions/sec processor is sub-

stituted and the page access time plus transfer time not changed significantly, the processor will be waiting on pages most of the time. Access time from rotating memories cannot be improved significantly, but transfer rates with head-per-track systems can be very high.

This suggests an approach based on the ability to read complete programs from the disc into main memory quickly, process them rapidly, and return them to the disc quickly. This minimizes the prorated memory usage by a program and allows high throughput without an excessively large memory.

The scheduler maximizes processor utilization within the constraint of system response times. Programs normally reside on the disc. The scheduler selects the next program to be transferred to memory. One factor in the selection is the amount of time before program transfer would begin (instantaneous access time). The program is transferred at 10^9 bits/second (e.g., a 10^6 bit program is transferred in 1 msec). The program is put on a queue of programs to be processed. Having the complete program in memory allows processing without paging to the next input or output by the program, and then writing the processed program into available disc space (generally the first available disc space) without regard to its previous disc location. The previous disc location is added to the available disc space.

Scheduler considerations include the distribution of available space around the disc, distribution of ready-to-be processed programs around the disc, and nearness of ready-to-be-processed programs to their response-time limit. Programs whose processing time exceeds a specified limit are not allowed to degrade the system response time of the other programs. Programs larger than the memory require partitioning into files or pages.

Main memory (or optionally a slower, lower-cost random-access memory) and the disc buffer the input-output activity of the programs.

The system could be organized to place FORTRAN users in one group, JOSS users in another group, etc. Each user group would have its own compiler and supporting operating system. A portion of the operating system would be common to all of the groups. The computer would be a "dedicated" FORTRAN system for a fraction of a second, then a "dedicated" JOSS system for another fraction of a second, etc. As a result, operating systems would be simpler and a change could be made in one system without affecting the other systems.

CONCLUSIONS AND OBSERVATIONS

The system described here achieves concurrency of fetching, arithmetic operations, and storing without the need for complex instruction lookahead hardware. The complexity of control is in software.

The bandwidth of the processor is over 100 million equivalent third-generation instructions per second. This rate will be achieved for some problems. However, delays due to waiting for operands or instructions in the data channels will lower the processing rate in many cases. Some types of probelms seem to inherently have a great deal of delay—for example, table-lookup using computed addresses.

Problems in which the flow of control and addressing are not data-dependent could run near the bandwidth of the system. (An additional requirement for this is that the addressing be such that the block transfer rate between main memory and the associative buffer is reasonable.) Optimizing the code consists of (1) minimizing the delays caused by instructions and operands not being available when needed, and (2) pipelining and overlapping the arthmetic operations.

To program for this processor in its machine language, a master control program (instructions) and channel programs (commands) are prepared. There are many chances to make an error and lose synchronism between instructions and commands. As a result of the difficulty in machine-language programming with this organization, even more programming would be in higher-level languages.

Initially, the compiler for this computer could be relatively crude and unsophisticated. As time passes the subtleties and characteristics of the design would be assimilated and experience gained by the compiler writers. As a result, midway through the fourth generation the computer should average 50-80 million equivalent third-generation instructions per second.

A more powerful data channel command set than is described here may be desirable for non-numeric applications.[8]

The only significant way to reduce software cost by hardware is to build a faster computer (with a lower cost per computation), which will then allow the programmer to reduce total costs by using algorithms that are simpler to program but require more computer processing.

REFERENCES

1 C S WALLACE
A suggestion for a fast multiplier
IEEE Transactions on Electronic Computers Vol EC-13 February 1964

2 M LEHMAN N BURLA
Skip techniques for high-speed carry propagation in binary arithmetic units
IRE Transactions on Electronic Computers Vol EC-10 December 1961

3 S F ANDERSON J G EARLE
R E GOLDSCHMIDT D M POWER
The IBM system/360 model 91: Floating-point execution unit
IBM Journal of Research and Development Vol 11 No 1 January 1967

4 D W ANDERSON F J SPARACIO
R M TOMASULO
The IBM system/360 model 91: Machine philosophy and instruction handling
IBM Journal of Research and Development Vol 11 No 1 January 1967

5 D H GIBSON
Considerations in block-oriented system design
Proceedings of the 1967 Spring Joint Computer Conference

6 G G SCARROT
The efficient use of multilevel storage
Proceedings of the IFIPS Congress Spartan Books 1965

7 H R BEELITZ S Y LEVY R J LINHARDT
H S MILLER
System architecture for large-scale integration
Proceedings of the 1967 Fall Joint Computer Conference

8 B CHEYDLEUR
Summary session, proceedings of the ACM programming languages and pragmatics conference
Communications of the ACM Vol 9 No 3 Marcj 1966

Optimal control of satellite attitude acquisition by a random search algorithm on a hybrid computer

by WILLIAM P. KAVANAUGH, ELWOOD C. STEWART and DAVID H. BROCKER
Ames Research Center, NASA
Moffett Field, California

INTRODUCTION

Computer implemented parameter search techniques for optimization problems have become useful engineering design tools over the past few years. Many, if not most of the techniques, are based on deterministic schemes which have inherent limitations when the system is nonlinear. Random search techniques have been suggested which propose to overcome some of the difficulties. References 1-3 give good general discussions of the merits of random techniques. Reference 4 develops an algorithm, based on random methods, to solve the difficult mixed two-point boundary value problem that results from an application of the Maximum Principle. The method was shown to be remarkably effective in solving a fairly complex fifth-order, nonlinear orbital-transfer problem. The purpose of this paper is to discuss the application of the random search algorithm to a still more complex problem to demonstrate its feasibility. The example chosen was the three-dimensional, large-angle, single-axis attitude acquisition control problem in which it is desired to minimize fuel expenditure to accomplish the acquisition. The equations are highly nonlinear since small angle assumptions cannot be made; the control torques are assumed to be limited. This problem is more complex than the orbit-transfer problem in that the dimension of the state vector[6] is greater by 1 and the number of degrees of freedom allowed the control action is greater. The same acquisition problem was dicussed in Reference 5 but a proportional control law was assumed. A random parameter search was used in that paper to find the optimal set of feedback constants for the given control system structure so as to minimize system performance (fuel). Systems performances will be compared to indicate the striking improvement in performance with optimal nonlinear control.

In the following sections we will state the control problem for notational purposes, review the random search algorithm developed in detail in Reference 4, discuss the hardware and software necessary for implementing the algorithm, and last, present the results of applying the method to the satellite acquisition problem.

Problem formulation

The problems considered are restricted to those for which the Maximum Principle is applicable. Although familiarity with the principle is assumed, a few remarks are necessary to properly pose the problems we will be concerned with in this paper. The system to be controlled is defined by the vector equation

$$\dot{x} = f(x,u,t) \tag{1}$$

where $x = (x_1, x_2, \ldots, x_n)$, $u = (u_1, \ldots, u_r)$ and $u \in U$ where U is the allowable control region. Interest will center on fixed-time problems because of their convenience in computer operations. It will be desired to take the system from a given state x(o) to a final target set S so as to minimize the generalized cost function

$$C = \sum_{i=0}^{n} \alpha_i x_i(T) \tag{2}$$

where $x_0(t)$ is the auxiliary state associated with the quantity to be minimized. The target set S, for the example chosen here, will be defined as $S = x_f \in R_n$, that is, a fixed point in n-dimensional space.

It is well known that application of the Maximum Principle to the stated problem invariably requires the solution to the following set of equations:

$$\left.\begin{aligned} u &= u(x,p,t) \\ \dot{x} &= f(x,u,t) \\ \dot{p} &= p(p,x,u,t) \end{aligned}\right\} \tag{3}$$

where $p = (p_1,...,p_n)$. We can see that the Maximum Principle yields a good deal of information about the nature of the control, that is, we know the function u(x,p,t), and we know the equations for x and p. However, at no time do we know the specific values of both x and p. For example, at the initial time, x(o) is generally known from the problem specifications, whereas p(o) is not known. The remaining boundary conditions required will be known at the final time by some combination of components from the x and p vectors. Thus, there is difficulty in solving these equations even numerically because the known boundary conditions are split between initial and final times.

Random search algorithm

In this paper we will use the algorithm developed in Reference 4 for solving the mixed boundary-value problem. Consequently, we will give only an intuitive account of this approach necessary for the later example application.

A direct way of solving the mixed boundary-value problem is to convert it into an initial-value problem. From Equations (3) it is clear that for any arbitrary value of p(o), there will be sufficient information to determine a final state x(T). Since this state will generally be different from the desired state $x_f(T)$, we introduce a vector metric J (see Ref. 4 for the significance of using a vector metric rather than a scalar metric) to measure the distance between x(T) and $x_f(T)$. It is convenient conceptually to think of the components of the vector quantity J as hypersurfaces in an n-dimensional space of the components of the p vector. Then the boundary-value problem is equivalent to finding the simultaneous minima of all the hypersurfaces. It is important to note that in this case the minimum values are known, i.e., zero.

Deterministic approaches for finding the minima of the hypersurfaces have a number of difficulties. For example, the gradient technique requires the calculation, or possible experimental measurement, of the partial derivatives of the surfaces at each step of an iteration process. If the surfaces are discontinuous, have many relatively rapid slope changes, or regions of zero slope, gradient approaches will fail. The random search techniques overcome these difficulties. References 1 through 4 discuss the virtues of these methods in greater detail. In particular, it is demonstrated in Reference 4 that many of the hypersurface abnormalities mentioned above actually occur in even a moderately complex problem.

The random search approach to be used here was described in considerable detail in Reference 4. The approach is based on a direct search of the hypersurfaces by selecting the initial condition vector p(o) from a gaussian noise source, followed by an evaluation of the corresponding values of the hypersurfaces. It was shown that the pure random search is not practical for moderately high-order systems because of the slow convergence to the minimum. However, by making the search algorithm adaptive, the convergence properties were shown to be greatly improved. This was accomplished by: (a) varying the mean value of the gaussian distribution on any iteration so as to equal the initial condition of the adjoint vector on the last successful iteration, and (b) varying the variance of the distribution so that the search is localized when the iterations are successful but gradually expanded in a geometric progression when not successful. Thus, the mean provides a creeping and direction-seeking character to the search while the variance provides an expanding and contracting character.

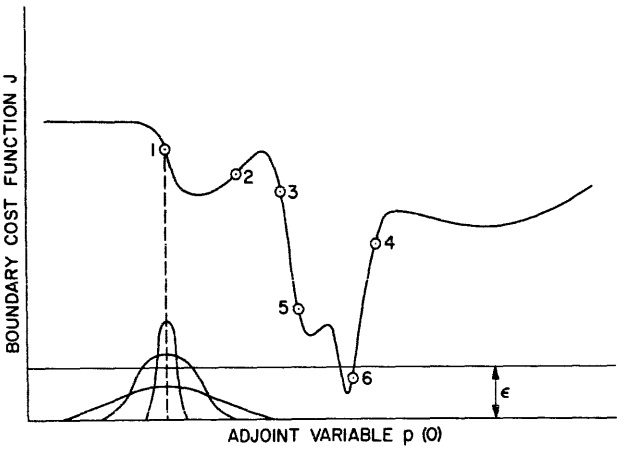


Figure 1 – Typical boundary cost function surface

The behavior of the algorithm is illustrated in Figure 1 by the typical boundary function surface given in only two dimensions. Starting at point 1, the mean of the distribution is made equal to the corresponding value of p(o); the search starts with the small variance indicated and gradually expands in geometrical steps until a lower point on the surface is detected, such as at point 2. Then the mean of the distribution is made equal to the corresponding value of p(o) and the search repeats. The search continues as indicated by the typical numbered points until a value of zero or some small value, ϵ, near zero is reached.

The type algorithm has some desirable properties that enable it to find the minimum. For example, it has a local minimum-seeking property that is due to the small variance used on those iterations which are successful. The algorithm also has a global searching property that enables it to jump over peaks, which is due to the expansion of the variance when the iterations are not successful. Further, it will not matter

whether the surface is discontinuous, or has many peaks, valleys or flat regions.

Implementation

The hybrid computer proved to be the most feasible way to implement the random search algorithm. A primary reason for this is that a relatively large number of iterations are required to find a solution. Reference 4 showed that approximately 8000 iterations were required on the average for a typical solution. Each iteration can be, from a computational point of view, divided into two steps: (1) integration of the equations of motion on the interval [O,T], and (2) execution of algorithm logic. The analog computer is by far the faster machine in performing step one. Although the second step might be accomplished in approximately the same time with either machine it is best done digitally. Thus the conclusion is reached that a hybrid approach requires a great deal less computer time than a completely digital simulation. It is worth noting that an alternative approach with pseudohybrid techniques was investigated using an analog computer and something less than a digital computer. However, our experience shows that inaccuracies, limited storage and limited flexibilities in logical operations seriously limit the feasibility of this approach.

In the hybrid implementation, the analog computer was delegated the task of solving the state, adjoint, and control equations as given in Equation (3). It also served as the point at which the operator exercised manual control over the hybrid system. The digital computer was required to calculate the metric, provide storage, implement the algorithm logic, randomly generate the initial conditions for the adjoint equations and, finally, oversee the sequencing of events of the iterate cycle. This division of computational effort is shown schematically in Figure 2 which is a block diagram of the search algorithm. The superskript k shown in this figure designates the generic k^{th} iteration of a long sequence of iterations. For convenience the flow of information through the block diagram may be thought to start at p(o) which represents the adjoint initial conditions in analog form. They are applied to their respective initial condition circuits on the analog machine, and then that machine is commanded into an operate mode. The set of Equations (3) are solved on the time interval [O,T], and at time T those analog components we are interested in reading are commanded into a hold mode and the executive sequencing program instructs the A/D converters to read these variables. On the basis of this information the boundary cost function metric is computed digitally, and then tested by comparing it to the last smallest value discussed above. On the basis of this test information, the algorithm logic operates to control the mean value m^k and variance σ^k. A new random vector $p^k = m^k + \xi^k$ is then generated, converted to an analog signal, and applied to the p(o) initial condition circuits. The whole process is repeated for the next iteration.

Figure 3 is a hardware diagram of the hybrid system used. Shown are the two basic elements of the simulation, the analog and digital computers along with their coupling system, and peripherals. The coupling system is comprised of two distinct parts: (a) the Linkage Systems and (b) the Control Interface System. A discussion of the hardware used in these subsystems is given in the four sections to follow. The next (fifth) section discusses the sequencing of events through the subsystems during one iteration cycle in order to better describe the functioning of the hybrid system as a whole. Discussed in the final section is the flow graph for the algorithm.

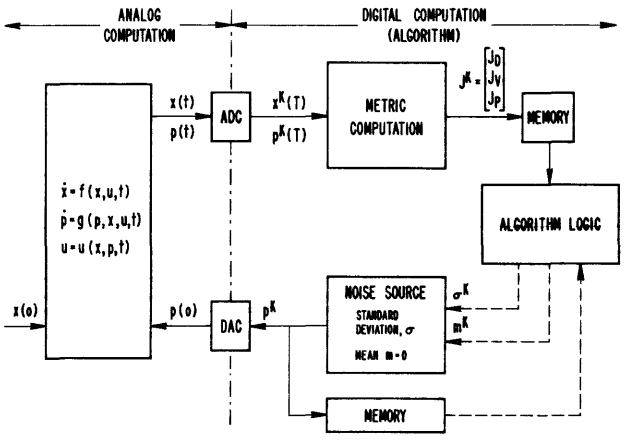


Figure 2 – Hybrid system block diagram for random search method

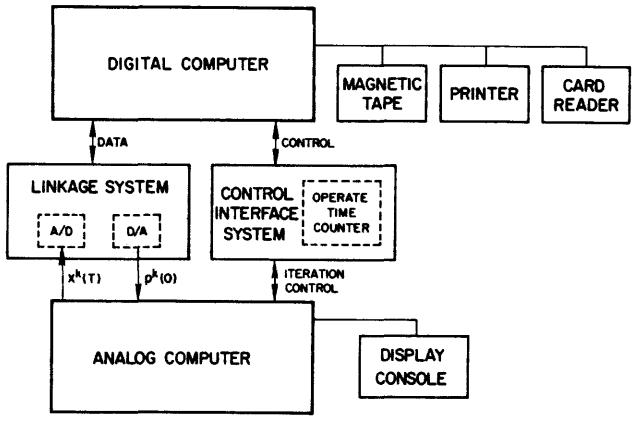


Figure 3 – Hybrid system hardware

Digital Computer

The digital computer used in the optimization program was an Electronic Associates, Inc. (EAI) model 8400. The particular machine used has 16,000 words of core memory with 32 bits per word. Memory cycle time is 2 microseconds. The machine uses parallel operation for maximum speed. Floating point operations are hardware implemented. The optimization program was coded in MACRO ASSEMBLY in order to keep the execution time to a minimum. The instruction repertoire includes special commands by which discrete signals can be sent to or received from the external world. External interrupts are provided which can trap the computer to a specific cell in memory. In an example to be discussed later, the optimization program utilized about 8,000 words of storage. Of these about 1,000 comprised the actual optimization executive program, the remaining 7,000 being used for subroutines, monitor and on-line debugging and program modification routines.

Analog Computer

The analog hardware consisted of an Electronic Associates 231R-V analog computer. Since the state equations, adjoint equations, and the control logic were programmed in standard fashion, analog schematics were not included.

The analog computer serves as the point at which mode control of the hybrid computer is accomplished. By manual selection of switches either of two modes can be commanded: (1) In the "search" mode the analog computer operates in a high-speed repetitive manner. Such operation is accomplished by controlling the mode of the individual integrators with an appropriate discrete signal. This signal is a two-level signal which is generated on the control interface in conjunction with the digital computer and, depending on the level, holds an integrator in either "operate" or "initial condition" mode. (2) In the "reset" mode, the integrators are placed in their initial-condition mode and held there.

For continuous type output, a display console was connected to the analog computer to provide visual readout of variables. The display contained a cathode ray tube (CRT) which could simultaneously display up to four channels, and enabled photographic records to be taken of the display quantities. The display was extremely helpful in determining if the algorithm was functioning properly.

Control Interface

The control interface between the analog and digital system is an Electronic Associates, Inc. DOS 350 (see Fig. 3). It is through this unit that the iteration process is controlled. An important task allocated to this subsystem is the operate-time control. This function is implemented through the use of a counter and is the key element in the control of all timing in the hybrid simulation. The counter is driven from a high-frequency source in the interface system allowing for a very high degree of resolution in the simulated operate-time. Also, the interface allows the digital computer to use any conditions in the analog computer which can be represented by discrete variables (binary levels) and to send discrete signals to the analog system to be used as control elvles. or indicators. An example of the former would be the hybrid system mode control which merely amounted to the operator depressing the "reset" or "search" switch on the analog computer. This action sets a binary level which is then sensed by the digital computer. An example of the latter situation is when the digital sends the operate command to the operate-time counter. The interface system allows patching of Boolean functions. Hence, some of the logic operations required for timing pulses, event signals, and other like operations were very effectively programmed on it.

Linkage System

The linkage system shown in Figure 3 houses the conversion equipment, the A/D and D/A converters. It is through here that all of the data pass between the analog and digital portions of the simulation. The linkage system is controlled by command from the digital computer.

Input to the digital computer is through the A/D converter via a channel selection device or multiplexer that selects the analog channel to be converted. Conversions were sequential through the analog channels at a maximum rate of 80,000 samples per second from channel to channel.

Output to the analog used the D/A converters, with each data channel having its own conversion unit. The maximum conversion rate of the D/A's used is 250,000 conversions per second.

Sequencing of events during one iteration

The sequencing of events during one iteration cycle are depicted in Figure 4. The instants of time t_1, t_2, . . . , t_5 shown in this figure are considered fixed relative to each other, and t_1 is conveniently regarded to be the start of the iteration cycle. We will consider the cycle to begin at t_1 with the analog integrators in an operate mode. As discussed previously, the elapsed time $(t_2\text{-}t_1)$ is controlled by a counter on the interface system. At t_2 an interrupt pulse is generated on the control interface which is sent to the digital computer

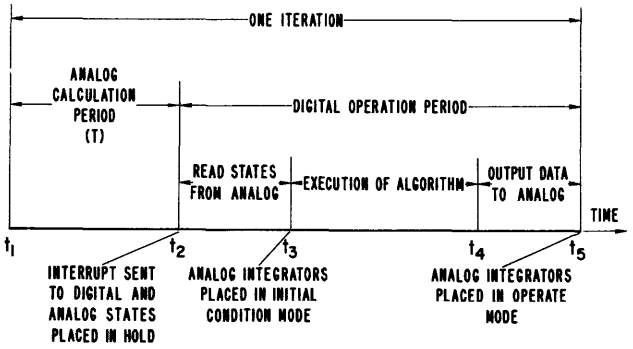


Figure 4 – Sequencing of events during one iteration

signaling it to commence its operations. Simultaneously, the pulse is sent to the analog to instruct the track-store units to hold their respective values which they had at time t_2. During the interval (t_3-t_2) the digital computer reads these analog variables with the A/D converter. At t_3 the digital sends a pulse via the interface to the analog console which commands the integrators to an initial conditon mode. At t_4, when the data required by the algorithm have been generated, the D/A converters send these values to the appropriate points in the analog portion of the simulation. The digital machine allows enough time for the transients to settle in the initial condition circuits of the analog before sending a command at t_5 that places the integrators in an operate mode and starts the counter. Since t_5 and t_1 are the same event, we merely repeat the above sequence for repetitive operation.

Some specific numerical values might be of interest. The total iterate time (t_5-t_1) is primarily composed of two parts: (1) (t_2-t_1) which in a later example problem was scaled in the simulation to 7.5 milliseconds, and (2) (t_5-t_2), which was primarily determined by the speed of the digital machine in computing, converting, and generating random numbers; this latter period was on the order of 7.5 milliseconds. Thus, the total iterate time for the above situation is on the order of 15 milliseconds (or 66 iterations per second). This figure is dependent on the control problem chosen and the exact form of the algorithm implemented.

Algorithm flow graph

Figure 5 is a program flow graph showing the software requirements on the iteration process. This basically constitutes a majority of the steps involved in the algorithm and the iteration control sequences utilized by the hybrid system. Note the inclusion of the event times t_1, t_2, ... discussed earlier in connection with Figure 4. The program is continuously recycling in a high-speed repetitive fashion.

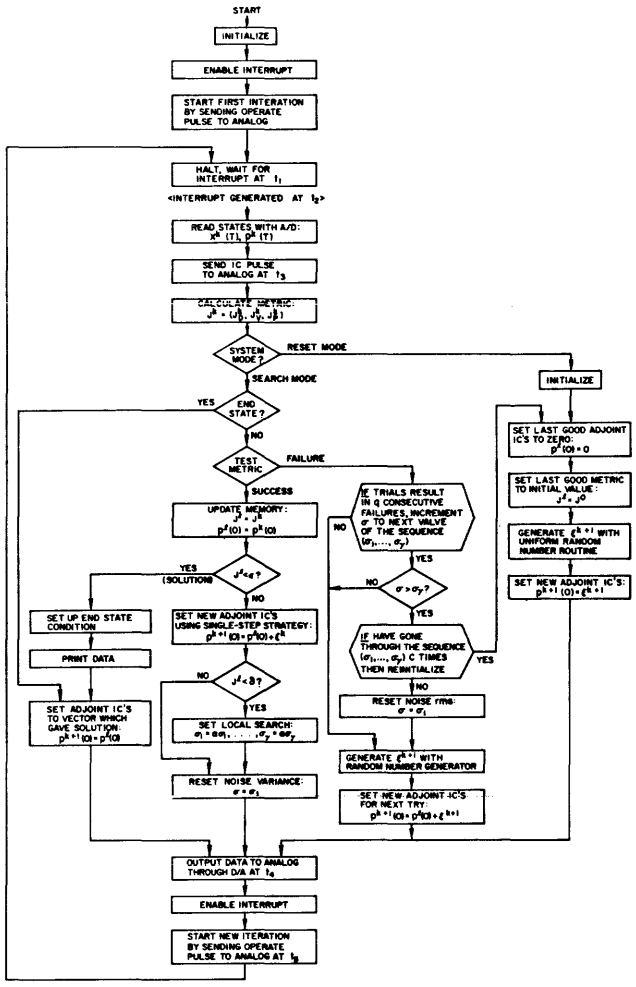


Figure 5 – Algorithm flow graph

There are three basic loops in Figure 5 corresponding to the three system modes in the optimization program: a reset loop, a search loop, and an end-state loop. The reset loop initializes the program. The search loop uses the algorithm to search for a solution to the problem. The end-state loop is entered by the digital program when a solution is found, and is used for generating graphic displays. The operator manually selects the search or reset mode as discussed in the section dealing with the analog computer. A more detailed description of these loops is given in Reference 4.

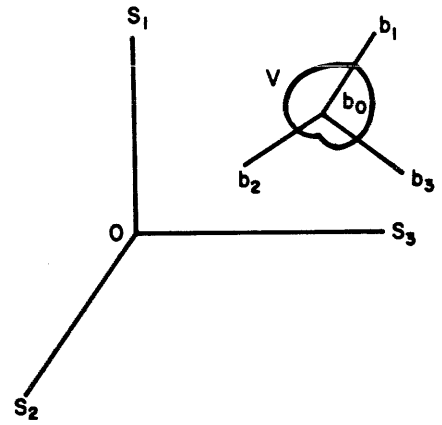
Application

In this section we will discuss the application of the random search algorithm to the single-axis attitude acquisition control problem. In the following we will first formulate the problem by giving a physical description of the problem and writing out the exact equations of motion. Second, we will outline the equations necessary for determining optimal nonlinear control as derived by means of the Maximum Princi-

ple; for comparative purposes we will also outline the optimal proportional control derived in Reference 5. Next, we will discuss the boundary conditions and the vector metric. In the final two subsections we will illustrate the computer results: in one we will give a variety of time history solution and fuel performances, and in the other, some cross sections through the boundary cost-function hypersurfaces.

Formulation

Consider a freely rotating vehicle V about a point b_o in inertial space defined by the set of axes (S_1, S_2 S_3) shown in the sketch.



A fixed set of body axes, b_1, b_2, b_3, is ascribed to the vehicle in the principle axes of inertia with origin at b_o. The orientation of any body axis with respect to inertial space will be specified by direction cosines, e.g., a_{13}, a_{23}, a_{33} where a_{13} is the cosine of the plane angle between S_1 and b_3, and similarly for a_{23} and a_{33}. The orientation of the vehicle is specified by a (3×3) direction cosine matrix. Since we are interested in single-axis orientation, we will only require knowing three direction cosines instead of nine. In the study we will orient b_3 in inertial space. The control torques required to orient the vehicle are produced by mass expulsion devices alined with each of the three axes. It is assumed the mass flow rates are used to vary the torques, and that they are bounded (except when examining proportional control). The vehicle is inertially unsymmetrical and is considered to be in a general tumbling motion at the initial time. The objective of control is to apply torques for a fixed period of time in a manner that will reduce total momentum to zero and orient the b_3 axis of the vehicle from any initial orientation to any other prescribed orientation in inertial space. Furthermore, we must accomplish this task with the control program that uses the least amount of fuel. This verbal statement of the problem will now be formulated more explicitly.

The dynamical equations of motion of a body of fixed inertia rotating about a point in inertial space and acted upon by external torques are given by the following set of equations:

$$\left.\begin{aligned} \dot{\omega}_1 &= \{(1-\beta)/\alpha\}\,\omega_2\omega_3 + Y_1/\alpha \\ \dot{\omega}_2 &= \{(\beta-\alpha)\}\,\omega_3\omega_1 + Y_2 \\ \dot{\omega}_3 &= \{(\alpha-1)/\beta\}\,\omega_1\omega_2 + Y_3/\beta \end{aligned}\right\} \quad (4)$$

where

$\alpha = I_1/I_2$ Roll to pitch inertia ratio

$\beta = I_3/I_2$ Yaw to pitch inertia ratio

$Y_1 = M_1/I_2$

$Y_2 = M_2/I_2$ } Controll accelerations, rad/sec^2 normalized to I_2

$Y_3 = M_3/I_2$

ω_i = body rates, rad/sec

$i = 1,2,3$

M_i = torque, lb/ft

The three kinematical variables (direction cosines) required to specify the orientation of b_3 are designated a_{13}, a_{23}, a_{33}. These variables are related to the dynamical variables by the following set of differential equations (see Ref. 5 for a discussion on this):

$$\left.\begin{aligned} \dot{a}_{13} &= \omega_3 a_{23} - \omega_2 a_{33} \\ \dot{a}_{23} &= \omega_1 a_{33} - \omega_3 a_{13} \\ \dot{a}_{33} &= \omega_2 a_{13} - \omega_1 a_{23} \end{aligned}\right\} \quad (5)$$

For this study the specific values of the roll and pitch inertia ratios were taken to be $\alpha = 1.15$, $\beta = 0.48$. Also, the maximum control torque acceleration permitted in the nonlinear controller situation was limited to approximately one-sixth the peak-acceleration required for proportional control, or $Y_{1max} = Y_{2max} = 2Y_{3max} = 7.5$ rad/sec^2.

To transfer (4) and (5) into state variable form, the following substitutions are made:

$a_{13} = x_1$; $\omega_1 = x_4$; $u_1 = Y_1/\alpha$

$a_{23} = x_2$; $\omega_2 = x_5$; $u_2 = Y_2$

$a_{33} = x_3$; $\omega_3 = x_6$; $u_3 = Y_3/\beta$

This gives us the following set of state variable equations:

$$\left.\begin{aligned} \dot{x}_1 &= x_6x_2 - x_5x_3 \\ \dot{x}_2 &= x_4x_3 - x_6x_1 \\ \dot{x}_3 &= x_5x_1 - x_4x_2 \\ \dot{x}_4 &= \{(1-\beta)/\alpha\}\,x_5x_6 + u_1 \\ \dot{x}_5 &= \{(\beta-\alpha)\}\,x_6x_4 + u_2 \\ \dot{x}_6 &= \{(\alpha-1)/\beta\}\,x_4x_5 + u_3 \end{aligned}\right\} \quad (6)$$

The objective of the control is to take the state vector from an arbitrary initial value x(o) to an arbitrary final value $x_f(T)$ in a fixed interval of time [O,T],

and use the least amount of fuel in so doing. A new coordinate, proportional to the total fuel used in all three axes, can be defined as follows:

$$x_0(t) = \int_0^t \sum_{i=1}^{3} |u_i(\tau)| d\tau \tag{7}$$

and we can then interpret the objective as the minimization of the terminal value $x_0(T)$.

Control Laws

The nonlinear optimal control can be derived by an application of the Maximum Principle. This derivation will not be given here but a summary of the equations necessary for computer implementation will be given. First are the adjoint equations which can be shown to be:

$$\left.\begin{aligned}
\dot{p}_1 &= p_2x_6 - p_3x_5 \\
\dot{p}_2 &= p_3x_4 - p_1x_6 \\
\dot{p}_3 &= p_1x_5 - p_2x_4 \\
\dot{p}_4 &= p_3x_2 - p_2x_3 - p_5x_6(\beta-\alpha) - p_6x_5(\alpha-1)/\beta \\
\dot{p}_5 &= p_1x_3 - p_3x_1 - p_4x_6(1-\beta)/\alpha - p_6x_4(\alpha-1)/\beta \\
\dot{p}_6 &= p_2x_1 - p_1x_2 - p_4x_5(1-\beta)/\alpha - p_5x_4(\beta-\alpha)
\end{aligned}\right\} \tag{8}$$

Second are the equations defining the optimal control vector at each instant of time:

$$\begin{aligned}
u_i(t) &= N_i \operatorname{sgn} p_{i+3}(t) \quad \text{if } |p_{i+3}(t)| > 1 \\
u_i(t) &= 0 \qquad\qquad\quad \text{if } |p_{i+3}(t)| < 1
\end{aligned} \tag{9}$$

where $i = 1,2,3$ and N_i is the maximum torque acceleration allowed in the i^{th} control axis. It is seen that the control torque is of the on-off character and that torque direction is obtained by assigning the correct sign to the "on" signal according to Equation (9).

An optimal proportional control law used in this paper for comparative purposes was taken from Reference 5 and is discussed briefly here for the sake of completeness. In Reference 5 the structure of the optimal control law was assumed to be of the form:

$$\left.\begin{aligned}
u_1 &= -Dx_4 - Ex_2 \\
u_2 &= -Fx_1 + Gx_1 \\
u_3 &= -Hx_6
\end{aligned}\right\}$$

The parameters D, E, F, G and H were left free, and by means of a random parameter search suitable values were found for which the stated objective of the problem (zero momentum and alinement of b_3 to S_3) was achieved. The search was repeated a number of times, each time observing system performance given by equation (7). An optimum parameter vector was selected from the set of parameter vectors which satisfied the problem objective and minimized the performance criteria. This parameter vector, in conjunction with equation (10), defines optimal proportional control. This control may be difficult to achieve in practice, however, since no bounds have been imposed on the thrust. When there are bounds, the control law is then referred to as optimal saturating proportional control.

The vector metric

The desired boundary condition at the terminal time was chosen to be zero momentum and alinement of the body axis b_3 with the inertial axis S_3; this is expressed by $x_f(T) = (0, 0, 1, 0, 0, 0)$. To satisfy these boundary conditions, it is necessary in the random search approach to introduce the vector metric J as discussed previously. Its general form was specified in Reference 4 to be $J = (J_D, J_V, J_P)$ where the subscripts on the components refer to displacement, velocity, and adjoint variables, respectively. However, in this application, since all terminal states $x_f(T)$ are fixed, p(T) is completely free so that we may ignore the J_P component in the vector metric. For the present example J_D and J_V are taken to be

$$\begin{aligned}
J_D &= \sqrt{x_1^2 + x_2^2 + (x_3-1)^2} \\
J_V &= \sqrt{x_4^2 + x_5^2 + x_6^2}
\end{aligned}$$

It is clear that $J_V = 0$ implies $x_4, x_5, x_6 = 0$ which represents zero momentum as desired. Also, $J_D = 0$ implies the desired final orientation $x_1 = x_2 = 0$ and $x_3 = 1$. In actual practice we will only require

$$(J_D, J_V) < (\epsilon_D, \epsilon_V)$$

where the ϵ values are chosen to meet the problem requirements. For the specific problem discussed below, the value of ϵ_V chosen reduced a 10°/sec initial velocity error in each axis to approximately 0.75°/sec in each axis at time T. The ϵ_D chosen required that the b_3 body axis be oriented to within a few degrees of the S_3 inertial axis, from any initial orientation.

Time history solutions

In this section we will give some computer solutions for the satellite attitude acquisition problem. Our interest will center on results for the optimal nonlinear control obtained by implementing the random search algorithm discussed in the preceding sections. For comparison, we will also give results for no

control and the proportional control studied in Reference 5.

Table I summarizes the controllers to be studied, the initial conditions of the vehicle, and the resulting fuel requirements. It is worth emphasizing again that in the initial condition vector the first three components are the initial angular positions given in terms of direction cosines varying between −1 and +1. The latter three components are the initial angular velocities in degrees/second, and were chosen somewhat arbitrarily. The desired final condition vector is taken to be $x_f(T) = (0, 0, 1, 0, 0, 0)$; thus, the initial momentum is reduced to zero and the body axis b_3 is to be kept alined with the inertial axis S_3. Also, a solution is given for the more general case where initial misalinement exists.

TABLE I – Comparison of Control Systems

	Initial Condition	Fuel
No Control	(0,0,1,−10,10,10)	0
Optimal Proportional Control (no saturation)	(0,0,1,−10,10,10)	.65
Optimal Impulse Control	(0,0,1,−10,10,10)	.28
Optimal Nonlinear Control (with saturation)		
a. Initial alinement	(0,0,1,−10,10,10)	.31
b. Initial nonalinement	$(0,\frac{1}{\sqrt{2}},\frac{1}{\sqrt{2}},-10,10,10)$	.43

A. No Control – In Figures 6(a) and (b) are shown the time histories of the state variables describing the motion of the system when no control is used and the vehicle starts with the initial condition $x(o) = (0, 0, 1, -10, 10, 10)$ This corresponds to initial alinement of axes but with the initial angular velocities indicated. It can be noted that the angular positions vary over their entire range (−1, 1) during the time interval [O,T]. From these results it might be anticipated that with limited torque control, the wide range of angular position variations would persist over a substantial portion of the interval. In this event, the equations of motion are distinctly nonlinear and it is inappropriate to linearize them. The Maximum Principle allows these nonlinearities to be dealt with directly.

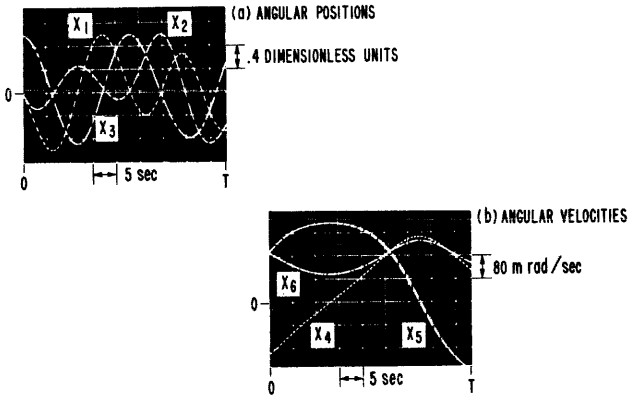


Figure 6 – Time history of state vector; no control; initial alinement of axes

B. Optimal Proportional Control – The time history solutions for the optimal proportional control derived in Reference 5 (as discussed above) are given in Figure 7 for the same initial condition as with no control (see Table I). As is well known, optimizing system performance under the assumption of proportional control often leads to impulsive-type control.

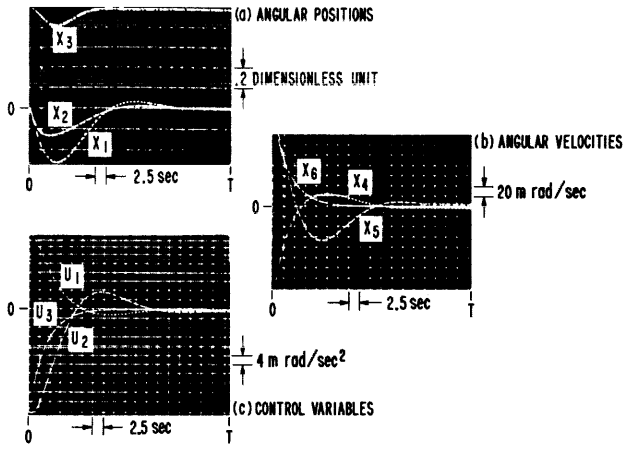


Figure 7 – Time histories of state and control vectors; optimal proportional control; initial alinement of axes

This tendency can be seen in Figure 7(c) where the initial torque acceleration rises to 44 mrad/sec² and then decreases relatively quickly to zero. A normalized fuel consumption as measured by the computer was $x_o(T) = 0.65$. This is nearly two and one half times the minimum theoretical value of 0.28 obtained by allowing an ideal impulse of torque.

In practice, the fuel requirements of 0.65 could only be attained with proportional control, i.e., without saturation of the torque. It was demonstrated in Reference 5 that the effect of saturation on the proportional controller is to increase both the consumed fuel

and the time required to accomplish the mission. However, the results given there were for initial velocities of the vehicle of only 4°/sec rather than the 10°/sec used herein.

C. Optimal Nonlinear Control—The solutions for the optimal nonlinear control obtained by implementing the random search algorithm are given in Figures 8 and 9. These figures correspond, respectively, to the cases indicated in Table I: initial alinement and no initial alinement.

The results given in Figure 8 are for initial alinement of axes in which the initial condition vector is x(o) = (0, 0, 1, −10, 10, 10), the same as for the other controllers discussed above. The solutions obtained are quire different from the previous cases as can be seen by comparing Figure 8 to Figures 6 and 7. It is noted that the maximum torque for optimal nonlinear control is approximately one-sixth the peak value for proportional control (note the different origins for the

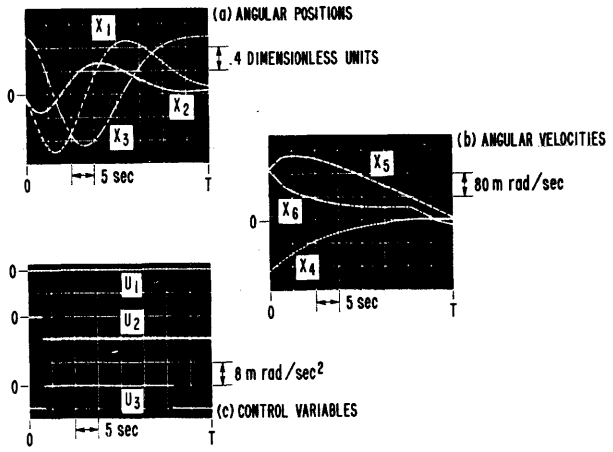


Figure 8—Time histories of state and control vectors; optimal nonlinear control; initial alinement of axes

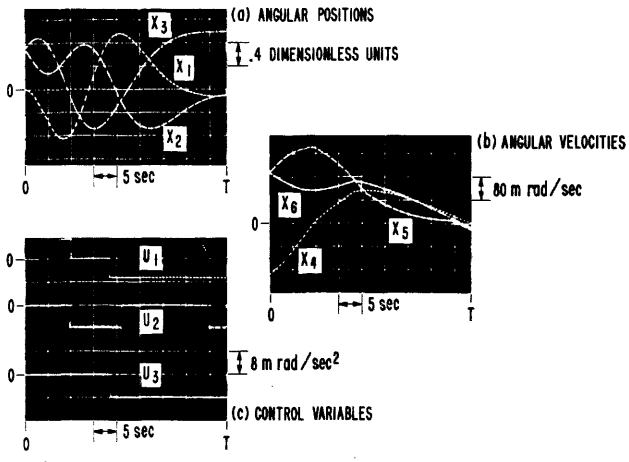


Figure 9—Time histories of state and control vectors; optimal nonlinear control; initial nonalinement of axes

three control functions). From careful examination of the time histories, it appears that the optimal control law is anticipatory, that is, it appears to act at the most advantageous moment in order to reduce the large excursions of the states to the desired end values. Fuel consumption was found to be $x_0(T) = 0.31$ which is only 1.1 times larger than for the optimal ideal impulse case. By comparison, the optimal proportional control is very inefficient. Proportional control uses 2.1 times the fuel for optimal nonlinear control. This factor would be even larger if the comparison were made to the more practical saturating proportional control system. Note also in Figure 8 that the operate times are nearly the same as for the proportional control system. Thus, we need not necessarily expect a time penalty for the practical constraint imposed by control saturation.

In the results given in Figure 9, the initial condition vector x(o) is not alined. The initial value is $x(o) = \left(0, \frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, -10, 10, 10\right)$ and corresponds to an initial rotation of b_3 of 45° in the S_2, S_3 plane. The time histories and optimal control thrust are seen to be similar to those with initial alinement. As for the fuel, we might expect that angular position error at the initial time would require a higher fuel consumption to obtain the same objectives. Indeed, experimental data show that fuel required is 0.43 which is 40 percent more than when the axes are initially alined.

Boundary cost function surfaces

It was pointed out previously that irregularity in the boundary cost function hypersurfaces reflects the need for a search algorithm that incorporates global as well as local-seeking properties. The highly irregular nature of these hypersurfaces for the attitude control problem is demonstrated in Figure 10. Here are shown typical two-dimensional cross sections through the hypersurfaces at a solution point. The curves were obtained by slowly varying one of the adjoint variables from −100 volts to +100 volts while all others were fixed at their respective solution values. Note in particular the irregular multipeak nature of the surfaces and the occasional noisy appearance. It is also worth noting the rather narrow valleys surrounding the optimum point. These surfaces clearly indicate the difficulties a deterministic method might encounter. For example, the gradient method would have ample opportunity of "hanging up" in the wrong valley. Furthermore, the two-dimensional curves give only a hint of the difficulties to be expected in the actual six-dimensional space.

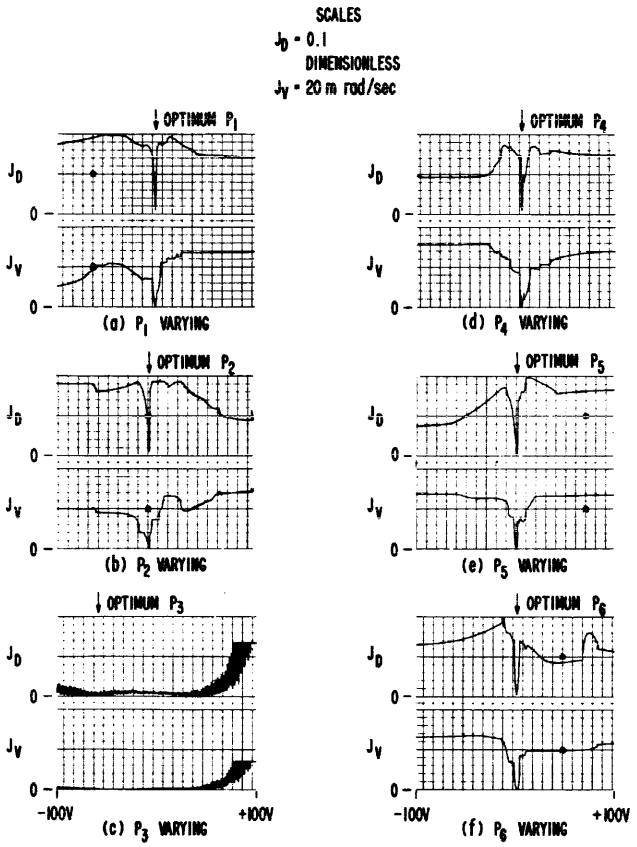


Figure 10—Example two-dimensional cross sections of boundary hypersurfaces

CONCLUSION

It was demonstrated in Reference 4 that the random search technique gave a practical approach to generating explicit optimal solutions for a moderately complex example problem under a wide range of conditions. An objective of the present study was to demonstrate the feasibility of using the random search technique in a different and more complex situation. The results presented in this paper gave a positive indication of that feasibility and together with Reference 4 suggest that possibly many high-order nonlinear problems will have complicated mappings and that the random search methods would be highly successful in coping with them. Furthermore, the results of this paper show quite explicitly what gains in system performance can be made in a given situation by using rigorously derived optimal control compared to a more conventional "optimal" control procedure. Such knowledge gives greater impetus to seek efficient ways of implementing the optimal nonlinear control. The open-loop aspect of the controls generated in this manner still limits any on-line capability. However, it is believed that off-line preliminary design capability for studying moderately complex nonlinear dynamical systems could be enhanced by this approach.

REFERENCES

1 D C KARNOPP
Random search techniques for optimization problems
Automatica vol 1 Pergamon Press 1963 pp 111-121

2 I N BOCHAROV A A FELDBAUM
Automatic optimizer for the search for the smallest of several minima
Avtomatika i Telemekhanika vol 23 no 3 March 1962

3 R R FAVREAU R G FRANKS
Statistical optimization
Proceedings 2nd International Analog Computer Conference 1958

4 E C STEWART W P KAVANAUGH D H BROCKER
Study of a global search algorithm for optimal control
Fifth Congress of International Association for Analog Computation Lausanne Switzerland August 1967

5 A SABROFF R FARRENHOPF A FREW M GRAN
Investigation of the acquisition problem in satellite attitude control
Wright-Patterson Technical Report AFFDL-TR-65-115 December 1965

Evaluation and development techniques for computer assisted instruction programs*

by M. TARTER, T. S. HAUSER, and R. L. HOLCOMB
Department of Biostatistics
The University of Michigan
Ann Arbor, Michigan

INTRODUCTION

Computer-Assisted Instruction presents the individual conducting educational research with the potential for more controlled and reliable experimental design and data collection. Certain features of programmed instruction also present new *challenges* to the researcher. Chief among these is a problem related to the individualization of programmed instruction. The authors have developed automated experimental design routines which are used to partially overcome these problems. This procedure will be referred to as "Computer-Assisted Pairing" or CAP.

An individual student's response to instruction is greatly affected by his background. Although it is possible to use such sophisticated statistical techniques as Analysis of Covariance to adjust for background differences, most educational studies rely on the simple statistical device of "paired-comparisons."

To accomplish this pairing, a pretest is usually administered to a group of students and, on the basis of the pretest results, the students are allocated to pairs. After N/2 pairs are formed a student within each pair is randomly assigned to the experimental group and his counterpart is assigned to the control group. At this stage the treatment, e.g., instruction, is administered to the proper group of students and the "placebo" to the counterpart group. A post-test is then administered to all students, and the differences $\{d_i\}$ between the ith pair of treated and untreated students' scores are recorded. The results can be analyzed by means of a simple Student-t test or a more sophisticated analysis can be conducted. (For example, if two separate post-tests are administered, a Hotelling T-test can be used to handle the resulting multivariate data.)

The above requires a rather idealized experimental environment. Such standard obstacles to straightforward analysis as missing or incorrectly recorded observations are more than likely to occur, especially in classroom teaching situations. However, aside from problems of data-collection, there is one major obstacle to the experimental design that is likely to interfere with the progress of the paired-comparison experiments in CAI research. It may not be possible to assemble and pretest all N students before administering the instruction sequence to the treatment pair members.

This problem is much more likely to occur in CAI studies than in research involving conventional classroom instruction. Since one of the primary advantages of programmed instruction is that the student can choose his own time of instruction, it would be artificial to require *all* N students to have completed a pretest before any student is allowed to proceed to the treatment, i.e., instructional sequence. In fact, an ideal experimental situation would keep information about the separation of pretest and treatment instruction from the student. In this way a single-blind, or even double-blind experiment could be conducted where the student need not know whether he is being given "treatment" or "placebo" instruction and the researcher need know only after the experiment has been completed.

Naturally the branching feature available in any of today's CAI languages could be used to administer such a double-blind study. However, the problem remains: How can at least a part of the desirable features of pairing be retained in the process of conducting a double-blind experiment, with treatment or placebo administered immediately after pretest completion. In other words, how can the pretest score of the Ith student be used immediately (before the remaining N-I students are tested) to branch the student to a treatment or placebo sequence.

*Research sponsored by the Public Health Services Biostatistics Training Grant NIH 5 T01 GM 00045-11, NIH GRS 67 SPH Project 78, and Hill Rhodes No. 9.

Of course any technique which pairs in "real time" rather than after the entire sample has been examined will be less efficient than the conventional method. In a sense, the convenience and advantage of using a double-blind experiment will be purchased at the cost of greater variance of the differences $\{d_i\}$. The best method of real time pairing, therefore, will increase the variance of the $\{d_i\}$ least. A considerable amount of theoretical research has yet to be done on the method proposed in this paper, in terms of variance reduction. However, CAP seems to be a reasonable approach and there is evidence to show that it is far superior to the one alternative method which has been applied.

The alternative pairing method

The alternative method is implemented as follows: The pretest is administered to each of the first N/2 (N even) students and each of these students is assigned at random to either the treatment or placebo group. The [(N/2) + 1]st student is then given the pretest and paired with the previously tested student whose pretest score is most similar; say the Ith student. If the Ith student was assigned to the treatment group, the [(N/2) + 1]st student is assigned to the control group and vice versa. The pretest is then administered to the [(N/2) + 2]nd student. The scores of all previously tested students, with the exception of students [(N/2) + 1] and I, are compared to the [(N/2) + 2]nd student's score. The student whose score is most similar to that of the [(N/2) + 2]nd student is then paired and assigned to the opposite group.

The disadvantage of this simple procedure is that the decision to pair is left to the last possible point, i.e., when half of the group is already allocated to the experimental or control group. Thus, there is no opportunity for two students among the first (N/2) to be paired with each other. It might also be noted that this design is particularly sensitive to non-random ordering of the students. For example, consider the case where students who present themselves for instruction early in the semester tend to be better students than those who use the teaching technique later (after conventional course exam grades are returned). In this situation, the simple sequential pairing method will tend to allocate a good and a poor student to the same pair which defeats the very purpose of pairing.

In the remainder of this paper, the group of students who have been tested and allocated to the treatment or control group (but not paired) will be referred to as the "pool." In the previous example, the "pool" is filled to the largest possible extent before being emptied, i.e., its members paired. The sequential pairing method proposed in this paper allows students to be removed from the pool at each stage rather than waiting until the pool is filled to capacity and then emptying it.

The CAP algorithm

The new pairing method will be referred to as the CAP algorithm because the implementation almost certainly requires high speed digital computation. Since in CAI investigations the pretest will probably be administered by console presentation, the same program which administers the pretest can be used to store previous students' pretest scores and implement the CAP algorithm.

The CAP algorithm can be divided into two parts. First, the main routine uses the Ith student's transformed score and decides whether he should be paired or entered into the pool. Second, a sub-program determines the appropriate transformation and applies it to the Ith student's score and the score of all other students within the pool.

The reason why "transformed scores" are necessary will become clear when the statistics underlying the CAP algorithm are discussed. Temporarily, the assumption will be made that the distribution of pretest scores is uniform on the interval [0,1]. This can be represented mathematically by the statement that the distribution f(x) of the pretest scores equals:

$$f(x) = I_{[0,1]}(x)$$

where $I_{[0,1]}(x)$ is the indicator function

$$I_{[a,b]}(x) = 1 \text{ if } a \leq x \leq b$$

$$= 0 \text{ otherwise}$$

(If I(a) is to be 1 and I(b) zero this function is written $I_{[a,b)}(x)$.)

The CAP main program performs the following operations: At the Ith insertion, i.e., immediately after the Ith student's prestest score Y[I], is available, a pairing criterion DEL is computed. The score within the pool that is closest to Y[I] is determined. Let the score closest to Y[I] be denoted by Y[MIN]. The expression $Z = |Y[MIN] - Y[I]|$ is calculated and if $Z \leq DEL$, then the MINth and Ith students are paired. If the MINth student had been assigned to the treatment group, the Ith student is assigned to placebo group and vice versa. If $Z > DEL$, then the Ith student is randomly assigned to either the treatment or control group and added to the pool. Naturally, if the pool size equals the number of students still untested at the Ith insertion then the [I + 1]st through Nth students are all paired. This is equivalent to setting DEL = 1 whenever the pool size equals N-1.

The most important part of the main program is naturally the computation of DEL. When only early scores are available, DEL should be small since there are at this stage many possible future scores, Y[J], which might satisfy the inequality $|Y[MIN] - Y[I]| \geq |Y[J] - Y[I]|$. Conversely when almost all students have been pretested, DEL should be large since there are fewer scores that can occur between Y[MIN and Y[I].

The "probability of mis-pairing," which will be symbolized by A, was used as a criterion for the determination of DEL. Two scores can be said to be mispaired at Stage I if one of the N-I "future" scores occurs between Y[I] and the MINth pool member paired with Y[I]. Now after the Ith insertion, what is the probability that one of the N-I remaining scores will occur in the interval beginning with Y[I] and ending with Y[MIN] (or beginning with Y[MIN] and ending with Y[I])? If the remaining N-I scores can be assumed to be independent and uniformly distributed on the interval [0,1] then the probability of mis-paring is exactly

$$A = 1 - (1 - |Y(I) - Y(MIN)|)^{N-I}.$$

By solving the above for $|Y(I) - Y(MIN)|$ one can determine that DEL is related to A by the function

$$DEL = 1 - (1 - A)^{1/(N-I)}.$$

If an allowance is made for a high probability of mispairing, large A, then DEL will be close to one. On the other hand, if one chooses a small probability A then DEL will be small. However, if A and consequently DEL are assigned small values then most students will remain unpaired until the pool size equals N-I, at which point DEL must suddenly become equal to −one. Therefore, the efficient implementation of the CAP main program depends upon some reasonable determination of A and hence DEL. To accomplish this the probability A is considered as a parameter, related only to the sample size N and constant throughout the pairing process. Under this assumption, the pairing criterion DEL is related to I through the formula

$$DEL = 1 - (1 - A)^{1/(N-I)}$$

where now both N and A are considered as parameters.

By use of the function given in the preceding paragraph the problem of finding an efficient pairing criterion has been reduced to the problem of estimating the parameter A. An overall criterion of efficient pairing therefore, must be introduced and the parameter A estimated on the basis of this criterion. The total pair separation was chosen as this criterion. Separation in the case of the score Y[I] and its eventual mate Y[MIN] is defined as simply the distance

$$|Y[I] - Y[MIN]|.$$

Clearly the best sequential pairing method is the one which yields the total pair separation closest to the pair separation possible for the ordinary pairing situation. In the ordinary pairing situation, complete information—in the form of knowledge of all N scores, is available before any student is to be paired. The most efficient sequential pairing algorithm would be the one which best used the limited information available at the Ith stage, i.e., that obtained from the I previously measured scores.

The estimation of the parameter A is made in the following way. For a given sample size N, the estimate A is chosen which minimizes the total pair separation. Estimates of A have been obtained for various sample sizes and used in the CAP program.

A program was constructed which simulated the above pairing process, and, therefore, could be used to estimate A. The total pair separation was measured for repeated samples of size 50, 100, 200, and 250 where A was chosen as .05* K,K = 0,... 18. For all samples the optimal value of A was found to be surprisingly large. Since the alternative method of sequential pairing, which was described earlier, is a special case of the CAP procedure where DEL = 0 for all I from 1 to N/2 and DEL = 1 for all I from N/2 + 1 to N, the observation that A and hence DEL should be large for small values of I tends to show that the CAP procedure greatly improves upon the alternative sequential pairing method. In fact for all sample sizes, the CAP procedure tends to reduce the total pair separation by a factor of at least two. In Table I the estimated optimal values for A are given, as well as two other estimates which are of interest. These are: First, the size of the pool. NI, when NI = N − I, i.e., at the time when DEL is set equal to one and all subsequent scores are paired. Second, the number of times NI = 0, i.e., the pool is depleted during the pairing process. Obviously, if the pool is depleted immediately before any stage I other than N then the Ith score must be entered into the pool.

Theoretical and simulation work has shown that the CAP main program provides a substantial improvement over the alternative simple sequential pairing method. Actual trials with real data are currently being conducted to check the implementation of the CAP technique.

One of the reasons trials with actual data are needed to test the efficiency of CAP is that the CAP main program requires the assumption that the pretest scores are uniformly distributed. Since this is obvious-

TABLE I – Pairing with and without optimal A

Sample Size N	Optimal A	Expected Pool Size When Pool Must be Emptied	Expected No. of Times Pool is Emptied	Total Separation Using CAP with Optimal A	Total Separation Using Alternative Pairing Method (A = 0)
50	.55	4.55	.200	1.35	2.64
100	.60	3.80	.250	1.54	4.02
150	.65	2.95	.550	1.68	5.01
200	.75	1.55	1.00	1.92	5.01

ly a very restrictive assumption, a CAP subroutine is used to preprocess the Ith and all scores within the pool as soon as the Ith pretest score is available. The remainder of this section will deal with the theory and implementation of this transform subroutine.

The transform subroutine

Let the sequence of N random variates, identically distributed with density function f(x), be represented by X[I],...,X[N]. Let the function F(x) represent the cumulative distribution function associated with density f(x). The random variables, F(X[1]), F(X[2]),..., F(X[N]) are uniformly distributed on the interval [0,1]. Therefore, the problem of transforming the pretest scores to suit the requirements of the CAP main program is related to the problem of cumulative distribution function estimation.

The sample cumulative or step function F*(x) would in ordinary circumstances be considered a good estimate of F(x). However, for the purposes of CAP preprocessing, it is a poor estimate. The step function F*(x) equals

$$F^*(x) = (1/n) \sum_{i=1}^{n} [I_{(X_{i,b})}(x) + (1/2)\, I_{[X_i,X_i]}(x)]$$

where X_i represents an arbitrary sample point. F*(x) is not a smooth or differentiable function. Also, since 2nF*(x) must be an integer for every value of x, F*(x) would distort the "local spacing" of the transformed values. Since in the CAP main program the spacing of consecutive points is particularly important, it is obvious that F*(x) is not a suitable transformation to uniformity.

What is required is a smoother estimate of F(x) which can be updated easily as new pretest scores are obtained. An estimate of F(x) which not only fulfills the above two requirements, but also is more efficient than $\hat{F}^*(x)$ has been investigated by two of the authors[1,2] and a few of its properties will be reviewed in this paper. This estimate will be represented as $\hat{F}_m(x)$ where

$$\hat{F}_m(x) = \frac{x-a}{b-a} + \sum_{k=1}^{m} \frac{(b-a)\,\bar{c}_k}{\pi k} \sin k\pi\left(\frac{x-a}{b-a}\right)$$

$$\bar{c}_k = \frac{2}{(b-a)n} \sum_{i=1}^{n} \cos\frac{k\pi(X_i-a)}{(b-a)}\, I_{[a,b]}(X_i)$$

and n represents the number of data points $X_1,...,X_i,...,X_n$, and "a" and "b" are two predetermined constants, preferably such that for most X_i the inequality $a \leq X_i \leq b$ will be satisfied. It is shown[2] that as m approaches infinity $\hat{F}_m(x) \rightarrow \hat{F}^*(x)$. Also for all densities with bounded variation, e.g., all continuous distributions commonly encountered in statistical research (the Normal, Cauchy, Laplace, Gamma, and Logistic) $\hat{F}_m(x)$ is a more efficient estimate than $\hat{F}^*(x)$. Here efficiency is measured in terms of Mean Integrated Square error $J(\hat{F}_m)$ where

$$J(\hat{F}_m) = E\int_a^b |F(x) - \hat{F}_m(x)|^2\, dx.$$

In Ref. 2 it is also shown that the constant m associated with the most efficient estimator of the form $\hat{F}_m(x)$ is usually less than 10. Consequently $\hat{F}_m(x)$ provides both an easily computed and a smooth estimate of $\hat{F}(x)$ and $\hat{F}_m(x)$ is actually more efficient than $\hat{F}^*(x)$. In Ref. 1 and 2, a rule for determining the optimal value of m is given. This stopping rule is based on an unbiased estimator of Mean Integrated Square Error $J(\hat{F}_m)$. Also[1] a computation scheme is given which allows the constants $\bar{c}_k$ of $\hat{F}_m(x)$ to be computed recursively. Since the estimator of $\hat{F}(x)$ should be revised after each new pretest score becomes available, the recursive computation of the $\bar{c}_k$ and hence $\hat{F}_m(x)$ represents a considerable saving in terms of computer time.

Implementation of CAP

The following is a brief outline of the implementation of CAP:

A. Since it is likely that no *a priori* information about the form of $\hat{F}(x)$ will be available, the first 20 students are added to the pool and assigned at random to the treatment or placebo group.
B. Using the previous 20 as well as the 21st pretest score the transformation $\hat{F}_m(x)$ is determined. The procedures used to compute $\bar{c}_k$ and the stopping rule for determination of m are given in Ref. 1.

C. The original 21 pretest scores
$$X[1], X[2], \ldots, X[21]$$
are transformed by means of $\hat{F}_m(x)$ to
$$Y[1], Y[2], \ldots, Y[21].$$
D. The score Y[MIN] is determined where
$$|Y[MIN] - Y[21]| \leqq |Y[J] - Y[21]|$$
for all J from 1 to 20.
E. An estimate of A has been read into the program to suit the eventual sample size N of this particular CAI experiment. The pairing criterion DEL is computed
$$DEL = 1 - (1 - A)^{1/(N-21)}.$$
F. If
$$|Y[MIN] - Y[21]| \leqslant DEL,$$
the 21st and the MINth students are paired and the 21st student is assigned to the treatment group if the MINth student is assigned to the placebo group or vice versa.
G. If
$$|Y[MIN] - Y[21]| > DEL$$
the pool size is increased to 21 and the 21st student is randomly assigned to either the treatment or placebo group.
H. This process is repeated as the 22nd through Nth student's pretest scores become available. However,
   1. For each new score, the constants $\bar{c}_k$ are updated and a new value of the transform Y[I] calculated for each student in the pool.
   2. If at any time before the last pair is formed, the pool is emptied, the next student is entered into the pool. (Equivalently, DEL is set equal to 0.)
   3. When the pool size equals the number of students still to be tested all subsequent student are paired with their closest counterparts within the pool. (Equivalently DEL is set equal to 1.)

By following steps A through H, each student within every pair has been randomly assigned to the treatment or placebo group. Also each student, i.e., the Ith, is available for the treatment even though the pretest has yet to be administered to N-I students.

The construction of CAI programs

Up to this point a method for conducting a CAI experiment has been described, but no comment has been made concerning the source of data for such an experiment. In this section a brief description will be given of a particular CAI project and the process of program development rather than evaluation will be emphasized.

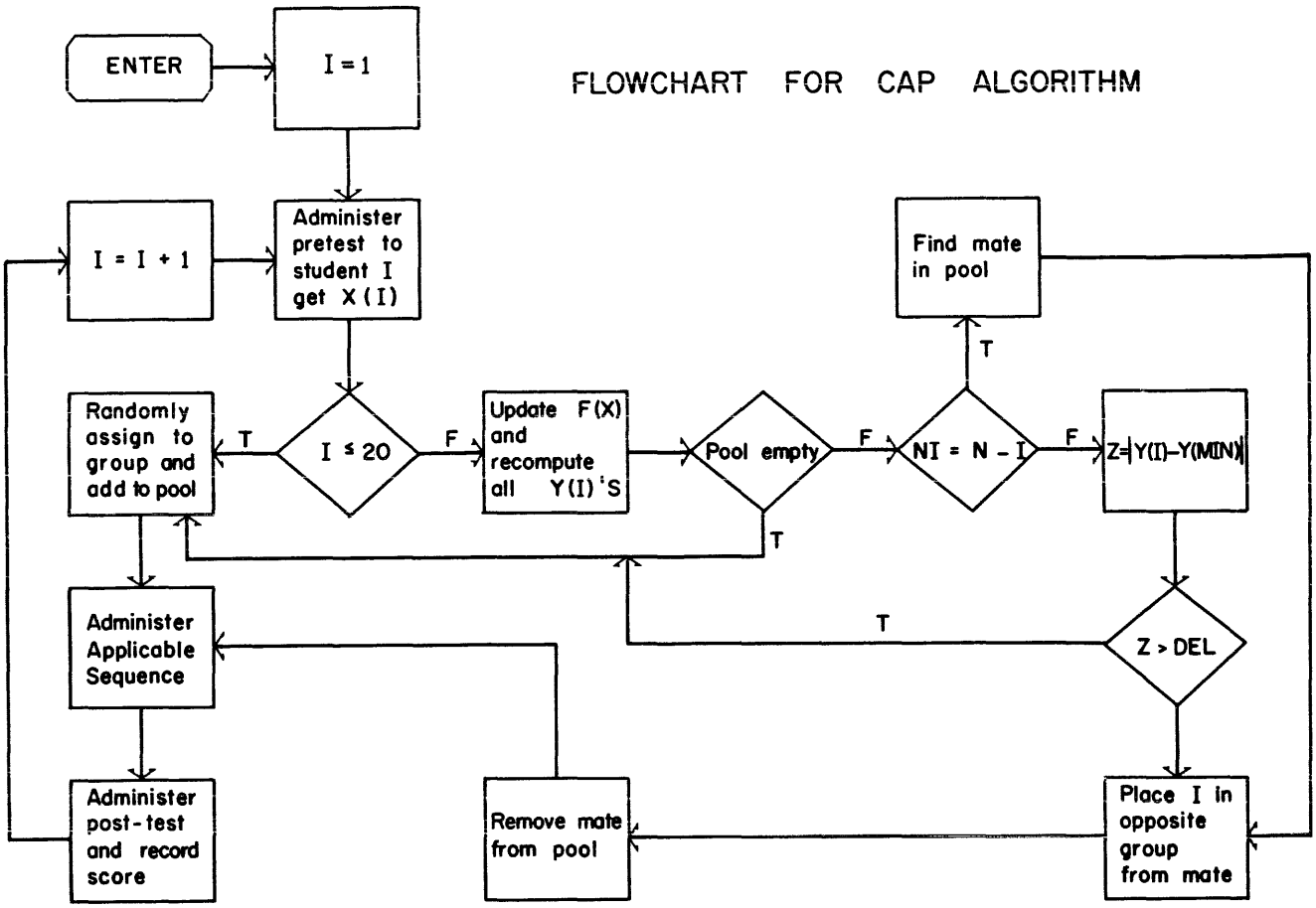
For the past two years the School of Public Health at The University of Michigan has been conducting an extensive experiment on the effect of Computer-Assisted Instruction within a section of a large university. This project has already generated ten programs of more than intermediate size, although much data has yet to be gathered before any final conclusions can be announced. Programs have been written in such diverse areas as Biostatistics, Epidemiology, Environmental Health, Public Health Dentistry, Public Health Education, and Industrial Toxicology. Several different procedures have been used to construct CAI programs and, therefore, our observations about these procedures may be of value to other workers in this field. The construction of CAI programs is an expensive process at this early stage of hardware development and our observations may suggest shortcuts and point out pitfalls.

The best place to start when discussing CAI program construction procedures is with the personnel who actually participate in the construction process. Four categories can be listed.

A. The person who will actually be responsible for the implementation of the completed programs and who initiates the construction process. At the University this will usually be the professor who wishes to use the material in one of his classes.
B. Subject matter oriented staff working under the supervision of the person in category A. Here the term "subject matter oriented" is used to distinguish this type of person from CAI programmers.
C. CAI programmers – with experience and training in education or psychology, but with little background in the specific subject matter areas to be programmed.
D. Teaching assistants, research fellows, trainees, and others, who might be called the transient work force.

A brief description of the process itself follows. The list given below is, of course, a highly idealized one. However, like any other form of computer programming there are definite clear-cut stages, e.g., flowcharting, coding, debugging, which must be carried out before a satisfactory working program is obtained.

1. An initial decision about the subject matter to be taught must be made.
2. The level of sophistication of the students who will take this program should be determined.
3. Should remedial sections be provided?
4. If yes to No. 3, how elementary should the material in the remedial sections be?
5. Should advanced sections be provided for the brighter students?



6. If yes to No. 5, how advanced should these sections be?
7. A list of the specific concepts that must be presented in a teaching program on this subject should be constructed.
8. The sequential order in which these concepts should be presented must be determined.
9. At least one question for each concept or fact that tests the understanding of the students must be written.
10. Information that must be presented to the student along with each concept that is presented must be determined.
11. A list of typical misconceptions by students should be made.
12. Constructive responses that would correct these misconceptions should be listed.
13. At least two general constructive comments to be presented to the students who respond with an answer that was not anticipated should be written.
14. Pictures or graphs that may be helpful to the students in the understanding of the concepts or facts presented should be obtained.
15. Appropriate use of slides, tapes and typeouts should be determined.
16. The general flow the program is going to follow should be decided upon.
17. The prepared materials in the computer language used must be programmed, i.e., coded.
18. The program must be entered into the computer.
19. The coding should be debugged.
20. The program should be tested (using students) to determine if it actually teaches.
21. Appropriate content revisions indicated by early student testing should be made.
22. Observations should be made of the perfor-

mance of at least ten students as they take the program and their comments and questions should be noted.

23. Further changes as indicated from students' reactions to the program should be made.

A number of methods have been used in writing programs in the School of Public Health. All of the methods involved the 23 steps described above.

The most successful of the alternatives we have tried involves the professor (A), a graduate student or staff member (B or D), and the programmer (C). The professor and his assistant (A and B, or D) completes steps 1-8 (the general outline of content materials to be used with a specification of the desired upper and lower limits of the materials). The student carries through steps 9-15 and discusses step 16 (general flow of the program with the possibilities of branching) with the programmer (C). The programmer handles steps 17-19. Then the responsibility is again assumed by the student and the professor for steps 20-23. Throughout this period, channels of communication have been established between the three people involved. When the work is distributed in this manner, all concerned seem to find the time and maintain interest in the program. The programmer is available for consultation in regard to possible uses of the computer, and the graduate student can solve the majority of the content problems on his own.

One problem we have found in implementing this method has been a lack of interest on the part of some graduate students. They have felt the typical pressures for research as opposed to teaching as a prerequisite for advancement within their own fields. The attitude of graduate students in general toward teaching was often a negative one. However, we have found that certain students who plan to teach in the future take this opportunity to develop teaching materials very seriously. This is also noted in Reference 3. They learn how important it is to break topics into small sections and sequence them in a logical order. They are often more strongly motivated after they observe students taking their programs. Frequently advanced graduate students (D) or professional staff (B) do not realize how a misplaced fact or lack of information can lead to miscomceptions on the part of the learner who is not familiar with a subject.

Our experiences lead us to believe that CAI programs must be written with full cooperation and communication between the professor (who devotes as much time as possible) and the programmer. To save the professor's time a student or staff assistant who is familiar with the subject matter and the programmer, carries through several of the 23 steps. Very few of the professors at the University of Michigan, School of Public Health have had the time needed for optimal participation in a project of this type. Therefore, the professor-student or staff assistant-programmer combination is usually the most feasible. This method also enables him to participate with a number of professor-student pairs. The quantity and quality of programs written is this way in our opinion represents a great improvement over other methods attempted. This statement, of course, is now being rigorously verified using CAP in conjunction with other evaluation procedures.

CONCLUSIONS

Computer-Assisted Pairing is a dynamic design for paired comparison evaluation of Computer-Assisted-Instruction sequences. It appears to be of considerable practical value since the pretest-treatment-posttest sequence can be made invisible to the subject. Simulation studies have shown CAP to be substantially superior to previously considered alternatives.

The application of the technique is certainly not limited to Computer-Assisted-Instruction. CAP can be applied fruitfully to almost any experimental situation where paired comparisons are needed. It is especially useful when task initiation and evaluation can be done by the computer.

ACKNOWLEDGMENT

The authors gratefully acknowledge the advice and support of Drs. Karl Zinn and Stanford Ericksen of the Center for Research on Learning and Teaching, University of Michigan and Associate Dean John Romani of the University of Michigan School of Public Health. Also we would like to thank Miss Wendy Hiller for help in checking the final manuscript of this paper.

REFERENCES

1 M TARTER R) HOLCOMB R KRONMAL
A description of new computer methods for estimating of the density of stored data
Proceedings 1967 ACM National Conference 511:9 1967

2 R KRONMAL M TARTER
The estimation of probability densities and cumulatives by Fourier series methods
Accepted for publication Jour Am Stat Assoc

3 M TARTER
Programmed instruction in statistics from the professor's point of view
The American Statistician 21:28-31 1967

4 V CLARK M TARTER
Preparation for basic statistics automated text
McGraw Hill 1967

5 J E COULSON
Programmed learning and computer-based instruction
Wiley 1962

6 W FEURZEIG
A conversational teaching machine
Datamation 10:6 1964
7 K L ZINN
Survey of materials prepared for instruction or research on instruction in on-line computer systems
Automated Education Handbook E Goodman (Ed) Detroit Automated Education Center 1965

Computer capacity trends and order-delivery lags, 1961-1967

by MICHAEL H. BALLOT and KENNETH E. KNIGHT
Stanford University
Stanford, California

INTRODUCTION

This paper examines the growth of computational facilities in the U.S. from the end of 1961 to September of 1967, exploring the dynamics of the growth process and attempting to link it to specific market events. The dynamics of supply and demand for general purpose computational capability points to many problems for both producer and user. This paper considers one of these pertinent to the ordering policy and planning of the firm seeking to acquire EDP equipment for, say, systems conversion; the delivery lag, or average time between equipment orders and delivery. This lag is studied and estimated using two separate empirical models.

Growth in computers

The tremendous growth of computers in this country, and abroad, is a well-known and accepted fact of our times, often referred to as the "Computer Age." A measure of this growth in recent years, based on the Monthly Computer Census reported in *Computers and Automation,* shows a definite exponential trend. The curves, fitted to data compiled for the cumulative number of machines installed and on order, are shown for three grouping: IBM, the *"Big 8"** (Burroughs, CDC, G. E., Honeywell, IBM, NCR, RCA, and Univac), and *All* companies as reported in the *Computers and Automation* Monthly Census (The "Big 8," Autonetics, Bunker-Ramo, Data Machines, DEC, Electronic Associates, EMR Computer Division, Philco, Raytheon, SCC, SDS, and Systems Engineering Labs). [See Figures 1, 2, and 3.]

For *cumulative machines installed,* the following equations, in logarithmic form, were estimated:

(1) for the number of machines installed, IBM,

$$\ln N_t^{in} = 3.362 + .0761t^{*} \quad , R^2 = .959;$$
$$(.00351)^{**}$$

(2) for the number of machines installed, the "Big 8,"

$$\ln N_t^{in} = 3.597 + .0781t \quad , R^2 = .989; \text{ and}$$
$$(.00188)$$

(3) for the number of machines installed, "All,"

$$\ln N_t^{in} = 4.233 + .0717t \quad , R^2 = .996.$$
$$(.00107)$$

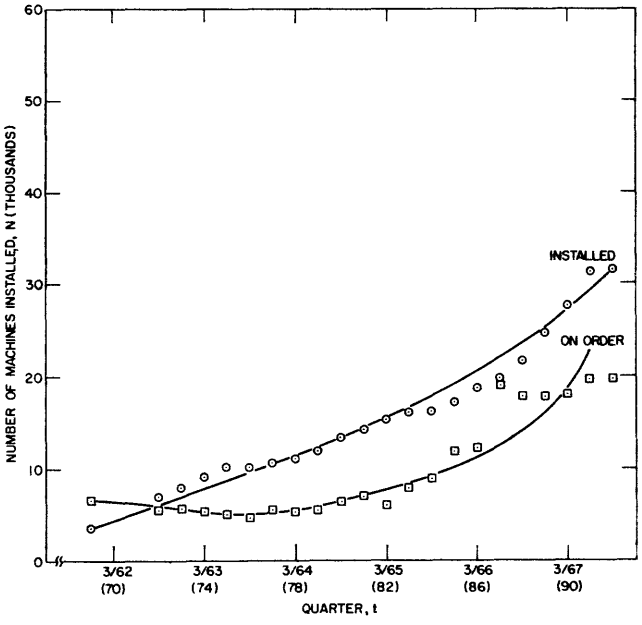


Figure 1 – Time series, machines, IBM

*Inclusion in this group was determined by the number of machines installed and on order and the average value of these.

*t, the time in quarters, is measured from 9/44 (t = 0), the date of introduction of the first general purpose digital computer, the Harvard Mark I.

**This is the standard error of the regression coefficients, s_b.

The curves fitted to *cumulative machines on order,* in logarithmic form, are as follows:

(1) for the number of machines on order, IBM,

$$\ln N_t^{o/o} = 3.347 + .0704t \quad , R^2 = .770; \\ (.00861)$$

(2) for the number of machines on order, the "Big 8,"

$$\ln N_t^{o/o} = 4.264 + .0631t \quad , R^2 = .843; \text{ and} \\ (.00609)$$

(3) for the number of machines on order, "All,"

$$\ln N_t^{o/o} = 4.365 + .0626t \quad , R^2 = .874. \\ (.00532)$$

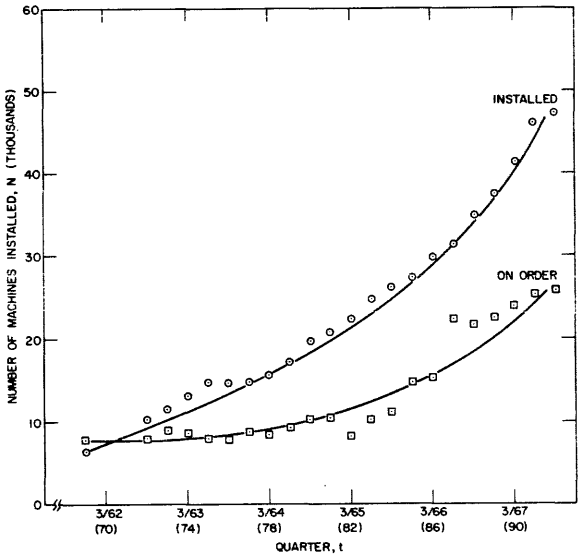


Figure 2 – Time series, machines, "Big 8"

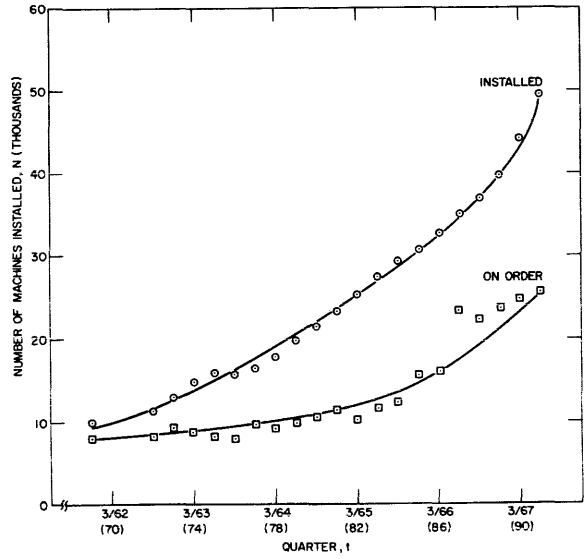


Figure 3 – Time series, machines, "ALL"

The results are gratifying, with high statistical significance and high correlation. The three groups show *quarterly growth in facilities of better than 7%,* with the growth of those of the "Big 8" manufacturers as high as 7.8%. These general results apply also to the curves fitted to the number of machines on order, with quarterly growth percentages about a point lower.

Tests were applied to the differences in these trends,* and only those of "All" *vs* the "Big 8," for machines installed, showed a statistically significant difference (at better than the 1% level). Tests were also applied to each group, comparing the trends for machines installed *vs* machines on order; again, only one comparison was statistically significant (at 5% or better), this time the installed and on order trends for the "Big 8."

Thus, most conclusions that are drawn from inspection of the results, intergroup and between installations and orders, can be only looked on as indicative. These indicate that new installations are growing faster than orders. This implies, given a constant machine mix in production, a small decrease in delivery lag. Further inspection of the statistics show that the "Big 8" (1.5% difference) and all companies reported (.9% difference) seem to be working appreciably on their backlogs, while IBM lags in this sense. These results can be interpreted as indicating that order backlogs are growing faster for IBM than for the other groups. This interpretation would be compatible with the success of the System/360 in the market coupled with the lag of IBM's production schedules for this system.* One factor needing amplification is the seemingly lower growth of installations of "All" compared with the "Big 8"; this may be explained by the decreasing percentage of the market for machines manufactured by the small-volume producers. The preceding observations, as a basis for analysis of computer growth and production lags seem, in general, valid but incomplete.

To further explore the growth in computational capability, quarterly time series for cumulative machine power** were constructed for IBM and the "Big 8", and exponential curves were fitted to these [see Figures 4 and 5]:

(1) for cumulative power installed, IBM,

$$\ln P_t^{in} = -6.88 + .140t \quad , R^2 = .934; \\ (.00848)$$

(2) for cumulative power installed, "Big 8,"

*See [1], p. IV - 19.

*See [2], pp. 138 *et. passim.*

**See [3] pp. 40-54, and See [4], pp. 31-35.

$$\ln P_t^{in} = -8.95 + .172t \quad , R^2 = .991; \\ (.00375)$$

(3) for cumulative power on order, IBM,

$$\ln P_t^{o/o} = -14.7 \ .249t \ , R^2 = .835; \text{ and} \\ (.0254)$$

(4) for cumulative power on order, "Big 8,"

$$\ln P_t^{o/o} = -13.3 + .235t \ , R^2 = 873. \\ (.0206)$$

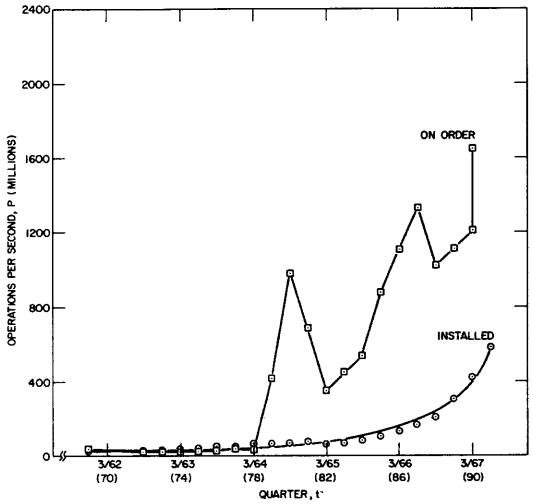


Figure 4–Time series, computational power, IBM

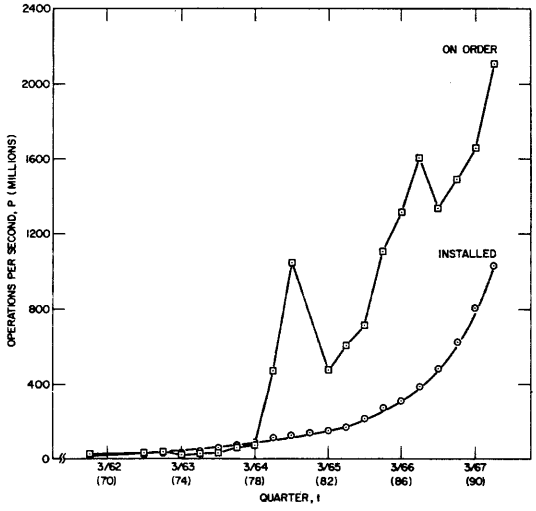
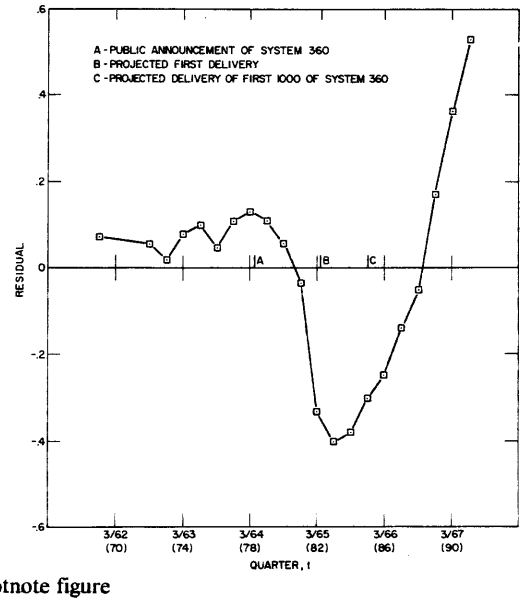


Figure 5–Time series, computational power, "Big 8"

Tests on trend differences were carried out on these results, and the difference between groups for the cumulative power installed, 3.2%, was statistically significant at better than the 1% level. We compared the power time series results for IBM and the "Big 8" with the machine time series for these two. The similarity between groups is evidenced in both, but the comparison of installations and orders shows opposite differences; that is, power orders seem to be advancing at a greater quarterly rate than installed power while the opposite is true for the machine series. The backlog of computational power is growing at about an 11% differential per quarter for IBM and about 6% for the "Big 8" (probably mostly due to IBM inclusion in this group). This trend can be credited to the large and powerful third generation computers, being ordered at an accelerating rate and being supplied with a considerable production lag. The results seem to further accentuate the premise that IBM is (or was)* having production difficulties with its new series. The growing orders for IBM computational power (at a quarterly rate of nearly 25%, almost 11% ahead of deliveries) is a testimony to the advanced selling of System/360, and, as previously asserted, is a production headache on a grand scale. The differential growth of computational power demanded for now and the future *vs* power supplied now is also evident in the statistics for the "Big 8." It should be remembered, however, that there is a damping effect due to the inclusive nature of this group. Further breakdown of the data may yield interesting inter-company results, but this is not the main purpose of this paper. Rather the preceding analysis is meant to lay the foundation for the study of the problem of delivery lags, as they effect the planning of the firm engaged in expansion replacement of its EDP facilities.



Footnote figure
Analysis of Residuals
$\ln P^{in} = \alpha + \beta$ t
for IBM

*Residuals ($\ln P_{obs} - \ln P_{calc.}$) of the regression (in logarithmic form) of $P^{in} = ce^{at}$ for IBM show steady increase at higher values of t. Observed values steadily diverge upward from the regression line for 12/66, 3/67 and 6/67, about one year after the first 1000 new System/360 computers were scheduled to be delivered. Graphically: (see footnote figure above).

Delivery lags

It has been conjectured that there exists a substantial lag between the ordering and installation of new and expanded computational equipment.* This section explores this problem and tries to estimate the lag for machines and for computational power. Two models are used: one assumes lags of 2, 3, 4, 5, 6, 7, and 8 periods (quarters) and tries to determine the best set of cumulative orders placed up to points in time within these periods to explain the growth of installations at the end of the period; the second assumes lags of 1 to 8 periods and tries to determine the best set of orders placed in single time periods during these over-all periods to explain the growth in installations at the end of the period under study.

A. Model 1

The first model used is of the form $\Delta X_t^{in} = f\left(\sum_{i=1}^{a} \Delta X_{t-i}^{o/o}\right)$ where a is the postulated delivery lag (2 to 8 quarters) and X is used to denote cumulative number of machine installations (N) or the sum of computational power (P), installed (in) and on order (o/o). Table below gives the best results for this linear formulation, for N & P with the various postulated lags, and with the constant suppressed as well as included.

TABLE I – Results, model 1

Group	*Best fit**	*F-ratio (or F_0)***	*R^2 (or R_0^2)***
IBM	$\Delta N_t^{in} = -1210 + .394\ N_{t-5}^{o/o}$ (.0372)	112.1	.911
	$\Delta N_t^{in} = .202\ N_{t-4}^{o/o}$ (.0119)	(286.8)	(.960)
Big 8	$\Delta N_t^{in} = -241 + .211\ N_{t-3}^{o/o}$ (.0213)	97.5	.890
	$\Delta N_t^{in} = .193\ N_{t-3}^{o/o}$ (.00781)	610.3	(.979)
All	$\Delta N_t^{in} = -255 + .205\ N_{t-3}^{o/o}$ (.0231)	78.4	.867
	$\Delta N_t^{in} = .187\ N_{t-3}^{o/o}$ (.00810)	(533.1)	(.976)
IBM	$\Delta P_t^{in} = -5.78 + .0894\ P_{t-4}^{o/o}$ (.0164)	29.5	.695
	$\Delta P_t^{in} = .0826\ P_{t-4}^{o/o}$ (.0190)	(56.9)	(.803)
Big 8	$\Delta P_t^{in} = -1.69 + .117\ P_{t-4}^{o/o}$ (.0156)	56.5	.813
	$\Delta P_t^{in} = .116\ P_{t-4}^{o/o}$ (.0101)	(132.3)	(.904)

*Best fit is determined by a combination of highest F-ratio, high R^2 and highly significant regression coefficients. Also note that the sample size drops from 17 for a = 2 to 11 for a = 8, for all regressions of the models.

**With the constant suppressed, all variances and correlations are computed about the origin rather than the mean in the BMD stepwise regression routine (BMD-02R). See 6.

*See [5]; January 1966, p. 17; May, 1966, p. 17; June, 1966, p. 135; and January, 1967, p. 125.

The regression results for machinery suggest that the best explanatory order variable for installations in period t is the machine orders accumulated up to and including period t-3, for All and the Big 8. For IBM, though, the lagged order variables are in the area of t-4 and t-5. This result thus is in line with earlier observations in Section I concerning IBM's faster growing order backlog. With a set at 3, 4, 5, 6, 7, and 8 periods, almost identical results were obtained for the regressions involving machines; this was also true for the power regressions with a at 4, 5, 6, 7 and 8 periods. The best fits for machines, as defined in Table I, were selected from seven sets of regression runs, and these were further narrowed down to the one representative equation for each group, as appears in the above tabulation. This selection was repeated for computational power, and these further results suggest that orders accumulated up to and including period t-4 are the best explanatory order variables for power installations in period t. This may well be due to the growing backlog of machine power for the groups under study. The production lag here, greater than that for machinery, is in line with prior thoughts and the time series data and discussion in the previous section.

The obvious problems in this regression model are auto-correlation and multi-collinearity. The first of these is most likely present in the model because of the influence of omitted variables going beyond the included periods and which are very probably serially correlated, affecting the assumed random disturbance term. The second problem is known to exist in the model from the correlation matrix given by the the program for all variables.* But both autocrrelation, by biasing and rendering less efficient $\hat{\alpha}$ and $\hat{\beta}_i$, and multicollinearity, by rendering the $\hat{\beta}_i$ nearly useless as a measure of relative effect, are not serious problems in this analysis. The model is not used structurally, but rather as an indicator of a single variable's explanatory power.

B. Model 2

The second model used is of the form $\Delta X^{in}_t = f\sum_{i=1}^{a} \Delta X^{o/o}$ with a again being the postulated delivery lag, varying from 1 to 8 quarters. The use of absolute first differences helps somewhat with the problems of autocorrelation and multicollinearity, drastically reducing the latter. Also, results can be more directly applied to the testing of the lag length, with support being derived from the lower limits suggested by the first model. The regressions were run for the three groups with machine numbers, and then two of these with computational power. The first set of results are shown in Table II.

These results, for number of machines, suggest that the lag may be in the neighborhood of 3 to 4 quarters, perhaps higher. (This is suggested by the occasional appearance in the regressions not shown here, of orders placed in period t-6, and the t-5 term in the second equation shown, as a third explanatory variable of some significance, plus the results of Model 1.) And, as was noted for the previous regressions, similar results were obtained for $a = 4$ to 8. Next is presented the results of the computational power relationship for IBM and the "Big 8," in Table III.

TABLE II – Results, model 2, machines

Group	*Best fit**	*F-ratio*	R^2
IBM	$\Delta N^{in}_t = 779 + \underset{(.0601)}{.289}\ \Delta N^{o/o}_{t-3} + \underset{(.0621)}{.476}\ \Delta N^{o/o}_{t-4}$	34.9	.853
Big 8	$\Delta N^{in}_t = 1610 + \underset{(.0406)}{.182}\ \Delta N^{o/o}_{t-3} + \underset{(.0466)}{.422}\ \Delta N^{o/o}_{t-4}$		
	$+ \underset{(.0588)}{.192}\ \Delta N^{o/o}_{t-5}$	35.0	.913
All	$\Delta N^{in}_t = 1670 + \underset{(.0652)}{.264}\ \Delta N^{o/o}_{t-3} + \underset{(.0658)}{.444}\ \Delta N^{o/o}_{t-4}$	27.8	.835

*See footnotes, Table I

*The small coefficient standard errors, though, might lead one to believe there is not high correlations amongst the explanatory variables, especially in the light of the not very large sample size. But the bane of time series is here too present.

TABLE III – Results, model 2, power

Group	*Best fit**	*F-ratio*	R^2
IBM	$\Delta P^{in}_t = 24.2 + \underset{(.0612)}{.0940}\ \Delta P^{o/o}_{t-4} + \underset{(.0625)}{.0866}\ \Delta P^{o/o}_{t-6}$	1.53	.234
Big 8	$\Delta P^{in}_t = 44.7 + \underset{(.0883)}{.111}\ \Delta P^{o/o}_{t-4} + \underset{(.0850)}{.0614}\ \Delta P^{o/o}_{t-5} + \underset{(.0896)}{.0913}\ \Delta P^{o/o}_{t-6}$	1.15	.278

*See footnote, Table I.

These results, showing extremely low correlation and not statistically significant, must be considered in the light of recent relatively heavy queue switching, with orders moved from the backlog of one computer manufacturer to another having smaller delivery lags in any one time period. Given the discontinuous nature of the power mix between the manufacturers included in the analysis, large power on order discrepancies from one period to the next would be quite common. The power on order series lacks any semblance of uniformity or direction. Thus, the results, though barely suggestive, do indicate that the delivery lags for power are 4 to 6 quarters. These longer lags, weighted by the larger machines, appear consistent with the results suggested in Model 1 and the time series for power (see Section I).

CONCLUSION

Exponential curves were fitted to time series data which depicted the growth of computational capability, as measured by number of machines and computing power installed and on order. For the first of these series, the growth of machines installed exceeded 7% per quarter, and orders, 6% per quarter. The second series for power showed trends in the area of 15% and 25% for installations and orders, respectively.

These lags present a vexing problem for the firm planning for future computer capability. For one plan to be brought to culmination, with the computer going on line, requires a 9 to 18 month delivery lag as well as a several month installation - on line lag. Then the new facility becomes obsolete economically due to the tremendous technological change in the computer manufacturing industry.* Clearly further research on this problem, involving the trade-off between economic costs and conversion expenses, is needed.

*See [3], p. 54. Knight find average advances in computing power, given the equivalent captial cost, greater than 80% per year, 1950-62. See [4], p. 34. Knight's updating, 1963-66, reveals an average advance of about 140%.

REFERENCES

1 E A McCRACKEN J M CARR JR
Statistical methods
ESSO Research Laboratories Humble Oil and Refining Company undated

2 T A WISE
The rocky road to the marketplace
Fortune October 1966

3 K E KNIGHT
Changes in computer performance
Datamation September 1966

4 K E KNIGHT
Evolving computer performance
Datamation January 1968

5 *Look ahead*
Datamation various issues

6 W J DIXON (editor)
Biomedical computer programs
Health Sciences Computing Facility UCLA 1965

7 J JOHNSTON
Econometric methods
McGraw-Hill Book Company Inc 1963

Error estimate of a fourth-order Runge-Kutta method with only one initial derivative evaluation

by A. S. CHAI
Hybrid Computer Laboratory
University of Wisconsin
Madison, Wisconsin

INTRODUCTION

In the numerical solution of differential equations it is desirable to have estimates of the local discretization (or truncation) errors of solutions at each step. The estimate may be used not only to provide some idea of the errors, but also to indicate when to adjust the step size. If the magnitude of the estimate is greater than the preassigned upper bound, the step size is reduced to achieve smaller local errors. If the magnitude of the estimate is less than the preassigned lower bound, the step size is increased to save the computing time.

The 4th-order Runge-Kutta method has the advantage that it provides an easy way to change the step size, but it does not provide as simple a way to get error estimates as does Milne's predictor-corrector method.[1] Several methods[2,3,4,5] for achieving error estimates have been derived and are briefly as follows:

1) One-step method

The one-step method provides all the information for the error estimate in one step. The important one-step method is Sarafyan's pseudo-iterative formula[2] which is a 5th-order Runge-Kutta formula imbedded in a 4th-order Runge-Kutta formula as follows:

$$k_0 = hf(x_n, y_n)$$
$$k_1 = hf(x_n + h/2, y_n + k_0/2$$
$$k_2 = hf(x_n + h/2, y_n + (k_0 + k_1)/4)$$
$$k_3 = hf(x_n + h, y_n - k_1 + 2k_2)$$
$$k_4 = hf(x_n + 2h/3, y_n + (7k_0 + 10k_1 + k_3)/27)$$
$$k_5 = hf(x_n + 2h/10, y_n + (28k_0 - 125k_1 + 546k_2 + 54k_3 - 378k_4)/625)$$

The 4th-order formula is

$$y_{n+1} = y_n + (k_0 + 4k_2 + k_3)/6$$

and the 5th-order formula is

$$\bar{y}_{n+1} = y_n + (14k_0 + 35k_3 + 162k_4 + 125k_5)/336$$

The estimate is

$$E_{n+1} = y_{n+1} - y_{n+1}$$

The work of Luther and Konen[6] (Legendre-Gauss) and of Luther[7] (Newton-Cotes, the 2nd formula, and Lobatto) also yield suitable pseudo-iterative formulas.

Really, pseudo-iterative formulas are 5th-order Runge-Kutta integration schemes used to estimate the error of the 4th-order Runge-Kutta integration.

The pseudo-iterative formula can be used to estimate the error at the first step, which can be done by no other method. But it requires about 50% more computing time for the additional derivative evaluations.

Merson's and Scraton's methods with five derivative evaluations belong to one-step pseudo-iterative method, but they are only applicable in particular cases.[4,8]

2) Two-step method[4,5]

This method requires the computation of y_{n+1} and y_{n+2} with a step size h, and then the recomputation of y^*_{n+2} with a doubling of the step size. The error estimate is

$$E_{n+2} = (y^*_{n+2} - y_{n+2})/30. \qquad (1.1)$$

Since the error is of the order h^5, we can let

$$y_{n+2} = \bar{y}_{n+2} + \epsilon_{n+1} + \epsilon_{n+2}$$

where the two error terms relate to the two steps, and

$$y^*_{n+2} = \bar{y}_{n+2} + 32\epsilon_{n+1} + O(h^6)$$

Hence, Formula (1.1) really provides an estimate of the error,

$$\epsilon_{n+29/30} = (31\,\epsilon_{n+1} - \epsilon_{n+2})/30 + O(h^6)$$

which is close to the error, ϵ_{n+1}, in y_{n+1}. If the estimate is for the error, ϵ_{n+2}, in y_{n+2}, it requires that the errors,

ϵ_{n+1} and ϵ_{n+2}, in y_{n+1} and y_{n+2} be approximately equal.

This method can be used to estimate the error at each two steps starting at y_2 and requires about 37.5% more computing time than fourth-order Runge-Kutta integration without estimates.

3) Multi-step method[3,4]

Ceschino and Kuntzmann (Ref. 3, pp. 305-310) collected several multi-step formulas for the error estimate which was based on Refs. 9, 10. One of them is (also in Ref. 4);

$$E_{n+2}=[10\Delta y_{n-1}+19\Delta y_n+\Delta y_{n+1}-h(3f_{n-1}+18f_n+9f_{n+1})]/30 \quad (1.2)$$

Strictly speaking, (1.2) estimates the error

$$\epsilon_{n+7/10}=(10\epsilon_n+19\epsilon_{n+1}+\epsilon_{n+2})/30.$$

This method can estimate the error at each step. But this method cannot be used until the completion of the third step (i.e., y_3). The estimate is close to the error at the last step, because

$$\epsilon_{n+7/10}\cong\epsilon_{n+1}.$$

If the estimate is for the error, ϵ_{n+1}, at the last step, and if the estimate causes the step size to change, four additional derivative evaluations are needed. If the estimate is for the error, ϵ_{n+2}, at the present step, it requires that the errors at three successive steps, ϵ_n, ϵ_{n+1}, and ϵ_{n+2}, be approximately equal. This requirement may not be satisfied if the errors change rapidly.

In general, the derivative evaluations need most of the computing time. The 4th-order Runge-Kutta method already requires more derivative evaluations than the other methods, e.g., Milne's method;[1] hence the extra time for the additional derivative evaluations for the error estimate is too expensive, and should be avoided as much as possible.

The suggested method

Ceschino and Kuntzmann (Ref. 3, p. 308) showed the following formula

$$E_{n+2}=\frac{11}{30}\Delta y_{n+1}+\frac{19}{30}\Delta y_n-h\left(\frac{1}{9}f_{n+2}+\frac{19}{30}f_{n+1}+\frac{8}{30}f_n-\frac{1}{90}f_{n-1}\right)\quad n>0 \quad (2.1)$$

for estimating the local discretization error in y_{n+2} in each step in a fourth-order Runge-Kutta integration. The author has found that (2.1) has advantages over the other methods in Section 1 and shows this below. Also, (2.1) can be extended to $n=0$, because we can form

$$y_{-1}=y_0+10\Delta y_1+19\Delta y_0-3h(f_2+6f_1+3f_0) \quad (2.2)$$

which has an error of $O(h^6)$, and then evaluate

$$f_{-1}=f(x_{-1},y_{-1}) \quad (2.3)$$

Then, using (2.1) for computing the estimate E_2, Equations (2.2) and (2.3) can be employed when (2.1) is just started or when the step size changes.

Equation (2.1) requires f_{n+2}, which has to be computed for k_0 in the next step if the step size does not change; hence, no additional derivative evaluations in each step are needed except for E_2 where one additional evaluation for f_{-1} by (2.3) is needed. This method requires about 12.5% more computing time for evaluating f_{-1}, when f_{-1} is not available, but no additional time if $n>0$. Hence, this method has an advantage in computing time over the one-step and two-step methods.

Equations (2.1-3) can estimate the error at each step after the first. Equation (2.1) estimates the error

$$\epsilon_{n+41/30}=(11\epsilon_{n+2}+19\epsilon_{n+1})/30$$

which is closer to the error, ϵ_{n+2}, than the estimates in the two-step and multi-step methods. Hence, this method has another advantage over the two-step and multi-step methods. The departure of the estimate from the local discretization error in y_{n+2} will be shown in the next section.

The derivation of (2.1) was shown in Ref. 3 and, as well as the derivation of (2.2), is briefly shown in Appendix 1.

Equation (2.1) can be employed to be a corrector to get errors in y of the order h^6. The convergence theorem and the experiment result are shown in Appendix 2.

The departure of the estimate

The departure of the estimate from the local discretization error is (Ref 3, pp. 306-308):

$$E_{n+2}-\epsilon_{n+2}=-\frac{1}{2}hf_{y_{n+1}}\epsilon_{n+1}-\frac{19h}{30}\epsilon'_{n+1}-\frac{11}{5400}h^6f^v_{n+1}+O(h^7) \quad (3.1)$$

To give the reader some picture of the departure, let us consider a differential equation:

$$y'=ay,\ y_0=y(x_0)$$

Where a is a constant and is not equal to zero.

The formula of the local discretization error of the order h^5 in y_{n+2} by several known 4th-order Runge-Kutta formulas can be found in (Ref. 3, p. 81). In this example, the local discretization error of the order h^5 is

$$\epsilon_{n+2}=-\frac{h^5}{120}\left(f_{y_{n+2}}\right)^3f'_{n+2}=-\frac{h^5}{120}a^3y''_{n+2}.$$

Since $y = y_0 e^{ax}$,
this gives

$$\epsilon_{n+1} = -\frac{h^5}{120} y_0 a^5 e^{ax_{n+1}},$$

$$\epsilon'_{n+1} = -\frac{h^5}{120} y_0 a^6 e^{ax_{n+1}},$$

$$f^v_{n+1} = y_0 a^6 e^{ax_{n+1}},$$

and

$$f_y = a.$$

Hence the departure of the estimate is

$$E_{n+2} - \epsilon_{n+2} = \frac{1}{135} h^6 a^6 y_0 e^{ax_{n+1}}.$$

The relative error is

$$\frac{E_{n+2} - \epsilon_{n+2}}{\epsilon_{n+2}} = \frac{8ha}{9} = .889ha$$

of which the magnitude is less than 10% if $|ha| \leq 0.1$.

To verify the theory an experiment was run to solve

$$y' = y,\ y_0 = 1,\ h = 0.1 \text{ and } x = 0 \text{ to } 10.$$

The local discretization error at x_{n+2} is $y_{n+2} - y_{n+1}e^h$. The experimental results for the relative errors of the estimates for x = .2 to 10, were between .08211 and .08225. Then the equation was changed to

$$y' = -y$$

with the same parameters. The local discretization error at x_{n+2} is $y_{n+2} - y_{n+1}e^{-h}$. The experimental results for the relative errors were between −.09620 and −.09637, or slightly less than 10%. Hence the experimental result agrees with the theory.

If Equations (2.2) and (2.3) are employed for computing f_{-1}, the departure, which is derived in Appendix 3, is

$$E_2 - \epsilon_2 = -\frac{1}{6} h f_{y_1} \epsilon_1 - \frac{19}{30} h \epsilon'_1 - \frac{11}{5400} h^6 f^v_1 + O(h^7)$$

As before, the relative error for E_2 in

$$y' = ay,\ a \neq 0$$

is approximately equal to

$$\frac{5}{9} h a = .556\, h a$$

which is about 5% if $|ha| \cong 0.1$.

The experimental results for E_2 in

$$y' = y \text{ and in } y' = -y,$$

are .0518 and −.0593 respectively, where h = 0.1.

Another experiment is to solve a system of nonlinear differential equations

$$\begin{aligned} y'(x) &= z(x) \\ z'(x) &= (2\, y(x) - 1)\, z(x) \\ y(o) &= 0.5,\ z(o) = y'(o) = -0.25 \end{aligned} \tag{3.2}$$

The step size, h, was 0.1 and x ran from 0 to 5.

The local discretization error at y_{n+1} is

$$\epsilon_{n+2} = y_{n+2} - \frac{\alpha(y_{n+1} - \beta) - \beta(y_{n+1} - \alpha)e^{(\alpha-\beta)h}}{y_{n+1} - \beta - (y_{n+1} - \alpha)e^{(\alpha-\beta)h}}$$

where

$$\left.\begin{matrix}\alpha \\ \beta\end{matrix}\right\} = \frac{1 \pm \sqrt{1 + 4(y^2_{n+1} - y_{n+1} - z_{n+1})}}{2}$$

The local discretization error ϵ_{n+2} and the relative error of the estimate are shown in Table 3.1. The values of the relative errors are about 0.1 in general except when the curve of ϵ_{n+2} approaches zero rapidly. Hence the estimate is in general suitable for practical purpose.

TABLE 3.1

x	Local error	Rel. error
.2	.7558E−08	−.6193E−02
.3	.7047E−08	−.4302E−02
.4	.6301E−08	−.1157E−01
.5	.5371E−08	−.2127E−01
.6	.4320E−08	−.3440E−01
.7	.3223E−08	−.5614E−01
.8	.2145E−08	−.9597E−01
.9	.1144E−08	−.1944E 00
1.0	.2710E−09	−.8569E 00
1.1	−.4447E−09	.5242E 00
1.2	−.9877E−09	.2258E 00
1.3	−.1352E−08	.1517E 00
1.4	−.1544E−08	.1189E 00
1.5	−.1586E−08	.9782E−01
1.6	−.1502E−08	.8015E−01
1.7	−.1317E−08	.6420E−01
1.8	−.1059E−08	.4473E−01
1.9	−.7522E−09	.1596E−01
2.0	−.4229E−09	−.5341E−01
2.1	−.9004E−10	−.6064E 00
2.2	.2301E−09	.3657E 00
2.3	.5275E−09	.2065E 00
2.4	.7931E−09	.1637E 00
2.5	.1021E−08	.1448E 00
2.6	.1211E−08	.1329E 00
2.7	.1362E−08	.1254E 00
2.8	.1477E−08	.1195E 00
2.9	.1556E−08	.1157E 00
3.0	.1605E−08	.1126E 00
3.1	.1626E−08	.1103E 00
3.2	.1624E−08	.1083E 00
3.3	.1603E−08	.1068E 00
3.4	.1565E−08	.1054E 00
3.5	.1516E−08	.1043E 00
3.6	.1457E−08	.1031E 00
3.7	.1391E−08	.1025E 00
3.8	.1320E−08	.1018E 00
3.9	.1247E−08	.1012E 00
4.0	.1172E−08	.1005E 00
4.1	.1087E−08	.1005E 00
4.2	.1024E−08	.9993E−01
4.3	.9525E−09	.9958E−01
4.4	.8836E−09	.9921E−01

4.5	.8176E–09	.9892E–01
4.6	.7548E–09	.9876E–01
4.7	.6954E–09	.9867E–01
4.8	.6395E–09	.9861E–01
4.9	.5872E–09	.9860E–01
5.0	.5385E–09	.9827E–01

Local discretization error and relative error of estimate at y in $y' = z$ and $z' = (2y - 1)z$

ACKNOWLEDGMENTS

The author is indebted to Professors C.A. Ranous and V. C. Rideout for aid in expression in writing this paper and Professor C. W. Cryer for discussion of convergence theory. The author particularly thanks his former advisor Professor H. J. Wertz who suggested this problem and has often given encouragement.

The author is also indebted to the referees for their constructive comments.

All the experimental results were obtained on an SDS 930 computer (38 bit = 11.4 digits in mantissa) in the Hybrid Computer Laboratory at the University of Wisconsin, Madison, Wisconsin.

Appendix 1

Derivation of (2.1) [3, pp 305-308] *and (2.2)*

It is easy to derive

$$y(x_{n+2}) = -\frac{8}{11}y(x_{n+1}) + \frac{19}{11}y(x_n) + h(\frac{10}{33}\bar{f}_{n+2}) + \frac{19}{11}\bar{f}_{n+1} + \frac{8}{11}\bar{f}_n - \frac{1}{33}\bar{f}_{n-1}) - \frac{1}{180}h^6 f^v_{n+1} + O(h^7) \quad \text{(A1.1)}$$

Substitute

$$y(x_n) = y(x_{n+1}) - \Delta y(x_{n+1}) - \epsilon_{n+2}$$

and

$$y(x_{n+2}) = y(x_{n+1}) + \Delta y(x_{n+1}) \epsilon_{n+2}$$

then, after simplification, we get

$$\epsilon_{n+41/30} = \frac{11\epsilon_{n+2} + 19\epsilon_{n+1}}{30} + O(h^7)$$
$$= \frac{11}{30}\Delta y(x_{n+1}) + \frac{19}{30}\Delta y(x_n) - h(\frac{1}{9}\bar{f}_{n+2} + \frac{19}{30}\bar{f}_{n+1} + \frac{8}{30}\bar{f}_n - \frac{1}{90}\bar{f}_{n-1}) - \frac{11}{5400}h^6 f^v_{n+1} + O(h^7)$$

Now, therefore, the estimate is (2.1).

Equation (2.2) is easy to be derived by

$$y_{-1} = 10y_2 + 9y_1 - 18y_0 - 3h(f_2 + 6f_1 + 3f_0) + O(h^6)$$

Appendix 2

A convergence theorem of (2.1)

Equation (A1.1) can be rewritten to

$$y_{n+1} = -\frac{19}{11}\bar{y}_{n+1} + \frac{30}{11}\bar{y}_n + \frac{19}{11}\Delta y_n - \frac{19}{11}\Delta y_{n-1} + h(\frac{10}{33}f_{n+1} + \frac{19}{11}f_n + \frac{8}{11}f_{n-1} - \frac{1}{33}f_{n-2})$$

Then

$$\bar{y}_{n+1} = \bar{y}_n + \frac{19}{30}(\Delta y_n - \Delta y_{n-1}) + h(\frac{1}{9}f_{n+1} + \frac{19}{30}f_n + \frac{8}{30}f_{n-1} - \frac{1}{90}f_{n-2}) = y_n + \emptyset(x_n, y_n)$$

The following statement of a covergence theorem is similar to Byrne and Lambert[11].

Assume that (1) f(x,y) is continuous for $x \in I$ and $\| y \| < \infty$, (2) f satisfies a Lipschitz condition and (3) $0(x_n, y_n)$ is defined for all h such that $x + h$ and $x - 2h \in I$.

(A2.1) is consistent because

$$\lim_{h \to 0} (x,y)/h = f(x,y), \; x \in I, \; \| y \| < \infty.$$

A convergence theorem is

Let y_0 be the initial value and

$$\| y_1 - y(x_1) \| + \| y_2 - y(x_2) \| \leq hL$$

where L is a non-negative constant and let there exist three non-negative numbers L_1, L_2, and L_3 such that

$$\| 0(x_n, w_n^*) - 0(x_n, w_n) \| \leq h(L_1 \| w_n^* - w_n \| + L_2 \| w_{n-1}^* - w_{n-1} \| + L_3 \| w_{n-2}^* - w_{n-2} \|)$$

hold for the vectors w_n^* and w_n. Under these conditions, the hypotheses (1-3), and the consistence property, (A2.1) is convergent.

The proof can be similarly employed as in Byrne and Lambert[11] except that one more term,

$$h L_3 z_{n-2} = h L_3 \| y(x_{n-2}) - y_{n-2} \|$$

is added in the right-hand side of the inequality

$$z_{n+1} \leq z_n(1 + hL_1) + hz_{n-1}L_2 + hg(h)$$

and thereafter. This proof does not include the point x_{-1}, so we have to assume that $x_{-1} \in I$ if (2.2) is employed.

Hence (A2.1) can be used as a corrector to get errors to $O(h^6)$ except at y_1. However y_2 should be corrected by twice the estimate at x_2.

Table A2.1 shows the accumulative errors of non-corrected and corrected solutions in the experiment to solve the system of non-linear equations (3.2). The theoretical solution is

$$y(x) = \frac{1}{1 + e^x}$$

The accumulative error was obtained by subtracting $y(x_n)$ from y_n (non-corrected) or y_n (corrected by (2.1)).

In Table A2.1 the errors of the non-corrected solutions are greater than those of the corrected.

x	Non-corrected	Corrected
.2	1.531×10^{-8}	2.929×10^{-10}
.3	2.228×10^{-8}	5.402×10^{-10}
.5	3.364×10^{-8}	1.759×10^{-9}
1.0	4.327×10^{-8}	6.483×10^{-9}
2.0	2.837×10^{-8}	6.314×10^{-9}
3.0	2.555×10^{-8}	-5.898×10^{-10}
4.0	2.104×10^{-8}	-2.966×10^{-9}
5.0	1.352×10^{-8}	-2.749×10^{-9}

TABLE A2.1
Errors in non-corrected and corrected solutions of y in Equation (3.2).

Appendix 3

The departure of (2.1) when (2.3) is employea

If (2.2) is employed for y_{-1}, since y_0 is the initial value, we can assume that y_0 has no error, then

$$y_1 = y(x_1) + \epsilon_1$$

and

$$y_2 = y(x_2) + \epsilon_1 + \epsilon_2$$

Hence

$$f_i = f_i + f_{y_i}(\epsilon_1 + \ldots + \epsilon_i),\ i = 1,2$$

Now

$$\Delta y_0 = \Delta y(x_0)$$

and

$$\Delta y_1 = \Delta y(x_1) + h f_{y_1}\epsilon_1$$

with

$$y_{-1} = y(x_{-1}) + 19\epsilon_1 + 10\epsilon_2$$

and

$$f_{-1} = f_{-1} + f_{y_{-1}}(19\epsilon_1 + 10\epsilon_2)$$

then

$$E_2 \cong \frac{11}{30}\epsilon_2 + \frac{19}{30}\epsilon_1 - \frac{1}{6}hf_{y_1}\epsilon_1 - \frac{11}{5400}h^6 f_1$$

Hence the departure of the estimate E_2 from ϵ_2 is

$$E_2 - \epsilon_2 \cong -\frac{19}{30}h\epsilon'_1 - \frac{1}{6}hf_{y_1}\epsilon_1 - \frac{11}{5400}h^6 f_1$$

NOMENCLATURE

$y' = f(x,y)$ - a system of differential equations where x is the independent variable and y represents the dependent variables. y, y′ and f are vectors.

h - step size.

y_n - solution of y at $x_n = x_0 + nh$ with an error of $O(h^5)$ obtained by a fourth-order Runge-Kutta formula.

y_n - solution of y at x_n with an error of $O(h^6)$.

$f = f(x_n, y_n)$

$f_n = f(x_n, y_n)$

$y(x_n)$ - theoretical solution of y at x_n.

Δy_n - increment function which is defined by $y_{n+1} - y_n$.

$\Delta y(x_n)$ - similar to Δy_n, but replacing y_n in Δy_n by $y(x_n)$.

- local discretization (or truncation) error in y_n which is defined by the difference of $y(x_n)$ from y_n with considering y_{n-1} to be the initial value, e.g.
$\epsilon_n = y(x_{n-1}) + \Delta y(x_{n-1}) - Y(x_n)$

E_n - estimate of ϵ_n.

REFERENCES

1 W E MILNE
Numerical solution of differential equations
Wiley New York p 66 (1953)

2 D SARAFYAN
Error estimation for Runge-Kutta methods through pseudo-iterative formulas
Technical Report No 14 Louisiana State University New Orleans (May 1966)

3 F CESCHINO J KUNTZMANN
Problemes differentiels de conditions initiales
Dunod Paris France pp 81 305-310 (1963) English translation by D Boyanovitch Prentic-Hall Englewood Cliffs N J pp 67 247-251 (1966)

4 R E SCRATON
Estimation of the truncation error in Rung-Kutta and allied processes
Comp J vol 7 pp 246-248 April 1964

5 L COLLATZ
The numerical treatment of differential equations
Third edition second printing English translation by P G Williams Springer-Verlag New York pp 51-52 1966

6 H A LUTHER H P KONEN
Some fifth-order classical Runge-Kutta formulas
SIAM Review vol 7 pp 551-558 1965

7 H A LUTHER
Further explicit fifth-order Runge-Kutta formulas
SIAM Review vol 8 pp 374-380 1966

8 A R CURTIS
Correspondence to Estimation of the truncation error in Runge-Kutta and allied processes
Comp J vol 8 p 52 1965

9 H MOREL
Evaluation de l'erreur sur un pas dans la methode de Runge-Kutta
Comptes rendus de l'Academie des Sciences Paris vol 243 pp 1999-2002 1965

10 J KUNTZMANN
Evaluation de l'erreur sur un pas dans les methodes a pas separes
Chiffres vol 2 pp 97-102 1959

11 G D BYRNE R J LAMBERT
Pseudo-Runge-Kutta methods involving two points
J ACM vol 13 pp 114-123 1966

Improved techniques for digital modeling and simulation of nonlinear systems

by JOSEPH S. ROSKO
United Aircraft Research Laboratories
East Hartford, Connecticut

INTRODUCTION

The engineer or scientist concerned with the mathematical description of physical systems is continually faced with nonlinear formulations. The nonlinearity may be represented in the form of a differential equation representing process or system dynamics. On the other hand, nonlinear control element characteristics such as hysteresis, saturation, backlash, or nonlinear damping, whose mathematical description is algebraic, may appear. In many instances the mathematical formulation for system description may become so unwieldy that an analytical solution is either impractical or impossible. It is particularly in situations like these that digital simulation has become an invaluable tool.

Historically, Euler integration, Newton-Coates quadrature formulas, predictor-corrector techniques, Runge-Kutta methods, and the techniques belonging to the realm of numerical analysis were first used to obtain approximate solutions to differential equations.[1,2] After it became commonplace to represent the linear components of feedback control systems by transfer functions, the Tustin[3] and other substitutional techniques[4,5,6,7] evolved. These methods represent each transfer function as an equivalent approximate discrete form. This permits the response of each block to be formulated as a single difference equation. Recently, space vehicle simulation and man-in-the-loop studies have necessitated the procurement of high precision and real-time simulations. The term real-time is used in this instance to denote the occurrence of events in a physical system and its simulation with the same time base. Although the simplicity of the algebraic response equations developed by the Tustin method make it attractive, the solution interval necessary for accurate closed loop simulations is prohibitive. A number of significant contributions have been made within the past four years toward the alleviation of these problems. Hurt[8] and Fowler[9] employ root locus procedures and Sage and Burt[10,11] have extended the quasilinearization technique to design more exact discrete system models.

This paper will first review three digital simulation techniques previously mentioned whose usage is widespread. Emphasis will be on nonlinear systems whose form is shown in Figure 1. Then a new real-time simulation method which is based upon purely

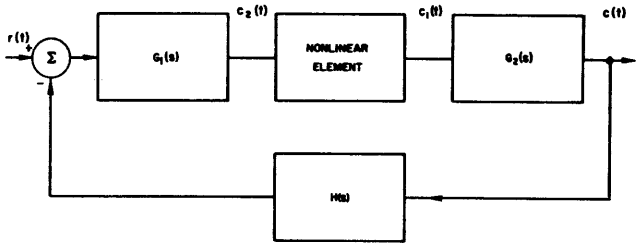


Figure 1 – A general nonlinear feedback system

algebraic procedures and does not require the use of ancillary computation during the design stage is presented in detail. An adaptive filtering technique is also described in which a time-varying compensatory device is employed in conjunction with the discrete system model to increase simulation accuracy. Finally selected examples are used to provide a basis of comparison and substantiate the results obtainable with the new methods.

Current simulation techniques

Tustin method

Tustin's mathematical formulation as applied to system simulation consists of making a discrete approximation to the operational integrating operator $(1/s)^n$ and obtaining a digitized representation of each transfer function. Once the various blocks have been discretized, algebraic difference equations representing the response of each block may be formed.

The linear components of a system are represented as transfer functions

$$G(s) \triangleq \frac{a_n s^n + a_{n-1}s^{n-1} + \ldots + a_1 s + a_o}{b_k s^k + b_{k-1}s^{k-1} + \ldots + b_1 s + b_o}, \quad k \geq n. \tag{1}$$

Begin by dividing the numerator and denominator Equation 1 by s^k.

$$G(s) = \frac{a_0(1/s)^k + a_1(1/s)^{k-1} + \ldots + a_{n-1}(1/s)^{k-n+1} + a_n(1/s)^{k-n}}{b_0(1/s)^k + b_1(1/s)^{k-1} + \ldots + b_{k-1}(1/s) + b_k} \tag{2}$$

Now a pulse transfer function may be formulated by substituting the Tustin discrete approximations for $(1/s), \ldots, (1/s)^k$ in Equation 2. This operation results in

$$G(z) = \frac{a_0\{\}_k + a_1\{\}_{k-1} + \ldots + a_{n-1}\{\}_{k-n+1} + a_n\{\}_{k-n}}{b_0\{\}_k + b_1\{\}_{k-1} + \ldots + b_{k-1}\{\}_1 + b_k}, \tag{3}$$

where $\{\}_i$ represents the discrete approximation to the ith order integrating operator $(1/s)^i$ as a ratio of polynomials in z.

Simplification yields

$$G(z) = \frac{A_0 z^k + A_1 z^{k-1} + \ldots + A_{k-1}z + A_k}{B_0 z^k + B_1 z^{k-1} + \ldots + B_{k-1}z + B_k}, \tag{4}$$

where the coefficients $A_0, A_1, \ldots, A_k$ consist of combinations of T, the independent variable increment, and the transfer function coefficients $a_0, \ldots, a_n$ and the $B_0, B_1, \ldots, B_k$ consist of combinations of T and the transfer function coefficients $b_0, \ldots, b_k$.

Applying this procedure to the system shown in Figure 1 results in the discrete transfer function $G_1(z)$, $G_2(z)$, and $H(z)$. In addition, it is necessary to insert a single period delay in the feedback loop to render a realizable simulation. The results of these digital modeling operations are shown in Figure 2. To

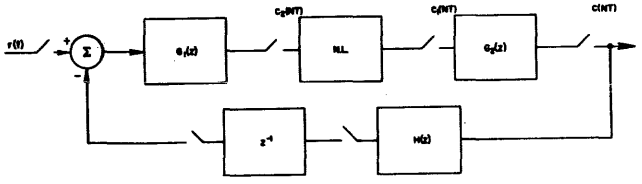


Figure 2 – Digital model of a general nonlinear system using the Tustin method

simulate a system such as this, all that remains is to obtain a set of recursion formulas using conventional techniques.

IBM method

Hurt[8] and Fowler,[9] the developers of the IBM Method, specifically propose to achieve a more exact simulation for a given sample period by matching the closed-loop eigenvalues and the static gains of the analog and digital systems.

Their procedure is now presented in an abridged form.

1. The initial discrete system shown in Figure 3 is obtained from the continuous system by forming the z-transform of each block and then inserting an incremental delay in the feedback loop to insure realizability.

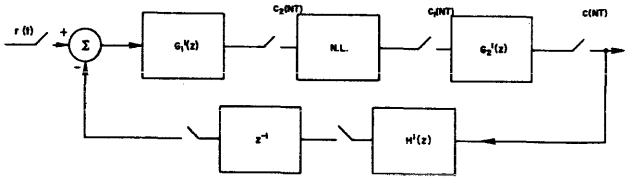


Figure 3 – Initial configuration; IBM digital model of a nonlinear system

2. Next each block is assumed to have a step input and the static gains are matched between the analog and digital block transfer functions. This is accomplished by multiplying the discrete transfer function by a constant parameter, applying the final value theorems to both transfer functions with the assumed step input and equating the results to determine the constant parameter. As an example, the first block in the forward path is selected.

$$\lim_{s \to 0} G(s) = \lim_{z \to 1} G_1 G_1^1(z) \tag{5}$$

For integrators, the approach is to multiply its z-transform by the sampling interval, T.

3. After replacing the nonlinearity by some nominal gain, G_3, the root locus of each system is matched by inserting a gain element (G_4) in the forward path of the discrete system. The results of these operations are illustrated in Figure 4.

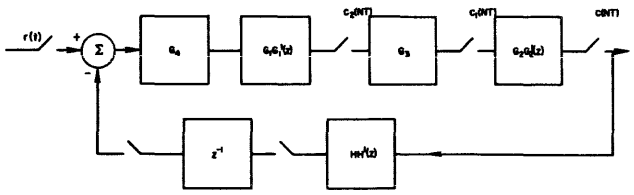


Figure 4 – Intermediate configuration; IBM digital model of a nonlinear system

4. The last step involves matching the discrete system to the excitation. This is achieved by attaching an "input transfer function" to the discrete system. The matching is accomplished by forming the response, in operational form, of both the continuous

and discrete systems, then obtaining the z-transform of the linearized continuous system response and equating it to the discrete response to evaluate I(z). For the system under consideration

$$Z\left[R(s)\cdot\frac{G_1(s)G_3G_2(s)}{1+H(s)G_1(s)G_3G_2(s)}\right]=$$

$$\left[R(z)I(z)\cdot\frac{G_4G_1G_1{}^1(z)G_3G_2G_2{}^1(z)}{1+z^{-1}HH^1(z)G_4G_1G_1{}^1(z)G_3G_2G_2{}^1(z)}\right]. \quad (6)$$

The final form of the digital model which is equivalent to the general nonlinear system at sampling intervals is presented in Figure 5.

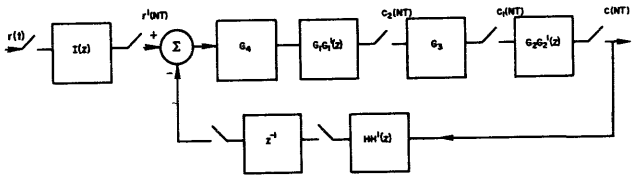


Figure 5 – Final configuration; IBM digital model of a nonlinear system

Sage-Burt method

The Sage-Burt approach[10,11] is the first attempt at discrete system modeling which considers the full impact of nonlinearities. However, this technique for modeling and simulation requires a considerable amount of tailoring and preliminary design computation.

Their procedure is now presented in summarized form by considering the system in Figure 1.

1. Individual blocks are discretized by extracting the transfer function from the aggregate system and formulating an optimization problem. The first block in the forward loop of Figure 1 is selected to illustrate the procedure.

If Figure 6, A(s) is the ideal transfer function, A(z) is the desired transfer function, and F(z) is the fixed

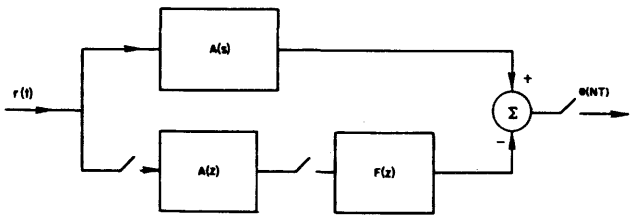


Figure 6 – System for optimization problem

portion of the system. The fixed portion of the system is selected to be unity except where a pure delay must be utilized to render a realizable simulation. Error and total squared error for this situation are given as

$$E(z)=C(z)-R(z)A(z)F(z), \quad (7)$$

and

$$\sum_{N=0}^{\infty}e^2(NT)=\frac{1}{2\pi_j}\int_{\Gamma}E(z)E(z^{-1})z^{-1}dz, \quad (8)$$

where C(z) is the z-transform of the product R(s)A(s) and the contour of integration is the unit circle. A(z) is determined by applying the calculus of variations to Equation 8 such that it is minimized. The results are

$$A_0(z)=\left\{\frac{R(z^{-1})F(z^{-1})C(z)}{[R(z)R(z^{-1})F(z)F(z^{-1})]-}\right\}\text{P.R.}, \quad (9)$$
$$[R(z)R(z^{-1})F(z)F(z^{-1})]+$$

where P.R. refers to the realizable portion of the term within {} and the + and − subscripts refer to the spectrum factorization operator denoting extraction of multiplicative terms containing poles and zeros inside (+) or outside (−) the unit circle. Generally either a step or ramp is selected as a convenient excitation for this procedure.

2. The nature of the nonlinearity is considered by the inclusion and determination of a multiplicative constant parameter P_i associated with each block. The desired result is attained when the discrete system output $\underline{Y}_A(t)$ approaches the analog output $\underline{Y}_i(t)$, where $\underline{Y}_A(t)$ and $\underline{Y}_i(t)$ are m vectors describing the system output state for a given input.

The continuous state vector output, $\underline{Y}_i(t)$, and the excitation, X(t), are assumed to be completely known. The constant parameters $\underline{P}$, interpreted as a p vector are then adjusted to minimize the performance index.

$$J=\frac{1}{2}\sum_{N=0}^{k-1}\left\|\underline{Y}_i(NT)-\underline{Y}_A(NT)\right\|_R^2 \quad (10)$$

subject to the constraint

$$\underline{Y}_A[(N+1)T]=f[\underline{Y}_A(NT),P] \quad (11a)$$
$$\underline{Y}_A(0)=\underline{Y}_i(0) \quad (11b)$$
$$\underline{P}[(N+1)T]+\underline{P}(NT). \quad (11c)$$

By application of standard variational calculus procedures, it is found that the optimum parameter vector $\underline{P}$ is determined by solution of Equations 10 and 11 together with the adjoint difference equations,

$$\underline{\lambda}_Y(NT)=\nabla_Y f'[\underline{Y}_A(NT),\underline{P}]\underline{\lambda}_Y[(N+1)T] \quad (12a)$$
$$+R[\underline{Y}_i(NT)-\underline{Y}_A(NT)]$$
$$\underline{\lambda}_P(NT)=\nabla_P f'[\underline{Y}_A(NT),P]\underline{\lambda}_Y[(N+1)T] \quad (12b)$$
$$+\underline{\lambda}_P[(N+1)T]$$
$$\underline{\lambda}_Y(KT)=\underline{\lambda}_P(KT)=\underline{\lambda}_P(0)=0. \quad (12c)$$

Sage and Burt consider the computational problems inherent in this discrete two-point nonlinear boundary value problem and suggest quasilinearization methods for its solution. The interested reader is referred to Sage and Burt[10,11] or Bellman[12,13] for particulars concerning this method.

New simulation method

Tustin's method provides excellent simulations of open-loop systems and has attained widespread acceptance and usage. However, when this technique is applied to the discretization of closed-loop systems, the necessity of inserting a delay in the feedback path to obtain a realizable simulation shifts the normal location of the closed-loop poles appreciably in many instances and results in the consequent performance degradation. Real-time simulation is generally ruled out since a smaller than tolerable time increment must be utilized to achieve adequate response representations. The IBM method and the Sage-Burt method rectify this situation by more perfect modeling of the closed-loop system. The new simulation method to be presented in this section combines the simplicity of Tustin's design procedure with the more exact modeling philosophy of these more recent approaches to yield a comparable real-time simulation capability.

One initiates the design process by employing the Tustin method to discretize each transfer function. After the necessary unit delay is inserted in the feedback path, the resultant digital system model of a general nonlinear system is that illustrated in Figure 2. Next a discrete filter with pulse transfer function D(z) is placed at the normal input to the digital model. Figure 7 depicts this situation where the transfer function of the digital compensator has the form

$$D(z)=\frac{_0+a_1z+\ldots\ldots+a_mz^m}{b_0+b_1z+\ldots\ldots+b_nz^n} \qquad (13)$$

with all coefficients being real. The first step in the determination of the coefficients in D(z) is to form a stable discrete overall transfer function of the continuous system with the nonlinearity represented as a nominal gain, G_3

$$K_d(z)=G(s)\Big|\left(\frac{1}{s}\right)^n=(\,)_n,\ n=0,1,2,\ldots,k, \qquad (14)$$

where

$$G(s)=G_1(s)G_3G_2(s)\,/\,[1+H(s)G_1(s)G_3G_2(s)].*$$

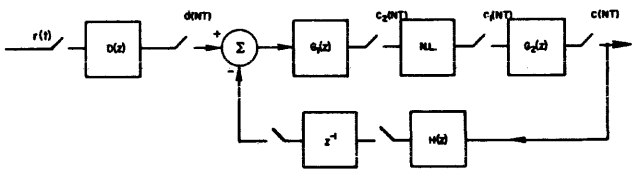


Figure 7 – New discrete general nonlinear system model

*( )$_n$ represents the Tustin discrete approximation to the nth order operational integrator $(1/s)^n$.

Next the overall pulse transfer function of the simulated system is formed by applying the Tustin element to each block in Figure 1 resulting in

$$K_A(z)=\frac{D(z)G_1(z)G_3G_2(z)}{1+z^{-1}H(z)G_1(z)G_3G_2(z)}, \qquad (15)$$

where:

1. D(z) has the form of Equation 13.
2. $G_1(z)$, $G_2(z)$, H(z) denote the approximate/discrete transfer functions for $G_1(s)$, $G_2(s)$, and H(s) respectively; each obtained utilizing Tustin's method.
3. G_3 represents a nominal gain of the nonlinear element.**

All that remains is the determination of the coefficients of the polynominals in D(z). This may be accomplished by equating Equations 14 and 15, expanding the pulse transfer function, and then equating the coefficients of like powers in z.

$K_A(z)=K_d(z)$

$$\frac{D(z)G_1(z)G_3G_2(z)}{1+z^{-1}H(z)G_1(z)G_3G_2(z)}=G(s)\Big|\left(\frac{1}{s}\right)^n$$

$$=(\,)_n,\ n=0,1,2,\ldots,k.* \qquad (16)$$

With all pulse transfer functions determined for the linear portion of the system, the corresponding recursive relationships for digital simulation can be obtained.

Adaptive filtering

Application of the new design technique results in a discrete system model consisting of numerous pulse transfer functions, system nonlinearities, and a digital compensator. Since the design procedure considers nonlinearities as fixed nominal amplification factors, the coefficients in D(z) are time invariant.

For simulations where greater accuracy is a requisite, an adaptive filtering scheme is proposed. Figure 8 is a suggested digital model where $G_1(z)$, $G_2(z)$, and H(z) retain their definitions from Figure 7. In this case, the form of the filter given as Equation 17 has time-varying coefficients.

$$D(z)=\frac{a_0(t)+a_1(t)z+\ldots+a_m(t)z^m}{b_0(t)+b_1(t)z+\ldots+b_n(t)z^n} \qquad (17)$$

The coefficients may be obtained in a manner analogous to Equation 16 for the fixed filter case.

**This assumption is exactly the same made by Hurt.[8]

*See previous footnote.

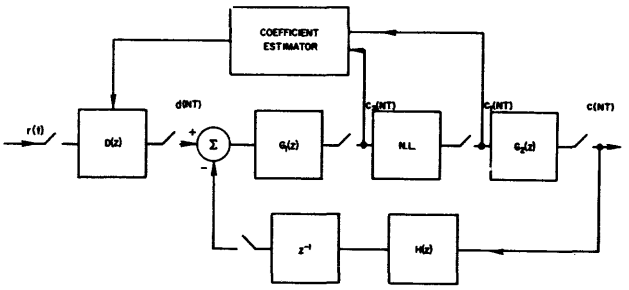


Figure 8 – Discrete general nonlinear system model with adaptive filtering

The filter coefficients will, in general, be explicit mathematical functions of the quantity A(t) representing the pseudo-gain of the nonlinear element. As shown in Figure 8, the Coefficient Estimator plays the dual role of identifying the gain of the nonlinearity and calculating the time-varying coefficients of D(z). The simplest determination of gain identification by direct measurement procedures is

$$A[(N+1)T] = \left.\frac{c_1(t)}{c_2(t)}\right|_{t=NT} \qquad (18)$$

where it is noted from Figure 8 that the best possible measurement with this compensatory device is delayed by one sample period.* More refined identification procedures utilizing both direct measurement and extrapolation would introduce further accuracy improvements.

An example for methods comparison

To illustrate the design procedure and to present a basis for comparison, the second order nonlinear system shown in Figure 9 will be used. This system was originally employed by Sage and Smith.[11]

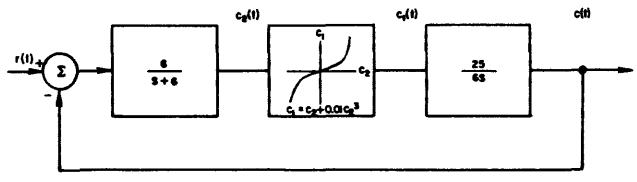


Figure 9 – Nonlinear system employed as an example

By applying the Tustin approximation for $(1/s)^n$ to $G_1(s)$ and $G_2(s)$ discrete transfer functions $G_1(z)$ and $G_2(z)$ may be obtained. Then by utilizing Equations 13-16 in succession with $H(s) = H(z) = 1$, one obtains an explicit form for the digital compensator D(z). With these preliminaties complete, a set of difference equations may be written and subsequently implemented on a digital computer.

$$\begin{aligned} d[NT] = \{&(8-50AT^2)d[(N-1)T] \\ &+(12T-4-25AT^2)d[(N-2)T] \\ &+(12T+4)r[NT] \\ &+(25AT^2-8)r[(N-1)T] \\ &+(50AT^2-12T+4)r[(N-2)T] \\ &+(25AT^2)r[(N-3)T]\} \\ &\frac{1}{4+12T+25AT_2} \end{aligned}$$

$$\begin{aligned} c_2[NT] = &\frac{1-3T}{1+3T}c_2[(N-1)T] \\ &+\frac{3T}{1+3T}\{d[NT]-c[(N-1)T] \\ &+d[(N-1)T]-c[(N-2)T]\} \end{aligned} \qquad (19)$$

$$c_1[NT] = c_2[NT] + 0.01\,c_2^3[NT]$$

$$\begin{aligned} c[NT] = c[(N-1)T] + &\frac{25T}{12}\{c_1[NT] \\ &+c_1[(N-1)T]\} \end{aligned}$$

Figure 10 illustrates the result of applying a positive step input of magnitude 10 to the discretized models employing the Tustin, IBM, Sage-Burt, and new simulation methods. A similar comparison of the responses produced for a sinusoidal excitation applied to the IBM, Sage-Burt, and new simulation models is made in Figure 11. These results clearly indicate that the new method for discrete system modeling and real-time simulation is justified since a marked improvement over the Tustin simulation has been achieved through a minimal amount of design effort.

Three schemes for the identification of the system nonlinearity were chosen to form the basis of comparing the enhancement afforded to simulation accuracy through the utilization of adaptive filtering. The direct measurement method was first selected to indicate that even "stale" or delayed parameter identification provides better results than a fixed coefficient filter. One-half period and full period linear extrapolation of the measured pseudo-gain were employed to illustrate further improvements. These identification procedures may be easily implemented as a component of the system simulation with the recursive formulas in Equation 20. Figures 12 and 13 compare the step and

DIRECT MEASUREMENT

$$A[(N+1)T] = 1.0 + 0.01C_2^2[NT] \qquad (20a)$$

*In cases where the nonlinearity may be expressed as a simple explicit mathematical function, "direct measurement" may be conveniently replaced by "direct formulation."

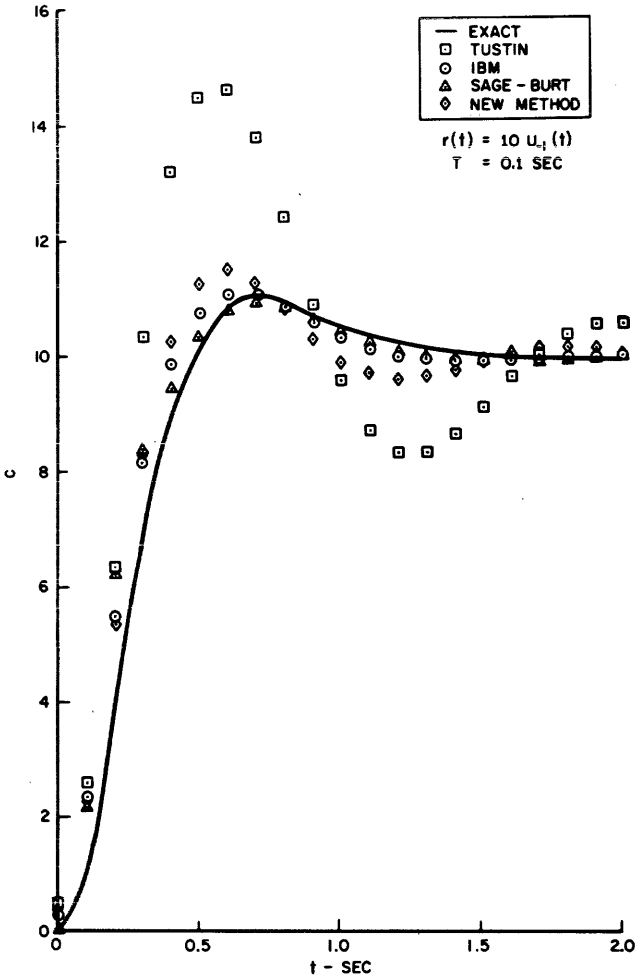
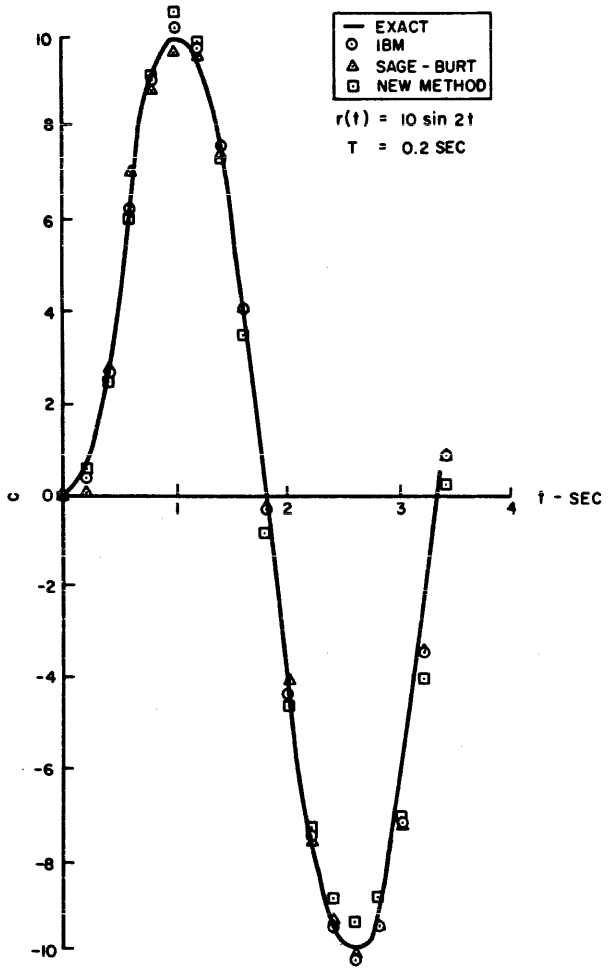


Figure 10 – Response of the example system to a step excitation



Fgure 11 – Response of the example system to a sinusoidal excitation tion

ONE-HALF PERIOD EXTRAPOLATION

$$A[(N+1)T] = 1.0 + 0.01\{1.5c_2[NT] - 0.5c_2[(N-1)T]\}^2 \quad (20b)$$

FULL PERIOD EXTRAPOLATION

$$A[(N+1)T] = 1.0 + 0.01\{2c_2[NT] - c_2[(N-1)T]\}^2 \quad (20c)$$

sinusoidal responses of the example system discretized utilizing the new modeling procedure for various adaptive filtering measurement schemes.

In order to compare the various discrete modeling and simulation techniques, simulations of the example system were performed for numerous sampling increments utilizing step and sinusoidal excitations. Then the mean-square error was calculated for each simulation technique for each sampling interval over the observation period 0-5 seconds. The error analysis of systems simulated with an applied step excitation of magnitude 10 shown in Figure 14 indicates that the new technique provides considerable improvement over the Tustin method and has accuracy comparable to the IBM method and the Sage-Burt quasilinearization method. Also, note the accuracy improvement and increased region of stability when Sage and Burt utilize quasilinearization. The IBM, Sage-Burt with quasilinearization, and new simulation techniques provide comparable low sensitivity to error as the simulation time increment is altered.

The resulting error analysis of systems simulated with an applied sinusoidal excitation is shown in Figure 15. A marked improvement over the Tustin method is exhibited by the system simulated using the new technique along with error sensitivity characteristics comparable to the IBM method.

Figures 16 and 17 illustrate the effects of adaptive filtering when applied to the example system for step and sinusoidal excitations respectively. Clearly, in all cases reduced error and reduced error sensitivity to the sampling interval is shown for simulation time intervals less than 0.25 second. As would be expected,

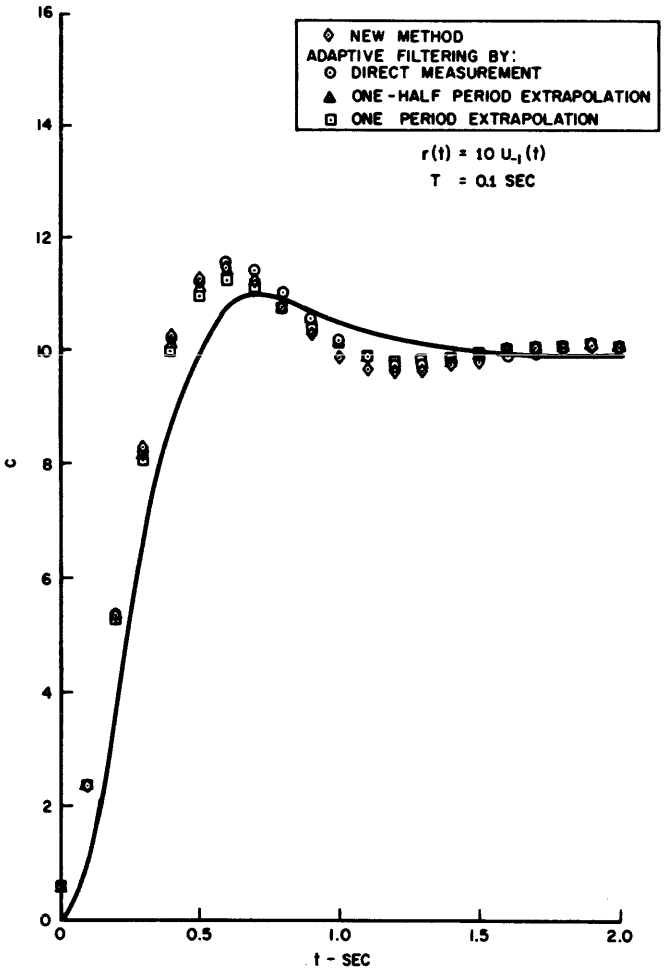


Figure 12—Step response of the example system for variations of the new simulation technique

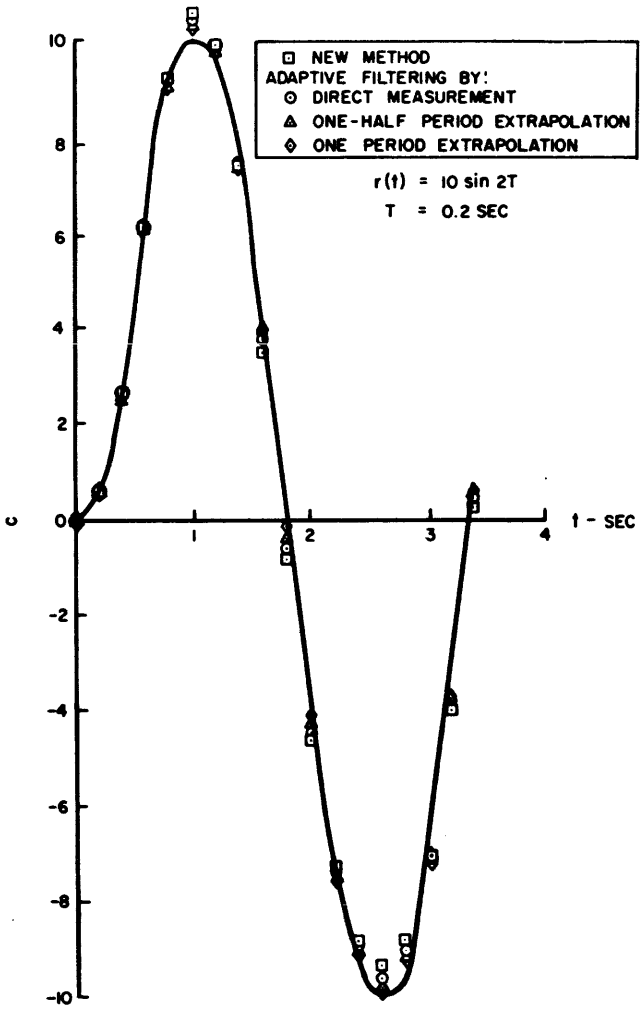


Figure 13—Sinusoidal response of the example system for variations of the new simulation technique

for small sampling intervals, the error decreases as transitions are made from a fixed filter, to an adaptive filter with direct (but delayed) process identification, to an adaptive filter with one-half period prediction, and finally to an adaptive filter with a one period prediction.

To adequately evaluate various digital simulation techniques, it is imperative to consider actual computation time in addition to simulation accuracy and design effort. For the selected example the number of additions, multiplications, and computation time for a single simulation interval is summarized in Table I*

TABLE I—Computation Time of a Single Simulation Cycle for the Selected Example

METHOD	Additions	Multiplications	Computation Time
TUSTIN	7	6	187.2μsec
IBM	6	10	257.6μsec
SAGE-BURT	8	9	256.8μsec
NEW			
non-adaptive	12	13	375.2μsec
½ period extrapolation	18	18	532.8μsec

*In obtaining this tabulation, no recursion formulas are reduced; i.e., it is assumed the response of each block in Figure 9 is desired.

These results are based upon an average of 9.6μ seconds for the floating point and instruction and 20μ seconds for the floating point multiply instruction on the PDP-6 digital computer.[14] One may note from this table that for this specific example the New Simulation Method requires twice the computation time of the classical Tustin method. It is also possible to deduce from Figures 14 and 15 that for all simulation time intervals greater than 0.02 second, the New Simulation Method exhibits less error than the Tustin method with one-half the simulation interval.

A common failing of both the adaptive filtering method and the Sage-Burt method is their inability to cope with large sampling intervals and their possible consequent introduction of instabilities under such conditions. It should be evident that in the former case the difficulty is attributed to parameter identification error, while in the latter, the quasilinearization procedure fails to converge.

CONCLUSIONS

A new method for digital modeling and system simula-

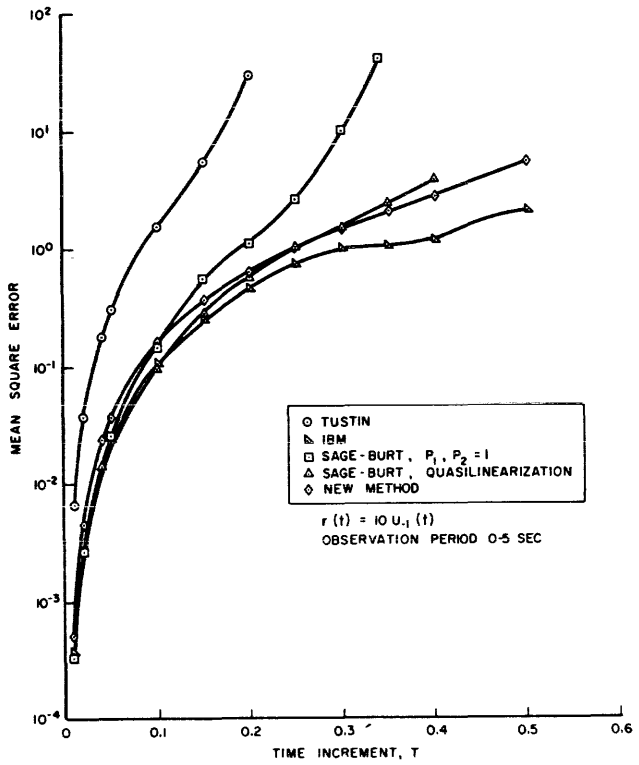


Figure 14—Error analysis of the example system with a step excitation

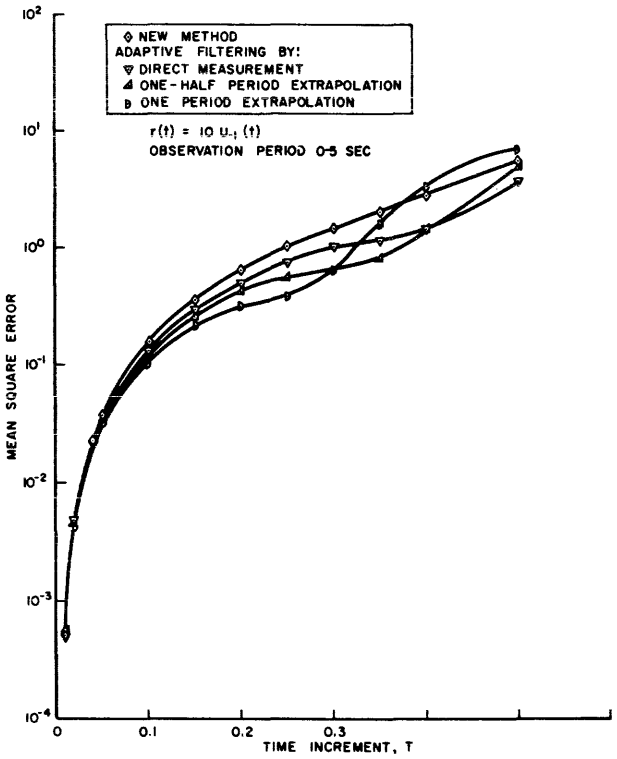


Figure 16—Error analysis of the example system with a step excitation showing the results of adaptive filtering

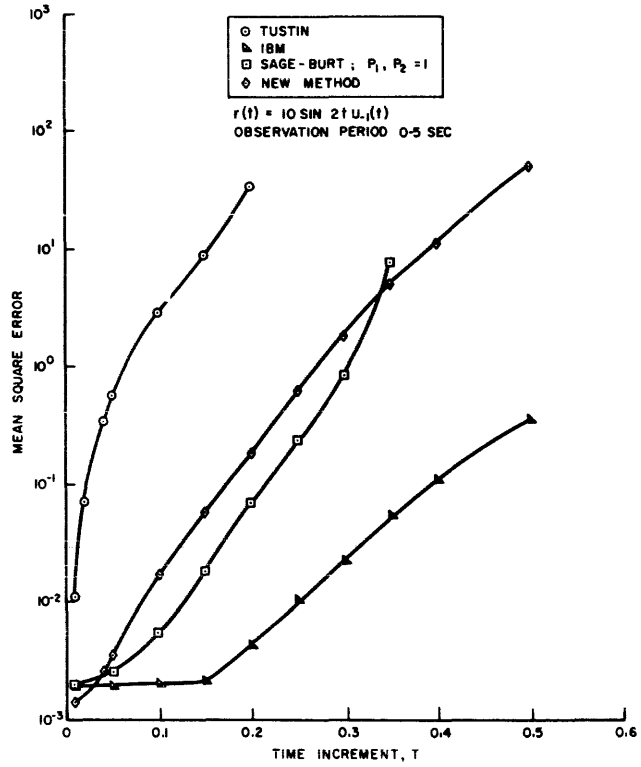


Figure 15—Error analysis of the example system with a sinusoidal excitation

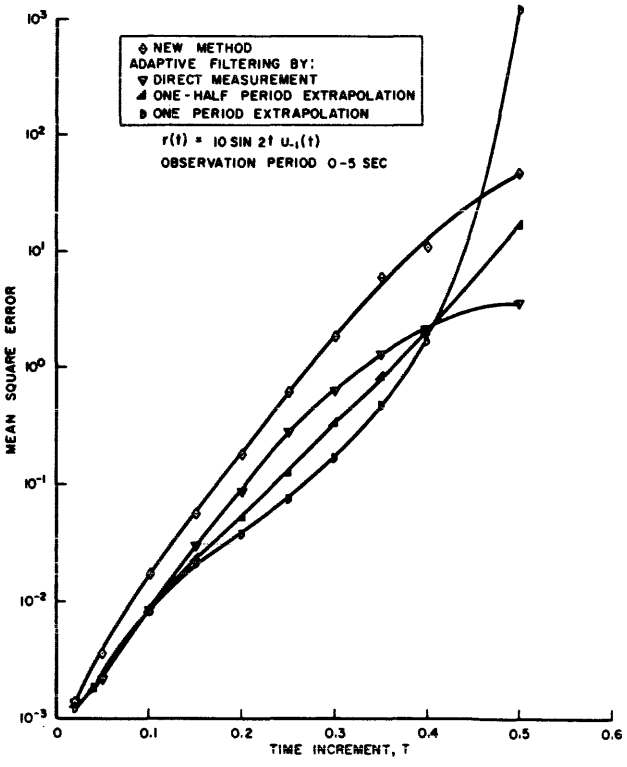


Figure 17—Error analysis of the example system with a sinusoidal excitation showing the results of adaptive filtering

tion has been introduced. The simulations performed utilizing this technique characteristically exhibit a high degree of accuracy improvement over the Tustin method and in many cases accuracy comparable to the IBM or Sage-Burt techniques. The modeling procedure encompasses the simplicity of the Tustin method, involves a minimal amount of tailoring, and does not necessitate the usage of ancillary computer programs for design purposes.

Error analysis indicates that the IBM method possesses the desirable characteristics of low simulation error and reduced sensitivity to simulation error. The Sage-Burt method and the newly developed method also offer low sensitivity to error for low sampling intervals which correspond to those generally used for adequate information representation. However, only the new technique offers an easily utilized modeling procedure. Although it must be emphasized that these results are for a selected example, the results of other work appear to be in complete concurrence.

For increased simulation accuracy utilizing the basic philosophy of the new simulation method, an adaptive filter whose characteristics are time varying is suggested. An example system simulated using an adaptive filter provides both reduced error and error sensitivity to the simulation time interval. However, as may be inferred from the presented results, in certain cases the utilization of an adaptive filter may limit the real-time simulation capability.

ACKNOWLEDGMENTS

The author wishes to thank Mr. R. Belluardo for the encouragement he provided.

REFERENCES

1 J B SCARBOROUGH
Numerical mathematical analysis
The Johns Hopkins Press Baltimore Md 1958

2 R W HAMMING
Numerical methods for scientists and engineers
McGraw-Hill Book Co Inc New York 1962

3 A TUSTIN
A method of analyzing the behavior of linear systems in terms of time series
Journal IEE vol 94 pt II-A
May 1947 pp 130-142

4 A MADWED
Number series method of solving linear and non-linear differential equations
Rept No 6445-T-26 Instrumentation Laboratory Massachusetts Institute of Technology Cambridge Mass April 1950

5 R BOXER S THALER
A simplified method of solving linear and non-linear systems
Proc of IRE vol 44 Jan 1956 pp 89-101

6 C A HALIJAK
Digital approximation of the solutions of differential equations using trapezoidal convolution
Rept No ITM-64 Bendix Systems Div Ann Arbor Mich August 1960

7 J T TOU
Digital and sampled-data control systems
McGraw-Hill Book Co Inc New York 1959

8 J HURT
New difference equation technique for solving nonlinear differential equations
Proceedings of the 1964 Spring Joint Computer Conference pp 169-179

9 M FOWLER
A new numerical method for simulation
Simulation vol 4 May 1965 pp 324-330

10 A P SAGE R W BURT
Optimum design and error analysis of digital integrators for discrete system simulation
Proceedings of the 1965 Fall Joint Computer Conference pp 903-914

11 A P SAGE S L SMITH
Real-time digital simulation for systems control
Proc IEEE vol 54 Dec 1966 pp 1802-1812

12 R E BELLMAN R E KALABA and R SRIDHAR
Adaptive control via quasilinearization and differential approximation
Rand Corporation Research Memorandum RM-3928PR Nov 1963

13 R E BELLMAN R E KALABA
Quasilinearization and nonlinear boundary-value problems
Rand Corporation Research Report R-438-PR June 1965

14 *Programmed data processor-6 handbook F-65*
Digital Equipment Corporation Maynard Mass 1965

Extremal statistics in computer simulation of digital communication systems*

by MISCHA SCHWARTZ and STEVEN H. RICHMAN
Polytechnic Institute of Brooklyn
Brooklyn, New York

INTRODUCTION

With the advent of the digital computer it is becoming more and more common to simulate the operation of rather sophisticated communication systems on the computer. The performance of systems under various types of operating conditions may be evaluated quite readily and economically prior to actual field usage.

The average error rate serves as a very common measure of performance for digital communication systems with a probability of error of less than 10^{-5} a desirable goal in most system design. Such extremely low error rates pose a real measurement problem, however. Generally with Monte Carlo simulation techniques used one would require data samples of the order of at least 10 times the reciprocal of the error probability to make valid performance estimates, leading to costly and time-consuming simulation runs.

The question of more efficient estimation of low error probabilities in communication system simulation is thus an extremely important one. We report here on encouraging results indicating that the methods of Extremal Statistics may reduce the data requirements in many simulation experiments by at least an order of magnitude.

Major applications of the field of Extremal Statistics[1] have heretofore been made primarily to such areas as Flood Control, Structural design, meteorology, etc. It is only relatively recently that applications to communications have begun to be made, with primary emphasis thus far on the analysis of data obtained from existing systems.[2,3] Thus, use has been made, in analyzing these data, of special plotting paper developed by Gumbel.[1] Our approach has differed in assuming from the beginning that all calculations were to be made by a high speed computer, that time was of the essence, and that we were interested in applying the theory to the simulation of broad classes of systems.

Extremal statistics is, as the name implies, concerned with the statistics of the extrema—maxima or minima—of random variables. As such it deals with the occurrence of rare events, exactly the problem encountered in simulating low error rate communication systems. It is found[1] that asymptotically (i.e., very large sample numbers of the random variable under study) many of the most common probability distributions follow a simple exponential law when expanded about an arbitrary point on their tails. Thus, the probability of exceeding a specified value or threshold x_0 assumes asymptotically the form

$$P_e = P(x>x_0)\,\frac{1}{n}\; e^{-\alpha_n(x_0-u_n)} \qquad (1)$$

The number n represents the number of samples used with α_n and u_n constants, depending on n, and the actual distribution of the random variable. In particular, the probability of exceeding u is $\frac{1}{n}$, providing another definition of u. Figure 1, for an arbitrary probability density function f(x), shows equation (1) graphically.

The gaussian (normal), Rayleigh, exponential, and Laplacian distributions are among the examples of the asymptotically exponential distributions. All of these distributions may be approximated by Equation (1) in the vicinity of u. How far from the vicinity of u one may move depends of course on the actual underlying distribution and the particular point (u) one expands about. As an example Figure 2 compares the exponential approximation to the actual probability of exceedance of x, P_e, for a gaussian density function. Here n has been arbitrarily chosen as 100. The actual probability P_e and its exponential approximation are then matched at $P_e = 10^{-2}$. It is readily shown that the point u_n about which one expands, is 2.32, and $\alpha_n = 2.68$.

*The work reported in this paper was supported under NSF Grant GK-527.

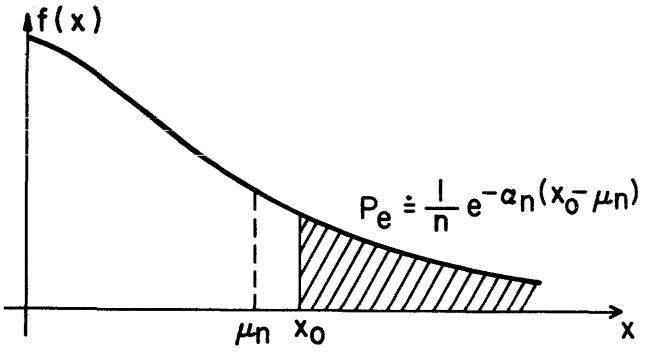


Figure 1 – Exponential approximation to probability of error

Comparing the probability of exceedance P_e for the actual gaussian and its exponential approximation, as plotted in Figure 2, it is apparent that the two are within 25% of one another at $P_e = 10^{-3}$ and differ by 50% at $P_e = 10^{-4}$.

This then points up the significance of the extremal statistics approach: if one is interested in estimating small probabilities of error, say of the order of 10^{-3} or 10^{-4}, it may be possible instead to first estimate much higher probabilities, say 10^{-2} in the example of Figure 2. If the exponential approximation is valid one should then be able to extrapolate down to the desired probability. Instead of the usual number of samples required to estimate P_e, say $10/P_e$, one can then work with a much smaller number n.

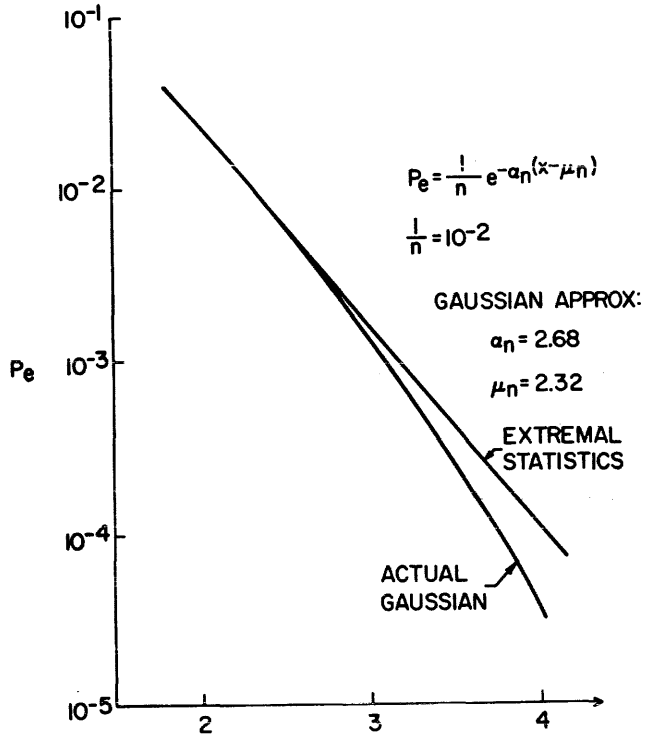


Figure 2 – Exponential approximation to gaussian statistics

There is of course one major problem, however. Since the underlying density function f(x) is in general unknown, or difficult to evaluate in the complex systems of interest to us, the two parameters α_n and u_n are unknown as well, and must be estimated. In the next section we discuss various ways of estimating α_n and u_n, and results of computer runs for two simple density functions, the gaussian and the exponential. The results are quite encouraging: even with additional samples needed to estimate α_n and u_n, one can still save at least an order of magnitude in the total number of samples required to estimate a given probability of error the traditional way.

In the final section, we discuss the computer simulation of two specific feedback communications systems for which probabilities of error have been estimated quite successfully using extremal statistics. (One of these systems is an example of one for which actual calculations or probabilities of error are quite difficult to make. In the example shown only bounds on the error have been obtained and the simulation results check these quite closely.)

Extimation of extremal parameters

We discuss in this section the use of extremal statistics to estimate small probabilities of error in the case of two known distributions, the exponential and the gaussian. The problem here is twofold: to first estimate the extremal parameters α_n and u_n, then to determine, using these estimates, how well the actual probabilities at the tails are estimated.

The exponential density function normalized to unit variance is given by

$$f(x) = e^{-x}u(x) \tag{2}$$

u(x) the unit step function, while the gaussian density function, again normalized to unit variance, is of course given by

$$f(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2} \tag{3}$$

One would expect rather good estimates of the probability at the tail for the exponential density function since it is already in the asymptotic form of equation (1). In the gaussian case, as pointed out in the previous section and as illustrated for one case in Figure 2, it is theoretically possible to extrapolate as much as two orders of magnitude away from the starting point 1/n before the quadratic exponential behavior of the gaussian function takes over and produces significant deviations away from the linear exponential behavior of extremal statistics.

The actual experimental behavior of the exponential approximation depends critically on the estimation of the two parameters α_n and u_n. To determine these we use the fact, as demonstrated by Gumbel,[1] that they are intimately connected to the asymptotic statistics of the extrema (maxima) of the random variable x. Specifically, if one generates n independent samples of x the probability density function of the *largest* (maximum), x_m, of the n samples is asymptotically $(n\to\infty)$ given by

$$P_n = \alpha_n \exp[-y - e^{-y}]$$
$$y \equiv \alpha_n[x_m - u_n] \tag{4}$$

Equation (4) is found to be valid for a wide class of density possessing exponential behavior at the density functions possessing exponential behavior at the tails, with the exponential, gaussian, and Rayleigh functions typical examples.

From Equation (4) it is readily shown that $u_n(n\to\infty)$ is a measure of the mode of $p_n(x_m)$, while $\alpha\frac{1}{n}(n\to\infty)$ is a measure of the dispersion. Specifically, one finds, using Equation (4), that

$$\frac{1}{\alpha} = \frac{\sqrt{6}}{\pi}\sigma_m \tag{5}$$

and

$$u = E(x_m) - \frac{\gamma}{\alpha} \tag{6}$$

Here σ_m is the standard deviation of the maximum (extremal) values, x_m, of x, $E(x_m)$ the expected value of these maxima, and $\gamma = 0.5772$ is just Euler's constant.

It is thus apparent that to estimate α_n and u_n one must first ensure $n >> 1$ (This is why Equation (1) is applicable to the *tails* of density functions, where $P_e << 1$), and then generate sufficient samples of the random variable x under test to measure their statistical properties. If N samples of the largest value of x in a group of n are to be made available this implies repeating the experiment nN times in all. It is the total number nN that is to be compared to the usual number $10/P_e$.

From the form of Equations (5) and (6) one would expect that for n and N large enough, good approximations to α_n and u_n would be obtained by averaging appropriately over the N samples of the maxima available. As noted later this was in fact found to be the simplest and most accurate procedure in actual experimentation with the computer. This estimation procedure is portrayed in Figure 3. There, as an example, $n = 10^3$ and $N = 10$ are chosen. The total number of independent samples or repeats of the computer simulation involved would thus be $nN = 10^4$. The resultant output samples would be grouped into $N = 10$ groups of $n = 10^3$ samples each. The largest sample, x_i, in each group would then provide $N = 10$ samples with which to estimate α_n and u_n.

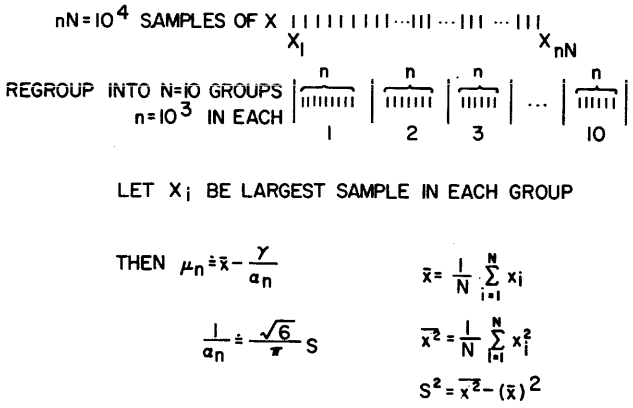


Figure 3 – Estimation of α and u_n

There is a tradeoff possible between n and N, given the fixed number of repetitions nN. Thus decreasing n decreases the range over which one would theoretically expect the asymptotic exponential approximation to hold (assuming perfect knowledge of α_n and u_n), but allows better estimation of α_n and u_n as N increases. Some analysis of the optimum choice of n and N has been carried in a recently completed doctorial thesis.[4]

In the actual computer simulations carried out n was taken as 500, $N = 20$, so that a total of 10,000 actual repetitions of the different experiments tried were performed. Normally this would provide relatively accurate estimation of probabilities of error as low as 10^{-3}. We were interested in extending the estimation to 10^{-4} and 10^{-5}.

As noted previously, an obvious initial estimate for α is to replace σ_m in Equation (5) by the sample standard deviation s, using the $N = 20$ samples available of the extrema. Similarly a first estimate for u is to replace $E(x_m)$ in Equation (6) by the sample mean $\bar{x}_m$ (again using the 20 extremum samples). (These are the procedures suggested in Figure 3.) Although the sample standard deviation is in general a rather poor estimator of the statistical standard deviation [var $(s^2) = 2\sigma_m^4/N$ for gaussian statistics] the experimental results obtained were surprisingly good for both the exponential and gaussian distributions: the estimates of α came within 10% of the true values in both cases. Similarly the estimates of u came within 3% of the true values, well within the confidence limits set by using the sample mean estimates.

These initial estimates were compared experimentally with several other approaches:

1. Initial estimates obtained using approximate solutions of the maximum likelihood estimates of α and u (these involve fourth sample moments).
2. Initial estimates obtained by an approximation procedure attributable to Kimball.[1]
3. Initial estimates obtained by using the first and second extremal values. (One might expect that the second extremal values should provide some information on α or u, and are available in a simulation run anyway.)

Using Equation (4) the maximum likelihood estimates of α and u are found to be given in terms of the N extreme (maximum sample values x_j, $j = 1 \ldots N$, by solution of the following two equations:[1]

$$\frac{N}{\alpha} = \sum_{j=1}^{N} x_j[1 - e^{-y_j}] \tag{7}$$

$$N = \sum_{j=1}^{N} e^{-y_j} \tag{8}$$

Here $y_j \equiv \alpha(x_j - u)$.

These equations cannot be solved specifically for α and u, but may be either iterated or approximated in various ways.

The two equations may be combined to eliminate u, resulting in the following equation in α alone.

$$\frac{1}{\alpha} = \frac{-\sum_{j=1}^{N}[x_j - \bar{x}_m]e^{-\alpha(x_j - \bar{x}_m)}}{\sum_{j=1}^{N} e^{-\alpha(x_j - \bar{x}_m)}} \tag{9}$$

Here $x_m = \frac{1}{N}\sum_{j=1}^{N} x_j$ is the sample average of the N extrema (maxima). Equation (9) lends itself readily to both approximation and iteration. (Note that since $\frac{1}{\alpha}$ is positive this equation indicates that values of x_j less than $\bar{x}_m$ will occur more frequently than those greater than $\bar{x}_m$. This is of course due to the asymptotically exponential character of the distribution of the extrema.)

An approximation of $\frac{1}{\alpha}$ suitable for simple calculation is obtained by expanding Equation (9) in an infinite series, retaining the first few terms, and reordering the resultant equation. This provides the following approximate estimate of α:

$$\alpha^2 = \frac{2s^2}{u_4}\left[\sqrt{1 + \frac{2u_4}{(s^2)^2}} - 1\right] \tag{10}$$

Here $s^2 = \frac{1}{N}\sum_{j=1}^{N}(x_j - \bar{x}_m)^2$ is the sample variance of the N extremal values, and $u_4 = \sum (x_j - \bar{x})^4$ is the sample fourth moment. (It is assumed, in deriving Equation (10), that the third sample moment $\sum (x_j - \bar{x})^3$ is zero. Although true for symmetrical distributions such as the gaussian, this moment is, on the average, negative for the extremal statistics. The assumption simplifies the equation considerably, however.)

The maximum likelihood Equations (7) and (8) for α and u, were iterated several times using as first estimates the different approximations noted earlier:

1. The asymptotic Equations (5) and (6) for α and u, with the standard deviation σ_m replaced by the sample deviation s and the expected value $E(x_m)$ replaced by the sample mean $\bar{x}_m$,
2. Equation (10) for α,
3. A technique attributable to Kinball[1].

Interestingly it was found by computer experimentation that Gumbel's asymptotic estimates of Equations (5) and (6) were both the simplest and fastest to implement on the computer, and in all cases tested also came closest to the true values on the initial try. (All four methods produced very nearly the same results after iteration.)

The first estimate using Equation (10) was also fairly close to the expected value of α on all experiments performed. The Kimball method noted was, however, quite inaccurate.

Using the best estimates of α and u obtained, extremal statistics were in turn used to estimate error probabilities for both the gaussian and exponential distributions. Results obtained compared favorably with those predicted theoretically using extremal statistics: with the total number of samples used, nN = 10,000, probabilities in the vicinity of 10^{-4} were estimated within 20%, while for probabilities as low as 3×10^{-5} the estimates were within 60%. (For low probabilities such as these, 100% variations still provide significant information. These compare with the more common statistical procedure of using 10,000 samples to estimate probabilities of the order of 10^{-3}, or at best perhaps 5×10^{-4}.)

The usefulness of the extremal statistics approach for estimating small probabilities is further seen in comparing confidence intervals with these obtained for the more common procedures. Since in the extremal statistics case it is the estimation of the parameter α that is more subject to large variations for N = 20 samples one may assume the parameter u fairly well estimated on the basis of the same 20 samples. Confidence intervals on the probability of error

curves may then be set using the standard deviation of the estimate of α. For the gaussian distribution and n = 500 samples it is then found that the $\pm 1\sigma$ confidence interval for α corresponds to the following bounds on the measured probability:

Theoretical probability	Upper bound	Lower bound
10^{-3}	1.1×10^{-3}	0.7×10^{-3}
10^{-4}	2×10^{-4}	0.5×10^{-4}
10^{-5}	5×10^{-5}	0.4×10^{-5}

Although the usual Tschebychef inequality confidence intervals (based on estimating probabilities by relative frequencies of occurrence) give roughly the same bounds at 10^{-3}, the lower bound deteriorates rapidly, below this probability, going to 0 at 10^{-4} samples available. One can of course say nothing, except possibly be extrapolation, of probabilities in the vicinity of 10^{-5} using the relative frequency approach.

Application to feedback communication systems

In the previous section we have discussed methods of estimation of the two extremal parameters α_n and u_n, using the known gaussian and exponential density functions to determine the most efficient ways of accurately estimating these parameters.

We now discuss the application of the method of extremal statistics to the simulation of digital feedback communication systems. Two systems of considerable interest currently have been investigated. One is a binary signalling system, the other an M-ary PAM type system, with the information transmitted as one of M possible amplitude levels. In both cases white gaussian noise is assumed added during forward transmission (this could be thermal noise introduced at the receiving antenna, front end receiver noise, or a combination of the two in general). The feedback path is assumed error free. Errors in signal detection due to the noise may occur and it is desired to estimate the probability of error in both systems.

Both systems may have potential usefulness in space-ground communications. In both cases the information is to be transmitted from space to earth. The forward transmission path is thus limited in power. It is assumed a feedback path from ground to vehicle with much larger power capability is available, so that the effects of noise may be neglected over this path. (Further computer simulation is planned to investigate the effect of noise on the feedback path. as well as other signal disturbances such as fading for example.) Such feedback systems hold great interest in the statistical communications field currently because of their expected high performance (low probability of error) in noise with a minimum of coding effort required.

The binary system investigated represents the application of sequential decision theory of statistics[5] to statistical communications. The operation at the receiver may be visualized by referring to Figure 4. Here v represents a received sample of the composite signal plus noise. As an example if one binary signal has a received amplitude of +A volts, while the other is −A volts (this is the example of bipolar signalling), the resultant probability density functions $f_1(v)$ and $f_2(v)$, respectively, appear in Figure 4.

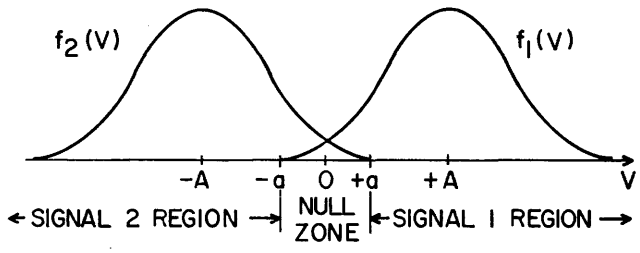


Figure 4 – Received signal space, binary feedback

In normal binary signalling one would declare signal 1 received if the received signal plus noise sample were to exceed 0, and signal 2 received if the received signal plus noise fell below zero. For small signal-to-noise ratios (SNR), as normally encountered in deep space-ground communication, the resultant probability of error would be intolerably high. To improve the performance a null zone of width $\pm$ a about 0 is set up. If the composite received signal falls in this region a decision is deferred and the transmitter asked, via the feedback path, to repeat the signal. The first composite signal sample is then stored and added to the second received sample after transmitter repetition. The two combined received samples are then tested and a decision made only if the sum exceeds +a or falls below −a. A third repeat is requested if the sum again falls in the null zone.

This procedure is repeated until the combined received samples fall outside the null zone. (To prevent the system from cycling indefinitely or for too long a time, the number of transmissions may be truncated after a specified interval and a definite decision made.) Such a sequential procedure with a statistically variable number of transmissions may be shown to asymptotically provide 50% decrease in average transmission time over an equivalent system without feedback with a fixed number of repeats[5] this for the same SNR and probability of error.

Although the scheme outlined is relatively simple to describe the analysis is quite complex, since the received signals summed are these falling in the null zone only. The statistical behavior of the composite random variable representing the sum of such received signals is difficult to determine and the resultant

probability of error not easily found. In fact asymptotic results only for the probability of error are available; those for small SNR and a large average number of transmissions, or those for high SNR and very small numbers of transmissions. The region in between can only be estimated by extrapolation or by bounding techniques. Computer simulation of such a system is, however, simple to perform. For small probabilities of error, however, the computer time involved can become quite large so that extremal statistics is an obvious answer.

The methods of extremal statistics were therefore used, in a computer simulation of this system, to estimate the probability of error. nN = 10,000 total repeats were used in each simulation, N = 20 for estimating α and u, n = 500 for the actual determination of the probabilities. Results are shown in Figure 5. Note that the experimental (simulation) points follow quite closely an approximate performance line obtained by extrapolating from Wald's asymptotic results, valid for large numbers of transmissions and small probabilities of error.[6] The curves of Figure 5 are for a SNR of ½. A reference curve, that showing the performance of a one-way system with no feedback and a fixed number of repeats, is included. Note that the two curves differ approximately by a factor of two in the transmission time, as would be expected from Wald's asymptotic results.

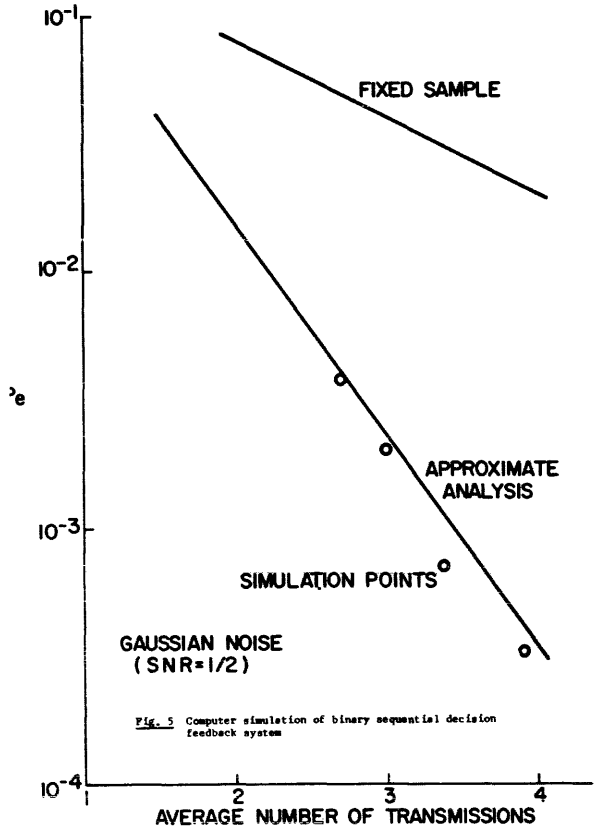


Fig. 5 Computer simulation of binary sequential decision feedback system

Figure 5 – Computer simulation of binary sequential decision feedback system

The second feedback communications system simulated, for which extremal statistics were used in estimating system performance (probability of error), was one first suggested and analyzed by J. P. Schalkwijk.[7,8] A simplified block diagram appears in Figure 6. The information to be transmitted consists of one M amplitudes, $\Theta = \frac{j}{M}$, j = 1, 2, ... M (the amplitudes are normalized to a range of 0 - 1 for convenience). T seconds are available for transmission and it is assumed that N signal transmissions, each lasting $\frac{T}{N}$ seconds, are made. (Note that this fixed transmission time scheme contrasts with the sequential binary scheme, requiring a variable transmission time, just discussed.)

On the k^{th} transmission the signal transmitted is of the form

$$s_k = g_k(\Theta - \hat{\Theta}_{k-1}) \tag{11}$$

with g_k a variable but known gain factor, and $\hat{\Theta}_{k-1}$ the receiver's maximum likelihood estimate of Θ, based on the first (k−1) transmissions. The receiver transmits $\hat{\Theta}_{k-1}$ to the transmitter via a noiseless feedback path. This system is of the information feedback type, since actual information as to the received signal plus noise is fed back to the transmitter.

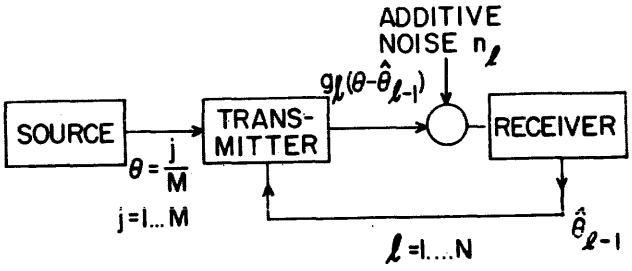


Figure 6 – M-ary information feedback system, N respects

On each successive transmission the receiver's estimate of the correct signal, Θ, improves so that $\hat{\Theta}_k$ approaches Θ more closely. After N repeats the nearest $\hat{\Theta}$ to Θ_N is chosen as the correct signal. It may then be shown[7,8] that the probability of error goes essentially as

$$P_e \sim \exp -\left[\frac{3}{2} e^{2(C-R)T}\right] \tag{12}$$

for large T, and R<C. Here the gain g_k of the k^{th} transmission has been optimized to provide minimum probability of error. The parameter R is the rate, in bit/second, of signal transmission:

$$R = \frac{\log_2 M}{T} \text{ bits/sec.} \tag{13}$$

and C is the so-called channel capacity in bits/sec. for this gaussian channel:

$$C = W \log_2 \left[1 + \frac{Pav}{n_0 W} \right] \tag{14}$$

Here $W = \frac{N}{2T}$ is the channel bandwidth, P_{av} is the average power available from the transmitter, and $n_0/2$ is the white noise spectral density.

(Equation (14) is exactly the channel capacity of the gaussian channel first obtained, using a random coding argument, by Claude Shannon. It represents the maximum rate of transmission, in bits/sec., available for error-free transmission over such a channel. Schalkwijk's scheme is the first specific signalling scheme known to obey Shannon's previously ideal signalling law, and this accounts for the interest stirred up in it.)

A computer simulation of this system was carried out, with the probability of error found using extremal statistics. Here, because of the relative complexity of the system, only a limited amount of simulation could be carried out. Specifically, for N = 25 repeated signal transmissions and R/C = 0.5, the probability of error was found to be 0.037, while for 50 signal transmissions, the probability of error was reduced to 2.3×10^{-3}. Both of these numbers compare favorably with results of an exact analysis, possible here, the complexity of the system notwithstanding, because of the additive *white gaussian* noise assumed and linear operations at both transmitter and receiver. (Had the noise been nonwhite or nongaussian, or had some simple nonlinear operations been included exact analysis would have been out of the question. Yet the simulation complexity would not have been substantially different, pointing out the utility of simulation here.) For the estimation of the probabilities here, 20 extremal values were again used to estimate α and u, while n = 50 samples were used in the estimation of probability. The total number of samples per simulation was thus 1,000.

REFERENCES

1 E J GUMBEL
Statistics of extremes
Columbia University Press New York 1958

2 E C POSNER
The application of extreme-value theory to error-free communications
Technometrics vol 7 no 4 November 1965 pp 517-529

3 J C ASHLOCK E C POSNER
The application of the statistical theory of extreme values to spacecraft receivers
National Telemetering Conference May 1966

4 M MUNTNER
Error control on real channels
Doctoral Dissertation Department of Electrical Engineering Polytechnic Institute of Brooklyn June 1968

5 A WALD
Sequential analysis
John Wiley New York 1947

6 M HECHT
Non-optimum sequential detection techniques
Doctoral Dissertation Department of Electrical Engineering Polytechnic Institute of Brooklyn June 1968

7 J P M SCHALKWIJK T KAILATH
A coding scheme for additive noise channels with feedback-part I - no bandwidth constraint
IEEEE Transactions on Information Theory vol IT-12 no 2 April 1966 pp 172-182

8 J P M SCHALKWIJK
A coding scheme for additive noise channels with feedback-part II - bandlimited signals
IEEE Transactions on Information Theory vol IT-12 no 2 April 1966 pp 183-189

MUSE: A tool for testing and debugging a multi-terminal programming system

by E. W. PULLEN and D. F. SHUTTEE
Control Data Corporation
Palo Alto, California

INTRODUCTION

Current literature on multi-terminal time sharing systems has a great deal to say about the macroscopic relationship of their parts and the statistics of their performance. However, very little is written about the efforts expended in putting them together or making them work efficiently and reliably, in the hands of the users. Traditionally checkout and testing of multi-terminal systems have assumed a method of an on-site marathon. Testing of the product outside the customer's shop has been virtually non-existent.

A major problem facing the system engineer in the checkout of such systems is just getting all the pieces together. In a large system with 64 terminals, for instance, it is not economically practical to assemble this equipment and make it available for several months, even when floor space and other logistic problems have been solved. To man this number of terminals, coordinate the activity at each one, and reproduce the system problem in such an environment is a virtual impossibility. Added to this is the problem of subjecting a multi-terminal system to various job environments and various types of users.

The system engineer could stand like the conductor of some great symphony orchestra. With baton poised, he would strike the note for a worst case loading of the system by his Teletype operators. But this would be unrealistic and uneconomical as the problem of checking a number of potential worst case conditions would still remain, as would the problem of repeating the results in case of system malfunction. A subtle, but nonetheless serious problem is determining the marginal performance of a system when one more terminal is added or a significantly different work load is imposed through one terminal. Effects of changes in background computing activities must also be superimposed on the multi-terminal environment.

The traditional approach is to do a best job with a few terminals. The system is then installed in the customer's environment with a larger—but probably not the maximum — number of terminals. Debugging and testing proceeds on-site until the customer is satisfied with the quality of system performance. The trauma associated with this approach is similar to bringing up ten or twenty little batch systems all at once, all interconnected, only worse.

The customer bears the burnt of equipment costs. He pays for terminals and lines he cannot put to use and concurrently he pays for computing time and machine hours which could be put to other uses. When the system programmers eventually leave—all too soon as far as the customer is concerned—there is little information on the effects of adding one, five, or ten more terminals to the system. When major design flaws are encountered during on-site debugging, the emphasis is on a quick fix. Often, the result is a very complicated product which cannot be readily updated or improved.

Unfortunately the end product may not be readily usable for another customer even when hardware configurations are similar.

The obvious solution is to do the job right in the first place. Time sharing systems must be balanced in their resource requirements (memory, disk space, data channel access, ets.) just as carefully as any power plant fly-wheel. Simulation of an entire system is an effective tool for testing and debugging in the early stages of planning and design. This type of simulation is most useful in defining the problems. Simulation, however, of interaction between user and parts of the system has been overlooked. Designs have been frozen before critical resource conflicts become apparent. Time-critical and resource-critical conditions fall by the way only to resurface in the customer's shop. At this point, a hasty fix can change a well designed system into a poorly constructed one.

Another submerged problem is associated with making large amounts of terminal and multiplexing

hardware available in the early development stages even when such equipment can be manned efficiently. This is the problem of hardware-software lag. Communications and terminal hardware designs must be fairly well defined before even a few prototypes can be made available for software development. When many terminals are required, flexibility in the hardware design is many times diminished. The software designer is once again confronted with unilateral hardware design decisions and an integrated system design is again frustrated.

Considerations such as the foregoing resulted in an attempt to simulate the user and communications environment in a program called MUSE (Multi-User Environment Simulator). This program allows the multi-terminal system to be extensively exercised independent of multiplexing and terminal hardware. As many as 64 users may be simulated at one time. This program, and to some extent the time-sharing system it tested, are the subjects of this paper.

The tested system

The MUSE program represents a joint effort of the RESPOND (Remote Sharing and Processing of Data) development project and the Quality Assurance Department. The purpose of this effort was to develop a tool for debugging and testing the CONTROL DATA 6000 Series TTY RESPOND time sharing system. The RESPOND system was designed to perform the following tasks:

- Define, store and retrieve files with respect to disk storage devices at the computer site
- Create programs in the form of executable files
- Submit these files for processing in the batch environment controlled by the SCOPE operating system
- Allow output results from the SCOPE processing to be placed on permanent disk files accessible to the user through his remote terminal

These capabilities co-exist with the full range of batch processing operations provided by the SCOPE operating system.

TTY RESPOND is a standard Control Data Corporation product, based on a special system developed by Control Data Corp. for the University of Texas and Aachen University in West Germany. The system operates within a hardware configuration which includes CONTROL DATA 6400, 6500 or 6600 computers with a minimum of 65 thousand words of central core storage (60 bits per word). A Control Data Corp 6676 Communications Terminal Controller acts as the multiplexer between the computer data channel and up to 64 Teletype terminals. These terminals may be connected to the multiplexer over DATA PHONE Service or directly through a standard signal and data set coupler interface.

A resident program includes the command processing code and housekeeping tables for the terminals. Scratch storage is requested and released dynamically from the SCOPE system as buffers are required for implementing the commands. The amount of core used is dependent upon the number of active terminals in the system, the type of activity at each terminal, and the amount of core available to the system. A request for core is placed in a queue if it cannot be satisfied. Queues are also maintained for disk requests, SCOPE processing operations, and output operations. Operation of these queues under a many-terminal environment was of primary interest during program debugging and testing.

Figures 1 through 4 illustrate the basic phases of RESPOND's operation. The Circular-Stack acts as an input delay-line buffer.

The Termstak is the basic element in coordinating the terminal's activity. The Jungle provides dynamically allocated scratch storage for terminal activity. The Job Table acts as the coordinating mechanism between the user and his disk files. Buffers for file management and output operations reside in the area labeled use file information.

Simulator design factors

Many factors could be considered in this development, since the application was rather new, effort was concentrated on the following six items:

Duplication of real-time characteristics of Teletype terminal, telephone and multiplexer system

Minimization of effect of simulator on the rest of the system, including the SCOPE processing environment

Flexibility in defining number of terminals and relationships between terminal activities

Flexibility in defining command strings and input data sent from the simulator to RESPOND system

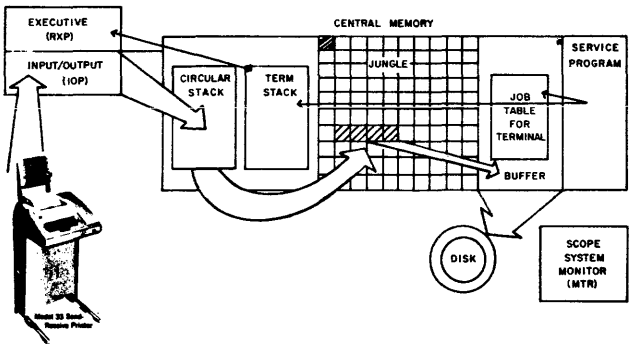
Control of terminating or overriding simulator performance from the computer console

Capability to move from a simulated environment to actual Teletype operations without disrupting continuity of activity at each terminal

These capabilities provided the potential for emulating a variety of job environments and user types as well as worst-case conditions. The override and simulator exit features were of particular importance in the debugging of both the simulator and RESPOND system.

Emulation of worst-case conditions was particularly important in testing and debugging the RESPOND

product. Worst-case conditions are defined as those which place the heaviest simultaneous load on the RESPOND queueing structure for resources. Two general sets of problems appear under these circumstances:



IOP Program

Periodically samples multiplexer
Receives 12-bit character into terminal I/O buffer
Separates and packs input characters into right-character buffers
Attaches input identification code and transfers buffer to circular stack

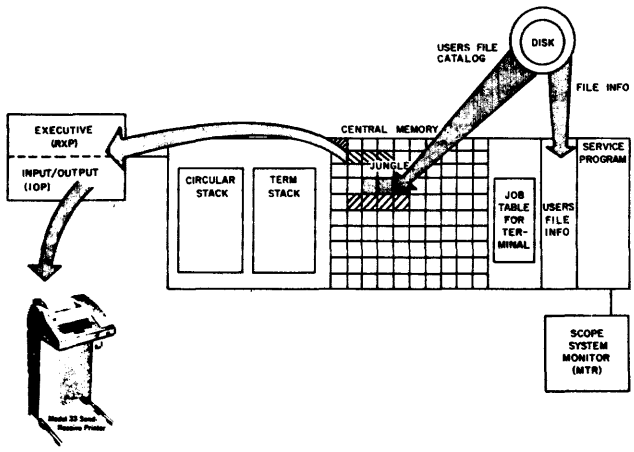
RXP Program

Detects and marks termination of input in TERMSTAK

Service Program

Analyzes input word in circular stack and parcels input into jungle unit assigned to terminal
Determines whether input is a command or incoming data
Updates job table for terminal
If a command, sets flags in TERMSTAK for required processing routine
If incoming data, assembles data in buffer for transfer to the disk and issues disk write request

Figure 1 – Input



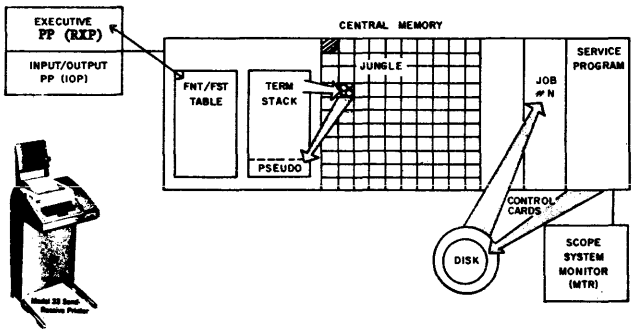
Service Program

Examines TERMSTAK to determine action required
Transfers information for referencing user's files into central memory from the disk
Changes, adds, or deletes information in user's files as required
Updates user's file catalog and job table
Places command-accepted response in jungle units

IOP Program

Sends response to remote terminal

Figure 2 – File maintenace



Service Program

Locates required files
Stores file information and TERMSTAK entry in wait stack in jungle
Transfers information from wait stack to pseudo-TERMSTAK entry when pseudo-terminal becomes idle
Generates a set of control cards and places them on the disk
Requests EXP to enter job in the FNT/FST table

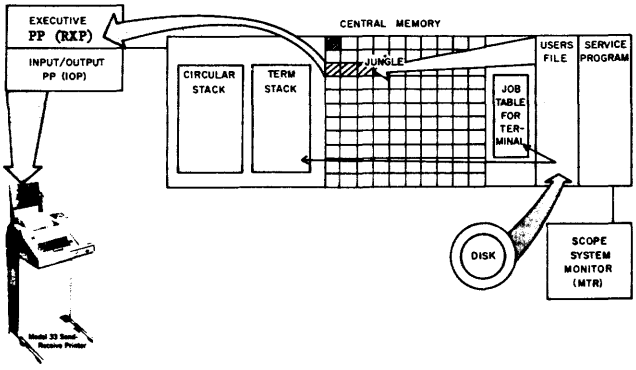
RXP Program

Makes initial entry in FNT/FST table
Intervenes at critical points to update FNT/FST and save output files

System Monitor

Multiprocesses jobs

Figure 3 – Scope processing



Service Program

Sets up output message or monitors terminal job tables for output messages
Calls up as much of output messages as IOP can handle during one output period
Places this output in jungle units
Sets output flag in TERMSTAK

IOP Program

Transfers output (one central processor word at a time) of five peripheral processor words to output buffer
Places output (one peripheral processor word at a time) into terminal I/O buffer
Transmits contents of terminal I/O buffer to terminals

Figure 4 – Output

TIMING PROBLEMS: Many users require execution of the same section of code, use of the same buffer areas, and entry into the same queues simultaneously.

SATURATION PROBLEMS: Queues become very long or full, causing rejection of a request for a resource. Buffers are filled, requiring the system to allocate and link to additional buffers.

For example, when an INPUT or a FILE command terminates, sorting buffers are in demand and, the disk request queue grows. Another example is the LOGOUT-LOGIN operation by which each user's file catalog must be transferred to from the disk.

Four general classes of users can be simulated by the MUSE program:

A polite general purpose user who accepts the rhythm of the system and is interested in exercising its capabilities.

A stereotype user from a particular job environment who represents a specific set of needs such as file management, short FORTRAN compilations or heavy execution requirements.

The impatient user who will run at his speed, rather than the system's speed, and continues relentlessly to enter commands into the system.

The hostile user who is intent on breaking the system.

Single terminal operations often suffice for the polite user and for the initial encounter with the hostile user. Multi-terminal simulation is the only way to adequately gauge system performance for stereotype users, impatient users or a gang of hositle users. Simulator design was oriented toward emulating the impatient, hostile and stereotype users. Since, however, data decks define both the number and type of user a change from any one group to another is easily accomplished.

The polite user certainly cannot be overlooked. Here is the capability to quickly and thoroughly exercise all variations of a command. For example, the RESPOND user may define a format to organize his data input stream. There are 512 variations on the FORMAT command's structure alone. The checkout of these variations is trivial when run through the simulator but extremely tedious when entered by hand more than once. Problems are easily repeatable when simulator testing is done.

Not all problems can be solved or even isolated by using the simulator alone. System malfunctions associated with misplaced files or records require some searching and guessing after an error is detected. This class of problems is accommodated by the design of the interface between MUSE and RESPOND which allows simulator operation to be terminated and activity to be transferred to a Teletype terminal. This mode of operation was also vital in resolving simulator and RESPOND communication problems.

Simulator design features

The MUSE simulator consists of two basic parts. The major part is essentially a FORTRAN program with several small assembly language (COMPASS) routines incorporated for efficient use of central memory after loading. A second part consists of extensions to the RESPOND executive program which allow communication with the simulator as though it were the system multiplexer. This interface program provides automatic switching of activity between simulator and Teletype terminals.

The FORTRAN program resides at one of the control points in the multi programming environment of the SCOPE system. This program requires a maximum of 9600 words of core memory for the 16-terminal version and 18000 words for the 64-terminal version. A character conversion table relates Teletype codes to internal codes in the same manner as in the RESPOND system.

The commands and data input submitted from the simulator to RESPOND are loaded by the simulator as data strings separated by control cards. The command and input strings reside on disk as separate files for each terminal. A simulator input buffer is filled with characters for each terminal from these files. When the buffer is filled, RESPOND performs a parallel read operation bringing in all characters for all terminals as though the simulator were a multiplexer. Output is transferred by a similar fashion from RESPOND to a simulator output-buffer then to a disk file for each specific terminal. Each data card record in a data string represents one discrete command or one data input line. Data strings may be entered by cards or from magnetic tape. Commands can be up to 77 characters per card and data records up to 80 characters per card with as many cards per input record as desired. The last two columns on a command card are used by the simulator to designate the number of times the command should be repeated under certain conditions.

Up to 36 diagnostics and other replies generated by RESPOND are entered into the simulator program's diagnostic table from data cards. When the simulator receives a message from RESPOND, it scans the diagnostic data cards loaded with the program. If a match is not found, the next line of input to RESPOND is issued. If a match is found, coded information entered with each command triggers a variety of actions. This allows a certain degree of recovery

within the simulator from conditions within RESPOND when that system is heavily loaded.

The simulator data control cards provide the following capabilities:

- The TERMINAL ID card identifies a particular part of the data string with one or more terminals. All possible or up to 35 terminals may be specified, individually or inclusively, to use the same data string.
- The TERMINAL NUMBER card specifies the number of the highest terminal to be serviced by the simulator during a particular run. This card is used in allocating fixed buffer space in the FORTRAN program among the terminals. This facilitates the printing of results files and examining flags when only a small number of terminals are simulated.
- The WAIT card is used to inhibit the issuing of a command at any terminal until a LOGOUT has been issued at one or more other specified terminals. This facilitates testing RESPOND supervisory control functions and sharing of files among several users.
- The CONTINUE control card is used in conjunction with coded information on the input data cards. It allows the movement of a selected group of simulator data from the terminal's current input data string to a re-entry file. The data will be issued at a later time to RESPOND.

The TERMINAL ID control card can be used to establish worst-case and balanced-load conditions with a minimum number of data cards. The WAIT control card can simulate supervisor-user interaction. Users may be held in a wait condition until public files have been created. The supervisor may be held in a wait condition for password list changes until several users have logged out. Users may wait for the exit of other users during tests of password and file sharing operations.

The capability provided by the CONTINUE control card is of particular importance. It accommodates the condition where a portion of the command string requires results from SCOPE processing. When these results are not immediately available, this portion of the string (that which ends with the CONTINUE card) can be placed on a reject file and re-issued later. When a diagnostic message received from RESPOND is included in the simulator's diagnostic table, a check is made to determine if the command is to be reissued. Any input statement may be reissued up to 99 times. Command sequences which do not elicit the correct response after a specified number of repeats are pulled from the string and placed in a reject file. When all entries in the command string for a terminal have been passed over once, the contents of the reject file are sent to RESPOND.

The following string of commands is an example of an application of this feature. Assume the user has a FORTRAN program P.

```
COMPILE P LIST PL BIN PB
STATUS                    ∞ 02
OPEN PL
DISPLAY FILE PL
EXECUTE PB INPUT  OUTPUT=RESULT
OPEN RESULT               ∞ 02
DISPLAY FILE RESULT
COPY RESULT TO PRINTER
∞CONTINUE∞
INPUT
```

The COMPILE operation is requested from the SCOPE processor, as are the EXECUTE and COPY operations. If the total system is heavily loaded, compilation may not be complete even after two STATUS requests have been issued and replies received. In such a case, the whole string from the first STATUS through the ∞CONTINUE card is moved to the reject file and the INPUT command will be sent to RESPOND. By the time all the other commands in the data string have been processed the compilation should be completed. The binary file can be passed to SCOPE for execution. If two attempts to OPEN the file resulting from execution are unsuccessful because the execution job has not completed, the string from OPEN RESULT through the ∞CONTINUE card will again be placed on the reject file to be serviced at a later time. This feature gives the simulator extensive recovery power from the time lags in the RESPOND-SCOPE processing cycle. When 40 or 60 terminals are being simulated this recovery becomes very important. This feature also lays the foundation for gathering some statistics on turnaround of terminal jobs in a particular computing environment and with different types of central processors.

A number of features are controllable by the user through sense switch settings while the simulator is running. These features include the ability to:

- Force the next entry in the command string on the RESPOND system after a preprogrammed time-out period
- Stop output of file contents from RESPOND to the simulator and proceed to the next command
- Call for an intermediate printout of the simulator's results file at any point in time without disrupting operations
- Display on the console screen the activities occurring at each terminal (for instance, the LOGIN command issued by the simulator and

the reply from RESPOND). This option may be exercised at any point in time for an arbitrary period of time and then rescinded

- Terminate the simulator run, print the results to that point, and exit the simulator from the computer system at any time

The ability to force commands is used to check input saturation problems and to overcome any breakdown in communications between RESPOND and the simulator. An impatient or hostile user may not wait for a reply from RESPOND to his last command. He may enter another, then another command, stacking up requests within the system. The forcing feature allows simulation of this action. Both the number of times a specific command is repeated and the time-out period required before another command is forced upon RESPOND are readily changeable parameters. Each occurrence of this action is recorded in the terminal's results file.

The ability to arbitrarily stop the output of file contents from the RESPOND system duplicates a real user's capability. This was not the reason, however, for including the feature in the simulator. In testing the commands which access SCOPE processing capabilities and recover files from SCOPE only the first several records are needed to identify a successful operation; however, it is often inconvenient to guess the number required. The STOP switch allows the tester to terminate output when he has enough data to answer his questions and before the full contents of the file is displayed. Quality Assurance testing activities make use of the simulator's ability to print intermediate results printout as well as the final results, and the visual display of terminal activity during the actual simulator run.

Results from the simulator are routed to two devices. All input and output strings for all terminals are displayed on the screen of the computer control console together with the terminal number, and computer clock time when entry was made or response received.

Figure 5 illustrates this type of display when 16 terminals LOGIN and perform a series of balanced COMPILE, INPUT and FILE operations. Through this display the simulator operator is aware of the progress of testing at all times just as if he were viewing the operations of 16 or 20 or even 60 terminals simultaneously. This display forms the basis for the designers' override actions. The contents of the log generated from this display indicate the relative performance of SCOPE and RESPOND in handling remote user's requests.

Another type of output is a listing produced by the simulator at the line printer. This listing includes:

```
11.26.27.GO
11.26.33.0 0/LOGIN GLEN 4754R619
11.26.33.0 1/LOGIN PASS01 46242619
11.26.33.0 2/LOGIN PASS02 46242619
11.26.33.0 3/LOGIN PASS03 46242619
11.26.34.0 4/LOGIN PASS04 46242619
11.26.34.0 5/LOGIN PASS05 46242619
11.26.34.0 6/LOGIN PASS06 46242619
11.26.35.0 7/LOGIN PASS07 46242619
11.26.35.0 8/LOGIN PASS08 46242619
11.26.35.0 9/LOGIN PASS09 46242619
11.26.35.010/LOGIN PASS10 46242619
11.26.35.011/LOGIN PASS11 46242619
11.26.35.012/LOGIN PASS12 46242619
11.26.35.013/LOGIN PASS13 46242619
11.26.35.014/LOGIN PASS14 46242619
11.26.35.015/LOGIN PASS15 46242619
11.26.41.0 0/  TIME    11 26 35
11.26.42.0 5/  TIME    11 26 35
11.26.42.0 6/  TIME    11 26 35
11.26.42.0 7/  TIME    11 26 35
11.26.42.0 8/  TIME    11 26 35
11.26.42.0 9/  TIME    11 26 35
11.26.42.010/  TIME    11 26 35
11.26.42.011/  TIME    11 26 35
11.26.43.012/  TIME    11 26 35
11.26.43.013/  TIME    11 26 35
11.26.43.014/  TIME    11 26 35
11.26.43.015/  TIME    11 26 35
11.26.43.0 1/  TIME    11 26 35
11.26.43.0 2/  TIME    11 26 35
11.26.43.0 3/  TIME    11 26 35
11.26.43.0 4/  TIME    11 26 35
11.26.45.0 0/  DATE    01/13/68
11.26.46.0 5/  DATE    01/13/68
11.26.46.0 6/  DATE    01/13/68
11.26.46.0 7/  DATE    01/13/68
11.26.46.0 8/  DATE    01/13/68
11.26.46.0 9/  DATE    01/13/68
11.26.46.010/  DATE    01/13/68
11.26.46.011/  DATE    01/13/68
11.26.47.012/  DATE    01/13/68
11.26.47.013/  DATE    01/13/68
11.26.47.014/  DATE    01/13/68
11.26.47.015/  DATE    01/13/68
11.26.47.0 1/  DATE    01/13/68
11.26.47.0 2/  DATE    01/13/68
11.26.47.0 3/  DATE    01/13/68
11.26.47.0 4/  DATE    01/13/68
11.26.51.0 0/.... COMPILE HUDAR
11.26.52.0 5/.... COMPILE HUDAR
11.26.52.0 6/.... COMPILE HUDAR
11.26.52.0 7/.... COMPILE HUDAR
11.26.52.0 8/.... COMPILE HUDAR
11.26.52.0 9/.... COMPILE HUDAR
11.26.53.010/.... COMPILE HUDAR
11.26.53.011/.... COMPILE HUDAR
11.26.53.012/.... COMPILE HUDAR
11.26.53.013/.... COMPILE HUDAR
11.26.53.014/.... COMPILE HUDAR
11.26.53.015/.... COMPILE HUDAR
11.26.54.0 1/.... COMPILE HUDAR
11.26.54.0 2/.... COMPILE HUDAR
11.26.54.0 3/.... COMPILE HUDAR
11.26.54.0 4/.... COMPILE HUDAR
11.26.58.0 0/.... INPUT FTN
11.26.58.0 5/.... INPUT FTN
11.26.58.0 6/.... INPUT FTN
11.26.59.0 7/.... INPUT FTN
11.26.59.0 8/.... INPUT FTN
11.26.59.0 9/.... INPUT FTN
11.26.59.010/.... INPUT FTN
11.26.59.011/.... INPUT FTN
11.26.59.012/.... INPUT FTN
11.26.59.013/.... INPUT FTN
11.26.59.014/.... INPUT FTN
11.26.59.015/.... INPUT FTN
11.27.00.0 1/.... INPUT FTN
```

```
11.27.00.0 2/.... INPUT FTN
11.27.00.0 3/.... INPUT FTN
11.27.00.0 4/.... INPUT FTN
11.27.04.0 9/00000010=0001
11.27.05.0 5/00000010=0001
11.27.05.0 6/00000010=0001
11.27.05.0 7/00000010=0001
11.27.05.0 8/00000010=0001
11.27.05.0 9/00000010=0001
11.27.05.010/00000010=0001
11.27.06.011/00000010=0001
11.27.06.012/00000010=0001
11.27.06.013/00000010=0001
11.27.06.014/00000010=0001
11.27.06.015/00000010=0001
11.27.06.0 1/00000010=0001
```

Figure 5 – Operating results from console display

- Input commands and results by terminal for each terminal declared in the control card
- The input string for each terminal as it appears in the data deck
- All rejected (unsent or unprocessed) entries in the input string for each terminal
- A matrix of operating statistics for each terminal which includes flags for abnormal terminations, the number of commands repeated and rejected, character counts of latest input and output strings, activity codes, and many more items useful in isolating problems within RESPOND, SCOPE or MUSE

Figure 6 is an example of the Results File kept for each terminal. This file is usually the main item of interest. It is identical to the results achieved from a Teletype terminal with the exception that the time between command entry and system response is included. This provides a means for collecting additional performance data and comparing different hardware configurations.

Statistical information is gathered during a simulator run and formatted as a matrix. This matrix provides a quick index to trouble areas. With a little experience it is possible to quickly pinpoint the terminal and command which has a malfunction. The items in this statistical file include command codes, character counts of most recent input-output strings, number of commands processed, and repeated, terminal status (active, waiting output, etc.) and latest contents of the character input-output buffers.

Console capabilities of the Control Data Corporation 6000 Series computer, operated through the SCOPE system, provide additional flexibility in RESPOND-MUSE operation. Simulator activity can be suspended by rolling the simulator out of execution. RESPOND can be returned to Teletype service by dropping the simulator from its control point or by changing the first character of the program name at the control point. Relative priorities and field lengths can be adjusted to give the simulator the aspects of a large compute-bound job.

The complete continuity between simulator and Teletype operations provides a mechanism for checking performance of the MUSE system or isolating RESPOND and the simulator communications problems.

Results of operation

By this point the reader is primed for a series of graphs and tables which illustrate the value of the system described. Our goal, however, was to design a tool for building and testing multi-terminal time sharing systems. The measurement of performance is actually a by-product of our efforts. With respect to the primary goals, we believe we have achieved the following results

- A better product than could have been built with more traditional testing and debugging methods
- A better understanding of the product's internal operations and the compromises made to achieve a specific level of performance
- The ability to adjust product performance given a specific job environment and service requirement to most efficient operating point
- A thoroughly tested and formally evaluated product; a product which will satisfy specific performance criteria with many more terminals than can be reasonably assembled or coordinated in a development environment
- Confidence that the system tested will continue to function under the worst-case stress conditions that can be imposed by a large number of remote users.

A valid question at this point: "Were we really able to find problems with the MUSE system which would have eluded us until a customer found them?" The answer is definitely affirmative. The following problems may not have come to light until several months after product release.

- When nine (or multiple thereof) users all request a sort of their input records at the same time there was a problem in handling that ninth request. This is understandable in our octal world, and not difficult to fix once isolated.
- When the thirteenth unsatisfied disk request entered the disk I/O queue, it created a problem.
- When the disk request queue became saturated, a path through one command requiring disk access did not handle rejects properly. This happened only when more than ten FILE commands occurred within a 30-millisecond period.

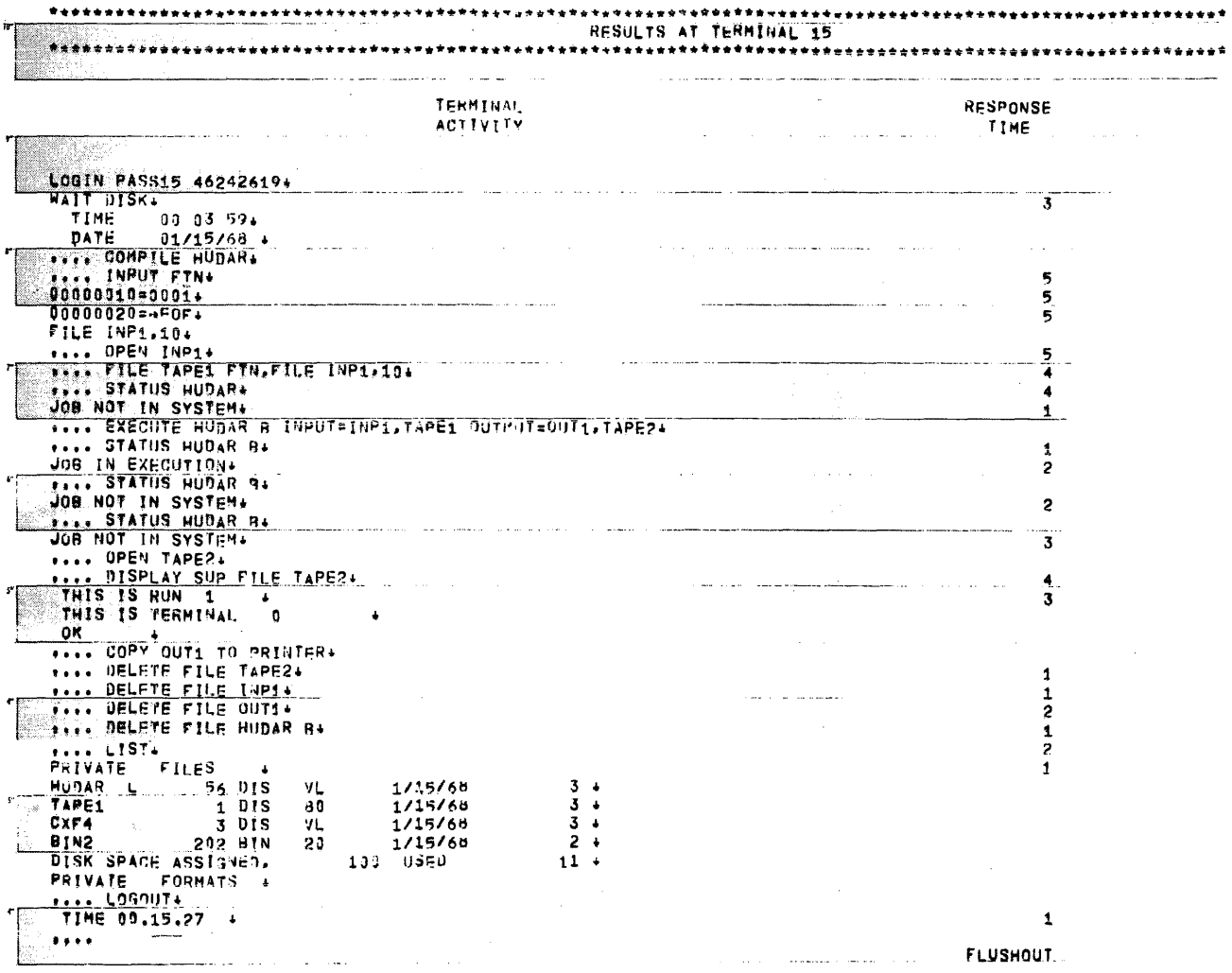


Figure 6 – Results at terminal 15 from a test with 16 terminals

- When many jobs were dumped into the SCOPE processing queue within a 90-millisecond period there were problems associated with jobs which had to wait for central memory storage.
- Timing problems occurred between the activities of the system supervisor in changing the contents seconds, the reject mechanism broke down.
- Timing problems occurred between the activities of the system supervisor inchanging the contents of public files and users trying to use these files.
- The simulator revealed a basic design flaw that resulted in a major internal change to the program which now insures one-to-one mapping of the active user's file catalog in core and on disk.
- A problem which plagued single terminal operations disappeared during multi-terminal simulation. This was traced to expansion of RESPOND's field length under multi-terminal activity. This bit of intelligence led us to the problem's solution.

To this list can be added the ability to check, with minimum trauma, all 512 variations of the FORMAT command or all xxx variations of the ZZZ command. Many single terminal problems associated with long strings of commands were also solved far less painfully through the consistency, repeatability, and rapidity of the simulator's operations.

The by-product of performance statistics on RESPOND has proven particularly interesting and encouraging. Data has been collected for a total of sixteen simultaneous users operating in the following two environments:

- File Management activity with a small amount of program compilation and execution (approximately 6%)
- General programming activity with equal emphasis on program creation, compilation, and execution operations.

All of the data, thus far, has been collected on a Control Data Corporation 6400 Series Computer having 65,000 words of central memory, and utilizing a dual channel 6638 disk unit for mass storage. The simulator itself acted as a backbround program requiring between 1700 and 2700 seconds of central processor time and 9600 words of central memory during the testing period.

The distribution of 'wait' times for the file management environment is shown in Figure 7. The wait time is defined as the interval between the entry of the last character of a command (the RETURN key of the Teletype) and the arrival of the first character of the reply from the RESPOND system. The mean 'waiting' time is 1.88 sec. The maximum waiting time recorded was 29 sec on the compilation of a FORTRAN program having 1321 statements. In this case the user chose to wait for the compilation to be completed. Commands used heavily in this environment included the FILE. INPUT, and DISPLAY. Table I contains a summary of the test results for this environment and details on its composition.

The results for a general programming environment are shown in Figure 8. The simulated user was placed in a position of waiting until a compilation or execution was complete, and this has been considered as a single waiting time. The programs compiled contained from 21 FORTRAN statements to 1321 statements.

The mean writing time for this environment was 2.6 seconds, reflecting the 17% of commands requesting SCOPE processing. The maximum waiting time recorded was 33 seconds, and was associated with a wait for a large compilation to be completed. Input and output operations, however, still make up a large portion of the user's activity, as can be seen from the command mix statistics in Table I.

These two environments are compared in Figure 9 and Table I. In Figure 9 the waiting probability is plotted against waiting time. Note that a waiting time of longer than 3 seconds would occur only 7% of the time in a file management environment, and 17% of the time in the programming environment even when the average user chose to wait for his compilation or execution to be completed before the next command. In both environments, a wait of 4 seconds would occur less than 10% of the time. Table 1 includes central processor operating statistics for TTY RESPOND under each environment. For file management activity, an average of 13.7 commands are processed per each central processor second used by RESPOND. For programming activity the average raises to 15.4 commands per central processor second used by RESPOND. In the latter case, fewer input and file operations are issued to RESPOND and the computing load is shifted to SCOPE.

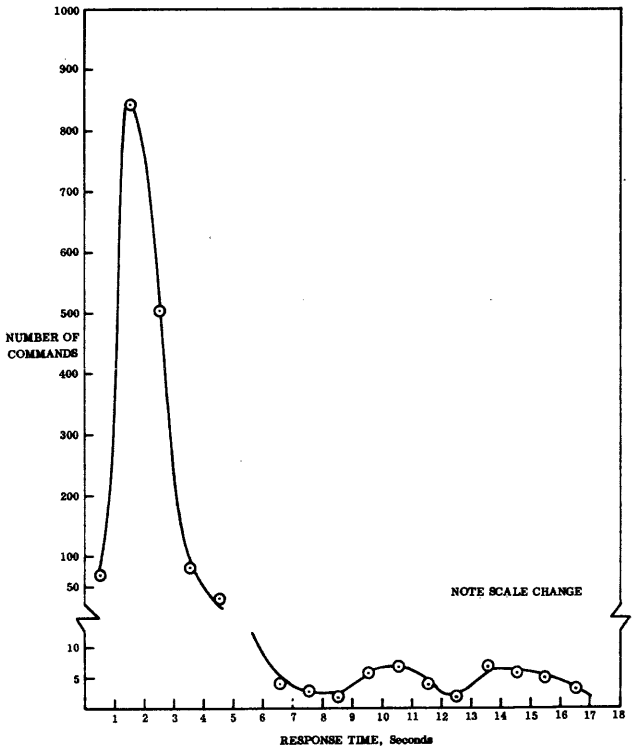


Figure 7—TTY RESPOND response time distribution for a file management environment

Guidance for the selection of LOGIN times, average number of commands per terminal and LOGIN time spread (Table I) were obtained from a number of unpublished sources and from Allan Sherr's MIT monograph on analysis of time shared systems.[1]

Additional study is planned to determine variation in RESPOND's performance with the hardware configuration consisting of

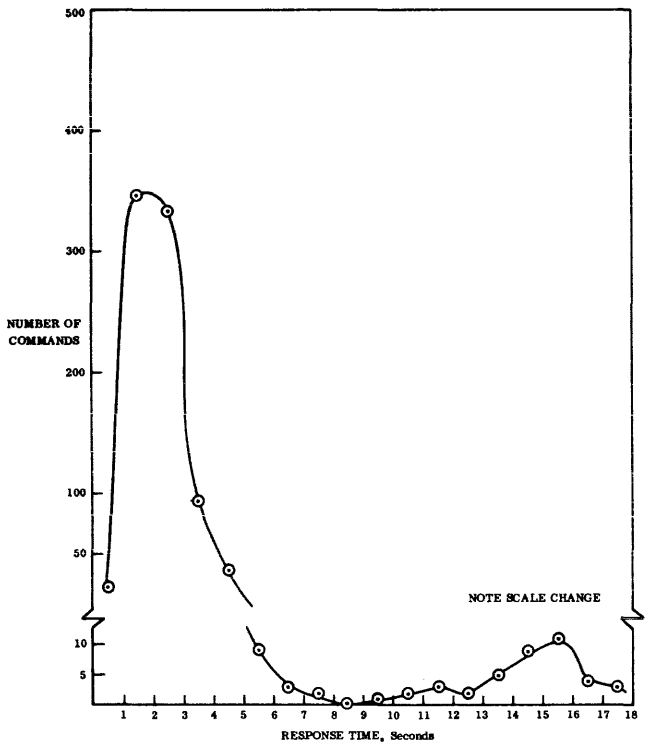


Figure 8—TTY RESPOND response time distribution for a programming environment

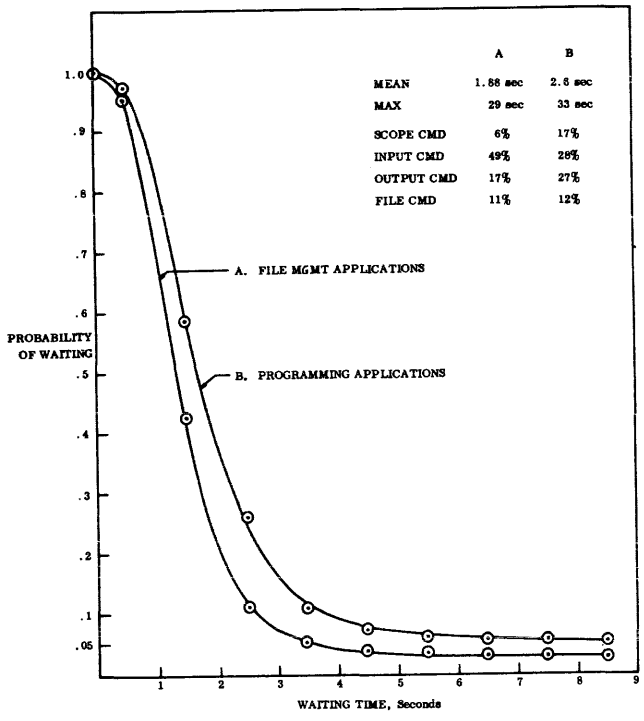


Figure 9—User waiting time probability distribution

- The 6500 dual central processor computer
- The 6600 computer (with its dual arithmetic unit)
- Single channel disk and multiple dual channel disk configurations
- Central processors with 131,000 words of memory
- Sixty-four simultaneous remote users

Variation of performance as a function of different background program activity will also be investigated. Currently, the background activity is provided by the computational needs of the MUSE program itself which uses 20 to 30 times as much central processor time as the entire RESPOND system.

Like any tool, the MUSE program is only as good as its user and the data which this user provides. It has, however, proven to be an excellent approach for solving many multi-user time sharing problems in a development center environment with a minimum cost for terminal equipment.

Simulator limitations

The MUSE program is designed to simulate activities of a remote terminal user, the remote terminal, the communications line, the systems multiplexer and the data channel to RESPOND's I/O buffers. The simulation of the user is only as good as the data entered to represent his terminal operations.

The MUSE program does not simulate the computer system itself. Changes in central processor memory size, the number of mass storage devices, the number of data channels, or other changes in computer site hardware can be checked through the simulator's performance, but they are not adjustable from the simulator end. Some aspects of the SCOPE processing environment, such as core and central processing time, can be changed by giving more of these resources to the MUSE program. This, however, does not represent a satisfactory method of varying the SCOPE environment or resource requirements.

An additional control card is required to simulate a user's think-time defined as the time between the end of a RESPOND output operation and the beginning of the next input string from the simulator. Currently commands such as HOLD and STATUS are used for this purpose but the control is poor and not straightforward. A THINK Control card would allow a specified time to pass before the next command was issued from the simulator.

More operator override features are desirable. The ability to assign the computer console keyboard to any terminal in place of the simulator would greatly increase the debugging potentials of the program. Better control over the balance of the SCOPE processing environment should be developed. This could be a

TABLE I – Summary of results and test specifications

Application	File Management	Programming
Mean waiting time	1.88 sec	2.6 sec
Maximum waiting time	29 sec	33 sec
Total RESPOND overhead	116.4 CPU sec	58.1 CPU sec
Background Program (MUSE)	2681.7 CPU sec	1729.8 CPU sec
Usage of CPU time		
Average LOGIN time	34 min	26 min
Range of LOGIN times	26 min to 44 min	24 min to 29 min
Total Nr. of commands processed	1592	896
Average Nr. of Commands per terminal	100	56
Range of commands per terminal	52 to 151	all 56
Command Mix (over all terminals)		
SCOPE Processing	6%	17%
Input Operations	49%	28%
Output Operations	17%	27%
File Operations	11%	12%
Delete Operation	10%	12%
System Access	7%	4%

SCOPE Processing Mix (to service RESPOND users)

Range of Processing under SCOPE .166 CPU sec to 19.2 CPU sec per job

Range of program size 21 FORTRAN statements to 1321 FORTRAN statements per program

companion program to simulate the system card reader.

The task of gathering data, and using this data to build up command strings is a major problem to be attacked. The big question is where to look for this data, given the myriad of command languages and processing features available to the user, and the relatively small, limited, and specialized applications of current time sharing systems.

CONCLUSION

In a system which provides time shared service to remote users there are a number of questions which must be resolved if a concept of service is to be maintained.

The question is not "How many terminals can be serviced?" but rather, "What is the marginal effect on existing users when one more terminal is added to the system?" This is repeated again and again with the job environment as a parameter, to determine the average responsiveness as a function of users and job mix.

The question is not "What features can be added to the system?" but rather, "How does the addition of a feature or a change in the processing method affect existing users?" The economic concept of 'Pareto-optimality'[2] must be considered to insure that some users are not much worse off because a new feature has been added for other users.

What is the effect of higher speed terminals on the system's performance? Which part of the system saturates first, and what effect does this saturation have on the rest of the system?

How does a system designer know that his creation really works for 64 Teletypes, or 32 display terminals? How does a quality assurance group certify that a product meets multi-terminal performance requirements? How is the product tested in a multi-terminal environment? One of the biggest questions is "What are the significant elements of exchange?"

Emphasis in developing the MUSE program has been to build a tool which could be used to answer some of these questions. Emphasis has been on concepts of multi-terminal system development and testing rather than performance measuring or predicting. We needed a tool that looked like a user. The MUSE program is doing a good job providing this capability.

The fact that RESPOND and MUSE can operate together independently of any multiplexer or terminal equipment also provides a means of evaluating the performance of new and different computer hardware configurations as well as collecting statistics on existing ones. This paper is a report on a different approach and a new tool in checking, testing, and even developing better time sharing systems. Hopefully, the idea will catch on. Not the use of a specific program but the idea of testing to see 'what happens when,' rather than installing to find these answers.

APPENDIX

RESPOND COMMAND LANGUAGE
MESSAGE (message contents, 70ch max)
LIST FILES or FORMATS
PEN File Id
FORMAT (record number, length, step size, tab and skip functions)
INPUT File Id, Format Id
#EOF
FILE File Id, FILE File Id(item select options)
HOLD
DELETE File Id (item select options) or Format Id
DISPLAY File Id (item select options)
#STOP
ASSEMBLE File Id LIST Result Id BIN Binary Id
COMPILE File Id LIST Result Id BIN Binary Id
EXECUTE Binary File Id INPUT = Input File Ids OUTPUT = Output File Ids BIN Binary File Ids
COPY File Id To Peripheral Device
COPY Peripheral Device TO File Id
SUBMIT Control Card File Id INPUT=Input File Ids, OUTPUT-Output File Ids BIN =Binary File Id's
STATUS File Id sent to SCOPE
&PASSWORD* LIST or CHANGE (Items) or ADD (Items) or DELETE (Items)
*TRANSFER FILE or FORMAT, User File Id or Format Id TO Public File Format Id
*DUMP (Creates archive type)
*SYSEXIT (RESPOND system exit)
LOGOUT

The reader is referred to the TTY RESPOND Reference Manual[3] for detailed descriptions of the systems command structure and capabilities

REFERENCES

1 A L SCHERR
An analysis of time-shared computer systems
MIT Research Monograph No 36 1967
2 T A MARSCHAK
Economic theories of organization
Handbook of Organization Theory J G March editor McGraw Hill 1966
3 *TTY RESPOND reference manual*
Control Data Corp Publication Nr *60189300* 1967

*Available to system supervisor only

Diagnostic engineering requirements

by JOHN J. DENT
International Business Machines Corporation
Kingston, New York

INTRODUCTION

There is a maze of diagnostic techniques in use today: Diagnostic Programs, Micro Programs, Test Panels, On-Line Diagnostics, Error Recording Techniques, Automatic Recovery Procedures; and the list goes on. The purpose of this paper is to take a look at the basic concepts applicable to Error Detection and diagnosis, in order to put into perspective the value of the various techniques for satisfying future requirements.

The terms malfunction, failure, and error are used rather loosely to refer to any deviation from the expected operation of the system, caused by a design error, fabrication error, equipment malfunction or program error. The most common method of automated diagnostic testing uses computer programs written in machine language. However, the same concepts apply whether these are implemented by software, hardware, or firmware.

Uses of diagnostic tests

The design of a diagnostic test is influenced by its intended use or test environments. Perhaps one of the most unique test environments is *Engineering Test*; that is, testing the very first model of a newly-designed system. Engineering Test is characterized by multiple design and fabrication errors, multiple component malfunctions, and multiple errors (bugs) in the diagnostic test itself. The purpose of Engineering Test is to verify the design. "Functional Tests" are written to determine whether the system operates in precisely the same manner described in its functional description or specifications. A test generated automatically from design automation tapes would be of no use in verifying the design itself. Functional tests are generally not sensitive to engineering changes. This is of particular value in the Engineering Test phase when there are a number of engineering changes. In addition to functional tests, random tests and worst case patterns must be developed to "shake down" the design.

The *Manufacturing Test* requirements are similar to the Engineering Test requirements. These tests must be designed to cope with multiple errors resulting from faulty components and fabrication errors. *Field Maintenance*, on the other hand, is slightly different. The assumption is that the system is in good operating condition and that failures are repaired as they occur. The diagnostic designed for this use can take advantage of the single error assumption.

There are several situations which require a *quick and thorough checkout* of the system: Ship Test, Installation Test, Acceptance, Early Morning Checkout, etc. In all such cases, the assumption is that the system is working. The requirement is for a rapid test to verify the fact. This is a GO/NO-GO type of decision requiring a fast and thorough test without regard for isolation.

Basic approaches to testing and diagnosing

The *Start-Small* approach is a building block approach where the first test starts with the smallest amount of circuitry possible. Each additional test adds a small increment to the circuitry tested. When a given test fails, the assumption is that the failing circuit is within the group of circuits added by that test. Figure 1 illustrates the general flow of the Start-Small approach. Notice that the flow is sequential. The sophistication of this approach is found in the design and sequencing of the individual tests. The assumption is that the first failure found in the test sequence is repaired before proceeding past that point. This approach is effective for multiple errors.

The *Multiple Clue* approach bases its diagnosis on the analysis of a series of individual test results. Figure 2 illustrates how the test flow is similar to the Start-Small approach. The difference is that a failure does not terminate the test. All tests are run, failure information is stored, and the diagnosis is determined by analysis of this failure data. Assuming a single error, this analysis can be quite sophisticated. Assuming more than one error, the analysis is extremely complex, if not impossible.

The *Start-Big* test approach has a more complex flow, as illustrated by Figure 3. Testing starts with a

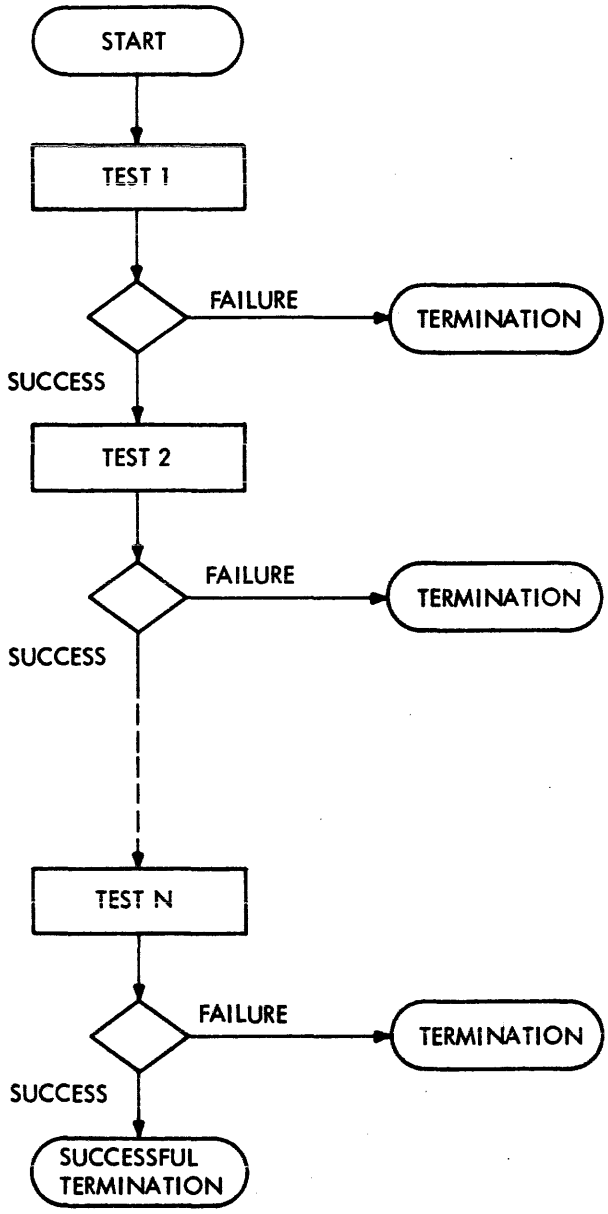


Figure 1—General flow diagram of start-small (building block) test approach

large portion of logic and, if successful, proceeds to another large portion of logic. A failure changes the test sequence by branching to a test designed to further pinpoint the trouble. The test results of each test determine which test is executed next. This approach appears to be the optimum strategy; under successful conditions, it provides a rapid checkout; under a single-failure condition, it provides rapid isolation. However, it is extremely complex in design and presents some hazards. One incorrect branch can lead the maintenance engineer astray. Furthermore, this approach is not applicable for multiple errors.

Figure 4 shows how the three test approaches are rated against the three test environments. The Start-Small test is extremely useful for Manufacturing

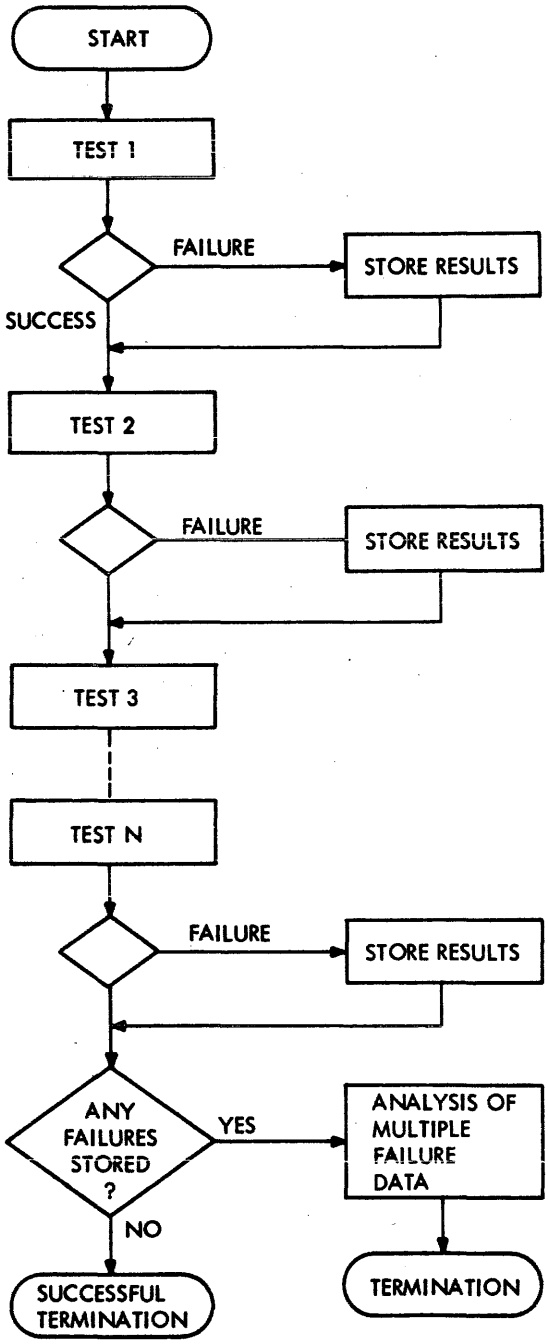


Figure 2—General flow diagram of multiple-clue test approach

Test, since the initial tests are simple and particularly suited for multiple failures. The Start-Small approach is also useful for field maintenance, along with the multiple clue approach to give better diagnosis in certain areas. The Start-Big approach satisfies the need for a quick checkout.

A mixed strategy can be developed to satisfy the different test environments with one series of tests. The Start-Small approach satisfies two of the three test conditions and can form the foundation of the mixed strategy. The multiple clue approach can be

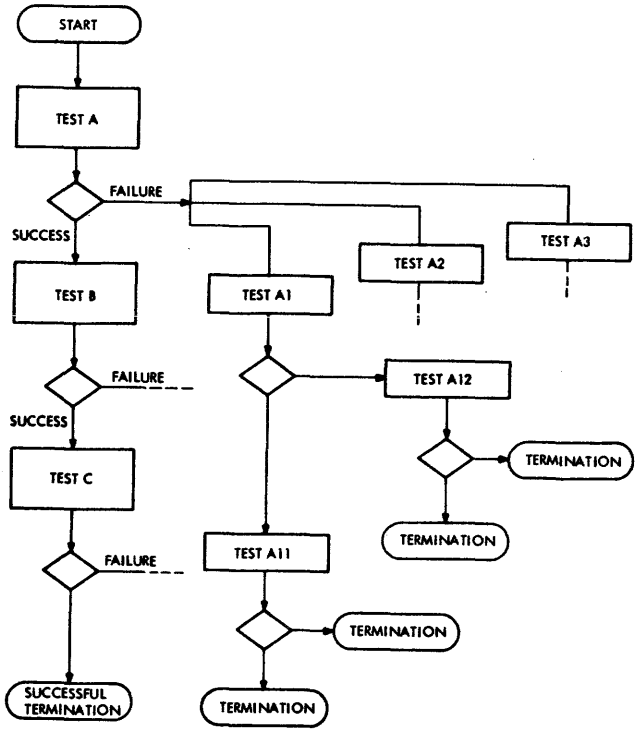


Figure 3 – General flow diagram of start-big test approach (process of elimination)

used where further isolation is required. The Start-Big approach can be used, sparingly and cautiously, to speed up the test process.

All of the techniques mentioned so far depend on the results of more than one test for proper diagnosis. They are susceptible to incorrect diagnosis of the intermittent error, and the diagnosis of a failure depends on the ability to recreate the failure under test conditions. The test environment is an artificial one and, while operating conditions may be simulated, they cannot be duplicated. Therefore, there is no guarantee that a failure which occurs during normal operation can be detected and isolated under test conditions. Error-detection circuitry, along with error-environment recording techniqes, has been used to detect errors during normal operation (on-line) for analysis at a later time (off-line). The ultimate diagnostic seems to be error-check circuitry with enough diagnostic resolution to isolate to the replaceable parts. This creates another diagnostic problem. Verifying the design and functioning of all the error-check circuitry is, in itself, a formidable task.

Field engineering

If Diagnostic Engineering is looked at from the Field Engineer's point of view, a very interesting perspective is obtained. Here is a man who is sent to company schools for extensive training in circuits, logic, theory of operation, programming, operating

	APPLICABLE STRATEGY		
	START SMALL	MULTIPLE CLUE	START BIG
ENG/MFG TEST	X		
FIELD MAINTENANCE	X	X	
QUICK CHECKOUT			X

Figure 4 – Test use vs. strategy

systems, diagnostics, etc. He is furnished with a complete set of manuals describing in detail the operation of the system and its subassemblies: hundreds of pages of logic diagrams; documentation for the operating programs; diagnostic program writeups, flow charts and listings. He is furnished with many maintenance aides such as test panels, failure indicators, oscilloscope, tool kit and spare parts.

The problem the Field Engineer faces is that, when a failure occurs, it is not always obvious where to begin looking. The problem could be a program error or a hardware malfunction. Should he spend more time investigating the customer's error symptoms or should he try to recreate the error under test conditions? Should he go to the test panel or run diagnostic programs? Should he run all the available test programs in sequence, or should he select the test program for the suspected malfunctioning area? Can the user still operate a portion of the system while he isolates and repairs the malfunction?

There are no simple answers to these kinds of questions. The field engineer is exposed to a wide variety of problems and operational environments. He has at his disposal a wide variety of testing tools and techniques. He applies his judgment to each unique set of circumstances as he proceeds in the isolation and repair process. All the diagnostic tools, with their options and variations, should be ready and available as he needs them.

This interplay between Field Engineer and diagnostic tools is, in itself, a Human Factors problem. The Field Engineer must control or manage the maintenance system. He must continually select a procedure, execute the procedure, and interpret test results. While he has been furnished with sophisticated systems for automatic testing and diagnosis, the toughest problems have been left for his own ingenu-

ity. The tendency is to compensate for this by providing him with flexibility in the maintenance system. The Field Engineer must communicate with the maintenance system to select the option he wants. The diagnostic programs must communicate test results in meaningful terms.

The Diagnostic Control Program has evolved as a method of standardizing the interface between the Field Engineer and the numerous diagnostic programs. It provides a framework within which all of the individual diagnostics are developed. It provides a standard communication medium between the test engineer and the individual diagnostics. The control program automatically initiates and terminates tests as instructed by the test engineer.

Maintenance systems continue to get more complex with the increased complexity of the systems being installed. Techniques must be developed in the future to simplify maintenance from a Human Factors point of view.

Other factors affecting diagnostic techniques

Up to this point, the basic techniques for testing, detecting and isolating equipment malfunctions have been described. The continuous trend toward more sophisticated data processing systems has placed restrictions on the use of some of these techniques and has created requirements for new techniques.

The technological advance from the tube to the transistor, and now integrated circuits, has given us a significant increase in reliability. The failure rate per logical decision has decreased significantly. The same technological advance is providing us with higher logical density and more logical function per dollar.

In parallel with advances in equipment development is the development of new and more sophisticated applications for this equipment. The result is a continuous trend toward larger and more complex system configurations. As a result of this increased use of equipment per system complex, the typical large complex of today is susceptible to a higher failure rate, in spite of reliability improvements at the logic level.

System error management

Systems must be designed which are fault tolerant; that is, capable of continuing with their primary function in spite of individual equipment malfunctions. This implies the implementation of a total error management concept throughout the system. This concept involves program design, as well as system design. Basically, it includes the following:

1. The ability to detect system malfunctions while the system is operating.
2. The ability to recover by redoing the operation, or going back to some checkpoint in the system.
3. The ability to recognize and isolate a solid malfunction to a unit.
4. The ability to adjust the system (reconfigure) to continue operation without the failing unit.
5. The ability to repair the failing unit without impairing system operation.
6. The ability to return the failing unit to the system without impairing system operation.

All of the above have been implemented to some degree or another for various applications. In some cases, the solution to the problem is applications dependent. For example, in some tracking applications occasional pieces of tracking input can be lost without significant effect on the operation. However, the loss of one business order or stock transaction can be very serious. On the other hand, the tracking application must respond in real time—including the error-recovery procedures—while many business applications can tolerate an occasional lag in response.

System Error Management has been implemented successfully for specific applications. Many defense systems serve as good examples. In many cases the additional programming costs run high. More generalized solutions to the error management problem are required in the future.

SUMMARY

In conclusion, a few general observations can be made. Error detection and diagnosis through the use of automated testing and programmed analysis has several drawbacks:

1. The use of testing for diagnosis assumes the ability to recreate the error under a test environment.
2. Diagnosis to replaceable parts requires sophisticated programs.
3. The effectiveness of diagnosis for intermittent errors is questionable.
4. Automated testing for error detection provides no error detection during normal system operation.
5. Implementation of this technique on-line, concurrent with the operating program, adds more complexity to the maintenance system.

In spite of these drawbacks, the automatic testing approach will still be required in the foreseeable future for design verification and initial system shakedown.

After installation, the best test of the system is the actual operating programs. It is possible to design detection logic which approaches 100% detection of equipment failures during system operation. The same

error detection logic can be designed to provide diagnosis or isolation to replaceable parts for intermittent as well as solid failures. Error detection logic plays a vital role in error management or automatic recovery techniques. As the complexity and cost of programming increases, the economic tradeoff is swinging toward more error detection logic for the detection and isolation of errors.

Self-repair techniques in digital systems*

by FRANK B. COLE
Westinghouse Electric Corporation
Baltimore, Maryland
and
WILLIAM V. BELL
U. S. Army Electronics Command
Fort Monmouth, New Jersey

INTRODUCTION

The Army's use of sophisticated digital equipment at the lower organizational levels has led to an increased requirement for equipment availability without extensive downtime, maintenance, or logistics support. Self-repair by means of automatic spares switching was considered in an Army funded study as a possible solution to this requirement. This paper is based on that study.

There are two basic problems inherent in the self-repair of equipment by the use of spares switching.. One is the detection of the failure and the location of the failed element, while the second is the switching out of the failed element and the switching in of a spare. The study upon which this paper is based considered these aspects of self-repair. Modular redundancy[1] was not studied, and is mentioned in this paper only for purposes of comparison.

Three common methods of fault detection and location are coding, software diagnostics, and duplication:

(1) Coding is one technique for the detection and location of faults. Ordinary Hamming and similar codes can be used for detection (and perhaps even correction) of faults in memory and similar transfer type operations. In the arithmetic unit where information is altered during operation residue type codes must be used. The main disadvantage to coding is the extra time required to generate the codes. This is especially true of the arithmetic unit where a new code word must be generated after each operation.

(2) If the digital device in question is a programmable computer, then software can be used for the detection and location of faults. The discussion of diagnostic routines and reasonableness tests is beyond the scope of this paper, except to point out that they form a valuable adjunct to techniques such as duplication. The disadvantage to the use of diagnostics for the primary detection of faults is that there is usually considerable delay between the occurrence of the fault and the detection of it by the diagnostic.

(3) Duplication is applicable to all parts of a digital system, including control. It does not present the timing problems that the first two techniques do, and there exists the possibility of using the duplicate both as a fault detector and as a spare. For these reasons duplication, in conjunction with the other techniques, was selected for the fault detection and location function. Several recent papers[2,3] suggest other approaches to self-repair.

*This work was sponsored in part by the U. S. Army Electronics Command under Contract DA28-043-02343 (E).

Repair switching with duplication

A. Organization

Figure 1 illustrates one straightforward method of performing repair switching when duplication is used for fault detection and location. The module to be replaced consists of a main unit, an identical auxiliary unit, and a comparator which compares all significant outputs of the main and auxiliary units. When the comparator detects a difference, it is assumed that one of the three parts of the module has failed and the whole module is replaced.

In looking at Figure 1, the question comes to mind, "Why not simply switch out the failed unit (main or auxiliary) and use the working unit as the spare?" The problem is, of course, determining which unit has failed and which is working, However, if the problem can be solved, the hardware saving is considerable. One possible solution is the use of diagnostic programs to find the failed unit. The hardware implementation is shown in Figure 2, and the system would work as follows.

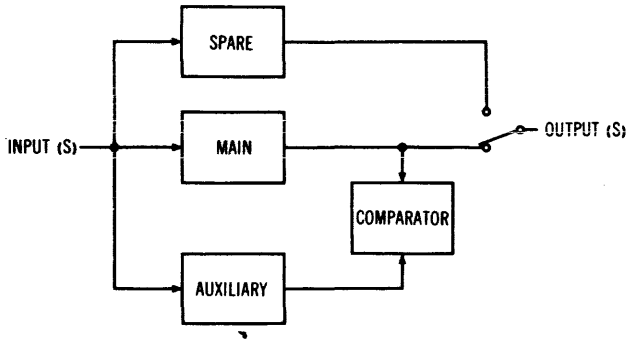


Figure 1 – Repair switching - external spare

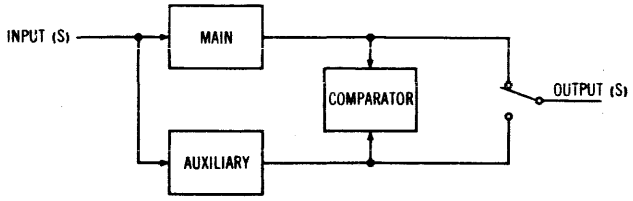


Figure 2 – Repair switching - no external spare

The main and auxiliary units operate in parallel, with identical inputs. All of the significant outputs are compared. As long as no failures occur, the outputs of the main and auxiliary units will be identical, but upon a failure in either unit (or the comparator) the outputs will be different and a program interrupt can be generated. A diagnostic program is then run. If the diagnosis detects no faults, then the main unit which is connected must be working and the failure must be either the auxiliary unit or the comparator. In addition, if the comparator output is observed during the diagnosis, and the difference does not occur again, the first difference must have been an error (noise) and not a hard failure. This approach can thus prevent errors from causing working units to be switched out and "thrown away". However, if the comparator does indicate a difference during a diagnostic, and the diagnostic indicates no faults, then the switch must be "locked" in position so that the main unit remains connected and the comparator output disabled so no further interrupts occur from the "repaired" module. If the diagnosis does detect a fault, then the main unit must have failed and the auxiliary unit's output must be switched into place permanently, and the comparator output disabled to prevent further interrupts from this module.

Thus, if the duplicated units themselves are used as spares:

(1) A means must be available to detect faults independently of duplication. Although diagnostic programs are the most promising method of fault detection, coding or any other acceptable scheme could be used.

(2) As long as the fault detector serves more than one module, permanent (non-power dependent) switch storage is required.

(3) Where applicable, this technique has some definite advantages over triple modular redundancy. First there is a hardware saving since duplication, rather than triplication, is required. Secondly, the reliability could be greater since only one out of two units need be working, rather than two out of three. On the other side of the coin, there are two disadvantages to this technique over triple modular redundancy. The first is that it is only applicable where a means of external (to the module) fault detection is possible, and the overhead (hardware, storage, time) of the external fault detection must be added to the system cost. Secondly, information is lost in the case of a failure and the program must be "restarted" in some manner each time a failure occurs.

One possible hardware implementation for the switching using internal spares is shown by Figure 3. It is assumed that the three control pulses, Reset Fault, Set Main Failed, and Set Auxiliary Failed, are generated by the diagnostic program or other fault detector. When the comparator gates detect a difference between a main and an auxiliary output, the J input of the Fault Flip-Flop is enabled and the flip-flop is set by the clock. The flip-flop is clocked to avoid its being set by spikes caused by temporary

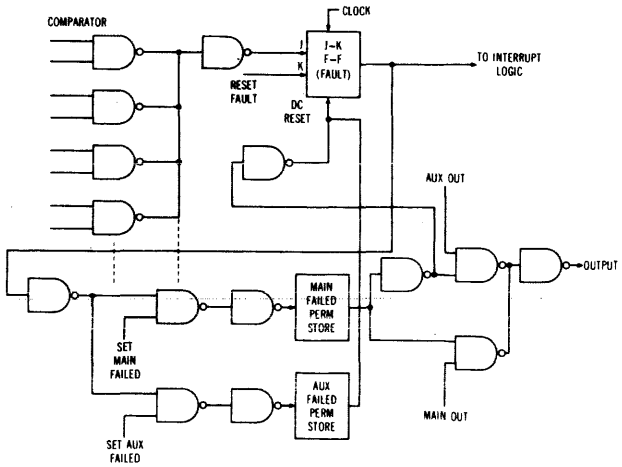


Figure 3 – Switch logic hardware implementation

differences between the main and auxiliary outputs during transition times. The setting of the Fault Flip-Flop puts the system in a fault detector mode. For example, it might cause an interrupt which causes the diagnostic program to take control.

Assuming that a diagnostic program is used for fault detection, the program would then reset the Fault Flip-Flop with the Reset Fault pulse and run a diagnostic of the area in question. If the diagnostic shows a failure, then the Set Main Failed pulse is generated. If the Fault Flip-Flop has been set during the diagnosis, as it would be in case of a hard fault, then the Main Failed pulse causes the Main Failed Permanent Store to be set. If the diagnostic does not show a failure, then the Set Auxiliary Failed pulse is generated and the Auxiliary Failed Permanent Store is set if the Fault Flip-Flop has been set during the diagnostic. In any case, the Reset Fault pulse is then applied to reset the Fault Flip-Flop.

Setting of the Main Failed Permanent Store disconnects the Main Unit from the module outputs and connects the Auxiliary Unit to the module output(s). It also prevents the Fault Flip-Flop from being set by any future difference between main and auxiliary output(s). Setting of the Auxiliary Failed Permanent Store simply prevents the Fault Flip-Flop from being set, thus assuring that the fault detector is never called on again by that module and the Main Failed Permanent Store cannot be subsequently set.

As a matter of practice the overall reliability of the system can usually be increased by including the switch gates as part of the following module. Of course, this also increases the system price since the number of switch gates is at least doubled (the number of switch gates would be equal to twice the number of places the output goes). For the remainder of this paper, the configuration with one set of switch gates will be assumed, and if the gates fail, the system fails.

Because of its size and relative importance, the memory module can be handled somewhat differently than a typical logic module. A memory configuration using duplication is shown in Figure 4. Each memory module contains a complete memory with parity. The permanent stores are completely under program control and no hardware comparator is provided as with the other modules. The memory operation is described below.

As each word is read from the memory the parity is checked. If no parity errors occur, the information is taken from the memory designated as main. However, if a parity error does occur, the information is taken from the memory not having the parity error. As each parity error occurs, a program (software) counter is incremented. If this counter (which is reset periodically) reaches a threshold, then it is likely that the memory causing so many parity errors has a failure. A diagnostic program is then used to check out this memory. If the memory fails the diagnostic, it is permanently switched out and the other memory is used alone. If a parity errors occurs in both memories simultaneously, the program is interrupted and a software decision is made depending upon the circumstances.

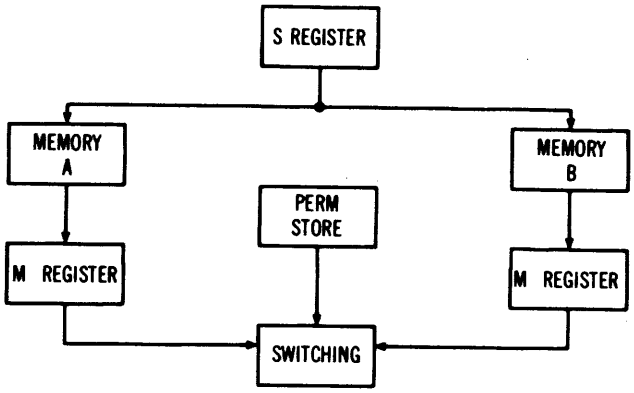


Figure 4 – Dual memory organization

B. Module size

The size of the self-repairing module is a very important consideration. The exact size of the optimum module is a function of the system being implemented, but some general comments can be made:

(1) The module must be large enough so that the number of internal components is significantly greater than the number of components required for switching.
(2) The division between modules must be made in such a way as to minimize the number of inter-module connections.
(3) If duplication without external spares is to be employed, the module should be selected for easy diagnosis or other external testing.
(4) The modules should be approximately equal in reliability.

C. Permanent store implementation

The permanent switch store must be independent of power as discussed previously. Of the approaches that could be used with presently available components, the two most promising are:

(1) Fuses – Fuses can be used in conjunction with solid state devices as shown in Figure 5. They have the advantage of being inexpensive and reliable and the disadvantages of being slow and non-resetable.

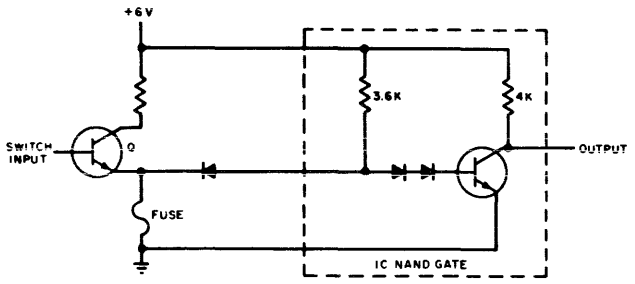


Figure 5 – Permanent store implementation

(2) MNOS Storage Devices[4] – The Westinghouse Research Laboratories have recently announced the development of Metal-Nitride-Oxide-Semiconductor field-effect transistor with a voltage variable threshold. Once the threshold has been set it will remain indefinitely, even though power is removed from the device. The threshold can also be reset by application of a voltage of suitable magnitude and polarity.

Prior to the advent of the MNOS device the fuse appeared to be the most practical implementation of a permanent store. Therefore the reliability calculations in the following section are based on the use of fuses. However, now that the MNOS storage device has become available with its advantages of resetability and high operating speed, it would probably be used in any self-repair implementation.

Reliability calculations

There are seven parts to the self-repairing module. These parts are:

1. Main Unit
2. Auxiliary Unit
3. Comparator
4. Main Failed Permanent Store
5. Auxiliary Failed Permanent Store
6. Output Switches
7. Fault Flip-Flop

The module will continue to operate successfully with certain combinations of failures, and fail with other combinations. In order to determine the module reliability, certain simplifying assumptions must be made in determining which combinations of part failure cause module failure. The assumptions which have been made are:

(1) The most destructive sequence of failures is always assumed. For example, if the Comparator fails after a failed Main Unit has been removed from the module, its failure will not fail the module. However, if the Comparator fails first, then the failed Main Unit will not be removed and the module will fail.

(2) A failure in the Output Switches will fail the module (and hence the system).

(3) A failure in the Fault Flip-Flop will fail the system. This assumption is made because if the flip-flop fails to a permanent set state, an interrupt will occur which the computer cannot clear. It will then have to ignore fault interrupts and the next failure will fail the system.

(4) Shorts to ground and shorted diodes are assumed not to occur at critical points (i.e., comparator, main and auxiliary unit inputs).

It should be noted that the first three assumptions are highly pessimistic. There are many sequences of failures and types of failures in these categories which could occur and not fail the system. Assumption (4) is optimistic, but not unrealistic. Molecular device failures tend to be opens rather than shorts. No significant data were found on the probability of these two types of shorts. The reliabilities calculated on the basis of these assumptions are thus pessimistic

Through the use of these assumptions, Table I was generated. This Table illustrates for which combinations of failures the module will continue to operate. The Output Switches and Fault Flip-Flop are not included in the Table because it has been assumed that a failure in one of these will fail the module. From the Table it is possible to select mutually exclusive combinations of states for which the module will work. With these combinations and making the further assumption of statistical independence between part failures, the reliability (R) of the module may be written as:

$$R = \{R_M R_A R_C + R_M R_{MF} R_{AF} (1 - R_A) + R_M R_A R_{MF} (1 - R_C) + R_M R_A R_{AF} (1 - R_C) (1 - R_{MF}) + R_C R_A R_{MF} R_{AF} (1 - R_M)\} R_S R_F \quad (1)$$

where the R's or reliabilities are the probability that a particular part will be working and the subscripts refer to the parts listed above.

Since the Main and Auxiliary Units are identical and the Main Failed and Auxiliary Failed Stores are identical, then $R_M = R_A$ and $R_{MF} = R_{AF}$. Thus it is possible to simplify the expression for R to:

$$R = \{R_M^2 R_C + R_M R_{MF}^2 - 2 R_M^2 R_{MF}^2 + 2 R_M^2 R_{MF} - 2 R_M^2 R_{MF} R_C + R_M R_{MF}^2 R_C\} R_S R_F \quad (2)$$

A similar fault table can be constructed for the memory. However, the memory table is simpler because there is no comparator. Such a table would show the memory to be working under the following three conditions:

TABLE 1 – Module Failure Combinations

Comparator	Auxiliary Unit	Main Unit	Main Failed Store	Aux. Failed Store	Module	Comparator	Auxiliary Unit	Main Unit	Main Failed Store	Aux. Failed Store	Module
F	F	F	F	F	F	W	F	F	F	F	F
F	F	F	F	W	F	W	F	F	F	W	F
F	F	F	W	F	F	W	F	F	W	F	F
F	F	F	W	W	F	W	F	F	W	W	F
F	F	W	F	F	F	W	F	W	F	F	F
F	F	W	F	W	F	W	F	W	F	W	F
F	F	W	W	F	F	W	F	W	W	F	F
F	F	W	W	W	W	W	F	W	W	W	W
F	W	F	F	F	F	W	W	F	F	F	F
F	W	F	F	W	F	W	W	F	F	W	F
F	W	F	W	F	F	W	W	F	W	F	F
F	W	F	W	W	F	W	W	F	W	W	W
F	W	W	F	F	F	W	W	W	F	F	W
F	W	W	F	W	W	W	W	W	F	W	W
F	W	W	W	F	W	W	W	W	W	F	W
F	W	W	W	W	W	W	W	W	W	W	W

F = FAILED
W = WORKING

(1) The Main and Auxiliary Units (memories) working.
(2) The Main Unit and Main Failed Store working and the Auxiliary Unit failed.
(3) The Auxiliary Unit, Main Failed and Auxiliary Failed Stores working and the Main Unit failed.

From these conditions the reliability (R) of the memory module may be written as:

$$R = \{R_M R_A + R_M R_{MF}(1-R_A) + R_A R_{MF} R_{AF}(1-R_M)\} R_S \quad (3)$$

Since $R_M = R_A$ and $R_{MF} = R_{AF}$ this equation may be rewritten as:

$$R = \{R_M^2 + R_M R_{MF} - R_M^2 R_{MF} + R_M R_{MF}^2 - R_M^2 R_{MF}^2\} R_S \quad (4)$$

The R's are all time dependent, that is the reliability of a unit depends on how long the unit has been operating. The most commonly assumed form of time dependence is –

$$R(t) = e^{-\lambda t} \quad (5)$$

where λ is the failure rate. Most data on component reliability are given in the form of λ or failure rate. For an exponential reliability curve the mean time to failure (MTTF) is $1/\lambda$. The MTTF is often used as a measure of system reliability.

For a non-self-repaired and non-redundant system, the system reliability is an exponential function of time, providing the component of subsystem reliabilities are also exponential functions of time. However, this is not true for a self-repaired or for a redundant system. The system reliability function is not an exponential, but a more complicated curve. Figure 6 shows the reliability, R(t), for a self-repairing module containing an accumulator register. The R(t) for the accumulator without self-repair is plotted on the same axes to illustrate the difference between curves.

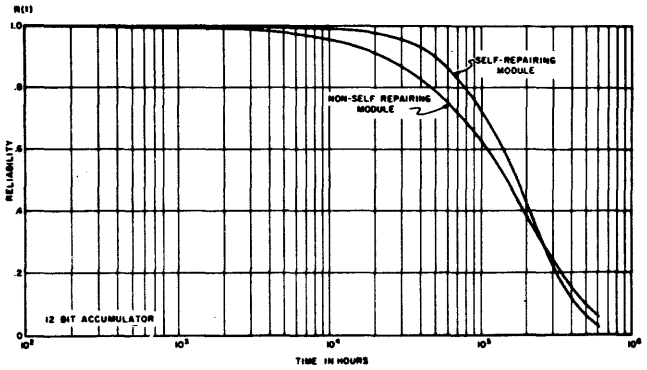


Figure 6 – Reliability versus time

A parameter which can be used in comparing various systems, then, is the reliability of the system at some mission time. The reliabilities of the various parts may be calculated in terms of the mission time and the part failure rates. Failure rates of .05 per 10^6 hours for JK flip-flops and .02 per 10^6 hours for gates are assumed.

Table II compares the reliabilities and component complexities for a hypothetical computer with and without self-repair. The computer is a simple 12-bit machine designed under the self-repair contract for the purpose of demonstrating self-repair techniques. The table shows both the increased complexity and the increased reliability due to self-repair.

CONCLUSIONS

A technique of implementing self-repair, via duplication and spare switching, for digital systems has been described. It has been shown that the use of this technique is feasible for increasing the reliability of these systems. The primary disadvantages of the technique are increased hardware and cost, and increased execution times. However, in systems where reliability is of utmost importance, the reliability increase can offset the disadvantages.

TABLE II – Hypothetical Computer Reliability and Complexity Showing Benefits of Self-Repair

	Module Complexity		Module Reliability			
			10,000 Hours		50,000 Hours	
	Self-Repair	No Self-Repair	Self-Repair	No Self-Repair	Self-Repair	No Self-Repair
I Register L	3.4	1	.9956	.9940	.9767	.9705
I Register R	3.4	1	.9956	.9940	.9767	.9705
N Register	3.5	1	.9969	.9964	.9836	.9822
Sequencer	2.6	1	.9948	.9871	.9693	.9371
Misc. Control	3.0	1	.9735	.9625	.8435	.8261
A Register L	2.4	1	.9951	.9796	.9656	.9021
A Register R	2.4	1	.9945	.9792	.9621	.9003
Q Register L	2.5	1	.9954	.9859	.9714	.9315
Q Register R	2.6	1	.9954	.9863	.9717	.9333
S Register	2.9	1	.9936	.9875	.9636	.9389
P Register	3.0	1	.9918	.9868	.9541	.9357
Memory	2.0	1	.9582	.8248	.5718	.3818
Total Model	2.2	1	.8861	.7012	.3536	.1696

REFERENCES

1 R H WILCOX W C MANN
Redundancy techniques for computing systems
Spartan Books Washington D C 1962

2 P W AGNEW R E FORBES C B STIEGLITZ
An approach to self-repairing computers
Digest of the 1967 IEEE Computer Conference

3 W G BOURICIUS et al.
Investigations in the design of an automatically repaired computer
Digest of the 1967 IEEE Computer Conference

4 T L CHU J R SZEDON C H LEE
Preparation and C-V characteristics of silicon-silicon nitride and silicon-silicon dioxide-silicon nitride structures
Solid State Electronics vol 10 p 897 1967

A study of the data commutation problems in a self-repairable multiprocessor

by KARL N. LEVITT, MILTON W. GREEN and JACK GOLDBERG
Stanford Research Institute
Menlo Park, California

INTRODUCTION

In recent years significant effort has been devoted to the development of techniques for realizing computer systems for which a significant problem of design arises from the unreliability of components and assembly as well as from special constraints on construction and operation.[1] Examples of such systems are computers for aerospace missions and on-line process control.

A significant effort has been directed in the past decade to attempt to solve various facets of the problem of realizing ultra reliable digital systems. This work, also summarized in detail in Ref. 1 has ranged from investigations of fault-masking techniques for various computer subblocks such as arithmetic processor and memory units, to studies of reliability enhancement policies for large systems. It is felt that most of the problems pertaining to enhancing the reliability of isolated digital subblocks are now understood, at least from the standpoint of being able to make sound engineering judgments concerning the utilization of the various techniques. Strictly passive redundancy techniques (e.g., replicated voting logic, error correcting coding methods) have in part been applied to the control and arithmetic processing sections of the Saturn IVb guidance computer, and it has been concluded[2] that the application of such techniques exclusively cannot economically satisfy the computation and reliability requirements of future advanced computers.

This conclusion has prompted many organizations to investigate dynamic error control mechanisms, in which the logical interconnections among the components of the computer may be altered.[3,4,5] In the system schemes investigated, the reconfiguration is employed only at very high functional levels, but it is well-known that there is potentially greater gain to be achieved by employing the reconfiguration at lower system levels.

It has also been recognized that there is considerable advantage to a system wherein the allocation of computation tasks is adjusted so as to be consistent with the available equipment. In such systems, which are colloquially said to embody "graceful-degradation," most of the available equipment is performing useful computations. Among the many references on this approach, in Ref. 6 a single processor structure is assumed, and in Refs. 7 and 8 a multi-processor structure is postulated. In these studies as well as many others, several important items are not treated in depth, namely those relating to:

(1) Diagnostic and replacement policies
(2) Logical design techniques for memory, control, and processing units so that diagnosis and repair are facilitated (or indeed feasible)
(3) Reliable commutation (or data switching) required for the execution of subsystem replacement
(4) The specification of software for the control of diagnosis and repair.

When the new sources of failure that are introduced by the mechanization of the above four items are included in the reliability analysis, it is not clear that the systems will perform as promised. This is an especially critical problem when reconfigurability is extended to low system levels or when the capability for graceful degradation is provided.

Hence, it is imperative that these four items be studied in detail, i.e., that design schemes be developed that minimize the new sources of failure, and that reliability analyses be carried out so as to accurately estimate overall system reliability. In this paper we develop a multiprocessor model—the structure which appears to be most appropriately matched to our computation requirements—and then study the data commutation problem within the framework of this multiprocessor organization.

A multiprocessor model

Many descriptions of multiprocessor systems have appeared in the literature,[8,9,10] and several contemporary computer systems[11] rely upon multiprocessing. Most of these previous descriptions have been concerned with (1) gross estimates of system reliability, assuming for example, that diagnosis and switchover are always executed correctly; (2) scheduling analyses and simulations to facilitate the determination of system responses to various inputs; and (3) the specification of software that will enable the optimal utilization of the hardware. One of the most important functions within a multiprocessor is the switching of data and control signals among the component blocks. The components needed for such switching are themselves sources of system error, and so it is necessary to design switching or commutation networks to achieve the utmost reliability.

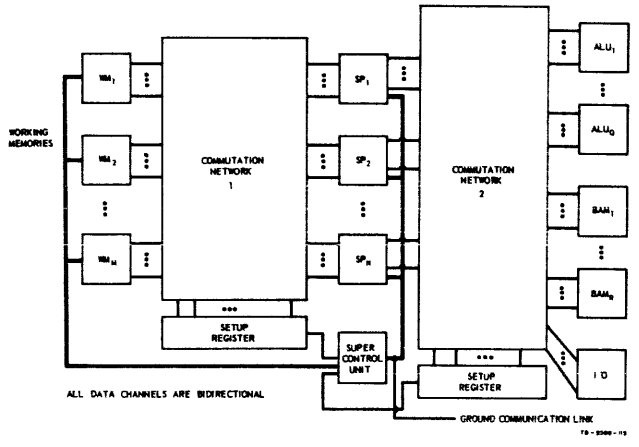


Figure 1 – Multiprocessor computer system block diagram

The model of a multiprocessor with which we will be concerned is depicted in Fig. 1. The system consists of a set of M high-speed working memories (WM); a set of N simple processor and control units (SP); a set of Q arithmetic logic units (ALU); a set of R back-up memories (BAM); an input/output device controller (I/O) for which we provide sets of spare registers, counters, buffers, and real-time clocks; two commutation networks (CN); a supervisory control unit (SCU); and two registers for the setup, i.e. the establishment of input/output links, of the commutation networks; it is convenient to view these registers as forming a component of the SCU. In Ref. 12 we describe in detail the functional requirements of each block type, and, in addition, we present a flow description of the system responses to input, interrupt, and error conditions.

It is envisioned that each SP unit will have the capability of executing comparatively simple decision and arithmetic algorithms, the capability of controlling program flow, and the capability of controlling processor allocation and scheduling. An ALU will be used for the execution of complex algorithms requiring extensive processing hardware. The SCU will function as a "referee" in all error control processes, and, in essence, represents the system "hard-core" although it can be superseded by an external command. The BAMs will store the task programs, diagnostic programs, and the setup programs for the commutation networks.

In addition to the interunit communication links provided by the commutation networks, a single data channel (probably serial), shown as bold-faced lines in Fig. 1 is provided, linking the SP and WM units with the supervisory control unit. It was noted by Alonso[10] that a complete multiprocessing system could be designed containing only this single data channel communication link, (although in practice it is doubtful that this link would be serial), but we have included the possibility of multiple-simultaneous communication between SP and WM blocks because of the additional flexibility thus provided and because at this stage it seems appropriate to work with a general model.

One important feature of the system, not shown explicitly in the figure, is that each defined block of the system will have *at least* one distinct power supply associated with it. Furthermore, it is assumed that the power can be disconnected from a faulty block without resulting in the propagation of errors into connecting blocks due to excessive loading on the part of the disconnected unit. Hence, one mode of error control would trivially involve the disconnection of a major system block, i.e., WM, SP, or ALU, upon the detection of a hardware failure. However, it appears that the overall system reliability is enhanced if some capability for repair is incorporated within these block types. We have found that the repair operation is particularly easy to carry out if each major system block is realized as a one-dimensional cascade of identical elements. We call such a logical realization a *byte-sliced* realization since it is natural to assign a byte (containing at present an undetermined number of bits) of each of the registers, adders, decoders, etc., to each element or slice in the cascade.

The repair operation for byte-sliced realizations, assuming each major block contains several redundant slices, then requires the routing of external data to and from only the working, i.e., unfailed slices, of the block, and also the "shorting" of data internal to the block, e.g., arithmetic carry or control information, around faulty slices.

In Refs. 1 and 12 logical design techniques are presented to demonstrate the feasibility of realizing in a byte-sliced structure, the memory, arithmetic logic, and microprogram control functions.

Thus, it is seen that the commutation networks will perform the function of establishing communication links between WM-SP units and between SP-ALU units, and also the function of repairing the units by routing data only to working byte-slices.

Commutation requirements

The data switching functions described in the previous section give rise to a variety of specifications for commutation networks. In this section we will classify the major network types, and in the remaining sections we will present a number of detailed designs for the various types.

In addition to meeting the particular switching requirements (e.g., number of active paths, number of configurations, etc), several engineering constraints should be considered in the design of practical networks. Thus, for any commutation function it is desirable to synthesize networks for which

(1) The design is economical
(2) The network setup is not difficult
(3) The data transfer is rapid
(4) Failures in the commutation network do not disable either the commutation network or the modules served by the network (this tolerance to CN failures should be achieved with minimal increase of network complexity), and
(5) If the commutation network is "repairable", the diagnostic routines should be easy to specify and of minimal length.

We have classified two types of data commutation for the task of module assignment, (i.e., the assignment of, for example, WMs to SPs) namely:

(1) Complete permutation – complete utilization
(2) Complete permutation – incomplete utilization, and three types of commutation for the task of repair namely,
(3) Incomplete permutation – order preserving
(4) Incomplete permutation – nonorder preserving
(5) "Shorting."

These five commutation functions are described schematically in Fig. 2 where the specific applications of each function are also listed.

The assignments associated with the *complete permutation – complete utilization* [CPCU(N)] function is obvious; the commutation network is to be capable of permuting in an arbitrary manner a set of N-input data lines (all lines active) emerging, for example, from a set of memories, to a set of N-output lines incident to, for example, a set of SP units.

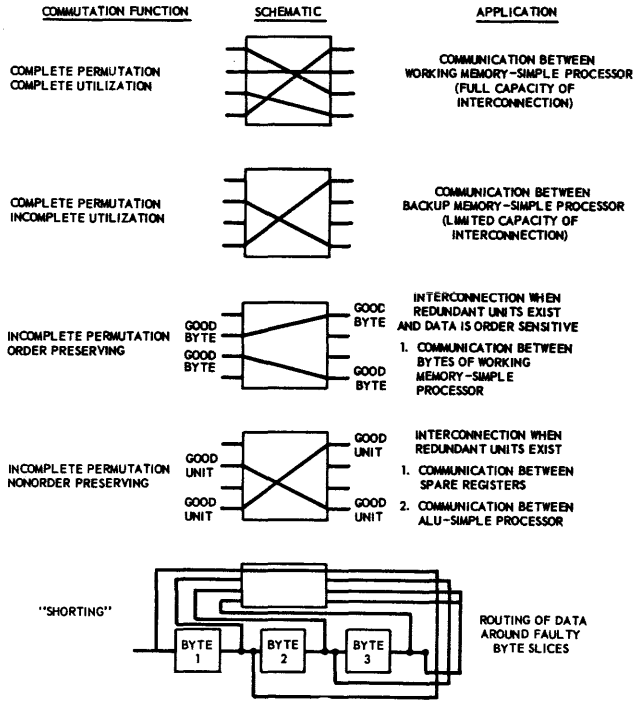


Figure 2 – Classification of data commutation requirements

In the illustration a data transfer path may represent a parallel set of lines containing one computer word (24-56). The assignments associated with the *complete permutation – incomplete utilization* [CPIU(N,m)] function differ from those associated with the CPCU function in that for the former, not all terminals are active at any one time, i.e., only a subset containing m-inputs of the total of N-inputs and outputs need to be interconnected at a given time. For the *incomplete permutation – order preserving* [IPOP(r,m)] function, a subset containing m inputs of the r-inputs, say for example associated with the working byte slices of a simple processor and control unit, is to be connected to a subset of the outputs, say associated with the working byte slices of an arithmetic logic unit, but with the restriction that spatial ordering of the input signals is to be preserved at the output. The preservation of order is clearly required in the example since the data to be commutated is a binary number. The assignments associated with the *incomplete permutation – nonorder preserving* [IPOP(r,m)] function differ from those of the IPOP case in that for the former preservation of order is not a requirement. It may be noted that this function differs from the CPIU function in that the CPIU function requires arbitrary specification of terminal pairs. For the *shorting* function the outputs of a given byte slice are either to be connected to the succeeding stage (slice) or "shorted" around that succeeding slice.

Corssbar solutions to the commutation network design problem

The obvious solution to the commutation network design problem relies upon the use of a single-level crossbar switch, similar to the type commonly found in central telephone exchanges. In Fig. 3 we display a schematic representation of a crossbar switch serving a set of WMs and SPs. Here the crossbar, where each single pole-single throw switch represents a crosspoint, performs both the CPCU(N) and IPOP (r,m) requirements. Clearly a crossbar with N^2r^2 crosspoint would be sufficient, but it was shown in Ref. 1 that actually $N^2(r^2-m^2)$ crosspoints are sufficient. In any event it is seen that $28 \cdot 10^4$ crosspoints are required to serve 25(=N) processors and memories, 32(=r) total bytes, and 24(=m) bytes required for computation. Since a multiprocessor of this complexity is not unreasonable, there is considerable motivation to seek more economical commutation network designs.

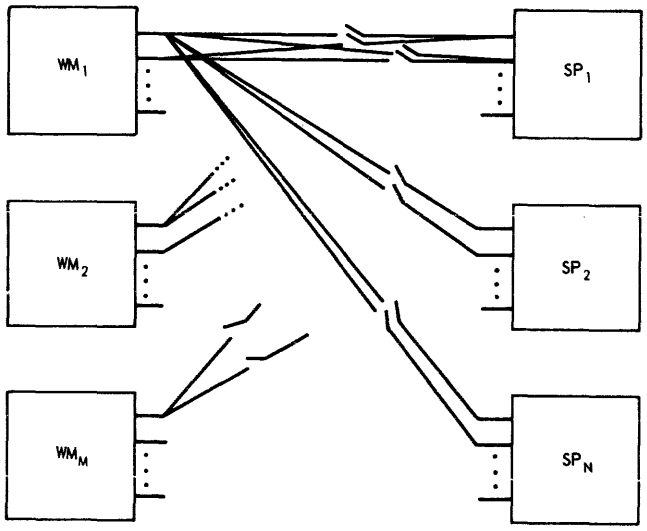


Figure 3 – "Crossbar" realization of commutation function

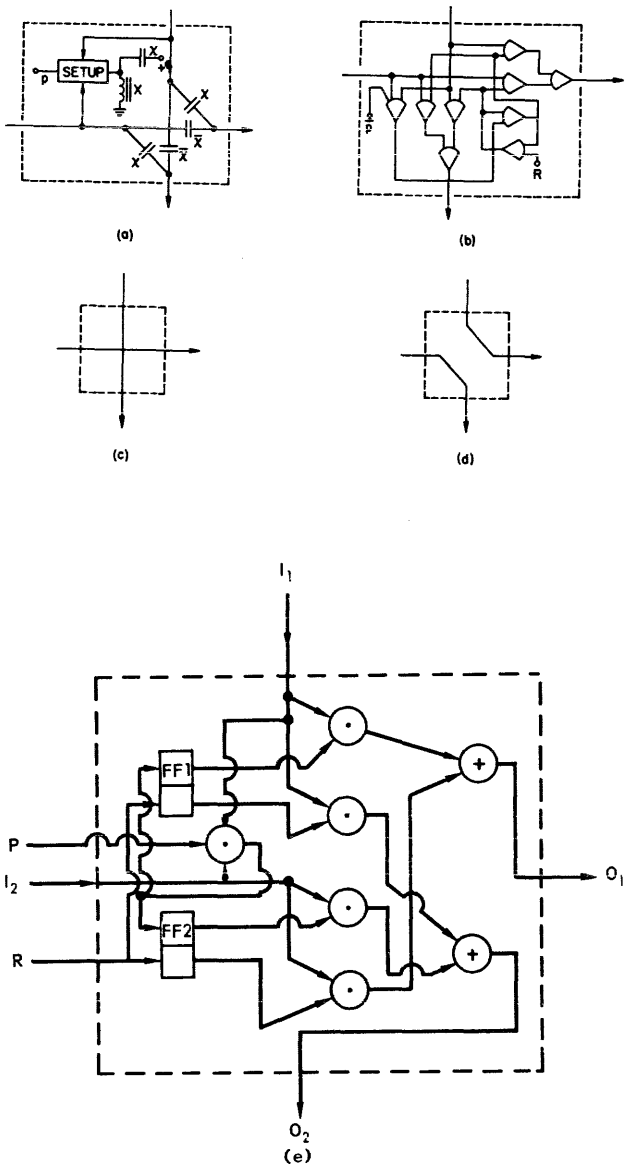


Figure 4 – Basic cell

The primitive building block of commutation networks

Most of the commutation networks to be described in succeeding sections will be composed of interconnections of the "cell" shown in Fig. 4. We have found that arrays of such cells provide a very attractive balance among the various engineering constraints listed in one section. In addition, the uniformity of cell types and the regularity of array structure make such arrays very well suited to advanced microelectronic technology (LSI).

In essence, the cell is a double-pole, double-throw reversing switch controlled by a storage element (e.g., a flip-flop), with some means provided for setting the storage element to the desired state. Figure 4(a) shows a relay-contact version, analogous to circuits in the MOS technology, and Fig. 4(b) a NOR gate-realization of the cell in question. The two modes consist of a "crossing" Fig. 4(c) and a "bending" Fig. 4(d) of the pair of input leads to the pair of output leads. Figure 4(e) depicts a redundant flip-flop version of the cell, for which any single component failure will result in one of two possible failure conditions, namely (1) the cell can realize only one of two possible modes, i.e., the bend or the cross, which we will call the "stuck-function" condition or (2) one output lead contains a faulty signal, which we shall call the "bad-output" condition. Table I summarizes the failure conditions resulting from various component failures of the cell of Fig. 4(e).

TABLE I – Failure conditions for basic cell

Component Fault	*Failure Condition*
Faulty OR gate	Bad-Output
Faulty 2-input AND gate	Bad-Output
Faulty 3-input AND gate	Stuck-Function
Flip-flop stuck in a mode	Stuck-Function
Same logic value on two outputs of a flip-flop	Bad-Output

(It is assumed here that if one of the flip-flops is stuck in a particular mode, the other flip-flop is permanently set to this mode, hence resulting in the stuck-function failure.)

Commutation networks for complete permutation – Complete utilization

1. Nonredundant networks

The synthesis of economical CPCU(N) networks based upon two-state cells has been discussed elsewhere.[13,14,15] The most efficient known procedure is based upon the construction indicated in Fig. 5(a) where the subnetworks P_A and P_B are themselves CPCU networks; for N even each subnetwork serves N/2 inputs otherwise P_A serves (N+1)/2 inputs and P_B (N−1)/2 inputs. It is easily shown that the number $N_1(N)$ of cells required is

$$N_1(N) = N\langle \log_2 N \rangle - 2^{\log N} + 1^*$$

which is asymptotically close to the lower bound of $\langle \log_2(N!) \rangle$ cells.*

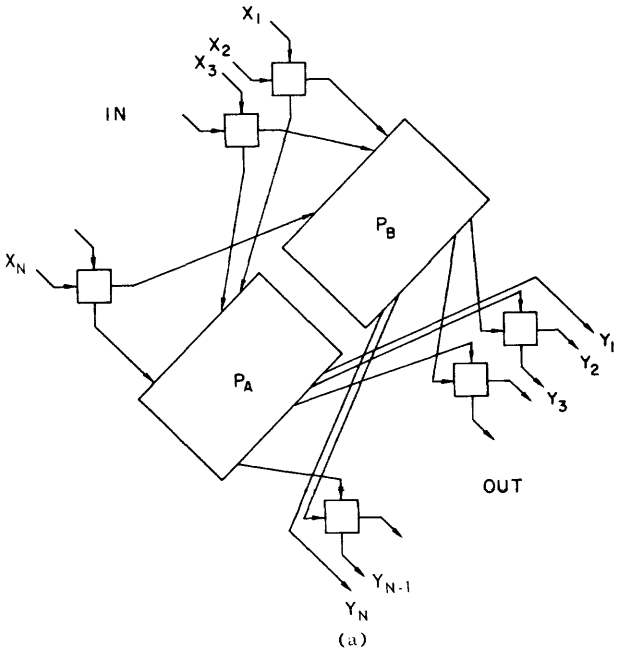
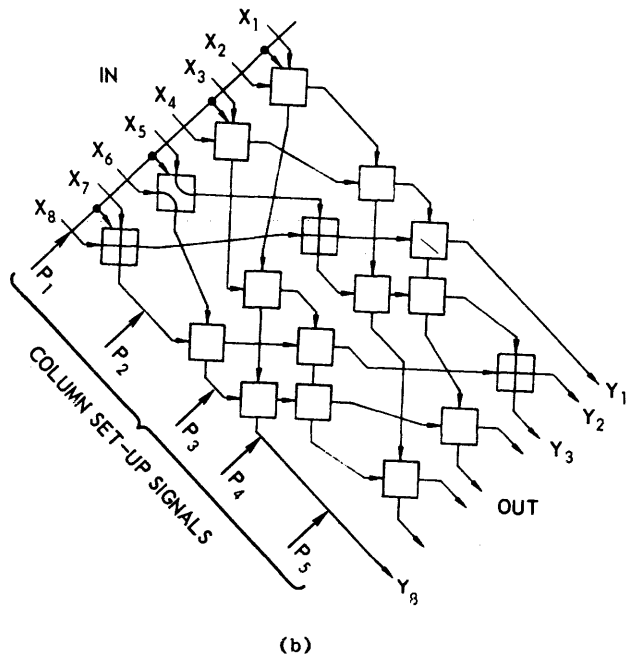


Figure 5 – Networks for complete permutation – complete utilization

*The symbol $\langle x \rangle$ denotes the smallest integer $\geq x$.

It is of interest to investigate efficient techniques for setting-up the network cells to realize the necessary mode for a particular permutation. Consider, for example the network for N = 8, shown in Fig. 5(b), and assume that we require the setting of the cells as shown. Referring to Fig. 4 we see that each cell is in the "crossing" mode upon resetting the flip-flop. The cell is set to the "bending" mode by the coincidence of logic 0 values on the data inputs and logic 1 value on the P input. Clearly then by applying a 1 to the P-input of all cells in a given level of the network – in Fig.5(b) a set of values $X_5 = X_6 = 0$, $P_1 = 1$ will set the cell serving X_5 and X_6 – and appropriately setting the N input signals, the network can be setup in a time interval proportional to the number of levels in the network. There are 2 $\log_2 N - 1$ levels in the CPCU(N) networks presented above.



2. Byte-sliced commutation networks

In this section we are concerned with the behavior of the CPCU networks under cell fault conditions and a simple technique of accommodating to these faults.

It is clear that for the case wherein the basic cells are double pole, double throw, reversing switches, each "bad-output" cell failure results in an error only on a single output of the network, and similarly each "stuck-function" failure results in an error on a maximum of two outputs. (In the latter case it is sometimes possible to accommodate to this failure type by appropriately setting the working cells of the network. This accommodation technique is discussed in detail in a later section). Unfortunately, for

the case of r-pole cells a single failure could result in the inability of the CPCU network to realize several of the WM-SP assignments.

This state-of-affairs is significantly improved by byte slicing the CPCU network as shown in Fig. 6. In this case, the bytes (where each byte is assumed to contain b-bits) of the WM's are permuted in separate networks, that is, the first CPCU has as inputs the first byte of each WM, the second CPU has as inputs the second byte of each WM, etc. The outputs of the first CPCU are ultimately directed to first byte of each SP, etc. It is thus seen that for this byte sliced realization, which of course, requires the distribution of the cell memory flip-flops among each of the CPCU's, a cell failure disables a single *byte*. These commutation network byte failures can be accommodated for in the identical manner proposed for other byte failures, i.e., by the use of the incomplete permutation-order preserving networks.

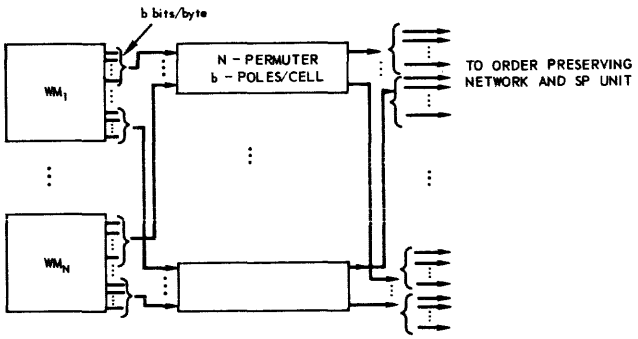
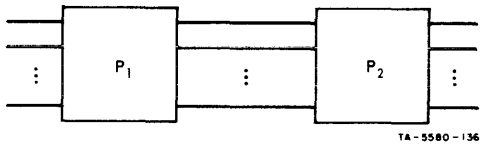


Figure 6 – Byte-sliced permutation network

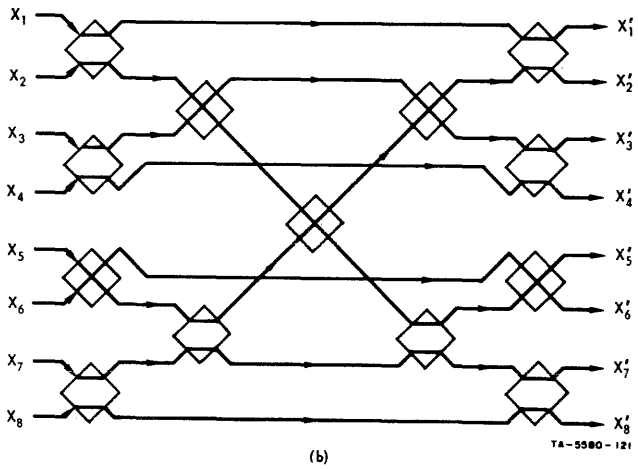
3. CPCU networks insensitive to cell failures

a. The "stuck-function" fault

For the moment consider only the case in which the network is fault-free or has precisely *one* bad switch (cell). Figure 7(a) illustrates a straight-forward solution to the single error correction problem. If P_1 and P_2 are both full permutation networks, then a fault occurring in one of them (such a fault being of the stuck-function type that does not disturb lead continuity) has no effect on the operations of the other network. Obviously this is a rather wasteful approach since all of the remaining switches in the network containing the fault contribute nothing toward forming the desired permutation. Instead let P_2 of Fig. 7(a) represent a permutation (CPCU) network and P_1 be a network specifically designed to undo the damage caused by a fault in P_2. Then if a fault occurs in P_2 it can be "repaired" by P_1 while a fault in P_1 causes no trouble because P_2 is a full permuter. What is required of the network P_1? A single fault in P_2 causes a simple interchange of some particular pair of leads at some cell within the network. This can only show up as a spurious reversal of exactly two leads at the output. Of course, one might be lucky enough to have the switch fail in the correct position so that no trouble would occur. In any event it would be *sufficient* that the network P_1 be capable of effecting the interchange of an arbitrary pair of input leads without changing the relative assignments of the other input leads.



(a)
TWO PERMUTATION NETWORKS IN TANDEM



(b)
A "DOUBLE-TREE" NETWORK

Figure 7 – (a) Two permutation networks in tandem; (b) A "double-tree" network; (c) stuck-function correcting networks

The "double'tree" (D-T) network of Fig. 7(b) can do this job. Any pair of input leads can be directed to some switch on the left of the center switch. At this switch (to the left of the center switch) the leads may be interchanged. Whatever switch settings are required to do this, we *copy* by reflection in the (imaginary) center-line to the right-hand part of the network with the exception of the switch that actually effects the interchange. The corresponding switch in the right-hand part of the network is set in the opposite state. The scheme is illustrated in Fig. 7(b) for the particular case of $N = 8$ in which we wish to interchange inputs X_2 and X_5. Here the switch settings to the right and left of the center-line are identical, and the center switch effects the desired interchange. Similar networks exist for all values of

N and are obtained by "pruning" the corresponding tree networks for the next largest power of two greater than N.

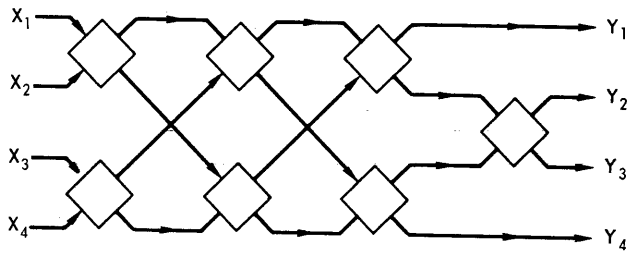
When one of these "double-tree" networks is placed in tandem with a full permutation network, we are then able to correct the effect of *one* switch failure wherever it may occur in the composite total network formed by P_1 and P_2. If P_2 is one of the CPCU type commutation networks we have already considered, it will be found that the input peripheral switches of P_2 and the output peripheral switches of the D-T network match up into tandem pairs of individual switches. Note for example the pairing of leads at the input of the network of Fig. 5(a) and the similar pairing of output leads in Fig. 7(b). Whenever such a pairing occurs, we may omit one of the two switches *if* we have provided for the possible failure of one or the other of them. We may therefore omit the entire column of peripheral output switches from any of the D-T networks whenever we adjoin them to one of the CPCU networks. The single error correcting capability is not affected by this "pruning" operation. Call the network that results from the removal of the output switches from the "double-tree" network the TDT(N) network (for truncated double-tree of N leads). Then single-error correction of any CPCU (N) network can be obtained by the tandem addition of one TDT(N) network. Furthermore, if one considers the possible effect at the output of the CPCU (N) of the *multiple* failure of switches, it turns out that the addition of p TDT(N) networks in tandem to any CPCU(N) network will suffice to correct up to p switch failures in the total network. The argument is much the same as the foregoing one, depending on the possibility of decomposing all such multiple failures into separate pairwise lead interchanges.

To estimate the cost of error protection according to the foregoing scheme, we note that the TDT(N) networks contain approximately (3/2)N switches. To correct p errors then takes about 3/2 (pN) switches. Thus if N is very large, we can correct a "few" errors at a cost that is small compared with the total number of switches in the network ($\approx N\log_2 N$). On the other hand, correction of multiple errors with TDT(N) networks does *not* furnish a recipe for creating arbitrarily reliable networks (in the Shannon sense) while still meeting the asymptotic cell count i.e., $\approx N\log_2 N$. We have not yet discovered how to do this although it appears likely that it is possible. If anything can be concluded from results obtained for small values of N, it seems that single fault correction should be obtainable at a cost of $\langle \log_2 N \rangle$, extra switches in excess of the $N_1(N)$. In particular we can exhibit specific networks that correct one fault and have switch counts as indicated in Table II.

TABLE II – Number of Cells in Single "Stuck-Function" Correcting CPCU Networks

Leads	Switches in Redundant Network	$N_1(N)$
2	2	1
3	5	3
4	7	5
5	11	8
6	14	11

In each case the number of switches shown in Table II is exactly $\langle \log_2 N \rangle$ larger than the corresponding value of $N_1(N)$. In the cases N = 2, 3 and 4 it is reasonably certain that these realizations are the minimal ones that exhibit the single fault correcting property. An example of a single fault correcting network for N = 4 is illustrated in Fig. 7(c).



(c)

As a result of extensive experimentation one feels impelled to make the following conjecture:

The cost of protection against (correction of) p faults in a CPCU(N) network is no more than the difference in cost between a CPCU(N) network and a CPCU(N+p) network. That is, the cost of correcting each additional fault, say fault i, is smaller than $\langle \log_2(N+i) \rangle$

If this conjecture is indeed true, then there would exist permutation networks of arbitrarily high degree of reliability whose cell-count would not exceed $K \cdot N_1(N)$ where K is a constant related to the probability of a switch failure.

b. An alternative single stuck-function-correcting construction

A different and slightly more economical [than the TDT(N) device] method of providing single fault protection in CPCU(N) networks stems from the observation that the construction of Fig. 5(a) can tolerate one cell failure in any peripheral cell if an extra cell serving outputs Y_1 and Y_N column is retained rather than deleted. The reason for deleting this cell in the first place was that exactly one peripheral cell

is unnecessary in the irredundant case. Since all of the peripheral switches are exactly equivalent in function, it makes no difference which one we delete. Hence, by retaining all of them we can ignore exactly one failure in any of them. If each of the "internal" permutation networks that implement the construction of Fig. 5(a) are augmented in the same way by replacing the deleted peripheral switch of the CPCU(N) configuration then we obtain a network, call it $S_1(N)$, that can tolerate *one* switch failure anywhere. An example of such a network, namely $S_1(4)$ is shown in Fig. 8. We notice first that the number of cells, $k = 8$, so that the network is *not* minimal [see Fig. 7(c)]. However, on counting the redundant cells required when $N = 2^r$ we find that the extra cells are exactly $N - 1$ in number. This means that in the special case $N = 2^r$ the networks $S_1(N)$ are more economical than those produced by annexing a TDT(N) network to a CPCU(N) network. Recall that this required about 3N/2 extra switches.

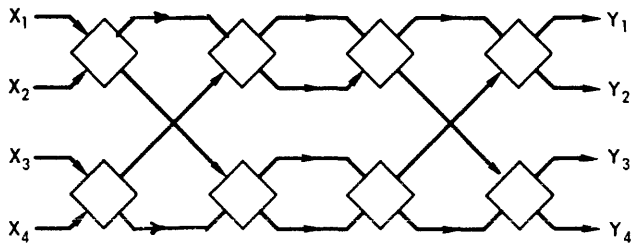


Figure 8—Non-minimal redundant 4-permuter—single stuck-function correcting

c. Correction of bad-output fault types

If one considers the effect of a single fault of this type on an otherwise full permutation network, it is apparent that exactly one output lead will receive an incorrect signal in response to the applied inputs. We can remedy the situation by adding one extra lead to the N-permutation network if we can also make sure that the input signals are applied to, and the output signals derived from the correct subset of N leads. One way of arriving at this situation is illustrated in Fig. 9.* The figure is drawn for the case $N = 4$, but the method is perfectly general. To permute N leads we employ an irredundant $N + 1$ permuter flanked on input and output with a "ladder" network having N switches. If the $N + 1$ permuter has a bad cell lead, this will show up as a failure of one of the signals on leads A, B, C, D or E of the "internal" $N + 1$ permuter to arrive at its specified output. By setting the switches of the ladder network in the obvious manner, the fault can be corrected, as illustrated in Fig. 9 for the case wherein input C does not arrive correctly at internal output 4.

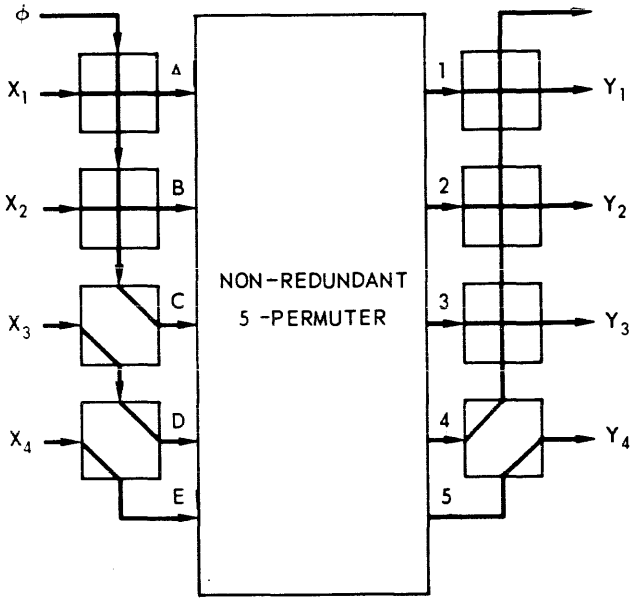


Figure 9—Network for correcting bad-output faults

The cost of correcting one failure by the method of Fig. 9 is 2N switches for the ladder networks plus $\langle \log_2(N+1) \rangle$ extra switches [assuming the CPCU(N) construction of Fig. 5(a)] to implement the $N + 1$ permuter rather than the N permuter. This cost, $\approx 2N + \log_2 N$ is asymptotically negligible compared with the cost of a CPCU(N) network for large N. It is obvious that the foregoing construction can be extended to correct multiple faults of the bad-output types.

Commutation networks for complete permutation—incomplete utilization

We are assuming here that the commutation network is to serve N-inputs and N-outputs [similar to the function of the CPCU(N) network], but in this case it is only necessary to provide simultaneous connections between m ($m < N$) inputs and outputs. Such a commutation function is referred to as embodying complete permutation—incomplete utilization and is denoted as CPIU(N,m). It is desired to specify a network which is more economical in terms of cell-count and/or is easier to set up than a CPCU (N) network which, of course, also achieves the CPIU(N,m) function.

It is easy to see that a CPIU(N,m) network must contain enough two-state cells to specify $\binom{N}{m}^2(m!)$ possible permutations; thus $N_2(N,m)$ the number of cells in the network must satisfy

*Clearly accommodation cannot be made for a bad-output failure on a cell immediately preceding a network. In this case the appropriate byte is disabled.

$$N_2(N,m) \geq \langle \log_2[({}^N]^2(m!)] \rangle$$

or

$$N_2(N,m) \geq 2\log_2\binom{N}{m} + N_1(m) - [N_1(m) - \langle \log_2 m! \rangle]$$

The above formula suggests that the CPIU(N,m) function could be realized, as depicted in Fig. 10, by a network composed of a CPCU(m) block (m-permuter) "sandwiched" between two "combination" networks.

It is assumed that a combination network serving m inputs and N outputs, denoted as a COM(m,N) network, is to have the capability of connecting the set of m inputs onto any specified set of m output leads, without regard to the order of these signals on the outputs. A similar definition applies to a COM(N,m) network where the number of inputs is assumed to exceed the number of outputs. A COM(N,m) network succeeded by an m-permuter would serve as a complete permutation-complete utilization network for the case of N-inputs, m-outputs, $m < N$.

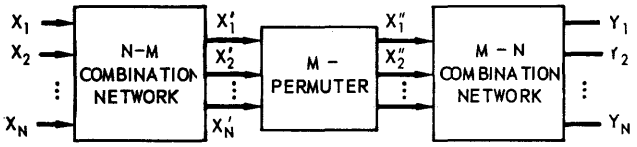


Figure 10—Schematic representation of decomposition of a complete permutation incomplete utilization network

A lower bound on the number of cells $N_c(N,m)$ required for an m, N (or an N, m) combination network is given by:

$$N_c(N,m) \geq \left\langle \log_2 \left[\frac{N!}{m!(N-m)} \right] \right\rangle.$$

Asymptotically, from Stirling's formula, we find that a network composed of N-1 cells might be sufficient to perform the combination function.

We have *not* found combination networks, composed of the 2-input basic cell that approach this N-1 cell bound, as closely as the CPCU(N) networks approach the $\langle \log_2 N! \rangle$ bound. However, it is not difficult to specify a COM(N,m) network which requires only N two-state cells but wherein each cell contains $m+1$ inputs.

Consider the two-state cell depicted in Fig. 11, with m horizontal inputs ($m = 3$ for the case shown), $I_1, I_2, \ldots, I_m$, m horizontal outputs $O_1, O_2, \ldots, O_m$, one vertical input, X_i, and one dummy (unused) vertical output. In the "cross" mode the three horizontal inputs are transferred unaltered to the three horizontal outputs. In the "bend" mode we effect the transformation $X_i \rightarrow O_m$, $I_m \rightarrow O_m$, $I_m \rightarrow O_{m-1}, \ldots, I_2 \rightarrow O_1$. A COM(m,N). network is easily synthesized as a cascade, containing N of these augmented input cells, as shown in Fig. 12 for the case $m = 3$, $N = 6$. The appropriate cell modes are shown for the case where it is desired to connect the inputs X_2, X_4, and X_5 to the three outputs, Y_1, Y_2, Y_3 that are the horizontal outputs of the last cell in the cascade. It is noted that the first cell in the cascade actually requires no horizontal inputs, the second cell only 1 horizontal input, ..., the mth cell only $m - 1$ horizontal inputs. However, if we consider the approximate cost of a cell to be proportional to the number of terminals served, then the cost of the combinational realization of Fig. 12 is of the order of mN. We can find realizations which although require a number of cells in excess of N, exhibit a cost measure significantly less than the network previously described.

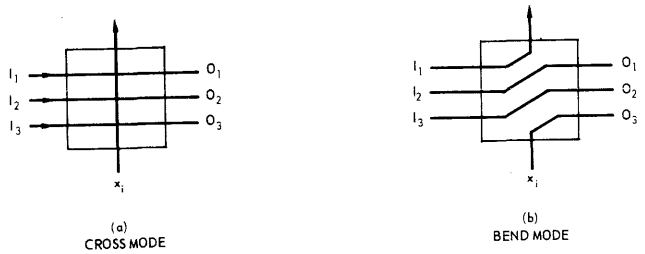


Figure 11—Basic cell with augmented set of inputs (a) cross mode; (b) bend mode

We will now recursively synthesize a COM(m,N) network composed of basic two input cells.

Consider the COM(m,N) network depicted in Fig. 13, where it is assumed that $2|m$ and $2|N$. The subnetworks Q_1 and Q_2 are themselves combination networks, each with half of the number of inputs of the total network. This arrangement requires a number $N_2(m,N)$ of cells which satisfies:

$$N_2(m.N) = 2N_2(m/2, N/2) + \frac{N}{2} - 1.$$

Solution of the recursion for $N = 2^r$, $m = 2^{r-1}$, (using $N_2(1,2) = 1$*) yields

$$N_2(2^{r-1},2^r) = 2^{r-1} + (r-2)2^{r-1} + 1$$

or

$$N_2(2^{r-1},2^r) = \frac{N}{2}\log_2(N) - \frac{N}{2} + 1.$$

*It is clear that a single 2 input cell where one of the inputs is *not* used is a COM(1,2) network.

For general m, N the number of cells required is

$$N_2(m,N) = \langle \text{Min } [m\log_2 n - 2m + N + 1, (N-m)\log_2(N-m) + m - N + 1] \rangle$$

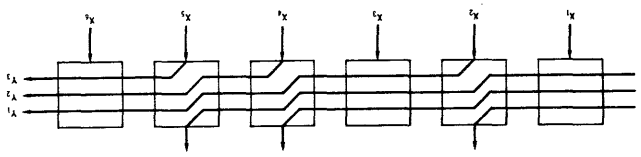


Figure 12 – An N, m combination network

A constructive proof that this network is capable of developing a path from the m inputs to an arbitrarily selected set of m outputs is as follows. Let the inputs and outputs be labelled X_1, X_2,..., X_m, and Y_1, Y_2,..., Y_N as indicated in Fig. 13. We will start by indicating the appropriate modes for the $\frac{N}{2} - 1$ (output) cells serving outputs $Y_2, \ldots, Y_{N-1}$, so that for an arbitrary selection of m outputs, exactly m/2 of these outputs are connected to Q_1 and m/2 to Q_2. If an output cell serves two selected outputs, it can be arbitrarily set to either mode. The output cells which serve the remaining set of selected outputs are then set to the appropriate mode so that the first of these outputs (including possibly Y_1) is connected to Q_1, the second connected to Q_2, etc.

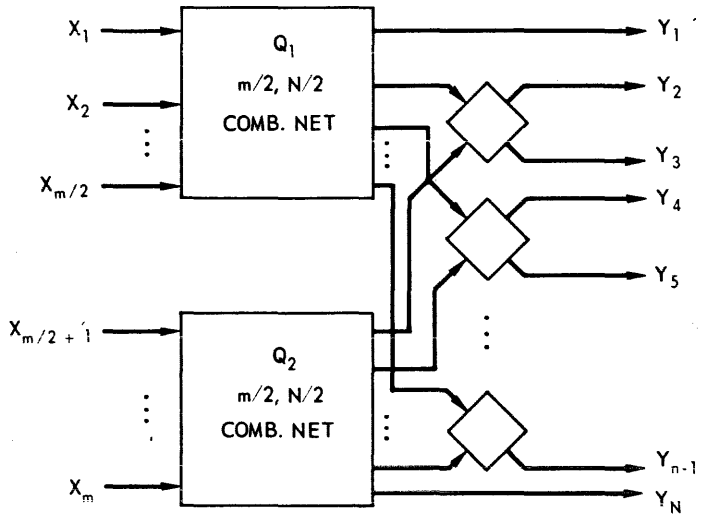


Figure 13 – Recursive approach to m – N combination network

The entire procedure is now repeated for each of the networks Q_1 and Q_2, etc., until the entire network has been set up. The network for m = 4, N = 8, is shown in Fig. 14, where the cell functions are indicated for the selected output set Y_2, Y_4, Y_5, and Y_6.

For the case of cell failures, techniques similar to those described in an earlier section can be applied such that the commutation function is maintained.

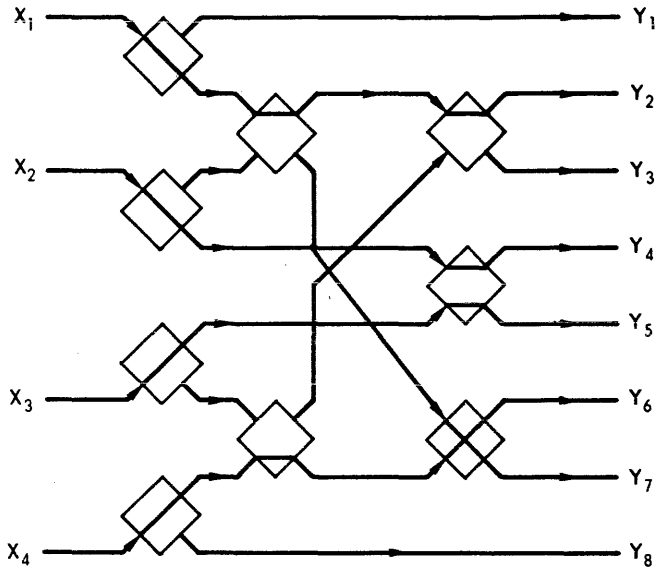


Figure 14 – 4-8 combination network

Commutation networks for incomplete permutation – order preserving and incomplete permutation – non-order preserving

It is recalled that the memory modules, arithmetic logic units, and simple processor and control units can be realized as a cascade of identical byte slices. These modules can continue to function, upon the occurrence of failures in slices, if several spare slices are provided, and if a commutation network is provided to route the signals between operating slices. The function of such a commutation network, denoted as an incomplete permutation – order preserving network, IPOP(r,m), is to set up connecting paths between an arbitrary set of m inputs and an arbitrary set of m outputs, both sets of which are subsets of the r-inputs and r-outputs $r \geq m$, so that the signal order is the same at both input and output. Goldberg[16] has described an efficient serial transfer IPOP network, but the data transfer rate appears to be insufficient for many applications.

It can be shown[12] that a lower bound on the number of two state cells required for an IPOP(r,m) network is between $\langle \log_2 \binom{r}{m} \rangle$ and $2\langle \log_2 \binom{r}{m} \rangle$ although the upper value appears to be tighter. It is possible to realize the IPOP(r,m) function in a network composed of 2r two-state cells, where each cell contains m + 1 inputs. It is seen that this network approximately satisfies the

$2\log_2(\frac{r}{m})$ bound for the case $m = r/2$ since

$$\lim_{r \to \infty} \log_2 \left(\left[\frac{r}{\frac{r}{2}} \right] \right) = r - 1.$$

The basic cell is the type shown in Fig. 11, and the network is displayed in Fig. 15 for the case $r = 6$, $m = 3$; the modes of the cells are such to realize order-preserving connections between inputs X_2, X_3, X_5 and outputs Y_4, Y_5, Y_6. Even though the number of horizontal inputs served by the first $m - 1$ cells in the cascade and the number of horizontal outputs served by the last $m - 1$ cells in the cascade can both be reduced, the cost of this network is of the order of $2mr$, a not inconsiderable cost. Similar to the situation with the other commutation functions, the cost of the realization is significantly reduced if the two input cell is used as the basic primitive block.

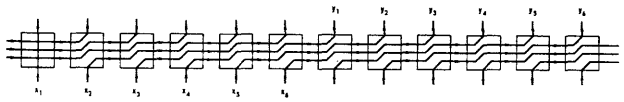


Figure 15—An incomplete permutation—order preserving network

A network composed of two-input cells which realizes the IPOP(r,m) function, displayed in Fig. 16 for the case $r = 8$, $m = 4$.

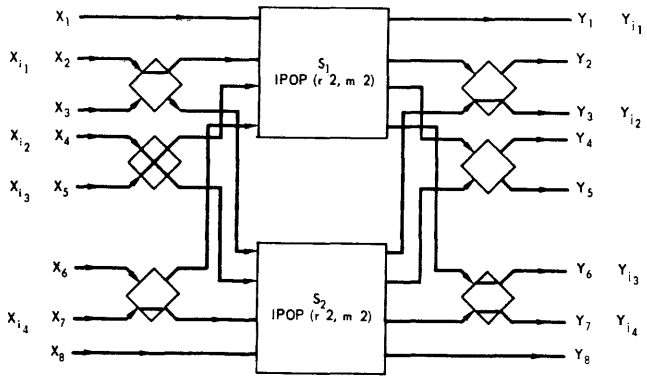


Figure 16—Recursive approach to incomplete permutation—order preserving network

The number $N_3(r,m)$ of cells required, for the case $r = 2^k$, $m = 2^{k-1}$, again using $N_3(2,1) = 1$, is shown to be

$$N_3(r,r/2) = r\log_2 r - \frac{3}{2}r + 2.$$

The recursion technique will yield a similar expression for arbitrary parameters r, m.

The following procedure, for the order preserving case will indicate the proper mode of each cell of the network of Fig. 16, for a given set of m inputs and outputs, and hence will prove that the network can function as an order preserving network. Let the distinguished set of m inputs be $X_{i_1}, X_{i_2}, \cdots, X_{i_m}$, where $i_\alpha > i_\beta$, if $\alpha > \beta$, and the m outputs be $Y_{j_1}, Y_{j_2}, \cdots, Y_{j_m}$ where $j_\alpha > j_\beta$ if $\alpha > \beta$. Consider each of the m inputs as residing in one of two disjoint groups. Group A_I contains those inputs that do *not* "share" an input cell with another distinguished input and group B_I contains those inputs which do share an input cell. We will similarly define groups A_0 and B_0 for the distinguished outputs. The goal is to assign X_{i_1} and Y_{i_1} to the same subnetwork (S_1 or S_2), X_{i_2} and Y_{j_2} to the same subnetwork, etc., and the procedure is as follows.

Assign X_{i_1} and Y_{j_1} to network S_1, by appropriately setting the pertinent input and output cells (except for the case where $X_{i_1} = X_1$ and/or $Y_{j_1} = Y_1$ in which case the assignment to S_1 is automatic). Then assign X_{i_2} and Y_{j_2} to network S_{2_j}; if X_{i_1} and X_{i_2} are in group B_I and/or Y_{i_1} and Y_{j_2} are in group B_0 the assignment to S_2 is automatic. Next assign X_{i_3} and Y_{j_3} to S_1, etc., until all of the m distinguished inputs and outputs have been set. This procedure is then applied to set the pertinent output and input cells of the networks S_1 and S_2, etc. It is clear that this assignment procedure can always be carried out. In Fig. 16 we show the setting of the input and output cells for the case $X_{i_1} = X_2$, $X_{i_2} = X_4$, $X_{i_3} = X_5$, $X_{i_4} = X_7$ and $Y_{j_1} = Y_1$, $Y_{j_2} = Y_3$, $Y_{j_3} = Y_6$, $Y_{j_4} = Y_7$

The same network will realize the IPNOP function although the set-up is somewhat easier than for the POP case.

The techniques for providing failure tolerant IPOP networks are not discussed in this section since they are quite similar to the techniques described previously. We note that a cell failure in the IPOP network (two-input cell type) can disable no more than two byte slices each for the input and output. Since it is assumed that redundant slices are provided, it is possible that a nonredundant network would be used and when cell failures are detected the slices which could not be served by the network would be discarded.

Commutation networks for "shorting"

In Sec. VII we described networks which for a redundant byte-sliced network can serve to route external data between the operating slices of distinct networks (e.g., between an SP and ALU). It was noted that internal data (e.g., control and carry infor-

mation) must be routed between the stages of the byte-sliced cascade. If a stage (or slice) has failed, then the internal data intended for that stage, which clearly comes from its immediate predecessor or successor, must be shorted around that failed stage. This shorting process must be accomplished reliably or else the entire network would be disabled.

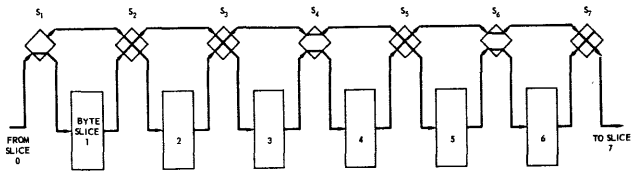


Figure 17 – "Shorting" networks

The shorting function is quite naturally effected with the two-input basic cell, as illustrated in Fig. 17. For convenience we have only indicated a signal flow to the right although it is clear that the network could be modified to handle bi-directional flow. We have shown the appropriate cell modes so that byte slice 2 and byte slices 5 and 6 are shorted out. We note that the network could recover from a single component failure within a cell, which results in either the stuck-function failure or the bad-output failure. However, a more severe cell failure which results in, for example, a permanent logical zero signal on both outputs of a cell would clearly disable the network, i.e., interrupt the signal flow.

Such a failure, which could only result from two component failures within a cell could be accommodated for by the redundant shorting network of Fig. 18. We have indicated the appropriate modes for cells S_1, S_1', S_2, S_2' such that byte slice 1 is shorted out, i.e., the output from slice 0 is directed to the input terminal of slice 2. We have also shown the appropriate modes for cells S_3, S_3', S_4 such that the network continues to function although both outputs of S_4' are faulty. In this case byte slice 3 cannot be used, but the signal flow is not interrupted. Similarly we have shown how the network accommodates to a double-output failure in S_5 in which case slice 5 is bypassed. This technique can be clearly extended to handle failures of greater multiplicity.

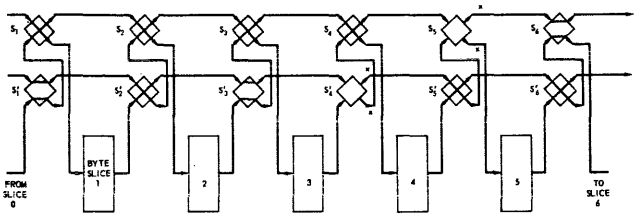


Figure 18 – Redundant "shorting" network

SUMMARY

In this paper we have studied in detail the logical design of networks which are well suited for realizing the various data switching or commutation functions required in a multiprocessor organization where the various modules are repairable. It is assumed that the memory, arithmetic logic, and possibly the simple processor and control modules are realized in a byte-sliced manner – a realization that has been demonstrated to be practical. These commutation networks might also be useful for certain logical functions within the various modules for example, in the distribution of the outputs of a decoder among control inputs of a set of registers. We feel that the designs we have presented, based upon the primitive two-input, two-output reversing cell represent adequate engineering solutions to all of the commutation problems posed, although some theoretical minimization problems still remain. These problems relate to minimum cost designs for the complete and incomplete permutation functions considering both the nonredundant realizations and the realizations which are tolerant to cell failures. In particular our designs for the incomplete permutation functions require a number of cells significantly in excess of the lower bound.

The delay in signal transmission encountered for the networks studied is significantly greater than the delay expected for a simple crossbar realization (approximately $2\log_2 N$ units compared with 1 unit), however, the fan-out and fan-in for the cell array is substantially less than in a crossbar, so that the overall delays may be comparable, for certain dimensions and circuit parameters. If less delay is desired, the networks could be synthesized from, for example, 4-input complete-permutation cells, which would result in one-half the delay, at the expense of somewhat greater total gate cost; however, the failure of such a cell might disable more network outputs than encountered for the two-input cell realizations.

ACKNOWLEDGMENT

The research reported in this paper was supported by NASA Electronics Research Center, Cambridge, Massachusetts under Contract NAS 12-33.

REFERENCES

1 J GOLDBERG K N LEVITT R A SHORT
Techniques for the realization of ultra-reliable spaceborne computers
Final Report-Phase I Contract NAS 12-33 Stanford Research Institute Menlo Park California (September 1966)

2 AES-EPO Staff
AES-EPO study program
Final Study Report volumes 1 and 2 IBM Electronics System

Center Owego New York (December 1965)
3 A AVIZIENIS
A set of algorithms for a diagnosable arithmetic unit
Tech Report no 32-546 Jet Propulsion Laboratory Pasadena California (1964)
4 A AVIZIENIS
A design of fault-tolerant computers
Proc Fall Joint Computer Conference (AFIPS) (1967)
5 W G BOURICIUS et al
Investigations in the design of an automatically repaired computer
Digest of the First Annual IEEE Computer Conference
IEEE Publication 16C51 (September 1967)
6 P W AGNEW et al
An approach to self-repairing computers
Digest of the First Annual IEEE Computer Conference
IEEE Publication 16C51 (September 1967)
7 R P HASSETT E H MILLER
Multithreading design of a reliable aerospace computer
Presented at 1966 Aerospace and Electronic Systems Convention (3-5 October 1966)
8 L J KOCZELA
Study of spaceborne multiprocessing
2nd Quarterly Report Volume II Contract NAS 12-108 Autonetics Division of North American Aviation Anaheim California (October 1966)
9 E C JOSEPH
Self repair: fault detection and automatic reconfigurability
Proceedings of the Spaceborne Multiprocessing Seminar NASA Electronics Research Center Boston pp 41-49 (31 October 1966
10 R L ALONSO et al
A multiprocessing structure Digest of the First Annual IEEE Computer Conference
IEEE Publication 16C51 (September 1967)
11 J F KEELEY et al
An application-oriented multiprocessing system
IBM Systems Journal Vol 6 no 2 (Entire Issue) (1967)
12 J GOLDBERG M W GREEN K N LEVITT H S STONE
A study of techniques and devices for the realization of ultra-reliable spaceborne computers
Interim Scientific Report no 2 Contract NAS 12-33 Stanford Research Institute Menlo Park California (November 1967)
13 V E BENES
Mathematical theory of connecting networks and telephone traffic
Academic Press New York 1965
14 W H KAUTZ K N LEVITT A WAKSMAN
Cellular interconnection networks
Accepted for publication in IEEE Transactions on Electronic Computers
15 A WAKSMAN
A permutation network
Accepted for publication in the Journal of the ACM
16 J GOLDBERG
Logical design techniques for error control
WESCON paper 9/3 Session 9 (September 1966)

A distinguishability criterion for selecting efficient diagnostic tests

by HERBERT Y. CHANG
Bell Telephone Laboratories, Incorporated
Naperville, Illinois

INTRODUCTION

Fault diagnostic procedures are usually derived by means of simulation methods.[1] One of the fault simulation methods is the *digital technique* in which the diagnostic information, viz., the diagnostic tests and test results of a machine, is generated with the aid of a computer program. The input to the program is a logical description of the machine and the output is a sequential testing procedure for the machine, along with the simulated diagnostic test results.[2] A *sequential testing procedure* is one where the next test to be applied depends on the outcome of the previous test.

The efficiency of a sequential testing procedure generated digitally depends largely on the method by which tests are chosen and ordered in the procedure. Some tests, when properly chosen and ordered, will yield a shorter testing procedure and better fault resolvability than others, and therefore, give rise to an "optimum" testing procedure. Previous results indicate that it is impractical to attempt to find a globally optimum testing procedure for any moderate size circuit;[3,4] local optimization techniques are therefore used. At each point in the test generation process, several candidates of tests are tried and evaluated, and the "best" one is chosen. Two criteria for evaluating the "goodness" of a test are available: the *check-out* or detection criterion and the *information gain* criterion.[2,3,5,6] Both criteria, however, share the drawback that tests are evaluated, based on the "ability" to detect or identify faulty *components*, rather than the smallest replaceable module(s) or circuit package(s).

In this paper, a new criterion, called the *distinguishability* criterion, for computing the figure-of-merit of tests to derive efficient testing procedures is introduced. The criterion is aimed at optimizing the diagnosability so as to identify failures only to the circuit package (or the smallest replaceable module) level. It appears that the distinguishability criterion is more practical since the impact of integrated circuits and modern packaging methods makes distinguishability among faults associated with the same module less necessary. The testing procedure so generated tends to yield shorter test sequences and better resolvability.

In Section 2, the sequential testing philosophy is reviewed and two of the existing criteria for selecting tests are briefly described. The distinguishability criterion is introduced in Section 3. Familiarity with References 2, 3, 5, and 6 is recommended.

A review of testing philosophy

The philosophy of sequential testing procedure has commonly been described as the "gedanken-experiment" or "black-box" philosophy.[2,7] A machine or processor is considered to be a black box with input and output terminals. A failure or fault is looked upon as a transformation of the fault-free or "good" machine into a different machine. For example, machine M_i may denote the output of gate Q stuck at low; machine M_j may represent the second input terminal of gate R stuck at high; and so on. Thus, if there are N possible failures in a machine,* the objective of a diagnostic procedure would be to identify one of the $N + 1$ (including the good machine) possible machines or black boxes. The procedure for accomplishing this is essentially a multiple branching experiment in the sense of Moore.[7]

In deriving a sequential testing procedure, a "test" or an input configuration is first applied and the "test result" or output configuration of each of the $N + 1$ machines is computed by simulation. A test is useful if it partitions the collection of $N + 1$ machines into several *equivalence classes*, each of which contains only those machines having the same output configuration. Another test can now be applied to one of the equivalence classes. Again by simulating the behavior of each of the machines in this class, the second test

*The single fault assumption is implied here.

may "partition" the set into smaller equivalence classes. This process is repeated for every equivalence class of machines until each failure or machine is identified, or the remaining subset of machines appear to be indistinguishable.

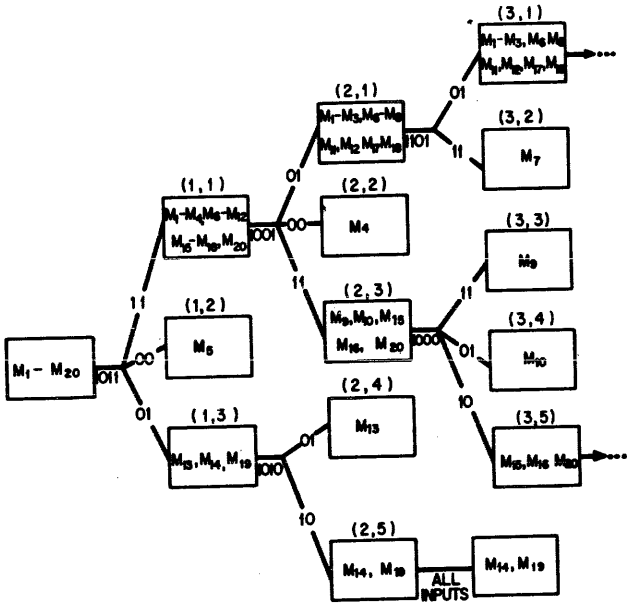


Figure 1 – An example of sequential testing procedure

An example of such a procedure is shown in Figure 1. Machines, or failures, are denoted as M_1, $M_2, \ldots, M_{20}$, with "M_1" representing the good machine. An input configuration or test 1011 is applied which partitions the set into three equivalence classes. Thus, if the output configuration is 11, one can conclude that either the machine is fault-free, or it contains one of the following failures: M_2, M_3, M_4, M_6, M_7, M_8, M_9, M_{10}, M_{11}, M_{12}, M_{15}, M_{16}, M_{17}, M_{18}, M_{20}. Similarly, if the output is 01, the machine contains one of the following failures: M_{13}, M_{14}, M_{19}. It is seen that if the output is 00, failure M_5 is uniquely identified; no more testing is necessary. The entry (i, j) denotes the equivalence class at i^{th} *level* and j^{th} *partition* in the testing procedure. By convention, the top branch (1, 1), (2, 1), (3, 1),... will always denote the equivalence classes containing the good machine M_1.

A second test 1001 may be applied to the set (1, 1) and further partitions (1, 1) into (2, 1), (2, 2) and (2, 3). A different test 1010 may be applied to the set (1, 3) in order to partition machines $\{M_{13}, M_{14}, M_{19}\}$. A third and a fourth, ..., test may be chosen along each sequence to continue the testing as far as necessary. Each failure, or equivalence class of failures, such as $\{M_{14}, M_{19}\}$ in the terminal equivalence class would be identified by a test sequence, such as 1011, 01, 1010, 10.

It is evident that there are many input configurations or tests one may choose, to partition the set(s) (i, j). The criterion for choosing a test depends on the purpose at hand. If the purpose is check-out or fault detection, the *check-out* criterion may be used. This criterion says that for any test T_k at the i^{th} level in the testing procedure, the efficiency of T_k can be measured by

$$\alpha_k \equiv \frac{n(i, 1) - n_k(i+1, 1)}{n(i, 1)} \tag{1}$$

where n(i, j) denotes the number of machines in equivalence class (i, j), and $N_k(i + 1, j)$ denotes the number of machines in equivalence class (i + 1, j) resulted from the partitioning by applying T_k. In other words, the test, which yields the maximum α_k and is therefore, considered to be the best choice, is the one that eliminates the largest number of machines from the good machine equivalence class (i, 1).[2,5]

If, on the other hand, the purpose is diagnosis, the *information gain* criterion may be used. For a test that generates an m-nary partition, its information gain is computed as follows. Let $(i + 1, j_1)$, $(i + 1, j_2), \ldots, (i + 1, j_m)$ be the m equivalence classes which result by applying test T_k to (i, j) (see Figure 2). Then the information gain for test T_k is

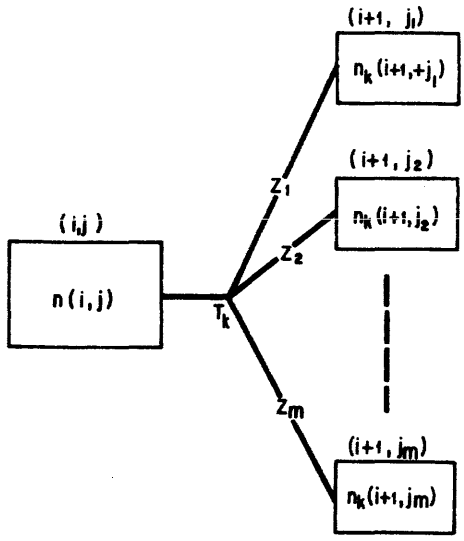
$$\beta_k \equiv -\sum_{s=1}^{m} \frac{n_k(i+1, j_s)}{n_k(i, j)} \log_2 \frac{n_k(i+1, j_s)}{n_k(i, j)} \tag{2}$$

where all faults in (i, j) are assumed equiprobable. The gain β_k defined in equation (2) is due to Mandelbaum[6] and is different from the one used by Seshu.[5] The information gain defined by Seshu is suited for binary partition. The definition is then extended to the m-nary partition case by saying that an m-nary partition can be replaced by a string of m-1 binary partitions. This is done by first computing the gain over the good machine equivalence class (i + 1, 1) with respect to each of the m-1 faulty machine equivalence classes and then taking the sum of the gains. The measure so computed tends to yield a good *detection* testing procedure,

Distinguishability criterion for test selection

The essential idea of the criterion to be described here lies in the notion of "fault distinguishability." The criterion is somewhat analogous to the one proposed by the author for combinational testing procedures.[8]

Suppose there is a collection of machines M_1, M_2, ..., M_{N+1} to be identified. One would like to apply a test to distinguish, if possible, each faulty machine from the good machine M_1, and from all other faulty



NOTATION
- (i,j) = EQUIVALENCE CLASS AT iTH LEVEL AND jTH PARTITION
- n(i,j) = NUMBER OF MACHINES IN (i,j)
- $n_k(i+1,j_s)$ = NUMBER OF MACHINES IN $(i+1,j_s)$, RESULTED FROM APPLYING T_k TO (i,j)
- T_k = TEST OR INPUT CONFIGURATION
- $Z_1, Z_2, \cdots, Z_m$ = TEST RESULTS OR OUTPUT CONFIGURATION

Figure 2 – Partitioning of machines

machines. In other words, there are $\frac{N(N+1)}{2}$ pairs of machines to be distinguished. A test is most useful, and therefore should be chosen, if it distinguishes the largest number of pairs of machines.

In general, a test T_k partitions the set of machines in (i, j) into m equivalence classes (i+1, j_1), (i+1, j_2) ,..., i + 1, j_m). Each equivalence class (i + 1, j_s) (s= 1, 2,..., m) contains $n_k(i+1, j_s)$ machines that are are indistinguishable by test T_k, where the sum $\sum_{s=1}^{m} n_k(i+1, j_s)$ is equal to n(i, j). Then, the test T_k actually distinguishes every machine in $(i+1, j_s)$ from every machine in $(i+1, j_t)$, for $s \neq t$. The total number of pairs of machines distinguished by T_k is therefore equal to:

$$\gamma_k \equiv n_k(i+1, j_1)\,[\,n_k(i+1, j_2) + n_k(i+1, j_3) + \ldots + n_k(i+1, j_m)\,]$$
$$+ n_k(i+1, j_2)\,[\,n_k(i+1, j_3) + \ldots + n_k(i+1, j_m)\,]$$
$$+ \ldots$$
$$+ n_k(i+1, j_{m-1})\,[\,n_k(i+1, j_m)\,]$$
$$= \sum_{s=1}^{m-1} \left[\, n_k(i+1, j_s) \cdot \sum_{t=s+1}^{m} n_k(i+1, j_t) \,\right] \qquad (3)$$

The value of γ_k reaches a maximum of $\frac{n(i,j)[n(i,j)-1]}{2}$ when m = n(i, j) meaning that the test T_k identifies uniquely each of the n(i,j) faults. It reaches a minimum of zero when m = 1, meaning that the test T_k cannot partition the set into smaller equivalence classes. This measure is compatible with the information gain criterion concept which has been discussed previously.[5,6]

As an example consider the case where three different output configurations and the corresponding tests T_1 T_2 and T_3 yield the following partitions:

	n(i+1, j_1)	n(i+1, j_2)	n(i+1, j_3)	n(i+1, j_4)	n(i+1, j_5)	n(i+1, j_6)
T_1:	1	1	1	1	1	1
T_2:	2	1	3			
T_3:	2	2	1	1		

There are six faults to be identified. From Equation (3), the distinguishability γ_k for each test is:

$$\gamma_1 = 1(1+1+1+1+1) + 1(1+1+1+1) + 1(1+1+1) + 1(1+1) + 1(1) = 15$$
$$\gamma_2 = 2(1+3) + 1(3) = 11$$
$$\gamma_3 = 2(2+1+1) + 2(1+1) + 1(1) = 13$$

The result indicates that T_1 is the best since it identifies each machine uniquely. T_3 is better than T_2 in that it isolates faults more finely.

The distinguishability criterion defined in equation (3) must be modified if one is interested in isolating faults only to the module level, rather than the faulty component level. For most practical purposes, whenever a fault is identified, it is the module(s), i.e., the smallest replaceable part(s) of a machine such as a plug-in circuit package or an integrated circuit card, that will be replaced. Thus, distinguishability among faults associated with the same module would be of no interest. For example, suppose the six faults just considered are associated with three modules: faults f_1, f_2, f_3 associated with one module p_1, f_4 and f_6 associated with another module p_2, and f_5 with module p_3.

Test T_2 partitions these faults into three equivalence classes:

	$(i+1, j_1)$	$(i+1, j_2)$	$(i+1, j_3)$
T_2	f_1^1, f_2^1	f_4^2	f_3^1, f_5^3, f_6^2

where f_i^j denotes a fault f_i assoticated with module j. The distinguishability between faults f_1 of $(i+1, j_1)$ and f_3 of $(i+1, j_3)$, or f_2 of $(i+1, j_1)$ and f_3 of $(i+1, j_3)$ is no longer relevant. Since, whenever either f_1, or f_2 or f_3 occurs, module p_1 must be replaced. So the emphasis of distinguishability criterion should be such that a test is considered most efficient if it distinguishes the largest number of pairs of faults $\{f_i, f_j\}$'s where f_i and f_j each is associated with a *distinct* module. The measure of test efficiency can be redefined by removing from γ_k the number of all those fault pairs that are distinguished by T_k but are associated with the same module.

Let $n(i, j)_p$ denote the number of faults in (i, j) that are associated with module p. Then,

$$n(i, j) = \sum_p n(i, j)_p$$

where $\sum_p$ is defined to denote summation over, and only over, all modules with which the faults in (i, j) are associated. In the previous example, $n(i+1, j_3)$ can be expressed as the sum of $n(i+1, j_3)_1$, $n(i+1, j_3)_2$ and $n(i+1, j_3)_3$, where $n(i+1, j_3)_1$, $n(i+1, j_3)_2$ and $n(i+1, j_3)_3$ denote, respectively, the number of faults in $(i+1, j_3)$ associated with modules p_1, p_2 and p_3. Thus, of the m partitions generated by test T_k, there are

$$\sum_{s=1}^{m-1} \left\{ \sum_p [n_k(i+1, j_s)_p \cdot \sum_{t=s+1}^{m} n_k(i+1, j_t)_p] \right\}$$

fault pairs which need *not* be distinguished. The distinguishability measure γ_k defined in Equation (3) should therefore be modified as follows:

$$\lambda_k \equiv \gamma_k - \sum_{s=1}^{m-1} \left\{ \sum_p \left[n_k(i+1, j_s)_p \cdot \sum_{t=s+1}^{m} n_k(i+1, j_t)_p \right] \right\}$$

$$= \sum_{s=1}^{m-1} \left\{ \sum_p \left[n_k(i+1, j_s)_p \cdot \sum_{t=s+1}^{m} \left(n_k(i+1, j_t) - n_k(i+1, j_t)_p \right) \right] \right\} \quad (4)$$

The application of this distinguishability criterion to derive an efficient testing procedure may be illustrated by the following example.

Example Suppose in equivalence class (i, j) there are ten faults to be identified. The faulty components are associated with modules p_1, p_2 and p_3 in the following way:

p_1: $f_1^1, f_2^1, f_3^1, f_4^1, f_5^1, f_6^1$
p_2: f_7^2, f_8^2
p_3: f_9^3, f_{10}^3

Six tests T_k (k = 1,2,3,4,5,6) are available, each generating a partition of equivalence classes as indicated in Table I. To derive a testing procedure

TABLE I – Configurations of Partitionings of (i, j)

	$(i+1, j_1)$	$(i+1, j_2)$	$(i+1, j_3)$	γ_k	λ_k
T_1	$f_1^1, f_2^1, f_7^2, f_8^2, f_9^3, f_{10}^3$	f_3^1, f_4^1	f_5^1, f_6^1	28	16
T_2	$f_1^1, f_2^1, f_3^1, f_4^1, f_5^1, f_6^1, f_7^2$	f_8^2, f_9^3	f_{10}^3	23	21
T_3	$f_1^1 f_2^1 f_3^1 f_5^1 f_7^2 f_8^2 f_9^3 f_{10}^3$	f_4^1, f_6^1	-	16	8
T_4	$f_1^1 f_2^1 f_3^1 f_4^1 f_5^1 f_6^1 f_7^2 f_8^2 f_{10}^3$	f_9^3	-	9	8
T_5	$f_2^1 f_3^1 f_4^1 f_5^1 f_6^1 f_7^2 f_8^2 f_9^3 f_{10}^3$	f_1^1	-	9	4
T_6	$f_1^1 f_2^1 f_3^1 f_4^1 f_5^1 f_6^1 f_8^2 f_9^3 f_{10}^3$	f_7^2	-	9	8

that isolates faults to component level, the first step is to compute the figure of merit of each test using either Equation (2) (the information gain criterion) or Equation (3) (the distinguishability criterion at component level). Both criteria indicate test T_1 is the best choice to partition (i, j); T_1 is thus selected as the first test. The faults in (i, j) are then partitioned into three equivalence classes: $\{f_1^1, f_2^1, f_7^2, f_8^2, f_9^3, f_{10}^3\}$, $\{f_3^1, f_4^1\}$ and $\{f_5^1, f_6^1\}$. Similar computations can be made for selecting the next best test(s) to further partition each equivalence class. The resultant testing procedure, as shown in Figure 3, indicates that the average length of test sequence to isolate a fault is 2.7 tests.

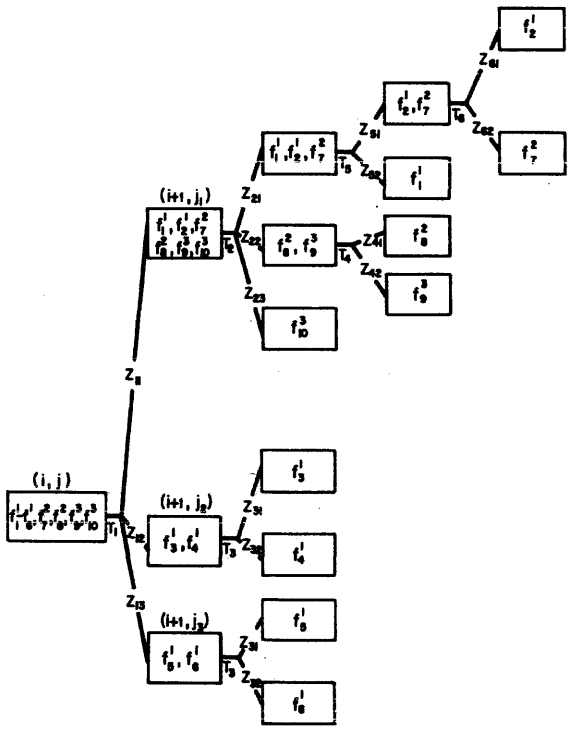


Figure 3 – A testing procedure isolating faults to component level

However, if the distinguishability criterion λ_k is used to derive a testing procedure that isolates faults to replaceable module level, one observes that test T_2 would be the best choice to partition (i, j) (see Table I). In this case test T_1 is less efficient since it distinguishes fewer fault pairs that are associated with different modules. A sample computation using equation (4) to compute λ_k is given:

$$\begin{aligned}\lambda_2 &= n_2(i+1, j_1)_1 \left\{ n_2(i+1, j_2) + n_2(i+1, j_3) \right\} \\ &\quad + n_2(i+1, j_1)_2 \left\{ \left[n_2(i+1, j_2) - n_2(i+1, j_2)_2 \right] \right. \\ &\quad \left. + n_2(i+1, j_3) \right\} \\ &\quad + n_2(i+1, j_2)_2 \left\{ n_2(i+1, j_3) \right\} \\ &= (6) \left[(2) + (1) \right] + (1) \left[(2-1) + (1) \right] + (1)\,(1) \\ &= 21\end{aligned}$$

Test T_2 partitions (i, j) into three equivalence classes: $\{f_1^1, f_2^1, f_3^1, f_4^1, f_5^1, f_6^1, f_7^2\}$, $\{f_8^2, f_9^3\}$ and $\{f_{10}^3\}$. Each equivalence class can be further partitioned by test(s) that yields the maximum figure of merit λ_k, computed by considering only those faults in each equivalence class. As an illustration, the λ_k's of various tests partitioning the set $\{f_1^1, f_2^1, f_3^1, f_4^1, f_5^1, f_6^1, f_7^2\}$ are computed and tabulated in Table II. In this case test T_6 is considered the best choice. The process of test selection is continued until all faults are identified to a single module. The resultant testing procedure is shown in Figure 4. The average length of test sequence to isolate a fault is 1.9 tests, which is much shorter than the 2.7 tests required to isolate faults to component level.

TABLE II—Configurations of Partitioning of Equivalence Class

	$(i+2, s_1)$	$(i+2, s_2)$	$(i+2, s_3)$	λ_k
T_1	f_1^1, f_2^1, f_7^2	f_3^1, f_4^1	f_5^1, f_6^1	4
T_3	$f_1^1, f_2^1, f_3^1, f_5^1, f_7^2$	f_4^1, f_6^1	-	2
T_4	$f_1^1 - f_6^1, f_7^2$	-	-	0
T_5	$f_2^1 - f_6^1, f_7^2$	f_1^1	-	1
T_6	$f_1^1, f_2^1, f_3^1, f_4^1, f_5^1, f_6^1$	f_7^2	-	6

Another point worth mentioning is that the program memory required to store the diagnostic testing procedure is often significantly reduced, when diagnosability is aimed at the module level. As an example consider the two diagnostic testing procedures of Figures 3 and 4. It can be shown that the size of the corresponding program statements required to realize the procedure of Figure 4, where faults are isolated to module level, is approximately *half* the size of the one required to realize the procedure of Figure 3, where faults are isolated to component level. For many real-time systems such as electronic telephone switching systems and air-borne computers, the resulted saving in memory could represent a significant reduction in cost.

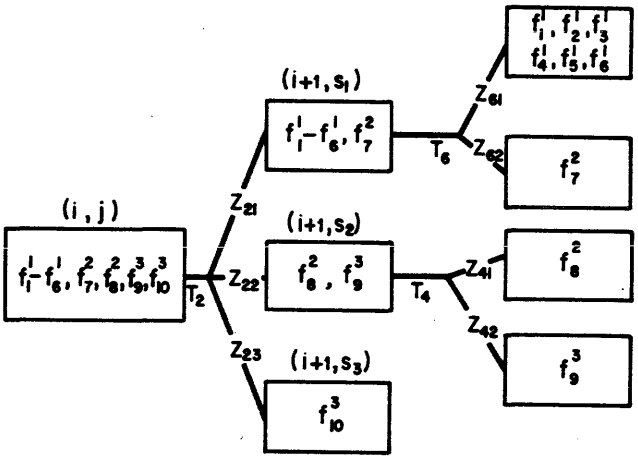


Figure 4—A testing procedure isolating faults to module level

CONCLUSION

A distinguishability criterion for measuring the efficiency of diagnostic tests has been developed. The criterion is compatible with the concept of information gain described in the literature,[3,5,6] if diagnosability is aimed at the component level. If, however, the diagnosability is aimed at the circuit package or module level, it is shown that the sequential testing procedure which is generated based on the distinguishability criterion tends to yield shorter testing sequences and better resolvability.

REFERENCES

1 E G MANNING H Y CHANG
A comparison of methods for simulating faults of digital system
Digest of First Annual IEEE Conference 1967

2 S SESHU D N FREEMAN
The diagnosis of asynchronous sequential switching systems
IRE Transactions on Electronic Computers Vol EC-11 August 1962

3 J D BRULE R A JOHNSON E KLETSKY
Diagnosis of equipment failures
IRE Transactions on Reliability and Quality Control Vol RQC-9 April 1960

4 J F POAGE
The derivation of optimum tests for logic circuits

PhD Thesis Princeton University 1963

5 S SESHU
On an improved diagnosis program
IEEE Transactions on Electronic Computers Vol EC-14 February 1965

6 D MANDELBAUM
A measure of efficiency of diagnostic tests upon sequential logic
IEEE Transactions on Electronic Computers Vol EC-13 October 1964

7 E F MOORE
Gedanken-experiments on sequential machines
Automata Studies Princeton University Press 1956

8 H Y CHANG
An algorithm for selecting an optimum set of diagnostic tests
IEEE Transactions on Electronic Computers Vol EC-14 October 1965

1968 SPRING JOINT COMPUTER CONFERENCE COMMITTEE

General Chairman

A. S. Hoagland
IBM Corporation

Vice Chairman

W. R. Lonergan
RCA - EDP

Bernard McGovern, Assistant
RCA - EDP

Technical Program

T. R. Bashkow, Chairman
Columbia University

Richard Auerbach
Computer Usage Development Corporation

Glenn Bacon
IMB Corporation

Jess Chernak
Bell Telephone Laboratories

Borge Christensen
General Electric Company

Chester Y. Lee
Bell Telephone Laboratories

Morton H. Lewin
RCA Laboratories

Sheldon Weinberg
Realtronics, Inc.

Finance

C. A. Erdahl, Treasurer
Price Waterhouse & Company

M. B. Basson
Price Waterhouse & Company

R. W. Liptak
Price Waterhouse & Company

Secretary

S. M. Matsa
IBM Corporation

Local Arrangements

H. L. Cooke, Chairman
RCA Laboratories

E. E. Andrews
RCA - EDP

Public Relations

J. M. Kinn, Chairman
IEEE

R. T. Miller
IBM Corporation

Special Events

A. A. Currie, Chairman
Bell Telephone Laboratories

R. L. Basford
Bell Telephone Laboratories

Exhibits

W. M. Carlson, Chairman
IBM Corporation

R. F. Welch
UNIVAC

Burt Totaro
UNIVAC

Registration

G. W. Jacob, Chairman
Sperry Gyroscope Co.

Solomon Scherr
Hazeltine Corporation

Charles Griebell
Sperry Rand Corporation

Patrick J. Ferrara
Sperry Rand Corporation

Ladies

Cecilie Smolen, Chairman
First National City Bank

Frances Zederbaum
IBM Corporation

Gretchen Remick
Computer Usage Development Corporation

Ellen Schaefer
IBM Corporation

Printing and Mailing

R. W. Thayer, Chairman
Princeton Printing Company

Advisor

Harlan Anderson
Time, Inc.

AFIPS Society Representatives

IEEE

S. Levine
The Bunker-Ramo Corporation

ACM

J. M. Spring
Computer Methods, Inc.

SCI

R. J. Doelger
Electronic Associates, Inc.

ASIS

Irlene Stephens
CCNY

AMTCL

W. J. Plath
IBM Corporation

REVIEWERS, PANELISTS, AND SESSION CHAIRMEN

REVIEWERS

A. Adler
Wilheim Anacker
James P. Anderson
Gary Bard
F. Bates
Frank Bevacqua
Ruth Block
Shelton Boilen
W. G. Bourichius
Robert C. Calfee
Harry N. Cantrell
Richard Caplan
S. H. Chasen
A. Ben Clymer
Steve Condon
Warren A. Cornell
Richard L. Crandall
James W. Daniel
H. A. Ernst
Monroe Fein
W. A. Fetter
Tudor R. Finch
R. Forbes
R. Stockton Gaines
Alonzo G. Grace, Jr.
Charles Gulotta
M. J. Haims
C. Halstead
P. J. Hanratty
Tom Hastings
R. A. Henle
B. Herzog
D. Hodges
Thomas J. Hogg
Mu-Yue Hsiao
H. Johnson
B. J. Karafin
H. B. Keller
Robert King
Eldo C. Koenig
Z. Kohavi
Mark Koschmann
J. Kurtzberg
Chester Lee
L. Lidofsky
Y. S. Lim
Arthur W. Lo
R. Mandell
Michael Marcotty
T. J. Matcovich
H. E. Meadows
M. J. Merritt
Gordon S. Mitchell
Thurber Moffett
Harrison Morse
Mervin E. Muller
Robert M. McClure
H. S. McDonald
I. D. Nehama
Robert L. Patrick
A. V. Pohm
W. J. Poppelbaum
Paul Reinhard
John R. Rice
V. C. Rideout
Lawrence G. Roberts
Robert F. Rosin
R. Roth
Wendell Sander
F. J. Samsom
C. L. Semmelman
S. Shapiro
R. L. Shuey
Warner V. Slack
Otto J. M. Smith
Robert Spinrad
A. Soudak
Thomas B. Steel, Jr.
William Sutherland
A. J. Sutton
R. N. Thompson
William Timlake
C. J. Tunis
H. VanBrink
E. Van Horn
J. V. Wait
C. J. Walter
R. H. Wanders
Robert Ward
Roger C. Wood
J. J. Yostpille
Edward Yourdon
M. S. Zucker

PANELISTS

J. A. Archibald, Jr.
Bruce W. Arden
Julius Aronofsky
J. D. Babcock
C. W. Churchman
M. E. Connelly
Philip A. Cramer
David C. Evans
M. M. Flood
R. M. Franklin
Ralph Gerard
Harry A. Gray
H. R. J. Grosch
Robert V. Head
William B. Helgeson
Linder C. Hobbs
Richard C. Jones
H. A. Kinslow
J. F. Lubin
Tom Marques
Andrew R. Molnar
Henry S. McDonald
Henry McDonald
J. D. McGonagle
T. William Olle
Ascher Opler
Budd J. Pine
R. C. Raymond
Walter Rosenblith
Thomas Rowen
Daniel W. Scott
J. E. Sherman
Warren G. Simmons
Robert Spinrad
Ithiel de Sola Pool
William R. Sutherland
C. J. Tunis
Robert Ward
Jerome B. Wiener
M. A. Woodbury
Kendall R. Wright

SESSION CHAIRMEN

C Bachman
W. Beam
C. R. Deininger
P. Dorn
G. Feeney
R. Forest
J. Githens
L. Hittel
E. L. Jacks
W. Keister
G. A. Korn
C. Lecht
T. McFee
B. Neff
A. Opler
V. C. Rideout
H. Sassenfeld
C. Simone
H. Teager
J. Wiener
R. O. Winder

AMERICAN FEDERATION OF INFORMATION PROCESSING SOCIETIES (AFIPS)

OFFICERS and BOARD of DIRECTORS of AFIPS

President
Dr. BRUCE GILCHRIST
IBM Corporation
Data Processing Division
112 East Post Road
White Plains, New York 10601

Vice President
Mr. PAUL ARMER *
The RAND Corporation
1700 Main Street
Santa Monica, California 90406

Secretary
Mr. MAUGHAN S. MASON
IBM Hybird Systems Center
2670 Hanover Street
Palo Alto, California 94304

Treasurer
Mr. WILLIAM D. ROWE *
The Mitre Corporation
5600 Columbia Pike
Bailey's Crossroads,
Virginia 22041

ACM Directors

Dr. ANTHONY G.OETTINGER
Computer Laboratory
Harvard University
Cambridge, Massachusetts 02138

Mr. J. D. MADDEN
ACM Headquarters
211 East 43rd Street
New York, New York 10017

Dr. ROBERT W. RECTOR *
Informatics, Inc.
5430 Van Nuys Boulevard
Sherman Oaks, California 91401

Dr. WALTER HOFFMAN
Computing Center
Wayne State University
Detroit, Michigan 48202

IEEE Directors

Mr. SAMUEL LEVINE
Bunker-Ramo Corporation
445 Fairfield Avenue
Stamford, Connecticut 06902

Mr. L. C. HOBBS
Hobbs Associates, Inc.
P. O. Box 686
Corona Del Mar, California 92625

Mr. KEITH W. UNCAPHER
The RAND Corporation
1700 Main Street
Santa Monica, California 90406

Dr. R. I. TANAKA *
California Computer Products, Inc.
305 N. Muller Street
Anaheim, California 92803

Simulation Councils Director

Mr. JOHN E. SHERMAN *
Lockheed Missiles & Space Corp.
D59-10: B-151
P. O. Box 504
Sunnyvale, California 94088

American Society for Information Sciences Director

Mr. HAROLD BORKO
Systems Development Corp.
2500 Colorado Avenue
Santa Monica, California 90406

* Executive Committee

Association for Machine Translation and Computational Linguistics-Observer

Dr. DONALD WALKER
The Mitre Corporation
Bedford, Massachusetts 01730

Special Libraries Association-Observer

Mr. BURTON E. LAMKIN
Library & Information Retrieval Staff
Federal Aviation Agency
800 Independence Avenue S.E.
Washington, D.C. 20003

Society for Information Display-Observer
Mr. WILLIAM BETHKE
1806 N. James Street
Rome, New York 13440

AFIPS Committee Chairmen

Abstracting

Dr. DAVID G. HAYES
The RAND Corporation
1700 Main Street
Santa Monica, California 90406

Admissions

Mr. WALTER L. ANDERSON
General Kinetics, Inc.
11425 Isaac Newton Square South
Reston, Virginia 22070

Awards

Dr. ARNOLD A. COHEN
UNIVAC
2276 Highcrest Drive
Roseville, Minnesota 55113

Conference

Dr. MORTON M. ASTRAHAN
IBM Corporation - ASDD
P. O. Box 66
Los Gatos, California 95030

Constitution & By-Laws

Mr. MAUGHAN S. MASON
IBM Hybird Systems Center
2670 Hanover Street
Palo Alto, California 94304

Education

Dr. MELVIN A. SHADER
IBM Corporation - SDD
1000 Weschester Avenue
White Plains, New York 10604

Government Advisory

Dr. HARRY HUSKEY
University of California
Division of Natural Sciences
Santa Cruz, California 95060

Harry Goode Memorial Award

Mr. ASCHER OPLER
T. J. Watson Research Center
P. O. Box 216
Yorktown Heights, New York 10598

IFIP Congress 68

Dr. DONALD L. THOMSEN, JR.
IBM Corporation
Old Orchard Road
Armonk, New York 10504

International Relations

Dr. EDWIN L. HARDER
Westinghouse Electric Corp.
1204 Milton Avenue
Pittsburgh, Pennsylvania 15218

Planning

Dr. JACK MOSHMAN
Leasco Systems & Research Corp.
4833 Rugby Avenue
Bethesda, Maryland 20014

Public Relations

Mr. ISAAC SELIGSOHN
IBM Corporation
Old Orchard Road
Armonk, New York 10504

Finance

Mr. WALTER M. CARLSON
IBM Corporation
Old Orchard Road
Armonk, New York 10504

Publications

Mr. STANLEY ROGERS
P. O. Box R
Del Mar, California 92014

Social Implications of Information Processing Technology

Mr. STANLEY ROTHMAN
TRW Systems
1 Space Park
Redondo Beach, California 90278

Information Dissemination

Mr. GERHARD L. HOLLANDER
Hollander Associates
P. O. Box 2276
Fullerton, California 92633

Technical Program

Mr. JACK ROSEMAN
Helidyne Corporation
1401 Wilson Boulevard
Arlington, Virginia 22209

Consultant

Mr. HARLAN E. ANDERSON
Time, Inc.
Time & Life Building
New York, New York 10020

Newsletter

Mr. DONALD B. HOUGHTON, 15-W
Westinghouse Electric Corporation
3 Gateway Center, Box 2278
Pittsburgh, Pennsylvania 15230

U. S. Committee for IFIP ADP Group

Mr. ROBERT C. CHEEK
Westinghouse Electric Corporation
3 Gateway Center
Pittsburgh, Pennsylvania 15230

JCC General Chairmen

1968 FJCC

Dr. WILLIAM H. DAVIDOW
Dymec Divisions
Hewlett Packard Company
395 Page Mill Road
Palo Alto, California 94306

1968 SJCC

Dr. A. S. HOAGLAND
IBM Research Center
P. O. Box 218
Yorktown Heights, New York 10598

1969 SJCC

Dr. HARRISON FULLER
Sanders Associates, Inc.
95 Canal Street
Nashua, New Hampshire 03060

AFIPS Executive Secretary

Mr. H. G. ASMUS
AFIPS Headquarters
9th Floor
345 East 47th Street
New York, New York 10017

1968 SJCC LIST OF EXHIBITORS

Academic Press, Inc.
Adage, Inc.
Addison-Wesley Publishing Company, Inc.
Addressograph Multigraph Corporation
Advanced Computer Techniques Corp.
Amp, Inc.
Ampex Corporation
American Telephone & Telegraph
Anderson Jacobson Inc.
Applied Data Research, Inc.
Applied Dynamics, Inc.
Applied Logic Corp.
Association for Computing Machinery
Audio Devices, Inc.
Auerbach Corporation
Auto-trol Corporation

Baldwin Kongsberg Company
Bolt Beranek and Newman, Inc.
Brogan Associates, Inc.
Burroughs Corporation
Business Supplies Corp. ofAmerica

Caelus Memories Inc.
California Computer Products, Inc.
Certex Inc.
Collins Radio Company
Comcor Astrodata, Inc.
Communitype Corp.
Computer Applications, Inc.
Computer Communications, Inc.
Computer Design Publishing Corporation
Computer Industries
Computer Methods Corp.
Computer Sciences Corporation
Computer Test Corporation
Computer Sharing Inc./Mauchly Group
Computers and Automation
Computerworld
Com-Share
Concord Control, Inc.
Control Data Corporation
Cybetronics, Inc.

Data Disc Inc.
Datamark Inc.
Datamation
Data Processing Magazine
Data Products Corporation
Datascan
Data Systems News

Datel Corp.
Di/An Controls, Inc.
Digi-Data Corporation
Digital Development Corporation

Digital Devices, Inc.
Digital Equipment Corporation
Digitronics Corporation
Dura Business Machines
Dynamic Systems Electronics

Eastman Kodak Company
Educational Computer Products
Elbit Computer, Ltd.
Electro-Mechanical Research Inc.
Electronic Associates, Inc.
Electronic Design
Electronic Memories

Fabri-Tek
Ferroxcube Corporation

General Computers, Inc.
General Design
General Dynamics, Electronics Div.
General Electric Company
General Instrument Corp.
General Kinetics Inc.
Geo Space Corporation

Hewlett-Packard Company
Honeywell, Computer Control Div.
Houston Instrument Div, Bausch & Lomb

IBM Corporation
IBM Industrial Products
Information Control Corporation
Information Displays, Inc.
Infotechnics, Inc.
Institute for Electrical and Electronics Engineers
Interdata

Kennedy Company
Keuffel & Esser Company
Kleinschmidt, Div. SCM Corporation

Laboratory for Electronics, Inc. Electronics Div.
Lancer Electronics Corporation
Litton/Datalog Division
Lockheed Electronics Company

Magne-Head, A Div. of General Instrument Corp.
Mandata Systems Inc.
Matrix Corp.
Memory Technology Inc.
Micro Switch, A Div. of Honeywell
Midwestern Instruments/Telex
Milgo Electronic Corporation
3M Company - Magnetic Products Div.
- Mincom Div.
- Dup. Products Div.
Modern Data Systems
Monitor Systems, Inc.
Motorola Instrumentation & Control Inc.
McGraw-Hill Book Company

The National Cash Register Company
The National Cash Register Company/
Industrial Products
National Computer Systems
Nissei Sangyo, Ltd.
North Atlantic Industries, Inc.

Omnitec Corp., A Subsidiary of Nytronics, Inc.

Peripheral Systems, Div. of Memorex
Prentice-Hall, Inc.
Presto Seal Mfg. Corp.
Pro-Data, Computer Services Inc.

RCA EDP Div.
RCA Electronic Components & Devices
Raytheon Computer
Redcor Corporation
Rixon Electronics, Inc.

Sanders Associates, Inc.
Sangamo Information Systems
Scientific Control Corp.
Scientific Data Systems
Shepard Laboratories
Software Resources Corporation
Soroban Engineering, Inc.
Spatial Data Systems, Inc.
Sylvania Lighting Products
Systems Engineering Laboratories, Inc.

Tally Corporation
Tektronix, Inc.
Teletype Corporation
Transistor Electronics Corporation
Tymshare, Inc.

United Telecontrol Electronics, Inc.
UNIVAC Division, Sperry Rand Corp.
URS Corporation
U. S. Magnetic Tape

Varian Data Machines
Vermont Research Corporation

Wang Laboratories, Inc.
Wanlass Electric Co.
John Wiley & Sons, Inc.

Xerox Corporation

AUTHOR INDEX