TR 62-1
JANUARY 1962

Burroughs

Burroughs D 850

Modular Data Processing System

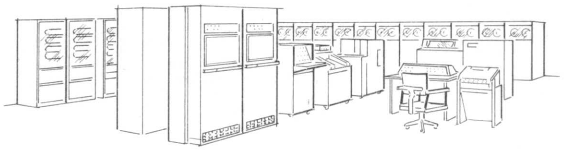
Burroughs Corporation

TR 62-1
JANUARY 1962

Burroughs D850

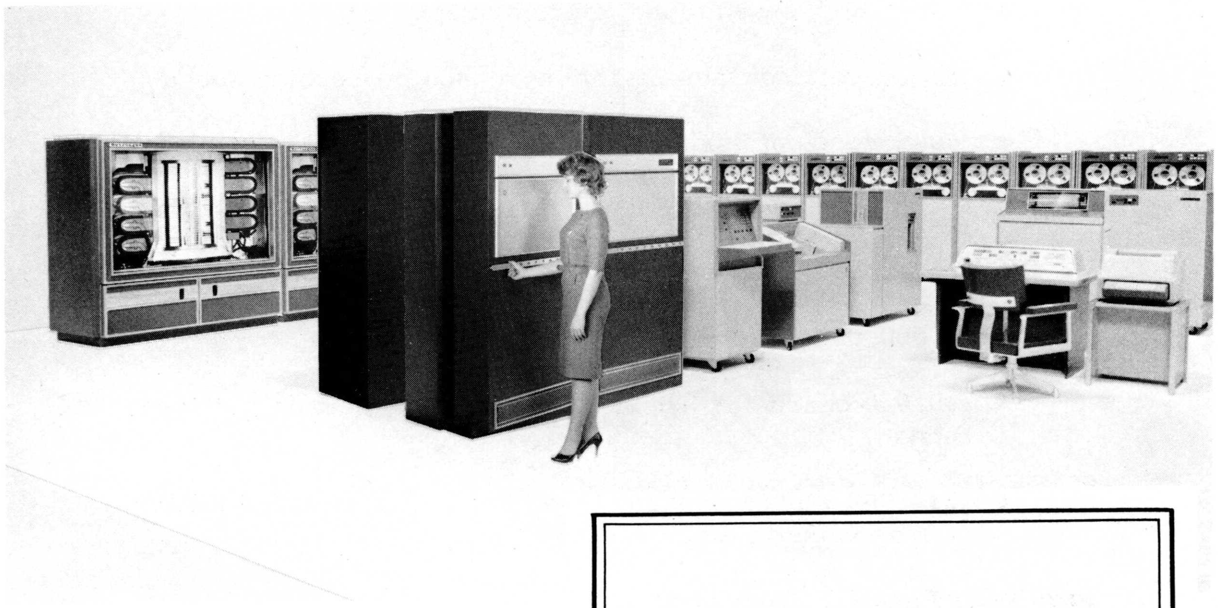
Modular Data Processing System

Burroughs Corporation



BURROUGHS

D850 MODULAR DATA PROCESSING SYSTEM



- UNPRECEDENTED SPEED
- TRUE PARALLEL PROCESSING
- EXPANSIBLE MODULAR DESIGN
- FORMAL LANGUAGE ORIENTED

SYSTEM CHARACTERISTICS

Unprecedented Speed

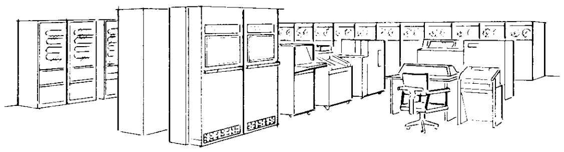
- 30 megacycle clock frequency
- 0. 12 μsec add time — 2. 08 μsec multiply time
- Ultra-high speed tunnel diode storage registers
- Effective "look-ahead" for increased processing speed
- 4096 words of 0. 4 μsec thin film store directly coupled with each processor module
- 0. 5 μsec cycle time achieved in 4-phase main memory modules
- 4, 000, 000 word per second I/O transfer rate
- 66, 660 character per second tape transfer rate per tape unit
- 212,000 word per second transfer rate from 14. 4 million word magnetic disc files

True Parallel Processing

- Simultaneous arithmetic and address processing
- Each main memory module addressable by all processors and I/O control modules
- Buffered simultaneous memory-to-memory block transfer
- Buffered I/O processing by satellite computers
- Simultaneous I/O operation of all I/O control modules

Expansible Modular Design

- 1 to 4 processor modules
- 1 to 15 main modular memories — 16, 384 50-bit words each
- 1 to 16 I/O control modules
- 1 to 64 I/O devices and satellite computing systems

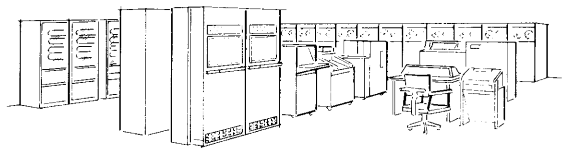


Formal Language Oriented

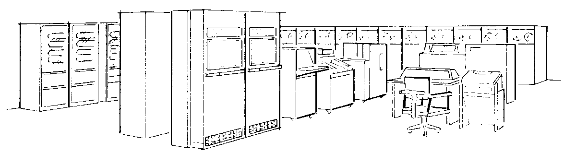
- Complete software package including:
  - ALGOL 60 Compiler
  - Automatic Operating and Scheduling Program
  - Automatic Diagnostic Routines
- Polish string machine language processing
- Compatible integer, floating point, and logical processing
- n-Level indirect addressing and indexing
- 15 high-speed index registers
- Direct use of any memory location for indexing
- Push-down stack for intermediate results
- Optional wired-in high-speed subroutines for intrinsic function evaluation

Plus

- Compatible satellite systems:
  - B5000 Information Processing System
  - D825 Modular Data Processor
  - B200 Family of Data Processing Systems
- Peripheral Equipment:
  - 800 word per minute card readers
  - 300 word per minute card punches
  - 700 line per minute line printers
  - Supervisory printer
  - Operators inquiry station



Introduction



Introduction

The D850 is the newest member of the Burroughs D800 series of modular data processing systems. It is a data processing system of scope, speed, computing capability, and sophistication which is not even approached by any other computing equipment yet built or in development. It is an order of magnitude more powerful than its predecessor, the Burroughs D825 Modular Data Processing System.

CHARACTERISTICS

A. General

1. The D850 central processor has an extensive order code embodying all of the operations requested. The computer data word is 48 bits plus parity, and the floating-point mantissa is 39 bits.

2. The high-speed memory furnished for a typical D850 system is made up of 4096 words of close-coupled fast-store plus 65, 536 words of main modular memory (4 modules) for a total of 69, 632 words of directly addressable random-access memory.

   A fully expanded D850 system will have 16, 384 words of close-coupled fast-store plus 245, 760 words of main modular memory (15 modules) for a total of 262, 144 words.

3. The modular bulk memory requirement is satisfied in the D850 by magnetic disc files of 12 or 24 discs each. The 12-disc model holds 7. 2 million full computer words and the larger unit holds 14. 4 million words. If one of each size is

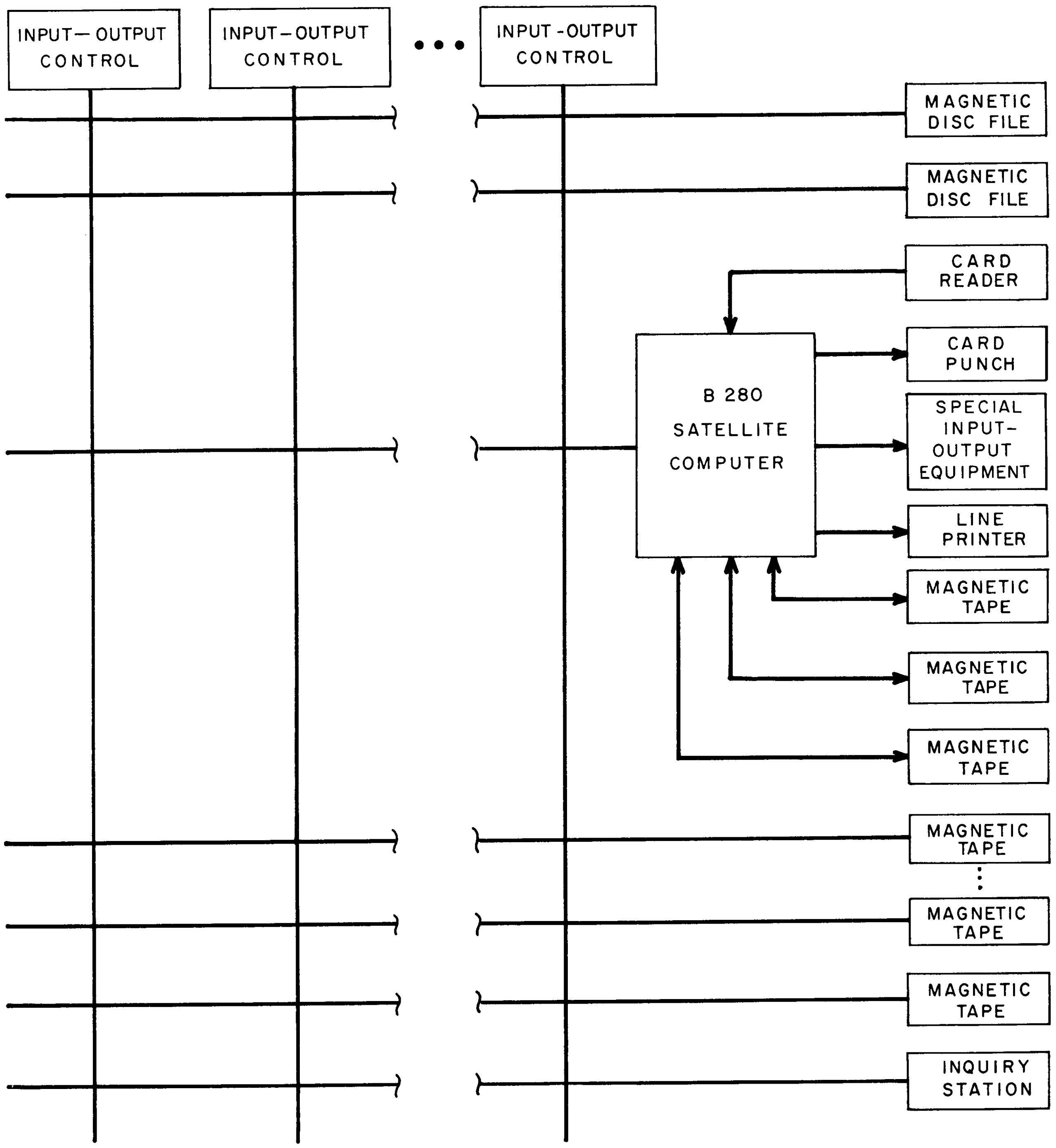
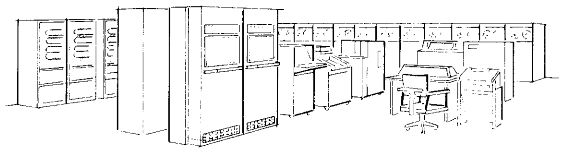


Figure 1. D850 Input-Output Complement Showing Relations to I/O Exchange and B280 Satellite Computer



utilized (with the smaller one being expansible at the site to the full capacity of the larger unit), 21.6 million words can be furnished. The two units can be utilized simultaneously, thereby satisfying a simultaneous in-out capability. The data rate of each disc file unit varies from 4.7 μsec per word to 12.12 μsec per word.

4. The D850 system offers the most comprehensive input-output system of any data processing system in existence. All input-output is completely buffered from processors. All high-speed I/O devices as well as the inquiry station are in direct communication with the modular memory. Each of the four proposed I/O control modules can control any of these devices and can channel the data transfer to any location in memory. As can be seen in Figure 1 the D850 I/O exchange is the same type of cross-bar matrix that is used for D850 intermodule communication. The figure also shows the manner in which the other devices are connected to the peripheral processor. The B280 peripheral processor is completely compatible in word length with the D850. Among its functions are formating, editing, and code translations (should the user have need for them). The B280 can be used both for off-line processing e.g. card-to-tape and for on-line input to the D850.

5. Input-output devices

   Referring to Figure 1, it can be seen that the D850 system can accept a variety of standard I/O devices fed both directly into the system or via the B280 satellite computer.

6. Motor generator sets which are used to provide 400 cps power for the central system modules are included in the D850 system. Each cabinet and device has its own internal air circulation system. No refrigeration equipment is required in a normally air-conditioned office space.

B. Speed

The hydro problem and the Monte Carlo problem have been analyzed in detail (see Section V and Appendices A and B). The D850 can perform one million cycles of the Hydro problem in less than 24 minutes and follow one million particles of the Monte Carlo problem in 18 minutes. These times are well substantiated by detailed machine language coding and timing as well as comparison with the same problems run on the Burroughs 220 computer.

C. Reliability

An analysis of D850 reliability which indicates an estimated availability of 0. 956 for a typical configuration appears in Section III. The D850 performs a parity check on every data transfer in the system, including input-output transfers and intermodule communications. All memory operations are parity checked.

D. Programming Aids

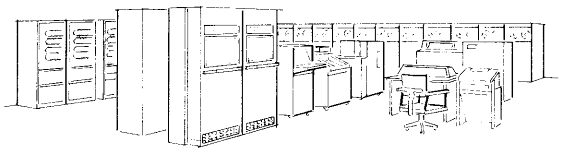
A complete software package can be delivered as an integral part of the D850 system. This package includes an ALGOL compiler; an automatic operating and scheduling program (AOSP), which is a master control program stored in totally shared memory; and a complete set of service and disgnostic routines.

The compiler will be based on an ALGOL-like high-level formal programming language possessing all the features of ALGOL 60 and certain extensions (particularly in the area of non-numeric processing) which will extend the scope and power of ALGOL. Fortran statements can readily be translated into ALGOL 60 and an automatic translation routine will also be included as part of the compiler. The AOSP is an executive routine which provides for proper sequencing of programs, equipment checks, and the man-machine interface. In the case of equipment malfunction, the AOSP will call up those diagnostic routines necessary to pinpoint the fault. Also provided will be service routines for debugging machine language programs, memory dumps, assembly of special purpose programs, etc.

The logical design of the D850 permits Polish parenthesis - free notation. This provides for the efficient compilation of ALGOL statements. It will also provide for the efficient compilation of algebraic languages other than ALGOL when they are written. For this reason, it is not likely the D850 will be outmoded by new software developments unless a language structure is introduced that is not now foreseen.

Delivery of programming data will be accomplished on or prior to the specified dates.

Further information on the D850 programming aids appears in Section V and Attachment 1.



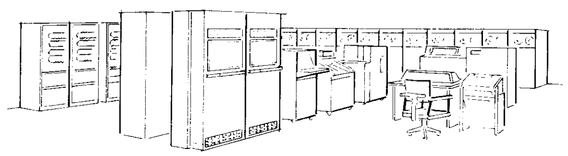
E. Training

Customer operating personnel can be given a thorough course in programming, operation, and the theory of the D850 system. The course can accommodate approximately 20 personnel and will last six weeks.

F. Environment

The environmental specifications of every module and unit of the D850 are tabulated in Section III. The tolerances specified are much wider than in normal commercial data processing equipment. These extra margins, especially in temperature, will usually mean extra reliability under normal office conditions.

I D 850 System Concept



I D850 System Concept

The Burroughs D850 Modular Data Processing System is a new addition to the modular processor family, joining the B5000 and the D825 as the only truly modular computer systems. The D850 possesses the same capability for growth as its forerunners without any need for reprogramming, the D850 can be expanded to obtain an increase in computing speed, in memory capacity, and in input-output data rate. The D850 retains all of the system reliability of the other Burroughs modular processors; no single component failure, no malfunctioning module can totally disable a completely modular system.

The D850 expands the range of the modular processor and pioneers new levels of data processing speed and capability. A D850 system using only a single processor module provides greater computing speed than any other computing system either built or proposed, yet the D850 is designed to make efficient use of four processor modules in a single system.

SYSTEM ORGANIZATION

The Burroughs D850 Modular Data Processing System is configured for its system complement of processor modules, main modular memories, input-output control modules, and high-speed peripheral devices. These units are interconnected as a matrix, graphically shown by Figure 1-1, with one to four processor modules and one to sixteen input-output control modules communicating with totally shared main modular memory. From one to fifteen modules of 16,384 words each are possible in the main modular memory supplemented by a block of 4096 words of magnetic thin film close-coupled fast store in each processor module making a total of 262,144 words of addressable memory in a maximum D850 system.

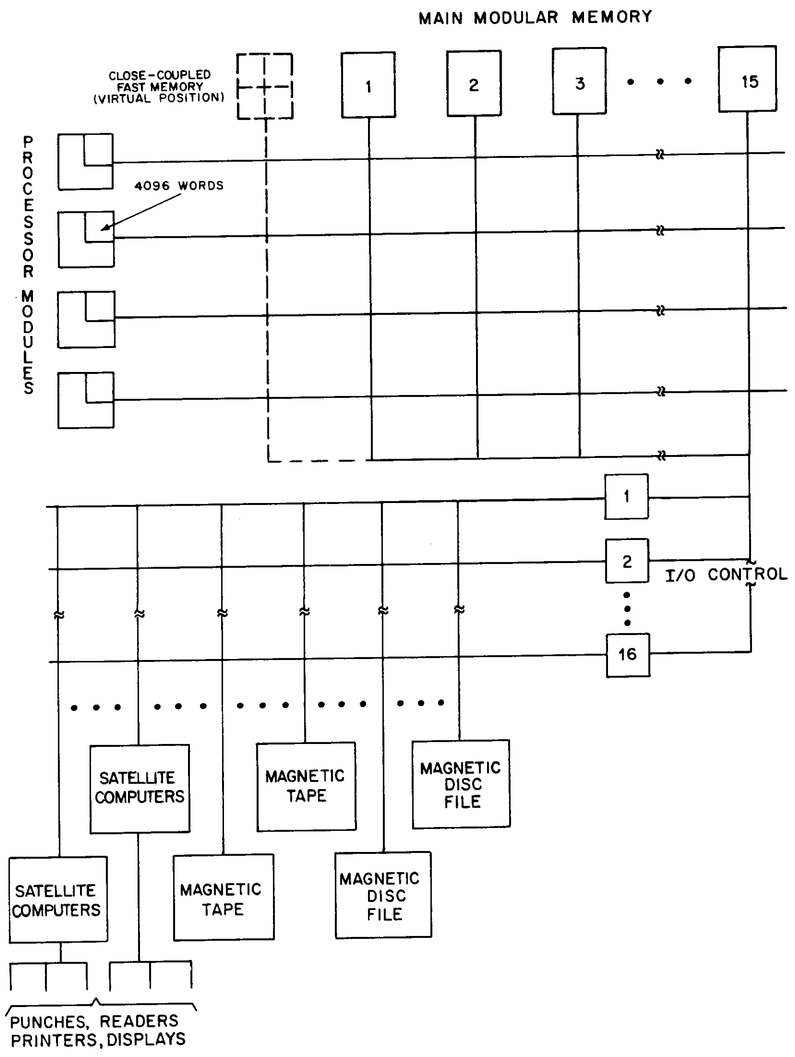
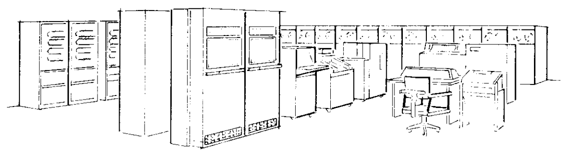


Figure 1-1. D850 System Block Diagram

Due to the extreme speed of the D850 system, direct access to it has been reserved for those devices whose data transfer rates approach its own. Magnetic tape units, magnetic disc files, magnetic drum files, and satellite computers communicate directly with the D850 system via the input-output control modules. Slower devices normally are interconnected by means of the satellite or peripheral computers. These computers are also used to preprocess data, performing code conversions, editing, and formating as well as handling off-line peripheral operations. The satellite computers which are compatible with the D850 include the B5000, the D825, and the B200 series systems. In addition to the high data-rate devices listed, the inquiry station despite its low data rate, communicates directly with the D850 to provide operator access.



PARALLEL OPERATIONS

In order to achieve extremely high computation rates, the D850 Modular Data Processing System carries on parallel processing of information wherever possible. In the processor module, addresses are computed and data fetched simultaneously with arithmetic processing, avoiding the usual sequential nature of these operations. Parallel access by each processor module and input-output control module to the main modular memory is a second class of parallel operation.

Parallel buffered input-output operations are fundamental to all Burroughs modular processors, with as many simultaneous operations possible as there are I/O control modules in the system. The D850 adds a new type of completely buffered data transfer mode, the memory-to-memory block data transfer.

OPERATIONAL STRUCTURE

The D850 is organized to perform smooth and efficient data manipulation. Many of the functions implemented as subroutines in other computers are accomplished directly by operators of the D850 powerful and flexible Polish string program. Still, subroutines remain vital data processing tools and no more efficient or convenient means of subroutine linkage exists than that provided by the pushdown operand stack of the D850 processor. The stack is also very useful as a store for intermediate results of computation.

The data formats of the D850 offer compatible integer and floating-point words. Mixed fixed and floating arithmetic operations are possible with the result automatically being in floating point.

All program and data blocks in the D850 are fully floatable, yet 18-bit addressing permits all 262,144 words to be directly addressed.

AUTOMATIC PROGRAMMING

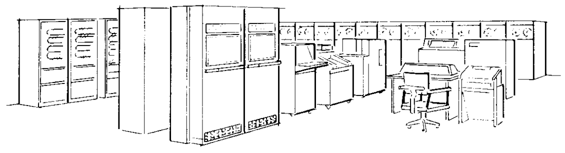
The Burroughs D850 programming package contains facilities for overall control of the system and for efficient problem language programming. Thus, Burroughs provides a complete working system including both the computer and facilities for its programming, program scheduling, maintenance, and general operation. The D850 system philosophy recognizes many factors as necessary for effective data processing. Machine speed is only one of these factors. Others are:

- An easily used programming capability
- Efficient program scheduling
- Avoidance of the unwitting use of malfunctioning equipment
- A routine facility for adding to the program library
- Ability to run any program despite the lack of availability of certain equipment modules
- The ability to make immediate use of additional modules when the system is enlarged.

In the D850, these features are provided as part of the integrated operating system known as the automatic operating and scheduling program (AOSP). The potency of this system is supported by the equipment features which have been designed into the system through the Burroughs philosophy of considering hardware and software as a whole. This philosophy has directed the design of the D850, starting with system inception and continuing through delivery to the ultimate user. In less sophisticated data processing systems, wide use has been made of programming systems, maintenance systems, debugging aids, and confidence checking and diagnostic systems, but the burden of integrating the use of these systems has fallen on a program scheduling department in the user organization.

To guarantee the most efficient use of the system and permit automatic scheduling of all programs and dynamic memory allocation, all the service packages are unified and made part of the AOSP. The inclusion of the programming and maintenance systems within the master program framework results in integrated scheduling, service, and control system which is essential for utmost system effectiveness.

Since high programmer productivity is essential to a completely efficient data processing operation, an ALGOL 60 compiler is included as part of the master operating system. This affords unification of all service systems within one framework.



The wide acceptance ALGOL 60 has received as a standard reference language commends it as the basis of a programming language. The compiler program supplied with the D850 is actually an extended implementation of ALGOL 60. Besides the basic ALGOL statements, the extended language includes:

- Special terms signaling the compiler to take advantage of special machine-programming features
- Symbolic machine language commands
- Instructions indicating input/output formating and execution
- Terms stating precedence requirements and eligibility for parallel processing

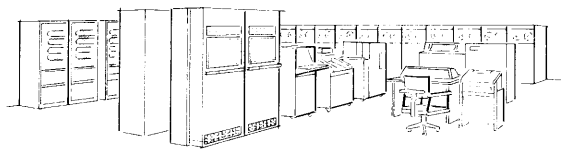
Having been developed as part of the overall operating system, the compiler generates object programs which are amenable to management by the scheduling function of the AOSP.

Programs generated by the D850 compiler reflect the block structure characteristic of ALGOL 60. The resulting systematic structure provides a standard program organization which facilitates program documentation and revision.

Program debugging is accomplished largely by reference to error messages printed out by the compiler and by running a suitable test problem. The automatic programming package includes routines for loading and dumping specifiable sections of memory.

The load and dump routines operating in a library maintenance program form the basis of an expandable program library. The library includes a set of confidence test programs whereby equipment modules can be exercised and evaluated for proper operation. Any or all of these test programs can be set up to be automatically scheduled by the AOSP for running at regular intervals. Equipment modules showing evidence of malfunctioning are passed over by the AOSP when selection is being made of equipment units to be used in running programs. The module is again eligible for use after that unit has been serviced and has passed a subsequent confidence test.

II D850 Module Descriptions



II D850 Module Descriptions

The D850 Modular Data Processing System with a unique system organization provides a data processing capability of heretofore unknown flexibility, efficiency, and speed. This capability has been achieved through the recognition that "brute force" use of high-speed circuitry is not in itself enough; data processing system design must strongly reflect the nature of the problems for which it is intended.

The D850 processor module is the result of a design effort that matches proven high-speed circuitry and memory techniques with the unequalled capability of the Burroughs Corporation in the implementation of efficient computer/programming language interface.

The D850 system, then, is organized in such a way that problems presented in an ALGOL-like formal programming language are executed with utmost efficiency. In such a language, each problem is resolved into a series of blocks. Within each block, a limited amount of data is needed; and most of this data is local to that block. Several of these blocks together form a larger block which also has local data; and so on.

The D850 system organization follows these lines. As may be seen from the D850 data flow chart of Figure 2-1, the large peripheral bulk store may contain new problems or sections of problems which have overflowed from the main modular memory. The main modular memory, a high-speed store, contains, generally, the entire problem to be solved. As blocks of program are required, they are moved to the close-coupled fast store, an extremely high-speed memory which contains program and data currently being processed.

Associated with the fast store is the processor proper. The processor reads through the program string and divides it into two groups--the group which directly manipulates data, and the group which prepares data for manipulation. Because of this division of actual processing and data sequencing control, it is possible to carry on in parallel the

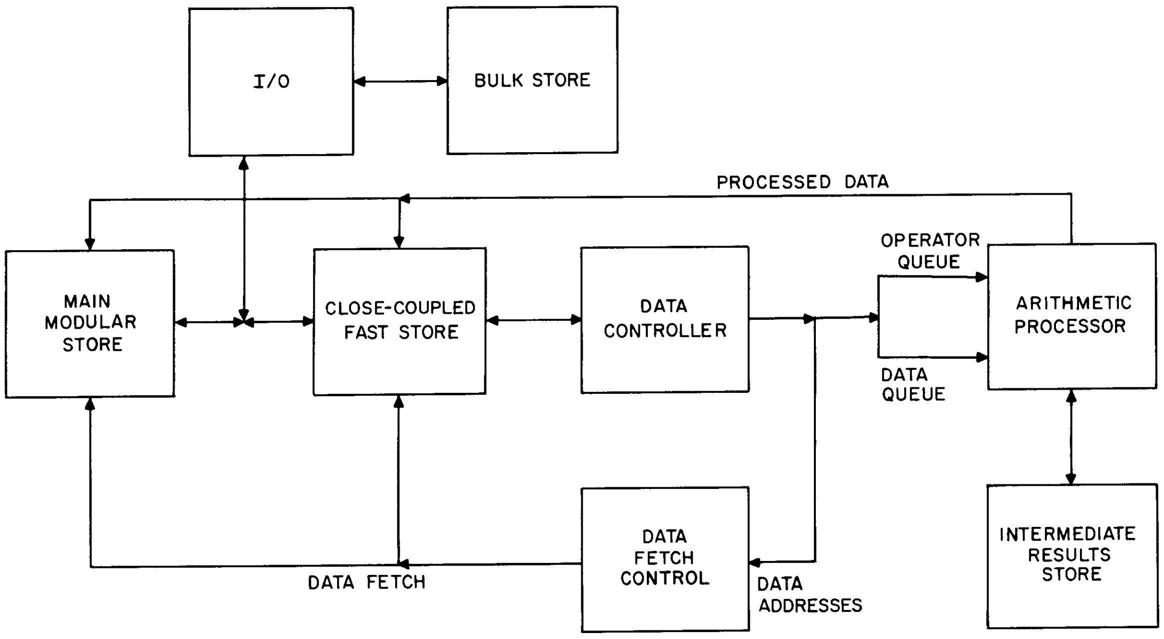
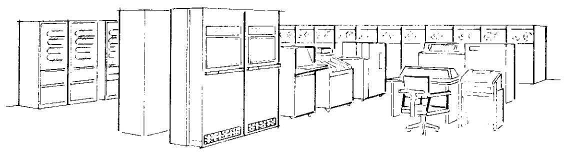


Figure 2-1. D850 Data Flow Chart

processing and housekeeping functions which in conventionally organized computers are sequential operations.

The address computer locates and fetches data from meory and places it in a waiting line or queue; simultaneously, the operators which are to manipulate the data are formed into a second queue. Thus, the operator and corresponding data are made available to the processing unit as it needs them without the necessity of waiting for the data fetch cycle. The processor proper also contains a temporary store or stack for retaining intermediate results, yielding an additional gain in speed. The stack is also used for certain other ready-storage functions, notably, subroutine working storage; and as such the property of unlimited depth is important. Thus, a property of the stack is that it may automatically overflow into the close-coupled fast store. This property is universal to the entire hierarchy of stores; data in any memory may be overflowed into the next slower member of the hierarchy, thus allowing the efficient handling of very large problems.

The true modularity of the system increases the operational capability. Up to four processor modules may be incorporated into a system; hence several problems may be run simultaneously, or several sections of a single problem may be processed concurrently. Moreover,



in such a modular system, failure of a single module does not mean system failure. Units which are still operational, under the control of the operating system program, will automatically assume the load of the failed unit; thus effectively minimizing system down-time.

PROCESSOR

The D850 processor module consists of these four functional sub-modules operating asynchronously with respect to each other:

- Instruction preprocessor
- Address computer
- Arithmetic unit
- 4096 word thin film memory

Additional features include:

- Syllable string programs
- 15 index registers per computer module
- Pluggable function logic for wired in algorithm and routines
- 1 to 4 processors per D850 system
- Tunnel diode clock rate access storage

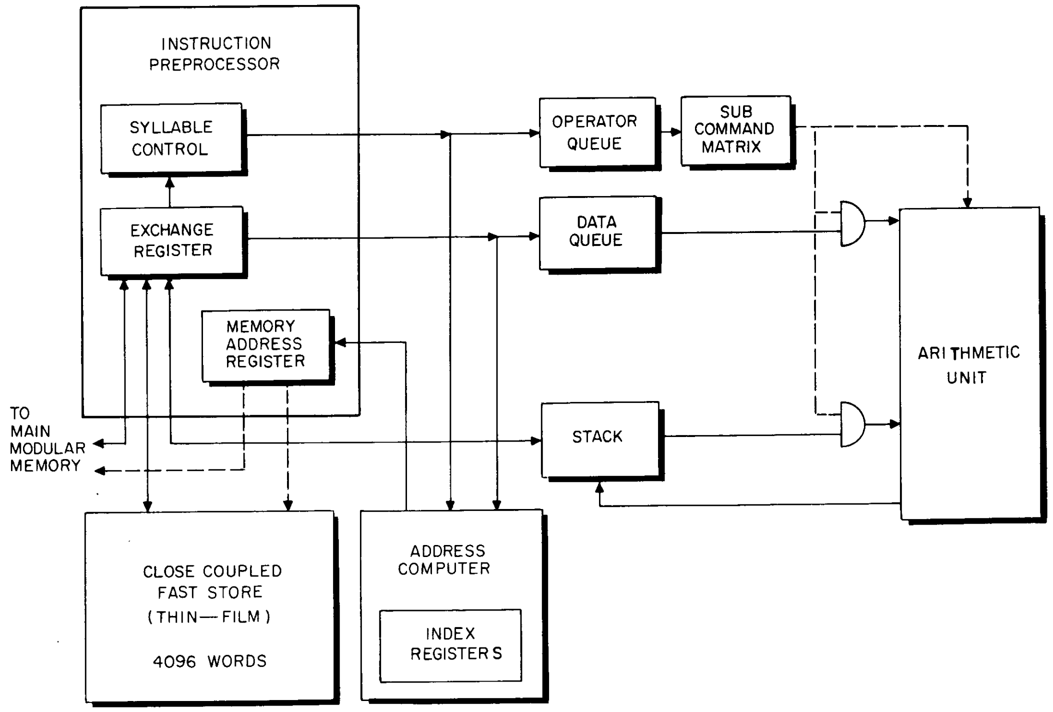


Figure 2-2. D850 Processor Module

Figure 2-2 is a block diagram of the D850 processor module. The instruction preprocessor and address computer perform the data control and data fetch control functions of the data flow chart.

Instruction Preprocessor

The instruction preprocessor sorts incoming program syllables and routes them to the appropriate destination: either the address computer, or the operator queue. An additional function of the instruction preprocessor is to control, via the exchange register, information flow between the memories (close-coupled fast store and main modular memory) and the arithmetic unit registers (operand queue and stack).

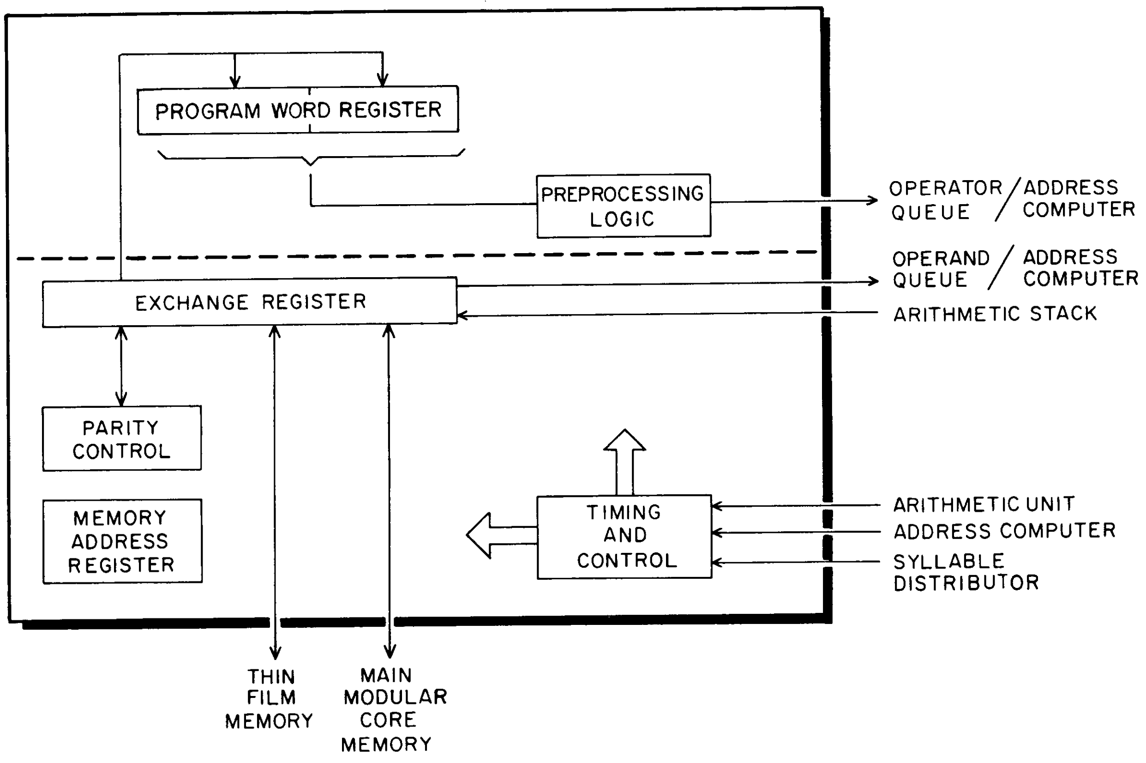
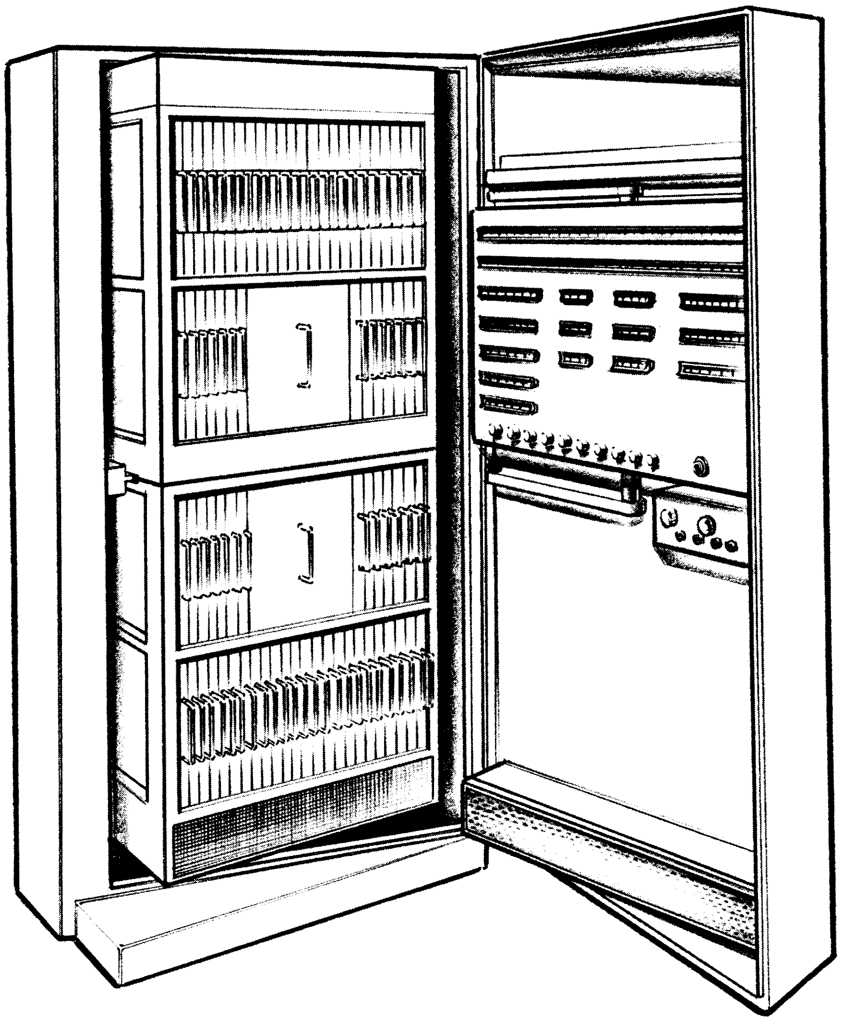


Figure 2-3. Instruction Preprocessor

The structure of the instruction preprocessor is shown in Figure 2-3. The program word register, a 96-bit register, holds two program words at a time for the preprocessing logic which examines the two-bit tag on each program syllable and routes the syllable body appropriately. The program address counter (PC) contains the address of the next instruction syllable to be examined. As one of the two words is exhausted the register is refilled from memory, giving the effect (to the preprocessing logic) of a continuous stream of instruction syllables.

The exchange register serves as the buffer register to memory and as a distributor for routing incoming information. Associated with the exchange register are the parity logic and control word detection.



Computer Cabinet With Two D850 Processor Modules Installed

Address Computer

The address computer (Figure 2-4) concerns itself with the formation of addresses which enable data to be fetched from memory. The computer is programmed by the address operators (either explicit or implicit in the addresses themselves), and its data are address and literal syllables and indirect addresses fetched from memory. The outputs of the address computer are direct memory addresses. Included in the computer is an 18-bit three-input adder used in forming addresses, an 18-bit base address register (BAR) for relativization of addresses, 15-index registers (IXR1-IXR15), each of 18 bits, and several miscellaneous registers.

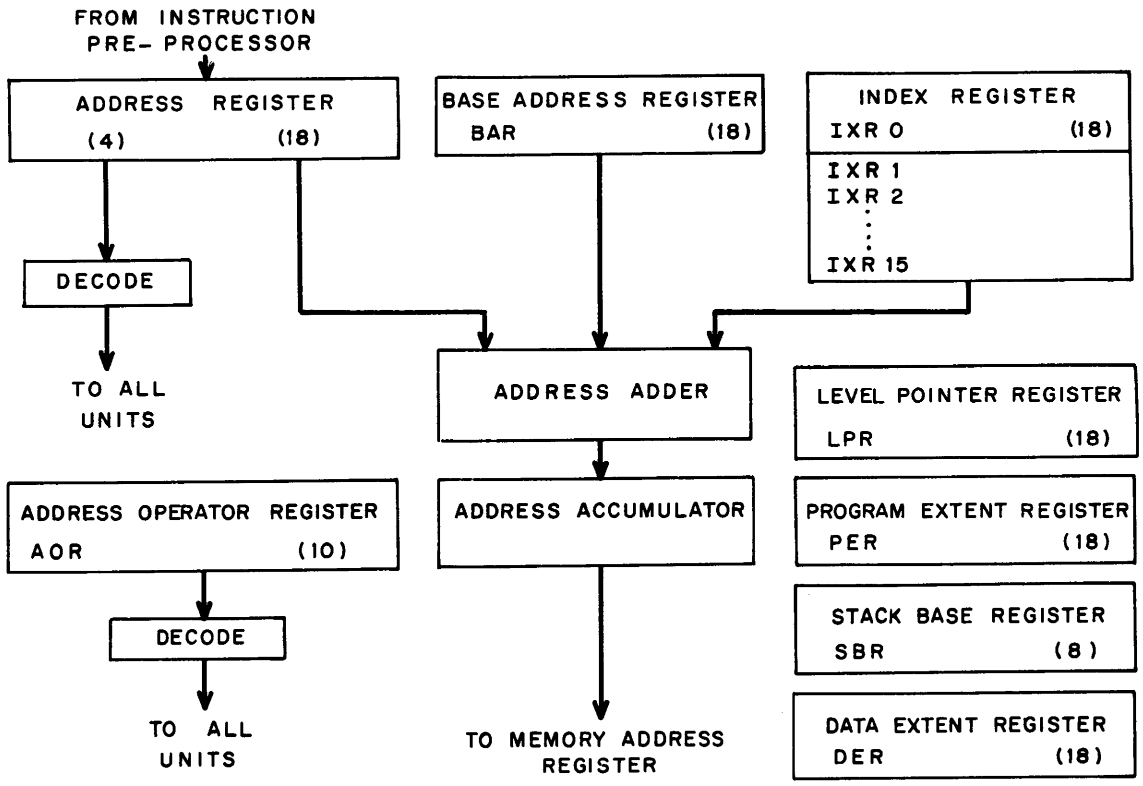
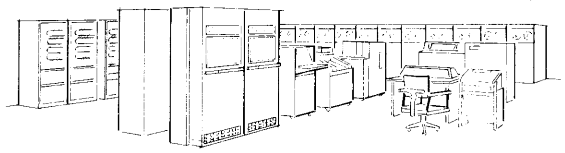


Figure 2-4. Address Computer

As address operators are sorted from the program string, they are transmitted to the address computer where they are held in the address operator register. The address operator contains a four-bit operation code (e. g., "Add to Address Accumulator"), and a six-bit address which references any of the registers in the address computer and certain other registers such as the top of the stack, etc.



Addresses stripped from the program string are loaded into the address register. Here the four-bit identifier is decoded to determine the nature of the 18 low-order bits of the words. These four bits determine whether the word is to be treated as a literal, an indirect address, or a direct address; and whether it is to be indexed by the contents of the address accumulator (which could be holding the sum of a set of index registers) or the base address register. As each address is formed, it is transmitted to the memory address register and the data fetch is initiated.

Included in the address computer are the stack base register and data extent register which provide controls for accessing any portion of the data which may have overflowed from the high-speed thin film memory into the main core memory. The level pointer and program extent registers are also in the address computer. The former controls the entrance to new blocks of program, and the latter determines the positions in which control words for the current block of program may be placed.

Arithmetic Processor

In addition to the arithmetic unit proper, the arithmetic processor contains the devices which eliminate the necessity for explicitly addressed operands and temporary storage and which provide a measure of program look ahead: the queues and the stack.

A queue is a set of registers addressed by a pair of pointers. These pointers indicate the register currently being addressed or head of the queue, and the next register available for loading or the tail of the queue. As a register is read or loaded the corresponding pointer is stepped. If the two pointers indicate consecutive positions, then the queue is full or empty; the ambiguity being resolved by the direction from which the pointers converged.

The stack is another set of registers similarly addressed. As an item is inserted in the stack, the top of the stack pointer (TOSP) is advanced; when an item is removed, it is moved back. If the TOSP approaches the bottom of stack pointer (BOSP) as it advances, it is an indication that the stack is almost full; and room is made by automatically moving the item at the bottom to the thin film memory (close-coupled fast store). Conversely, when the TOSP approaches the BOSP from the opposite direction and the stack is almost empty, the thin film memory is referenced for data which resides in the thin film memory extended stack.

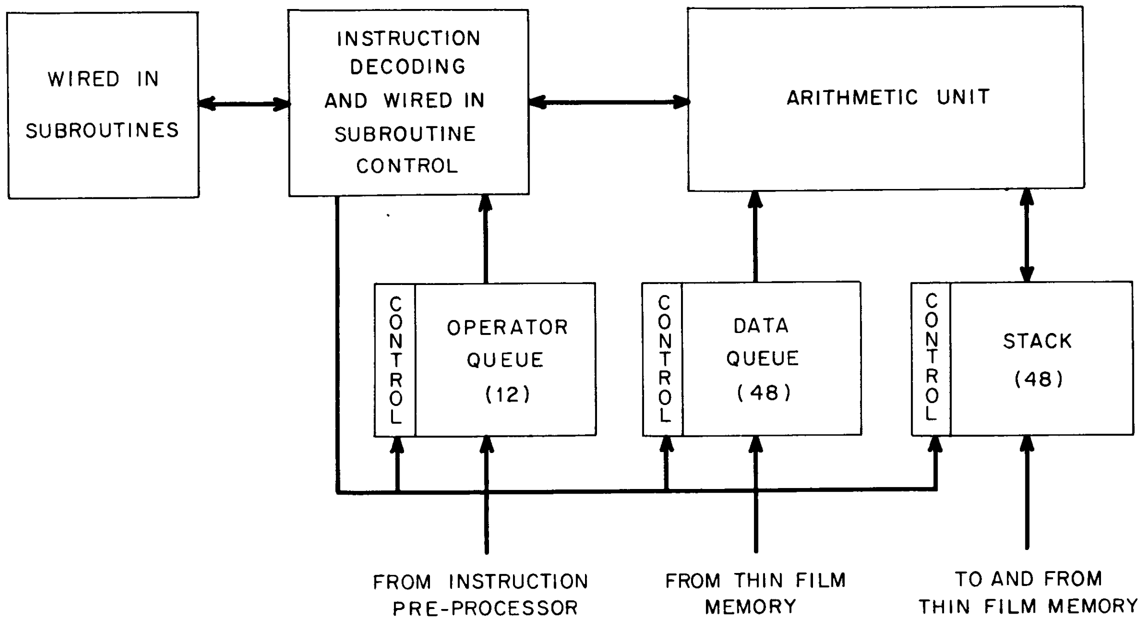
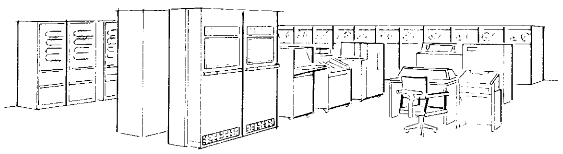


Figure 2-5. Arithmetic Processing Unit

The arithmetic processor (Figure 2-5) references two queues and the stack. The operator queue contains operators which have been selected from the program string by the instruction preprocessor while the data queue holds operands which have been determined and fetched under control of the address computer. The stack holds intermediate arithmetic results. As processing proceeds, data in the data queue and in the stack is "used up" under control of the operators in the operator queue. Speed is gained in arithmetic processing because all addressing is implicit and the time-consuming fetch cycle operates in parallel with previous arithmetic operations. Operators "consume" differing numbers of operands; therefore, the stepping of the queue and stack counter is controlled by the instruction decoding network. It is important to note that during long arithmetic operations, many positions of the queue are filled. In this way the thin film and core memories with their regular cycle times are matched to the extremely high but varying speed of the arithmetic unit.

Provision is included in all conditional operators for the programmer to indicate the desired direction in which the "look ahead" is to operate.



This feature will take advantage of the fact that normally conditional branches are biased heavily in a given direction and that the programmer can predict the direction of this bias and thus further enhance the "look ahead" feature.

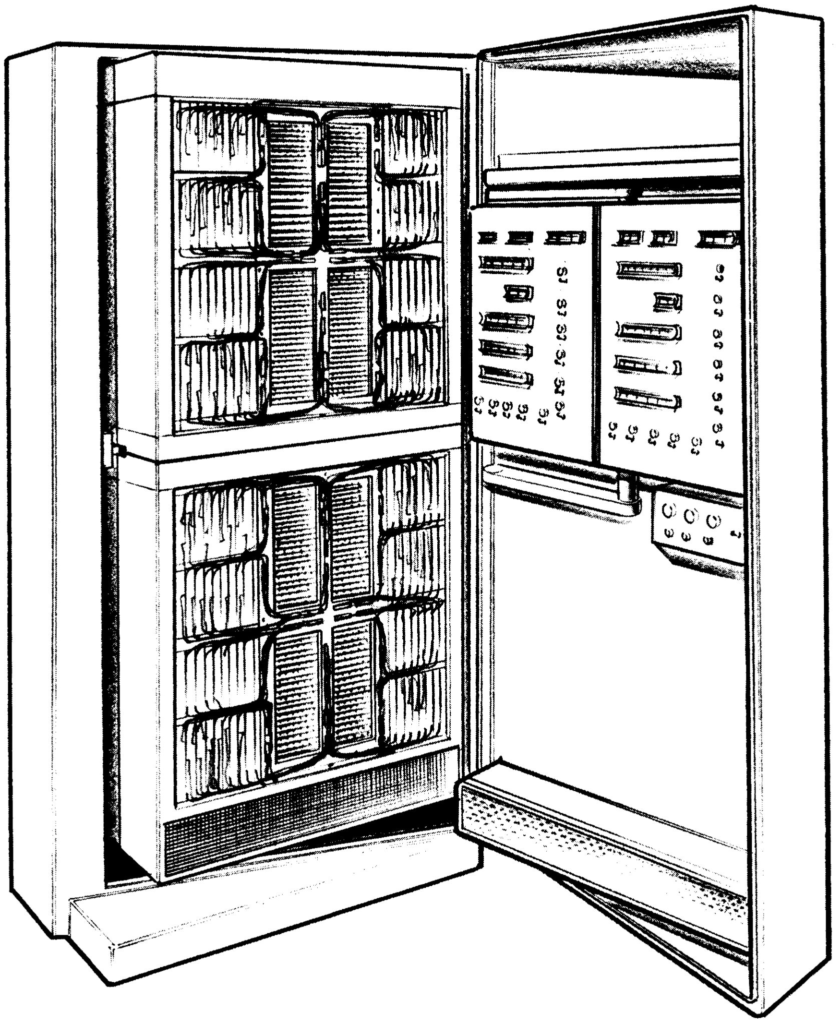
Another D850 feature which enhances its data processing power is the sophistication of its order code. The D850 is designed to operate on programs in high order programming languages such as ALGOL 60, and its order code contains operators which implement every standard function of the ALGOL 60 language (See Section 4 of Attachment 1). For example, SINX is the name of a variant of the function evaluate operation. A series of coefficients are stored in a unit-pluggable read-only memory. The first of the coefficients defines the order of the polynomial to be evaluated and others are the coefficients of each term to be evaluated. The coefficients can either be wired in or can reside in the data queue. Benefits are derived from this capability which result in faster compiling, shorter object program, and increased speed in the running of object programs.

MEMORY HIERARCHY

The memory or storage hierarchy of the Burroughs D850 is designed to be a practical and economical solution to the problem of providing memory fast enough to match hyper-speed processor modules and yet be capable of storing large amounts of randomly accessible data (up to 262, 144 words). A three-layered memory system is provided to meet these requirements. The highest strata takes the form of the operator queue, the data queue, and the stack. These are short special-purpose lists implemented with tunnel diode memory elements. The access time to each of these stores is 33 nanoseconds. They can be read nondestructively so that a rewrite cycle need not follow each access. The write cycle is accomplished in 67 nanoseconds. The number of registers to be implemented in each of the queues and the stack will depend on the results of extensive simulation tests presently in progress. A goal of these tests is the optimization of the interface between the arithmetic processor and the memory hierarchy of the D850. Tentatively, the queues and the stack have each been assigned eight positions.

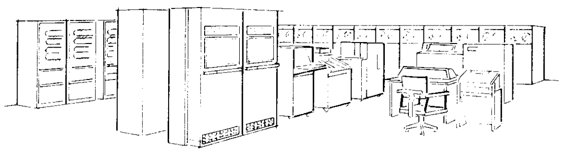
The next memory element in the hierarchy is the close-coupled fast store associated with each processor module. This is a magnetic thin film memory of 4096 full words. The read-write cycle time of this unit is 0. 4 microseconds and access to any word in it is obtained in 0. 250 microseconds. This memory is used to store active program blocks and local data and control words. In addition, the stack is extended into this memory whenever the number of stored items exceeds the capacity of the tunnel-diode registers provided.

The close-coupled fast stores, although integral to the processor module, are also part of the overall memory addressing scheme of the D850 system. The addresses 0000-4095 refer to the close-coupled fast store of processor number 1. The next three groupings of 4096 words (to address 16,383) each are reserved for the three additional processors which can be used in building the D850 to a maximum system. The remaining 245,760 cells are found in the main modular memory.



Packaging Four 16,384 Word Modules for the D850 Main Memory

The main modular memory is a system of from one to fifteen ferrite core modules of 16,384 words each. There are four sections of 4096 50-bit words in each module. All addresses in a module are, modulo four, cyclically distributed among the four sections so that although the cycle time of each bank or section is 2.0 microseconds, data rates of four times this can be obtained from consecutively addressed locations.



An additional feature of the memory system, one that is very useful in maintaining well organized program and data arrays, is the buffered block transfer mode. A control word, executed by a processor, can initiate a block transfer within the memory system. This transfer will proceed to completion without further program control in a manner analagous to a buffered input-output operation.

A memory system may concurrently process many different type of transfers; for example, a core to thin film transfer, a core to core transfer, and several input-output operations could be in simultaneous process by a memory system of no more than four modules.

INPUT/OUTPUT STRUCTURE

The input/output system of the D850 consists of up to 16 input/output control modules and various peripheral equipment such as magnetic discs, magnetic tapes, and a Burroughs B280 satellite computer for communication with punched card equipment, paper tape equipment, etc. Figure 2-6 shows the I/O control module.

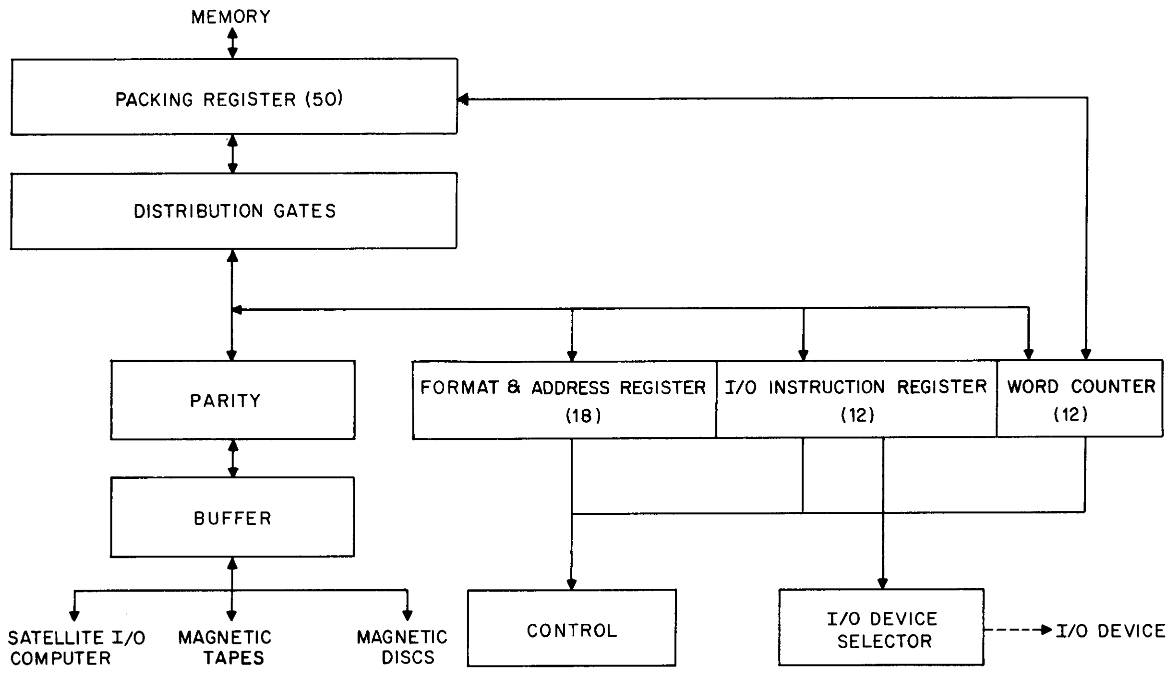
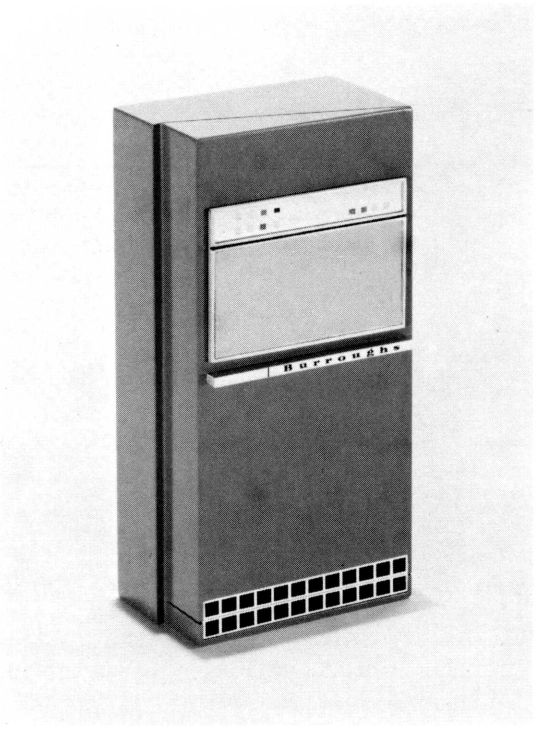


Figure 2-6. Input/Output Control Module

Input/output operations are initiated in the instruction preprocessor. Upon detection of an I/O control syllable, the succeeding 24-bit address (after processing in the address computer) is transmitted to the I/O control module. This is the address of a 48-bit I/O descriptor which the I/O control module references. The descriptor contains an I/O instruction (read type, write disc, print, etc.) and a unit address in its first 12 bits; the second 12 bits indicate the number of words to be transferred; and the last 24 bits carry the starting address of any formating information necessary.

In tape and disc operations, the operation of the system is clear. The appropriate device is selected, and words of the proper format are formed in the packing register for transmission. As each word is transmitted, the word counter is stepped and the operation is carried to completion. In operations dealing with the satellite computer, the format words in memory are referenced and transmitted to the satellite computer; this process continues until a tag in a format word indicates that this phase of the generation is complete. The format words are actually a program for the satellite computer, and contain all information as to assembly and disposal of the information. The appropriate number of computer words are then transferred between the satellite computer and the I/O control module following the same process used with tapes and discs.

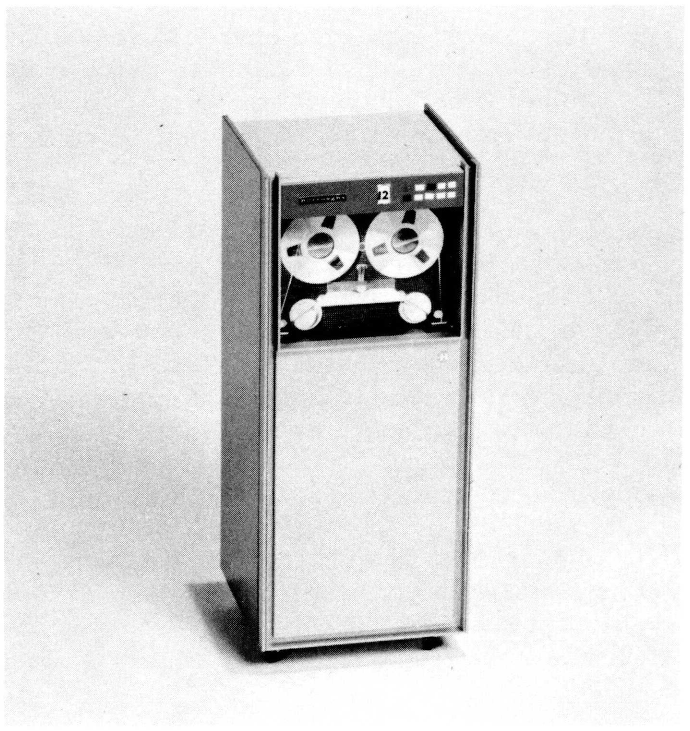
Alternatively, the satellite computer may operate directly upon the D850, tapes or discs by following the same input/output process.



MAGNETIC TAPE UNIT

The D850 magnetic tape system operates independently of the processor. Reading, writing, backspacing, rewinding, and erasing operations are under system control. A maximum of 16 of the magnetic tape units may be used with the system.

The D850 tape unit accepts data in either binary or single-frame alphanumeric form. Tape format is compatible with IBM Model 729-II and 729-IV magnetic tape units. Standard tape one-half inch in width is used. Tapes are mounted on reels which contain up to 3600 feet of tape and have a maximum diameter of 10-1/2 inches.



Data may be stored in two densities, either 555.5 or 200 frames per inch. One frame contains either six binary bits or one six-bit alphanumeric character. Tape speed is 120 inches per second, a transfer rate of 66,660 characters per second for a density of 555.5 frames per inch, and 24,000 characters per second for a density of 200 frames per inch. Packing density is selected by a switch on the unit.

Tape is rewound at a speed of 340 inches per second. Start or stop time is 5 milliseconds. The dual-gap read-write head of the D850 tape unit provides automatic checking of write operations.

Mounting and removal of tape reels is facilitated by quick-action reel locks. A reel brake release switch permits loading or unloading of the tape. A write ring must be installed on a reel to permit writing or erasing, thus preventing accidental destruction of files. After the tape reel has been mounted on the unit, activating the load switch causes the tape to be drawn into the slack loop mechanism and to be automatically positioned at the beginning-of-tape marker, ready for operation. An unload switch is used to reverse the load procedure for removal of the tape reel. The unit must be in local mode for these operations.

A ready indicator shows when the transport is in a ready state. A write warning light is turned on if the reel installed on the transport is equipped with a write ring.

Tape format consists of seven recorded channels across the tape. The information tracks, identified as 1, 2, 4, 8, A, and B, represent either single-frame alphanumeric or binary information. The C track provides a parity check for each frame. The nonreturn-to-zero method is used for recording.

Reading and writing can be done in either binary or alphanumeric mode, providing complete code flexibility to the system. The alphanumeric mode carries an even parity. That is, a parity bit is recorded in the C channel simultaneously with each character if an odd number of bits represents that character in the information channels. The binary mode carries an odd parity. A parity bit is recorded in the C channel if there is an even number of bits in the information frame.
It is possible to write interspersed binary and alphanumeric records in the binary mode.

A longitudinal check character is written after the last character of each record. This consists of a parity bit, automatically recorded in each track with an odd total bit count. These parity bits maintain an even number of ONE-bits in each track for the entire record length, regardless of the code used.

Both forward and backward read operations can be performed in either binary or alphanumeric mode. Regardless of reading direction, information is placed in memory in normal sequence.

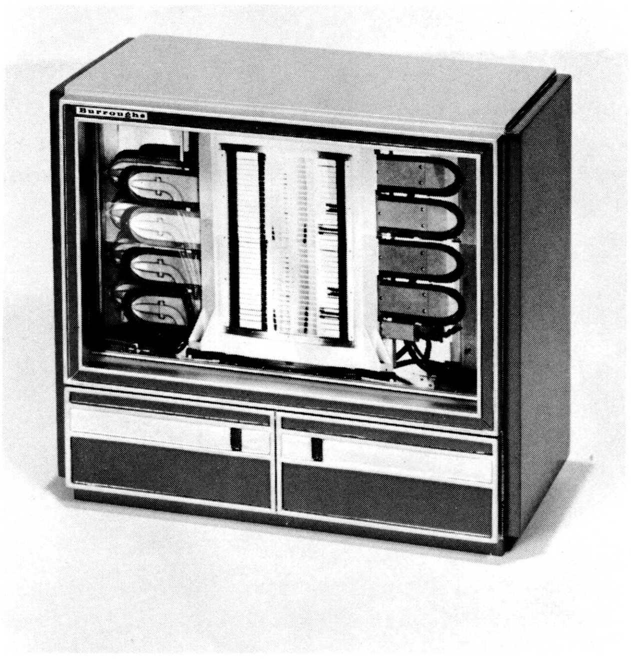
Write operations are performed in the forward direction, placing information from memory in ascending sequence on tape. Alphanumeric records may vary in length from one character to memory size in one-character increments. A group mark terminates an alphanumeric write operation. When the binary mode is used, one to 1024 words may be recorded in one-word increments. Writing the number of words specified by an output descriptor terminates a binary write operation. A 3/4-inch record gap is automatically supplied at the end of any write operation.

In addition to reading and writing, the tape system performs backspacing, rewinding, and erasing operations. When encountered, a backspace descriptor causes the designated tape unit to backspace one record. A rewind descriptor causes the unit to rewind to the beginning-of-tape marker. An erase descriptor is used to erase desired lengths of tape, as in extending a record gap over a tape flaw.

MAGNETIC DISC FILE

The magnetic disc storage unit is a unit organized so as to provide parallel information transfers of 24 bits. Each computer word is stored as two such segments with a longitudinal parity segment in each data block.

Each disc file can contain either 12 or 24 discs (with a 12 disc unit being expandable to 24) and has a capacity of 7,200,000 (12 discs) or 14,400,000 (24 discs) computer words.



The data transfer rate depends upon which of the six radial zones on the discs are involved in the transfer. The maximum transfer rate is 4.76 microseconds per word and the minimum rate is 12.12 microseconds per word. The access delay to a random address is a maximum of 100 milliseconds.

The Burroughs Corporation, as part of its effort for the development of proprietary products, is presently engaged in a program aimed at an improved disc file storage system which will be interchangeable with the current disc file but which will represent a great improvement over it with respect to:

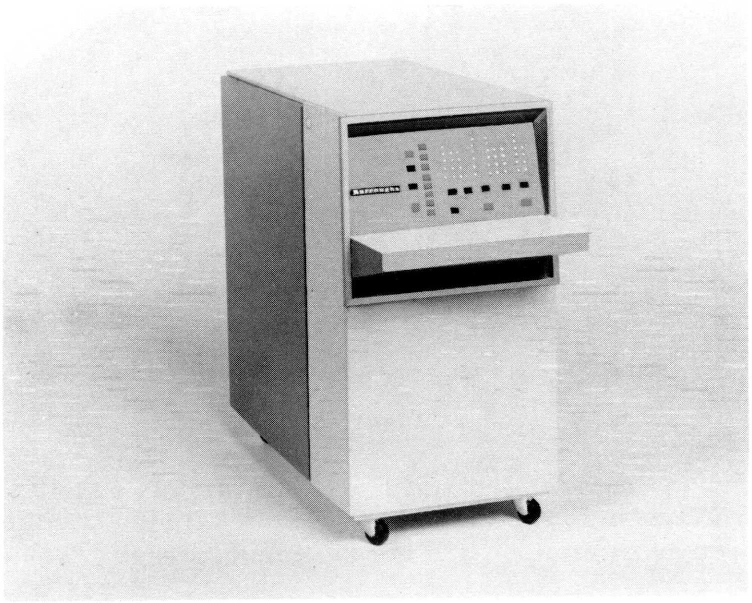
- increased storage capacity
- increased data rate
- radically reduced access delay

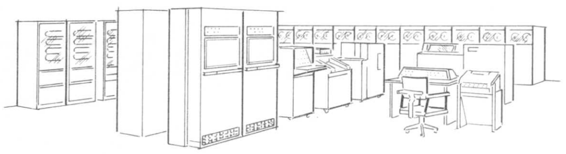
The goals for this program also include a much greater tolerance with respect to ambient conditions.

If this item is available in time for delivery with this proposed D850 system, it can be substituted for it. In any case it can be retrofitted into the D850 system at the installed site at a later date.

B280 SATELLITE COMPUTER

The B280 processor contains 4800 alphanumeric positions of core storage. Each position consists of a 6-bit information representation plus parity. The processor operates on fixed length 3-address, 12-character instructions; the data field is of variable length.





The Burroughs B280 processing system is used to interconnect devices of relatively low data rates into the D850 system. In addition to controlling data flow, the B280 also performs editing and formating functions. The B280 may be also used as an off-line processor for the following functions:

$$\begin{tikzcd}
\text{Card} \arrow[r, shift left] & \text{Tape} \arrow[l, shift left] \\
\text{Card or Tape} \arrow[r] & \text{Printer} \\
\text{Card or Tape} \arrow[r] & \text{Display/Recorder}
\end{tikzcd}$$

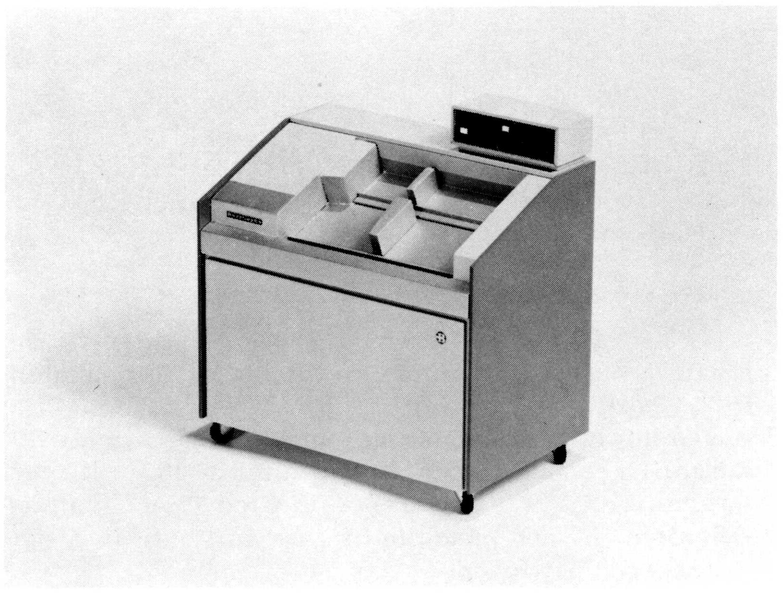
The B280 peripheral processor can handle a complement including:

- 2 input devices such as card readers
- 1 card punch
- 1 line printer
- 6 magnetic tape units or display/record devices.

Available with the system is an assembler and a generalized report generator.

CARD READER

The card reader is able to read punched cards under the control of the B280 processor at a maximum rate of 800 cards per minute. This unit also utilizes an immediate access clutch and is completely buffered. Transfer of the contents of the buffer to memory requires 2 milliseconds, while the time required to read another card, refilling the buffer, is 73 milliseconds.



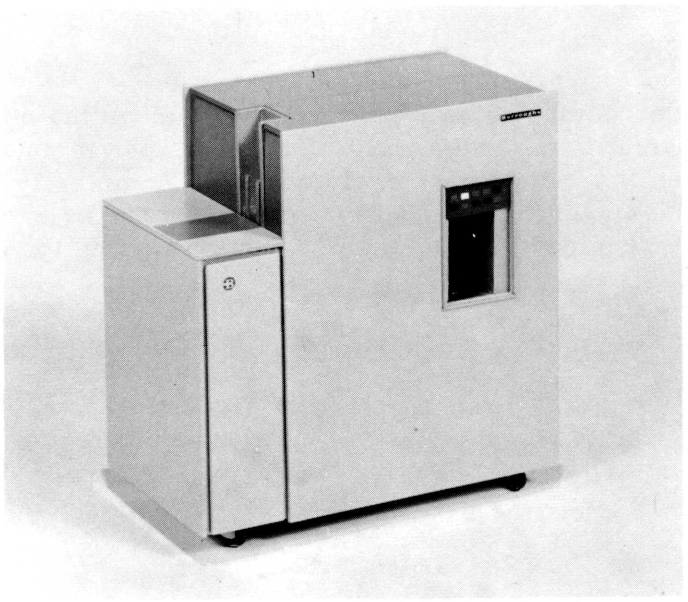
The information punched in the card is read and transferred one column at a time into the buffer.

The card reader also features character validity checking and monitoring ✓ of photoelectric read circuitry. No control panel is used. The card reader is capable of processing either 51-, 60-, 66-, or 80-column cards, of standard or post card thickness. Card stock thickness and length must be consistent during any one run. The card hopper and stacker will each hold approximately 2400 cards.

A maximum of two card readers may be used with any B280 system.

CARD PUNCH

The card punch has a maximum card punching rate of 300 cards per minute. The format of the output cards is under program control and no control panel is used. The high-speed punch is completely buffered.

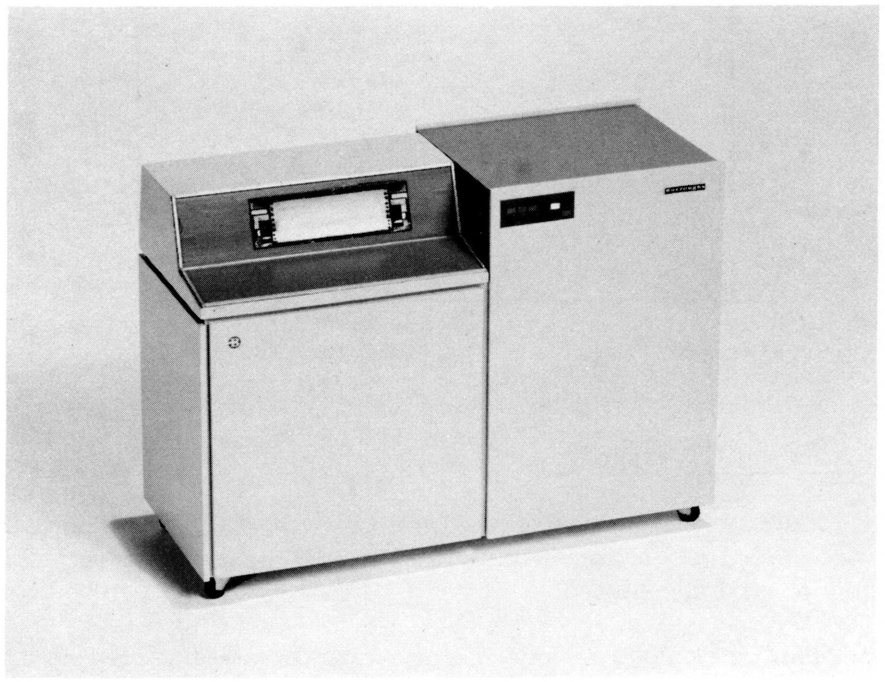


Transferring data from B280 memory to the buffer for punching is accomplished in 2.5 milliseconds. After the buffer is filled, a card is punched. If punching is continuous the punching operation will be completed in 197.5 milliseconds, or in a maximum of 267 milliseconds if punching is intermittent. The difference in time is necessitated by the clutching mechanism. When punching is intermittent, it is delayed until the clutch is properly positioned.

The card hopper on the card punch will hold approximately 1000 80-column, standard or post card thickness cards. Cards are fed from the hopper into the punch station and then into the output stacker which also holds approximately 1000 cards. Cards may be loaded and removed without stopping the unit.

LINE PRINTER

The line printer is a drum printer which prints 700 lines per minute when single spacing, or 650 lines per minute double spacing. Formating, editing (the insertion of commas, decimals, dollar signs, zero print control, etc.), form skipping and spacing are under B280 program control.



The transfer of data to be printed from storage to the print buffer is accomplished in 3.5 milliseconds. A line is printed as soon as the print buffer is filled.

There are 120 printing positions and 64 characters can be printed from each position as required by the data in the print buffer. Horizontal spacing is ten characters to the inch. Vertical spacing is either six or eight spaces to the inch and is under operator control.

The printer drum contains 64 characters in each printing position; 26 alphabetic, 10 numeric, and 28 special characters. The B280 processor installed as an input/output conversion system ("satellite systems") for large-scale computers often requires the printing of more special characters than the 11 normally available. For this reason either of two 28 special-character sets are standard equipment (see Figure 2-7). The additional 17 characters in each set are used to formulate problem statements for the ALGOrithmic Language (ALGOL).

COMMERCIAL SET		SCIENTIFIC SET	
Blank	=	Blank	?
.	+	.	#
)	C_R	)	[
(	≠	(	]
&	?	*	>
*	"	;	<
;	$	-	$
-	]	/	∨
/	[	,	∧
,	>	:	"
%	<	x	¬
:	Δ	#	Δ
@	≤	=	≥
#	≥	+	≤

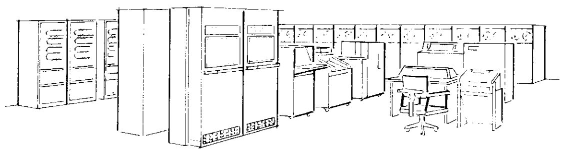
Figure 2-7. Special Character Sets

Tape Controlled Carriage

The tape controlled carriage on the line printer, in conjunction with the stored program instructions, controls the feeding and spacing of forms while they are being printed.

The form can be skipped to any one of eleven predetermined positions, as designated by a hole punched in the carriage control tape, under program control. The twelfth position in the control tape is used to designate the last printing line on a form. When a hole in this position is sensed, the line printer makes an impulse available to the central processor causing the print or skip instruction to take the end of page branch. Printing will occur on the line at which the 12 punch was sensed.

The carriage control tape has 12 columnar positions (1-12), called channels. Horizontal lines are spaced six to the inch, providing 132 lines (22 inches) for control of a form. For ease in preparation the form is somewhat longer than 22 inches.



Prepunched holes located in the center of the tape are used by a pin feed drive to move the tape past the sensing mechanism. Movement of the carriage control tape is synchronized with the movement of the form through the carriage.

The twelve carriage control tape channels are usually punched as shown below to control the following functions:

1. Channel 1 will normally be used to identify the first printing line or home position of a form.

2. Channel 2 is usually used to indicate the first body line of a form on which detail information also appears. In an invoicing operation, channel 1 would be used to indicate the first printing line on the form, in this case a name and address. Channel 2 would correspond to the first printed line of detail information.

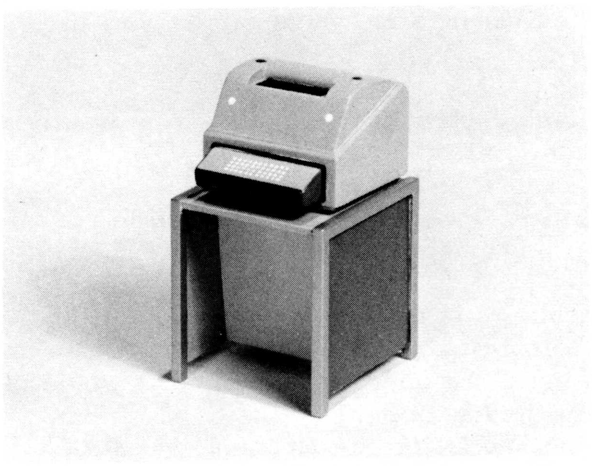
3. Channels 3-11 will normally be used to identify any one of 9 user-determined print positions. These 9 channels may be used in any desired sequence.

4. Channel 12 is reserved for punching the hole indicative of the last printing line in the body of a form. When a channel 12 punch is sensed, during a print or space operation, program contol will branch to location specified by the instruction.

INQUIRY/OPERATING STATION

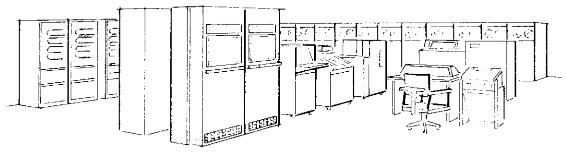
The active interface between the programmer/operator and the D850 system is the inquiry station, a teletypewriter which can print ten characters per second and has a keyboard for input messages. There are four special-function keys:

- Inquiry - sets a system interrupt condition to alert the D850 operating system of an input request
- Priority - establishes an inquiry as a priority request
- End-of-Message - signals that an input message is complete
- Error - signals that the last character was invalid



The man-machine interface is further implemented by system status indicators on the operating panel. These include AC power supply indicators and controls, on-line/off-line indicators for each module and system-ready indicators. Another function of the operating console is a micro-film projector. This device contains a file on micro-film of the D850 operating and maintenance manual library. By means of suitable controls, the operator can select any page from these manuals and cause it to be projected for view. A means is also provided for making a full-size copy of the page in a few seconds so that it may be taken for reference at another location.

III D 850 Design Considerations



III D850 Design Considerations

MECHANICAL

Design emphasis throughout the D850 system has centered upon speed-but speed with reliable operation. Hence, in selecting packaging and design techniques for the D850, constant attention has been given to reducing the amount of cabling integral to the system. The highest attainable packaging densities consistent with proven reliability and off-the-shelf availability have been sought to provide elimination of cabling requirements when possible, or otherwise shortening of cabling lengths.

Figure 3-1 shows a room layout with adequate (3 feet) spacing between all equipments and adjacent walls, minimization of walking movements between related equipments, and consideration for visual blocks, lighting, and table surface adequacy.

Table 3-1 shows the operating range of environmental characteristics and power requirements for each of the units in the D850 system. The power ranges reflect the maximum permissible time variation to the motor generator set at ± 10%; maximum frequency variation is also ± 10%.

The central system may be shipped via rail/truck transportation. When not in a shipping container, the equipment will meet the shock requirements per MIL-E-4970, shock test procedure No. 4, for bench handling, servicing, and installation. For transit, the equipment will withstand the shock requirements per shock test procedure No. 2 with shock limited to 15 G's.

The equipment will be designed to withstand 15 to 55 cps vibration at 0.015 inch double amplitude and 2 1/2 G's maximum unless installed in a transit container.

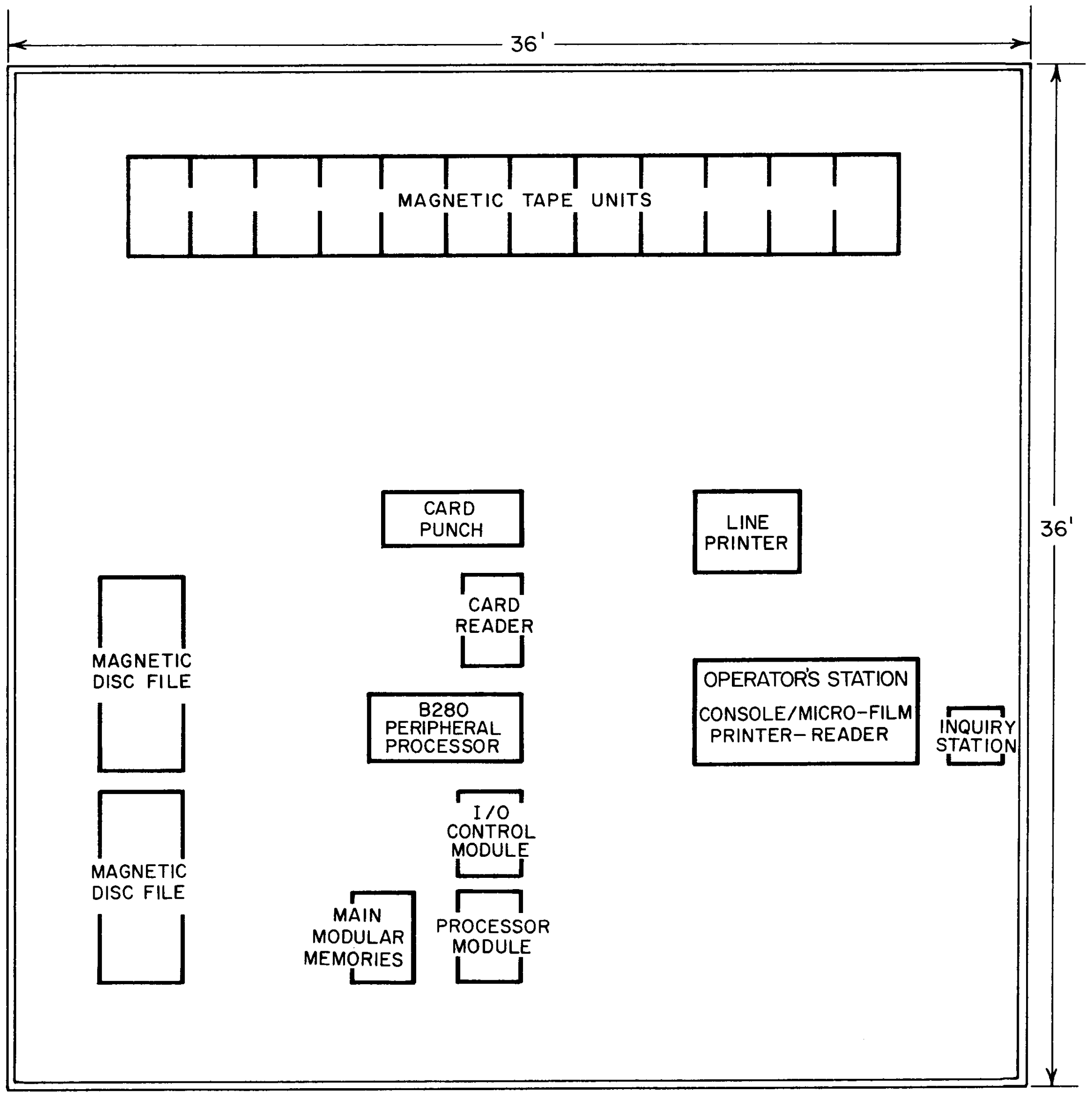
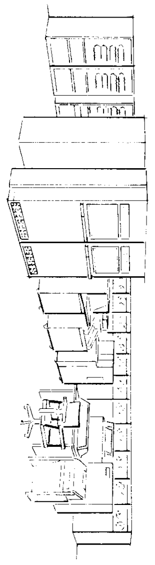


Figure 3-1. Typical D850 Installation Floor Plan

TABLE 3-1. SYSTEM ENVIRONMENT AND POWER RANGES

	Temp. °C	Relative Humidity (%)	Altitude (feet)	KVA Required	Voltage (volts)	Ø	Frequency (cps)
Processor Module	0-50	40-60	10,000	3.8	187-229	3	360-440
Memory Module	0-50	40-60	10,000	5.7	187-229	3	360-440
I/O Control Module	0-50	40-60	10,000	1.8	187-229	3	360-440
Magnetic Tape Unit	18-27	40-60	10,000	3.2	187-229	3	47-63
Magnetic Disc File	10-40*	40-60	10,000	5.0	187-229	3	47-63
Operating Station	16-38	40-60	10,000	0.8	187-229	1	47-63
B280 Peripheral Processor	0-50	40-60	10,000	1.0	187-229	3	47-63
Card Reader	12-32	40-60	10,000	2.0	108-132	1	47-63
Card Punch	16-38	40-60	10,000	3.0	108-132	3	47-63
Line Printer	16-38	40-60	10,000	3.0	108-132	3	47-63
Inquiry Station	16-38	40-60	10,000	0.35	108-132	1	47-63

*Records must be read at same temperature at which they were written, ±3°C



The equipment will use fungus resistant materials. The choice of materials will be selected from those considered as nonnutrients as defined by MIL-E-4158.

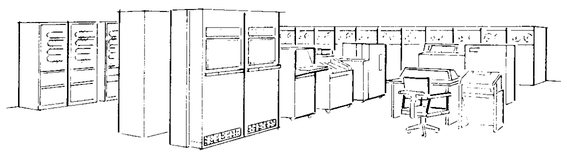
A standard multi-purpose cabinet houses all of the system elements for the computer, memory, and input-output control functions. Each system cabinet includes a unitized power supply to provide regulated power for the equipment it contains and an integral air filtering and circulating system. Cooling air enters each modular cabinet at the bottom through a shielded opening at the front and rear and is exhausted at the top. The exhaust opening is also shielded and is protected against the entrance of dust when the equipment is not in operation. Separate fans are provided for each of the sections located within each cabinet to ensure uninterrupted cooling for every section during periods of maintenance.

The r-f shielded cabinet measures approximately 40 inches wide, 28 inches deep, and 81 inches high. The hinged front door provides access to the internal rack-mounted equipment and also contains the operating and maintenance control devices. These control and indicating devices are mounted on an enclosed panel which can be rotated to permit operation of the control devices and viewing of the indicators from the inside surface of the door during periods of maintenance routines. Within each of the major systems modules, the circuitry is contained in plug-in assemblies which mate with a common back plane containing the circuit ground and interconnecting wire harnesses. Each modular unit consists of a hinged rack assembly which contains the plug-in assemblies, memory stack, and associated back-plane wiring. The hinged rack assembly permits complete accessibility to all subassemblies and back-plane wiring from the front of the cabinet. Specific packaging considerations for each of the major systems are described in the sections following.

At the operator's desk is an automated micro film projector/reader and printer for convenient display of instruction and maintenance manual pages as well as logical and schematic wiring diagrams.

Processor Module

The processor (Figure 3-2) with a clock rate of 30 megacycles per second requires high-speed transfers particularly in the arithmetic register, control, and adder sections of the machine. To limit transfer times in the processor to a maximum of 1 clock time requires that no signal leads exceed 3 to 4 feet in length and that leads between the arithmetic register and adder not exceed 1-1/2 feet. To accomplish this the processor backboard wiring area is 2 feet by 2 feet and plug-in package lengths are less than 1 foot. Control of lead length is main-



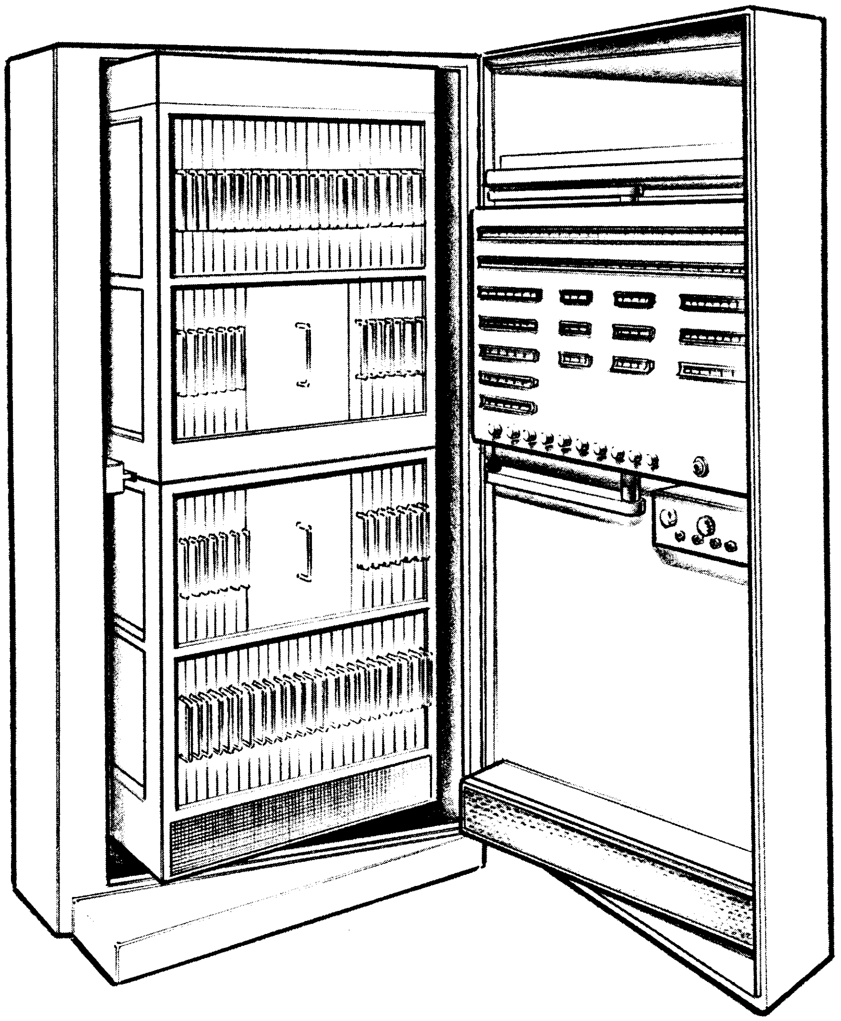


Figure 3-2. Packaging of the D850 Processor Modules

tained by packaging a large portion of the subsystem in a single package and locating critical areas in close proximity.

The processor volume is then less than 4 cubic feet for both the thin film close-coupled fast-storage memory and approximately 100,000 components of the processing and tunnel diode interim memory sections. This represents a component packaging density at the machine level of approximately 31,000 components per cubic foot for 100,000 components. At the component module level the component density is in excess of 100,000 components per cubic foot including cooling air spaces between packages, loss due to plug-in package structure and printed-circuit boards, and space between adjacent component modules.

The Burroughs logi-mod developed for the Mauler program and in production since early 1961 has a packaging density of approximately 120,000 components per cubic foot using standard, currently available components. The use of this logi-mod accomplishes the requirement for high-density packaging. The logi-mod consists of a component cluster attached to a header designed for plug-in to a printed-circuit board. The component cluster is an all-welded unit using standard off-the-shelf components with all components arranged vertically on a phenolic plaque; the plaque is drilled to accept the component leads to which interconnecting wires are welded to form the resultant component cluster. The header is a molded diallyl thalate piece containing input-output pins arranged in a 3 X 6 matrix.

Two sizes of logi-mod, each containing the same matrix arrangement of pins, are currently in production and satisfy all component packaging requirements for this system. The basic logi-mod measures 0.6 inch by 0.9 inch by 0.65 inch high; the larger unit has twice the width of the basic module.

The logi-mod is plugged into a mother printed-circuit board mounted to a plug-in package tray similar to a unit currently in use on another multimegacycle machine. The tray is a formed-metal package 13 inches high, 11 inches deep, and 3/4 inch wide and forms the ground system for the plug-in package; the grounded tray design permits minimum-length soldered ground connections from the logic elements and a low resistance path to the system ground. Spring contacts between the plug-in package and package guide runners provide additional ground paths to the system ground and in addition permits the package trays to perform as heat sinks for the logi-mods. Figure 3-3 shows formation of a plug-in package tray. The printed-circuit mother board can accommodate up to 130 logi-mods in a 10 X 13 matrix arrangement. Space between pairs of rows of logi-mods is provided for routing of coaxial lines and signal leads not incorporated into the printed-circuit pattern. Two right-angle male connectors are also mechanically attached to the tray and dip soldered to the printed-circuit board. Each plug-in package is also provided with coding pins on the rear which mate with receptacles in the back plane to prevent inadvertent insertion of a package into a wrong location.

The backboard connector panel forms the ground plane for the module and a mounting surface for the wiring harnesses and voltage buses. For this high-speed machine the voltage drops cannot exceed 1/2 of 1 percent. This is accomplished with the use of strip lines which route the voltages from the power supply to the plug-in packages with a continuous bussing arrangement. The strip line is a laminated bus using half-hard copper and a high temperature insulating material. The strip line is a rigid member for the horizontal and vertical runs at each rack assembly on the backboard connector panel and a flexible bus as it passes around the hinge and is routed to the power supply. The voltage

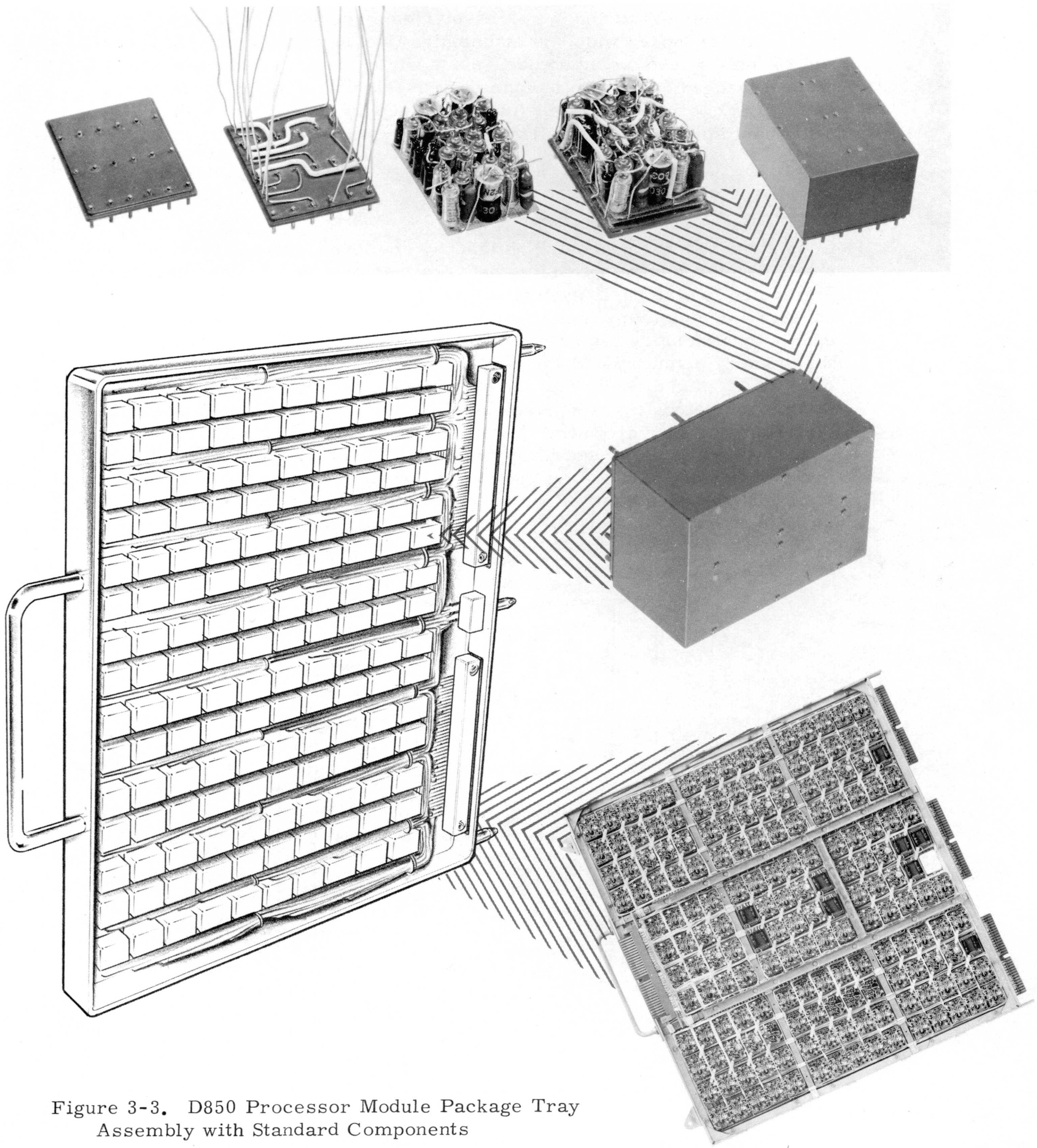


Figure 3-3. D850 Processor Module Package Tray Assembly with Standard Components

buses have a rectangular cross-section and are attached to the backboard panel with insulated stand-offs. At each package voltage, fingers run off from the bus to the package rack and panel connector.

The rack assembly for each processor module occupies a volume measuring 24 inches wide, 30 inches high (including fan mounting space), and 14 inches deep including back-plane wiring depth. The power supply is arranged behind the module rack assembly and permits the use of parallel cooling air paths. The power supply is designed with adequate heat sinks and also utilizes the cabinet structure as a heat sink.

Main Modular Memory

Consistent with the packaging philosophy employed in the processor module, interconnecting cabling between memory modules has been eliminated for the basic system and reduced for expanded systems by housing 4 complete main modular memories of 16,384 words (50 bits) each in one memory cabinet. Logi-mods are also used for packaging the memory circuitry and are plugged into printed-circuit board assemblies which plug directly into the backboard connector panel. Since the memory circuitry comprises smaller functional subsystems than the data processor circuitry, a large number of small printed-circuit board assemblies are used to improve ease of handling for test and maintenance and to increase the number of similar-type board assemblies for cost reduction. Each board assembly has a usable area measuring approximately 5 X 7 inches and can accommodate up to 1000 components with the logi-mods arranged in a 6 X 7 matrix.

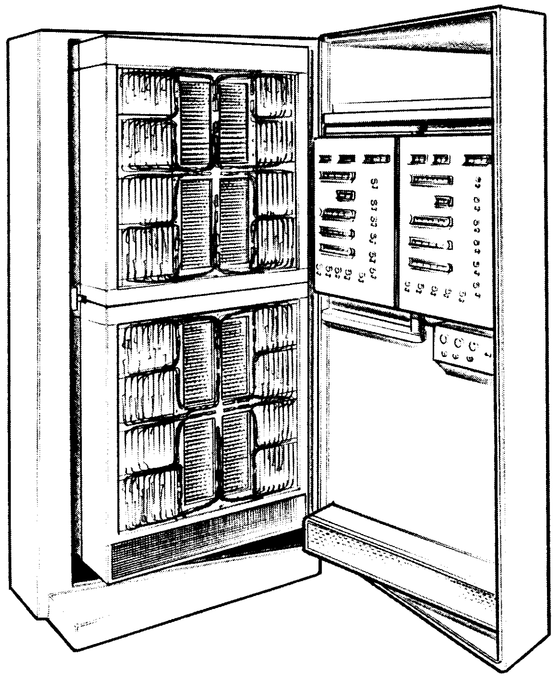
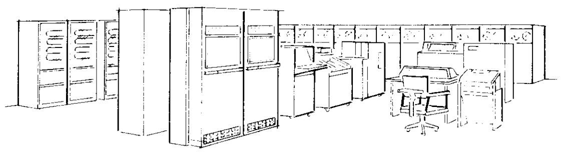


Figure 3-4. Memory Cabinet Internal Packaging Arrangement



The main modular memories are housed in a cabinet identical to the processor module. Each of the four memories is separately addressable and self contained in a rack measuring approximately 30 inches wide, 34 inches high, and 7 inches deep. The four rack assemblies are arranged in two tiers, one front and one rear, and hinged to provide access to the printed-circuit boards, to back-plane wiring of the rear racks, and to the power supply section located in the rear of the cabinet. Figure 3-4 shows the internal packaging arrangement for the memory cabinet.

The memory stack proper is a composite of four 4096-word, select phased banks. (See Figure 3-5.)

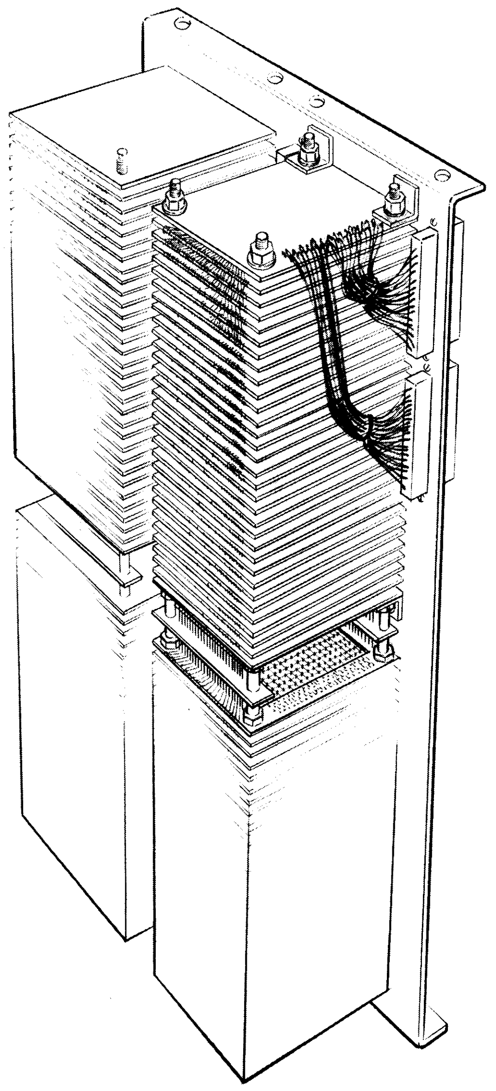


Figure 3-5. Memory Stack

Input-Output Control Module

The input-output control modules are housed in cabinets identical to the type used for the data processor and memory modules. The cabinet accommodates four I/O control modules. Each module is contained in a rack assembly identical in size to that used for the memory module. The printed-circuit card size is also the same profile size as the card assemblies used for the memory circuitry except that open circuitry is used for the component packaging. High packaging density is not required in this module and the board assemblies are manufactured by a completely automated process resulting in a lower manufacturing cost than the more densely packaged system elements and higher reliability. Wave soldering is used in the automated process to lessen board distortion and to afford maximum protection to the components by reduction of heat transfer during the soldering operation.

CIRCUIT DESIGN

The reliable systems operation of the circuits employed in D850 design is already well established. The operation of these techniques – DADOT logic, memory hierarchy construction, and intermodular switching and communication – is also well known such that the superlative system performance of D850 accrues from the selection and application of known techniques and not from revolutionary concepts of design.

Logic Circuitry

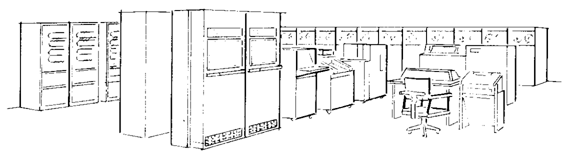
The Diode AND - Diode OR - Transistor (DADOT) buffer technique coupled with high-speed flip flops and shift registers has demonstrated marked stability and reliability for a wide range of applications. These include a 30-megacycle system and a large-scale (one million component) 10-megacycle system. DADOT is characterized by

- Simplicity of circuit configuration
- Optimum use of transistor gain-bandwidth product
- Noncritical transistor parameters
- Minimum component count

In essence,DADOT utilizes diodes for logic manipulation, and transistors as emitter-follower buffer amplifiers.

D850 Memory Hierarchy

The memory hierarchy actually comprises two physically separated storage areas; the main memory modules of ferrite core consturction,



and the thin film memories associated with each processor module. A set of small ultra-high-speed tunnel-diode memories is also used for interim storage (queues and stacks) with each processor module.

▲ Main Memory Modules

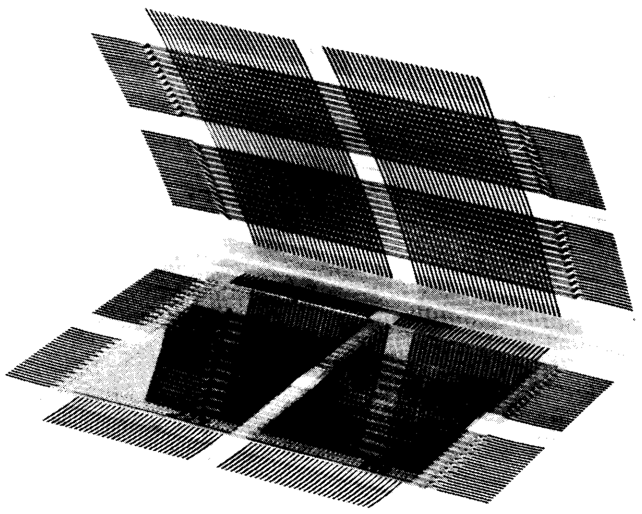
Modules of ferrite core memories, each having a capacity of 16,384 words, provide the D850 with an extremely flexible and expansible memory storage capability.

Speeds of 2-microsecond cycle time and 0.5 microsecond access time (for each of the four-phase banks exclusive of switch-interlock times) are achieved through linear-select word organization (as opposed to coincident-current bit organization). Burroughs pioneered in the R&D efforts on linear select memories and is thoroughly knowledgeable of both fabrication and application techniques and of developments throughout the industry in circuitry and core characteristics which contribute to continuing improvements in linear select memory designs. The cycle and access times provided for D850 operation pose no anticipated difficulties in technical achievement even for such a large-scale system application.

▲ Processor Close-Coupled High-Speed Storage

Each operational module (of which four may be used for the 262,144 word maximum system configuration) contains a 4,096 word magnetic thin film memory. This provides higher speeds of 0.4 microsecond cycle time and 0.25 microsecond access time to the processor modules. These memories are also word organized. A single thin film plane is shown in the illustration on the following page.

Each plane contains 64 groups of 24 bits. The complete memory module is constructed of 128 planes. The memory bits are rectangular spots of super-thin nickel-iron alloy vacuum-deposited on a glass substrate. The axis of preferred magnetization is aligned with that of every other bit on the plate. Parallel axes are established by deposition of the material under the influence of a strong uniform magnetic field. Precise control of the deposition process is required to obtain the necessary uniformity of characteristics from bit to bit and from plane to plane. The high bit denisty planes required for the D850 are currently in full production at the Burroughs Electronic Tube Division. These planes are used in the D825 Modular Data Processing System.



In operation, each bit is magnetized in one of the two directions allowed by the axis of easy magnetization. One of the directions represents a ONE, the other, a ZERO.

During READ, a drive current is applied along a line of bits representing a word in such a direction as to provide a strong field perpendicular to the axis of easy magnetization. The magnetization vectors are forced toward the perpendicular direction. Change in magnetization is extremely rapid because the super-thin films are constrained toward magnetization change by domain rotation rather than by domain wall motion. The change takes place during the rise time of the drive current. Sense lines pick up the change in magnetization as a voltage difference; the sense of the voltage shows whether a ONE or ZERO was present.

During WRITE, a separate information current is applied to each bit while the drive current is still present. The information currents are applied perpendicular to the drive current so that the direction of the information current biases the magnetization vector either toward the ONE or the ZERO state. When the drive current is removed, each bit becomes magnetized to the biased state.

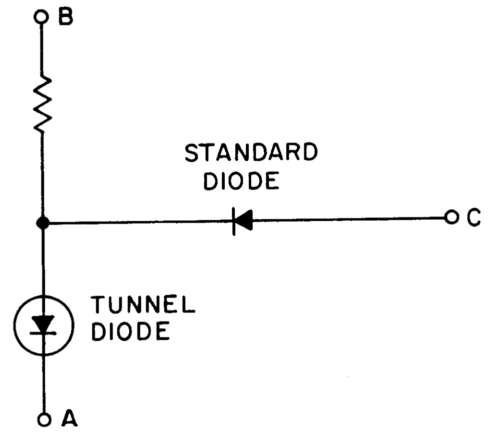
Thin film memory cells are inherently very high-speed devices. The speed of the memory is limited by practical considerations such as the transition times of the high currents used, the reaction times of the high gain sense amplifiers, propagation times along the sense lines, and system noise considerations.

▲ Interim Storage

Each operational module contains a number of small high-speed interim stores. The number of bits involved is too large for the economical use of flip-flop registers and it is questionable whether the practical speed of even a small thin film memory is sufficient. Thus the use of tunnel-diode memories is indicated.

Tunnel diodes have switching speeds of the same order as thin films. Unlike thin films, however, they are easily driven (currents on the order of milliamperes rather than amperes), easily sensed (voltages on the order of several tenths of a volt rather than millivolts), and provide memory with a non-destructive read-out characteristic. Thus, many of the speed limitations imposed by practical considerations in the case of thin films are eased in the case of tunnel diodes so that a considerable speed advantage is assured. Conservative estimates indicate an access time (READ only) of 0.033μsec and a cycle time of 0.1μsec.

The basic memory element shown below comprises a tunnel diode, a standard semiconductor diode, and a resistor.



A dc bias is provided at point B, positive with respect to A, such that the tunnel diode remains in the state to which it is set. Point C is so biased that if the tunnel diode is in its low voltage state (representing ZERO), the standard diode can conduct; if the tunnel diode is in its high voltage stage (representing ONE), the standard diode cannot conduct.

During READ, a negative voltage pulse is applied to A. If the memory element is in the ZERO state, the standard diode conducts and a negative voltage appears at C; otherwise, the standard diode does not conduct and no change in voltage appears at point C.

During WRITE, positive information voltage is applied to C for writing a ONE, or the voltage remains unchanged for writing a ZERO. At the same time, the same negative pulse used in READ is applied to A. If both exist, the tunnel diode switches to the ONE state. If only A exists, the tunnel diode remains as it was. During the application of A, the bias at B is reduced to zero and then returned to its original value. If C does not exist, the tunnel diode switches to ZERO. Figure 3-6 shows how a queue is implemented in this technique.

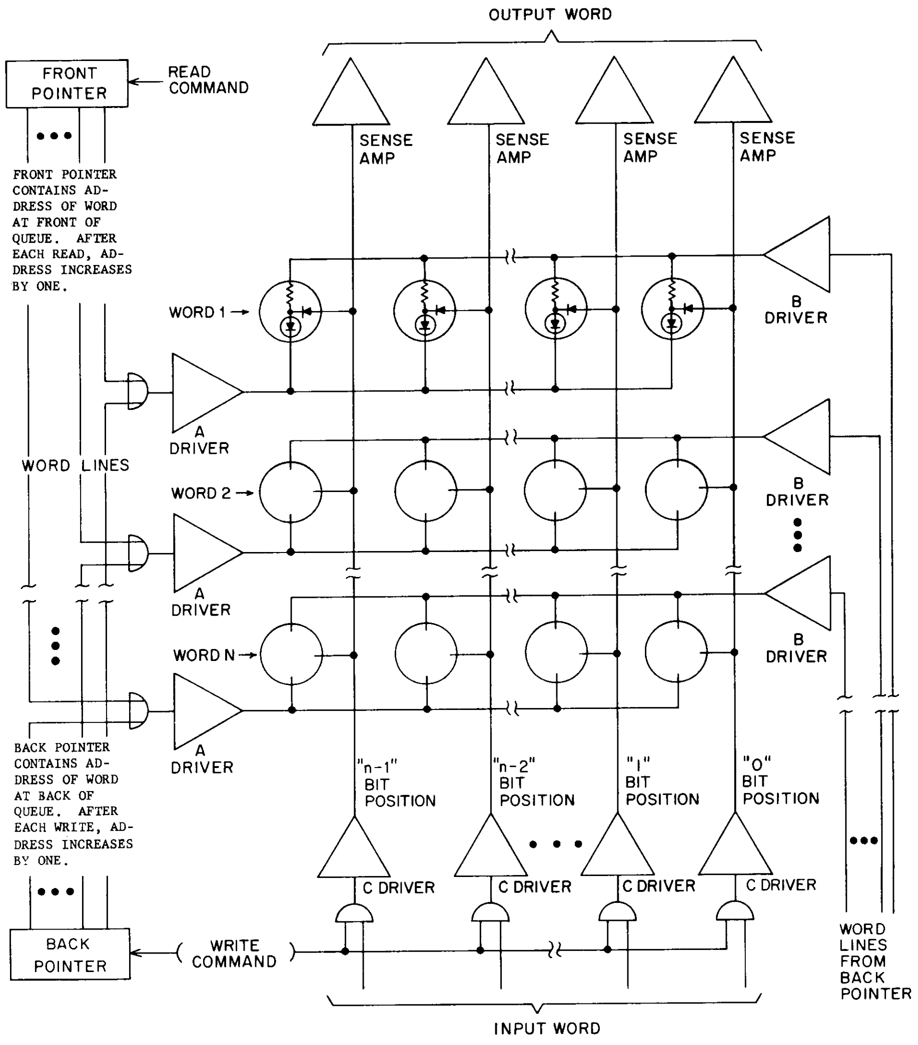
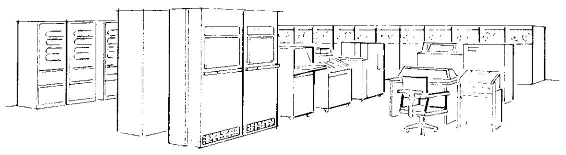


Figure 3-6. Queue Using Tunnel Diode Memory System



D850 Intermodule Communication System

Automatic high-speed interconnection control of the units of the D850 data processing system is provided. The system for implementing this interconnection control is hereafter referred to as the switch interlock. It can be considered the logical equivalent of a number of high-speed crossbar switches setting up in a single clock time and operating simultaneously to permit the transmission and control of a number of bits in parallel.

▲ Switch Interlock

The switch interlock is considered in two parts:

(a) the cross-point switch, which consists of the transfer gates which directly effect the parallel transfer of data from one module to another, and

(b) the bus allocator, which generates the control signals applied to the transfer gates.

There are two classes of data for which transfer channels are required. One is the information word, containing 49 bits plus parity plus tag, and the other is the address word containing 18 bits. A read-write indicator bit is transferred along with the address word.

Many transfer channels in the cross-point switch may be used simultaneously. However, in the event that a requestor unit seeks access to a "busy" memory unit or two or more requestor units seek simultaneous access to the same memory unit, interference will develop. It is the task of the bus allocator to preserve order in such cases.

The cross-point control logic is distributed among the memory, processor, and input-output control units together with the associated data transfer gates being controlled. The cross-point control consists of a flip-flop and associated gating for each of the transfer paths of the system.

Each unit of the system contains the transfer gates and associated control logic for accepting data from external sources. Each of the 15 main modular memories contains the gates and controls for accepting data (both address and information) from any of the possible 4 processors and 16 input-output control units. Each of the processors and input-output control units contains the gates and controls for accepting information words from any of the memory units.

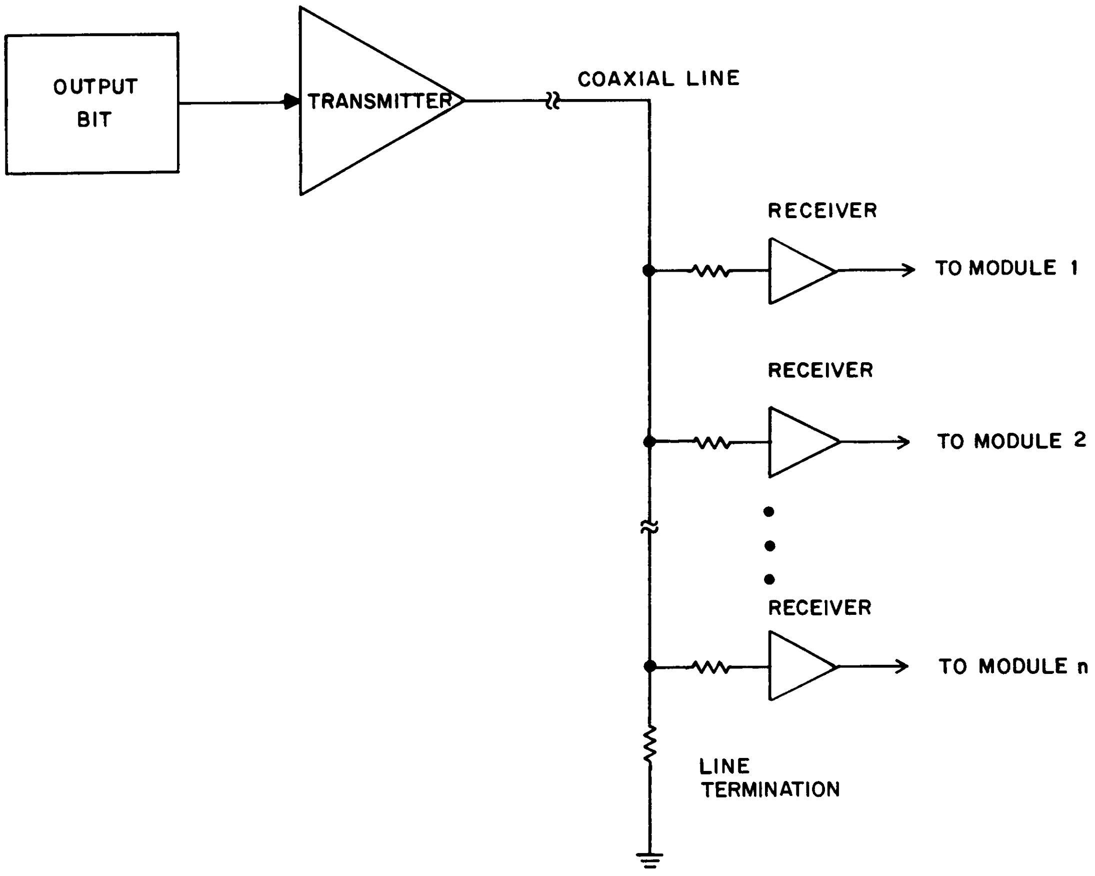
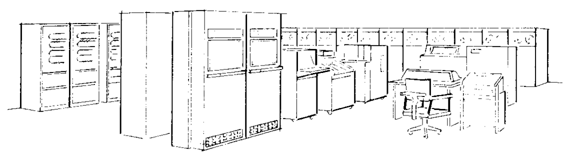


Figure 3-7. Transmitter-Line-Multiple Receiver Arrangement



The basic cross-point control logic is located within the memory units, where each of the main modular memories contains the cross-point flip-flops and associated gating. The cross-point logic within any memory unit allows any processor unit or input-output control unit to be interconnected with that memory unit.

Within each processor unit, there is control logic which operates in conjunction with the basic memory cross-point control logic. Each of the processors contains the gating required to allow any of the memories to transfer data back into the processor during read operations.

▲ Transmission and Reception

Transmission is always unidirectional. A bit of data from one module is processed to other modules by means of a transmitter which amplifies the signal representing the bit and sends it out over a transmission line to a multiplicity of receivers. The line is terminated at the last receiver on the line to minimize reflections. Each receiver presents a load which has a high impedance compared with the terminating impedance. Each receiving unit is thus effectively isolated from the line and no malfunction in a receiving module can effect the data path to other receiving units along this line. A receiver amplifies all signals above a given noise threshold and presents the data to the module associated with that receiver. The total delay is not more than (40 + 2L) nanoseconds, where L is the length in feet of the line between the transmitter and a receiver.

Thus, there is a transmitter and a terminated line associated with each bit of output data from each module, and a receiver associated with each bit of input data to each module. Figure 3-7 shows a diagram of a transmitter-line-multiple receiver complex for a single bit.

LOGIC DESIGN

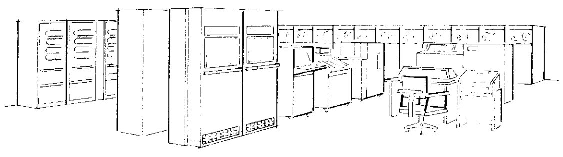
In general, the ultra-fast logic circuits used in the D850 Modular Data Processing System (delay through a "AND-OR" level is only 5 nanoseconds) are sufficiently fast to allow the use of straightforward logical techniques. Thus, for example, suppose that an 8-bit register is to be completely decoded and certain of the decoded outputs are to set flip flops. The decoding process is accomplished at minimum cost through two-level decoding using only 288 gates with 640 inputs. Time for decoding plus setting of flip flops requires only 20 nanoseconds - less than a single clock time.

In some areas, however, significant increases in processing speed can be obtained through the use of more sophisticated logical design. Such an area is the arithmetic unit, where the use of carry look-ahead has reduced time of addition by 70 percent - as compared to conventional logic. With carry look ahead, the arithmetic registers are divided into n-bit segments, and the output carry of the entire segment is determined by logical means. Thus, for a 40-bit operand, carry ripple time for only 40/n carries is required.

As the size of the carry look-ahead segment (n) increases, the number of components required in the adder increases proportionately - approximately as n^3. With the logical elements available, the optimum segment size in terms of cost and delay characteristics was found to be 8 bits. Independent adders and associated registers are provided for the exponent portion of data words for faster processing.

The full adder for the mantissa and exponent portions of the arithmetic word requires 751 decoders, mixers, and inverters, with a total of 1927 inputs; including the time to strobe information into the addend register, this is the equivalent in time of 12 logic levels, or 60 nanoseconds. Thus, two clock pulses are required for an addition, and one clock time for a shift. The arithmetic control logic is a part of the same mechanical assembly as the arithmetic unit, thus obviating delays (equivalent to several logic levels) which would be encountered if the commands were transferred in from another assembly. Transfer of new operand and operator words is accomplished at a 30-megacycle rate.

So that arithmetic decisions can be made at least as fast as the arithmetic unit is capable of executing them, the number of necessary control logic levels has been designed not to exceed the number of logic levels equivalent to one clock period. In some cases this necessitates the use of more complex arithmetic algorithms, but the additional cost is balanced by the increase in processing speed.



In addition to the adder, the arithmetic processor contains a multiplier/quotient register, a multiplicand/divisor register, an M/D counter, and associated peripheral equipment.

Multiplication

The multiplication algorithm employed utilizes both addition and subtraction of the multiplicand in the formation of partial products. Furthermore, single pulse time shifts of 2 and 3 positions are performed as required. The selection of these alternatives is based on an examination of the least-significant end of the multiplier.

For a 39-bit multiplier, a floating-point multiplication can be performed in 2.08 microseconds using this algorithm.

Depending upon the state of the least-significant end of tne multiplier register, the following alternatives are possible:

Alternative	Probability*	Clock Times
Shift 2	5/24	1
Shift 3	3/24	1
Add or subtract and shift 2	10/24	3
Add or subtract and shift 3	6/24	3

*First approximation; does not include end-effects.

With these four alternatives the average time required for each cycle is 2 1/3 clock times. Furthermore, there are (8n)/19 cycles, where n is the number of multiplier bits. Hence, total time for the execution of multiplication is (56n)/57 clock pulses. An additional 24 clock times are utilized for housekeeping in a floating-point multiplication. This includes exponent manipulation, normalization, and end-effects.

The expression for a floating-point multiplication (housekeeping included) is therefore:

$$(0.98n + 24)t$$

Using $n = 39$ and $t = 0.033$ microseconds yields a 2.08 microsecond average multiply time.

Division

The D850 divides using the standard nonrestoring division algorithm. Execution time is 4. 68 microseconds.

Square Root

A square root algorithm, similar in many respects to the nonrestoring method of binary division, has been devised for the D850. Execution time for the square root operation is the same as for division. The algorithm:

1. Subtract 0. 01 from the operand.

2. If the remainder is negative, the root bit is ZERO and an addition is indicated in step (4). If positive, the root bit is ONE and a subtraction is indicated in step (4).

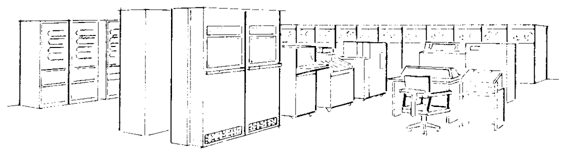
3. Shift the remainder left one bit.

4. Add the partial root (including the bit from step (2) with 11 appended to the remainder or subtract the partial root with 01 appended from the remainder. Return to step (2).

   The loop is terminated when the desired number of root bits have been generated.

The square root algorithm is illustrated below with an operand of 1/2. This root proves to be 0. 55 . . . in base 8.

Remainder 0	0. 1000000		
Subtract	1. 1011111		
Remainder 1	0. 0100000	1	
Shift Left	0. 1000000		
Subtract	1. 0101111		
Remainder 2	1. 1101111	0	
Shift Left	1. 1011111		
Add	0. 1011000		
Remainder 3	0. 0111000	1	
Shift Left	0. 1110000		Root
Subtract	1. 0101011		
Remainder 4	0. 0011100	1	
Shift Left	0. 0111000		
Subtract	1. 0100101		
Remainder 5	1. 1011101	0	
Shift Left	1. 0111011		
Add	0. 1011011		
Remainder 6	0. 0010111	1	



Note that the shift left one bit combined with the appendage of another bit to the partial root means that successive arithmetic operations are displaced by two bits. This is consistent with the generation of one bit of root for every two bits of operand. In division, the displacement of division and remainder consists of only one bit per operation.

The principle of the square root algorithm is similar to that for division. Paper and pencil methods call for a trial subtraction of $(2b\,R + x)x$ where R is the partial root, b the base of the number system, and $(0 \leq x < b)$. The largest x giving a positive remainder is to be found.

In binary arithmetic, x is either ZERO or ONE, and $b = 2$. This scheme provides for a subtraction of $(4R + 1)$ (this is the partial root with 01 appended). If the subtraction produces a positive remainder, the next root bit is ONE and the process continues. If, however, the subtraction "fails," a root bit of ZERO is indicated. As in the division procedure, there are now two alternatives: restore by adding $(4R + 1)$, shift the remainder left, the subtract $(8R + 1 + 4S + 1)$ which is the new root with 01 appended, or the following, which is equivalent and requires only one arithmetic operation: since

$$4(4R + 1) - (4S + 1) = 4S + 3,$$

$4S + 3$ is simply the new root with 11 appended. Hence, the two operations of add $(4R + 1)$ at bit location $2x$ and subtract $(4S + 1)$ at bit location $2x + 2$ is equivalent to the single operation, add $(4S + 3)$ at bit location $2x + 2$.

Special Algorithms

A unique feature of the D850 is the inclusion of provision for wired-in subroutines. Upon the detection of certain special operation codes, control of the arithmetic unit is transferred to a wired-in, read only memory. This memory contains the operators and constants necessary to perform the desired subroutine. The cycle time of this memory is comparable to the access time of the tunnel diode memories which form the queues and the stack; thus, no lengthy access to the thin film memory is required, and the time to perform the subroutine is essentially the sum of the times to execute the operations contained in the subroutine.

Three basic types of subroutines are available. The first operates upon the top of the stack, and employs the quantity there as the argument of a polynomial, continued fraction, etc; examples of this type are GAMMA(X), SIN(X), etc. The second type uses a fixed number of stack positions; examples of this type are MOD(X, Y) -- $X \equiv$ MOD(X, Y) modulo Y, or $P_2(X, A, B, C)$ -- $P_2(X, A, B, C) = AX^2 + BX + C$. The third type uses an arbitrary number of stack positions; in this type the top of the stack contains the number, n, of arguments to be used. An example of this type is the general polynomial evaluator, where the stack contains n, the degree of the polynomial, X, the argument, and C_n, C_{n-1}, ... C_0, the coefficients. Up to 32 subroutines of each type can be accommodated by the D850.

RELIABILITY DESIGN

The major reliability criterion for a continuously operating system is availability, which expresses the percentage of time that the system will be operating at full capability. The reliability parameters which affect availability are the mean-time-between-failures (MTBF) and average recovery time (ART) which should be maximized and minimized, respectively, to obtain maximum availability. Availability is conventionally expressed as:

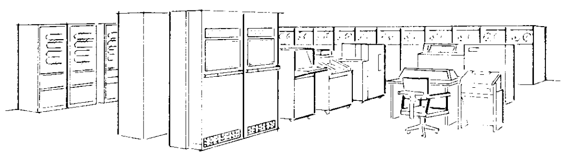
$$\text{Availabiltiy} = \frac{\text{MTBF}}{\text{MTBF + ART}}$$

By applying conservative part failure rates to the parts count, the MTBF for each module type comprising the electronic system elements has been estimated as follows for each module:

Module	MTBF (hours)
Processor	58
Main Memory	71
Input/Output Control	254

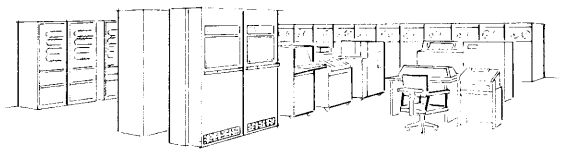
The various input-output devices, such as the magnetic tape units, card reader, card punch, and high-speed printer have not been included in the above estimates. This equipment is mostly mechanical and electro-mechanical, and is not subject to the catastrophic failure phenomena experienced by electronic equipment. The major failure mode of this equipment is wear-out, and highly reliable operation can be obtained through implementation of a sound preventive maintenance program.

The average recovery time for the system is estimated as 0. 5 hour, and the distribution of recovery times is generally exponential for this type of equipment.



A typical system containing 1 computer module, 4 memory modules, and 4 I/O modules would have an estimated availability, or percentage of up time, for all modules operating simultaneously of 0.956. The actual availability of the system to function usefully is considerably higher since satisfactory operation can be obtained, at some loss in speed, with one or more memory and/or I/O modules inoperative.

IV D 850 Programming



IV D850 Programming

For the man-machine-man communicative relationship, ALGOL 60 has been selected as the most flexible, expressive, and efficient of various advanced programming languages (FORTRAN, MAD, Math-matic, etc.) that provide algebraic statement forms for algorithmic solution. An introductory description of ALGOL 60 is furnished in Attachment 1.

- ALGOL is independent of any particular computer so that programs written in this language are available to all users regardless of the machine used.
- Because ALGOL is so closely akin to the problem formulation language of the programmer, time is minimized for formulation of solutions.
- ALGOL incorporates all means for expressing the solutions to problems of an algebraic nature.

In the D850 design, Burroughs has drawn upon its extensive past experience in ALGOL programming to provide a wealth of new design features in the D850, such that the system accomplishes automatically and directly many features of ALGOL 60 that can be accomplished only through complex program administration on other computers. Such features as automatic segmentation of large programs, recursive subroutines, compact program representation, efficient subroutine linkage, and program invariance with respect to location are all handled automatically by the D850. Additionally, because of the large number of programs which have already been written in FORTRAN language, provisions will be made in the compiler to accept these programs.

Since the logical design of the D850 permits Polish parenthesis-free notation, this not only provides for efficient compilation of ALGOL statements, but will also permit efficient compilation of other algebraic languages which may evolve. For this reason, it is not likely that D850 will be outmoded by new software developments.

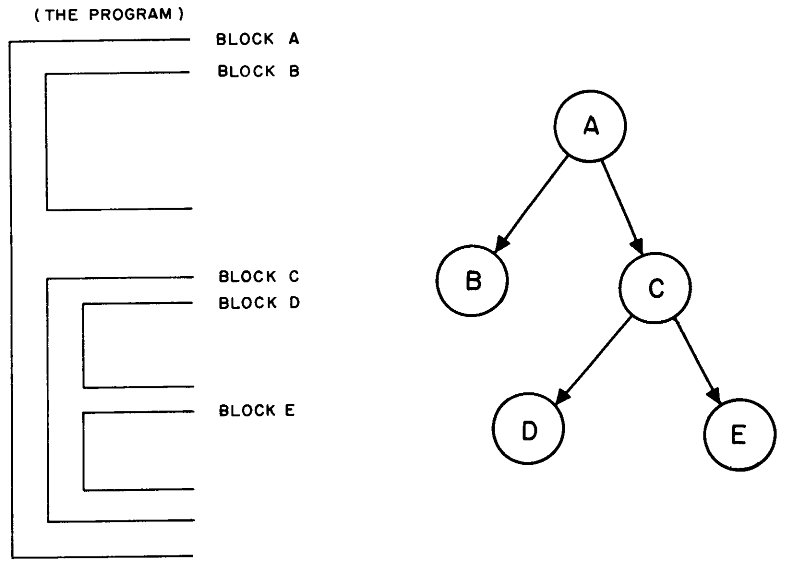


Figure 4-1. Program Structure

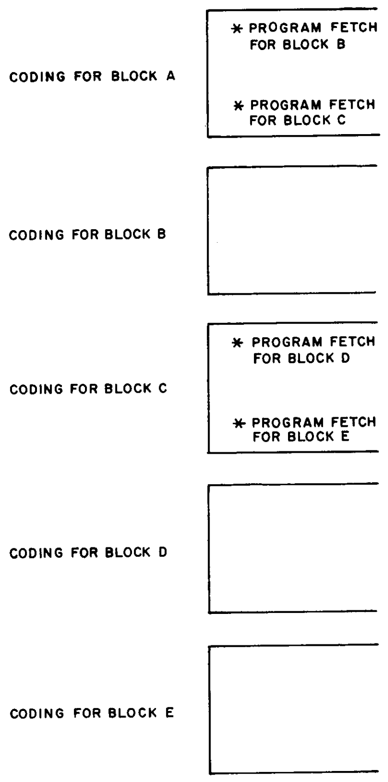
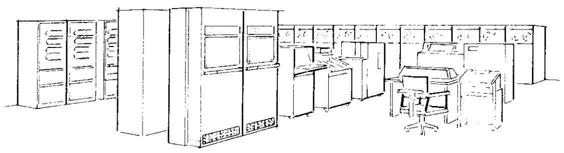

CODING FOR BLOCK A
* PROGRAM FETCH FOR BLOCK B
* PROGRAM FETCH FOR BLOCK C
CODING FOR BLOCK B
CODING FOR BLOCK C
* PROGRAM FETCH FOR BLOCK D
* PROGRAM FETCH FOR BLOCK E
CODING FOR BLOCK D
CODING FOR BLOCK E

Figure 4-2. Program Restructured



PROGRAM STRUCTURE

ALGOL 60 features a block structure in its program as explained in Section 6 of Attachment 1. A block is a computational unit consisting of a compound statement and data, or subroutines, that have meaning only within that block. Such data and code are called local to the block. The data are declared (typed and named) at the head of a block to provide information about the objects of computation to a compiler. The usual disposition of data declarations by a compiler is to reserve storage for the object named. The compiler emits code for statements referring to the data declared, using the addresses of the storage reserved.

Three fundamental rules of ALGOL blocks are:

1. A block may be entered only at its beginning (i. e. entry to the block can not be made at an arbitrary point).

2. Blocks may contain other blocks nested to an arbitrary depth.

3. A subroutine is a block.

Blocks provide a means for making automatic segmentation by a compiler possible. Since entrance to a block can only be through its beginning, it suffices to obtain code and assign space for the block when required in the execution of the program. Similarly, when a block is completed, the storage for that part of the program and its local data are no longer needed, and can be returned to other use. The total amount of storage that must be allocated to a program is that amount sufficient to contain the coding and data for the deepest nest of blocks.

A sketch of a program containing several levels of nested blocks is shown along with its tree arrangment in Figure 4-1. This same program may be restructured by the compiler into the arrangement of Figure 4-2.

The compiler converts the declarations for each block to address or control operators to reserve as much storage as necessary for the local variables and data associated with that block. Identifiers encountered in the program are converted to address syllables having a value corresponding to the position in which the identifier appeared in the declarations for the block.

As included blocks are removed from the coding block nest, the compiler will generate control and address operators to record the position in the current coding block, obtain the next block from the core storage, unload the local variables and data to the core storage, and transfer control to the beginning of the new block. Rule 1 associated with blocks assures that no transfer will be out of the particular chain of branches in a program tree, so that it

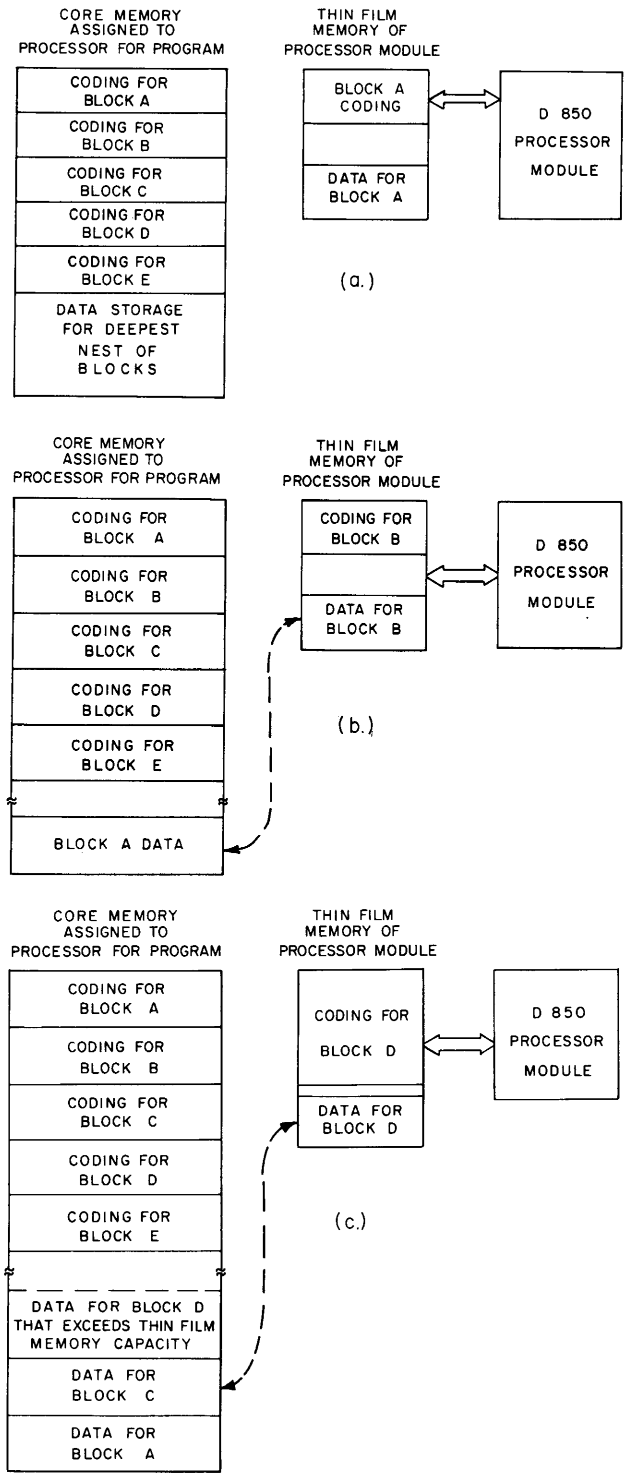


Figure 4-3. Program Execution Sequence

suffices to have at hand only enough storage to hold the largest block and some indication of the path through the program tree that led to the current block. If the path is represented by recorded return points, the total number of cells necessary to hold this information is known at compile time, and can be recorded at the beginning of the close-coupled fast storage of a processor.

Whenever a release of program storage occurs (by transferring to a higher level block), it is apparent that the local data associated with that block is no longer needed and may be released at the same time. This is possible because the dynamic establishment of storage, corresponding to declared variable and arrays each time a block is entered ensures that no data will be lost. If two or more blocks are to operate on the same data, then that data must be declared in the outermost block in which reference to the data takes place.

In the D850, the active program block and its local data are stored in the close-coupled fast store. As much of the balance of the program as is feasible to have inside the machine is stored in the main module memory. When a new program block is needed, it is transferred to the close-coupled store while the data associated with the previous block is transferred from thin film memory to the area in the main modular memory reserved by the compiler for this purpose. Figure 4-3 pictures the program execution of Figure 4-1.

In Figure 4-3(a), the first block of the program is in the close-coupled thin film storage, as is its local data. The core area assigned to the processor contains the balance of the program and the area set aside for data storage.

Figure 4-3(b) shows the processor executing the coding in block B. The data for block A has been transferred to the main modular memory locations indicated to make room for the local data of block B. The processor still has access to this data. Data handling in this particular way is based on a hypothesis that data local to each block is referenced with high frequency, thereby justifying its retention in the close-coupled thin film store.

Figure 4-3(c) illustrates the memory layout when the local data for a block, D in this case, exceeds the capacity of the thin film storage for a processor. Part of the data is kept in the close-coupled fast store, while the balance is kept in the main modular memory. The relative addressing facility of the address computer allows fetching of data from the appropriate memory to proceed in an automatic manner. The situation of Figure 4-3(c) could also arise from the execution of a recursive subroutine. The transfer of fixed data from thin film to core memory, and adjustment of the relative addressing markers occurs dynamically, making it unnecessary to know precisely the depth of recursion. The exhaustion of all storage assigned is detected in the D850 processor and creates an interrupt condition leading into the control program which can then assign more storage.

WORD FORMATS

The basic memory word of the D850 contains 50 bits; however, its internal structure may vary depending upon the use of the word. All words in core and thin film memories carry a parity bit and a data tag bit; the data tag bit signifies the initial and basic distinction between word types, which further distinguishes data words from program words.

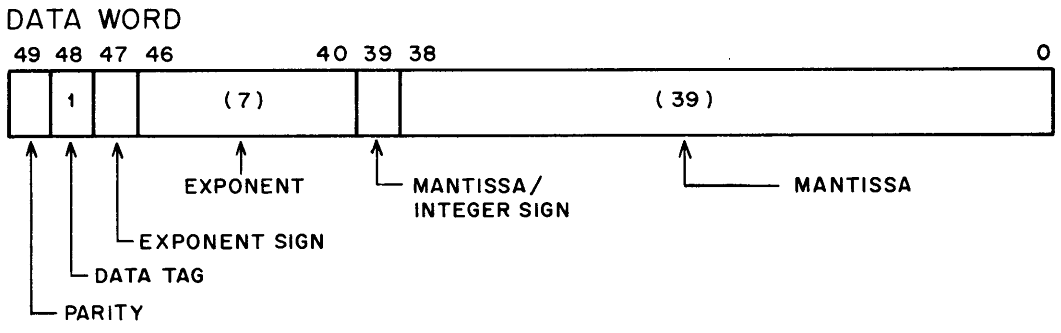
Data Words

Data words are of two types -- floating point words and integer words. The mantissa of a floating point word and the integer word are each 39 bits. Any integer word less than 8^{13} (approximately 5.5×10^{11}) can be accommodated.

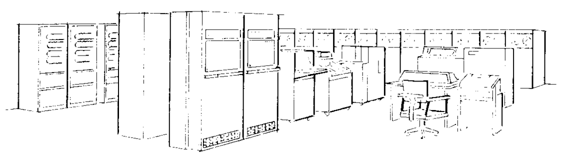
The floating point representation uses an octal-integer format of the form:

$$(\text{integer mantissa}) \times 8^{(\text{exponent})}$$

Full significance floating point values may range from 8^{-115} to 8^{141}; the approximate decimal equivalent range is 10^{-104} to 10^{127}.



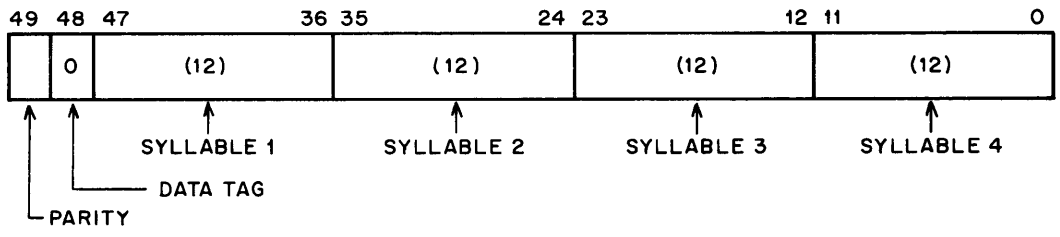
Integer words are identical to floating point words but have a zero exponent. The radix point in all data words is at the far right.



Program Words

Program words contain the program string and are identified by a ZERO in the data tag bit; they are composed of 12-bit syllables, which in the case of addresses are concatenated to form 24-bit address words.

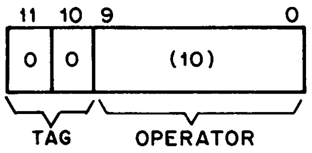
PROGRAM WORD



Depending upon the 2-bit tags in the syllable, the syllable is accepted as one of four types:

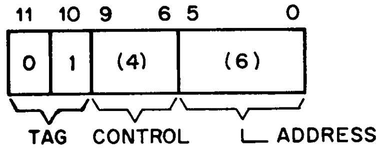
Syllable	Tag
Data Operator	00
Address Operator	01
Address	10
Control Word	11

DATA OPERATOR



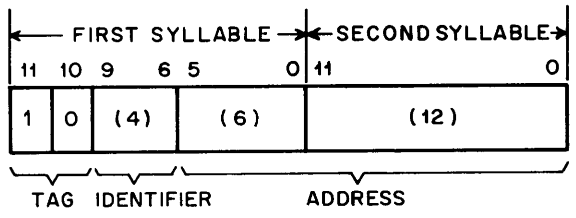
The data operator consists of the "00" tag and 10 bits of operation code; these operators indicate the operation to be performed upon the data in the data queue and/or stack.

ADDRESS OPERATOR



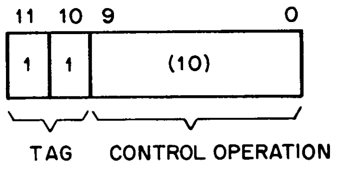
The address operator consists of the "01" tag, a 4-bit address operation code (control), and a 6-bit address. The address operation code specifies such an action to be performed with one of the hard registers (index registers, top of stack, etc.) and the address identifies the hard register.

ADDRESS

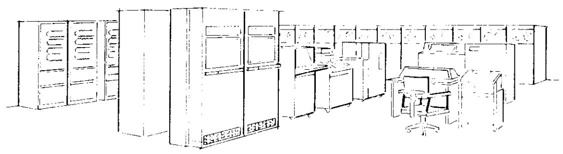


Upon identification of the address syllable, this syllable and the next syllable are taken together as a 24-bit word. The word consists of the 2-bit tag, 4 bits of identifier, and 18 bits of address. The identifier specifies whether the 18-bit address is indexed, relativized, indirect, etc. by a "10" tag.

CONTROL WORD



The control word syllable consists of the "11" tag and 10 bits defining a control operation. This syllable in general describes the way in which the memory addressing sequence is to be disrupted for a transfer of control, subroutine entrance, or store operations and is used to initiate memory block transfers and peripheral operations.



ORDER CODE

The D850 instruction repertoire comprises four broad groupings: data operators, address operators, stack operators, and control operators.

Data Operators

The data operators control the processing of data in the arithmetic unit; each has four basic variants.

Variant	Form	Action
00	OP	The operator acts upon the top of stack and the top of the data queue, replacing the top of the stack with the result; if it is a uniary operator, it operates on the top of the data queue.
01	OPs	The operator acts upon the top two positions of the stack replacing the top of the stack with the result; if it is a uniary operator, it acts on the top of the stack.
10	OPh	Same as variant 00; but the top of the stack is saved, and the result placed on top of it.
11	OPsh	Same as variant 01; but the second item in the stack is saved, and the result placed on top of it.

The data operation descriptions have the form:

Arithmetic Operators

Operation	Mnemonic	Execution Time (μsec)	Description
Add	ADD	0.12	Form the algebraic sum of the two operands.
Add	ADD	0.12	Form the algebraic sum of the two operands.
Subtract	SUB	0.12	Form the difference of the two operands.
Reverse Subtract	RSUB	0.12	Interchange the operands and perform SUB.
Multiply	MULT	2.08	Form the double precision product of the two operands, the less significant half of the product is destroyed by the next operation unless specifically referenced.
Divide	DIV	4.68	Form the quotient of the two operands; the remainder is destroyed by the next operand unless specifically referenced.

Arithmetic Operators (Continued)

Operation	Mnemonic	Execution Time (μsec)	Description
Reverse Divide	RDIV	4.68	Interchange the operands and perform DIV.
Square Root	SQRT	4.68	Find the square root of the absolute value of the operand; if the operand is negative set the Boolean indicator.
Absolute Value	ABS	0.12	Form the absolute value of the operand.
Round	RND	0.12	Round the operand according to the contents of the least significant half of the double-length accumulator.
Add Exponents	ADX	0.12	Algebraically add only the exponent fields of the operands.
Fix	FIX	(E+5)X0.033	Where E is the absolute value of the exponent; convert the operand to integer form; the least significant digits, if shifted off, are retained in the least significant half of the double-length accumulator.
Float	FLT	(Z+5)X0.033	Where Z is the number of zeros at the least significant end of the operand; shift the operand right until the first nonzero digit is reached, appropriately adjusting the exponent field.
Change Sign	CHS	0.067	Change the sign of the operand.
Set Sign Plus	SSP	0.067	Set the sign of the operand to "plus".
Set Sign Minus	SSM	0.067	Set the sign of the operand to "minus".

Note: Any arithmetic operation which causes overflow or underflow of exponent or mantissa sets the Boolean indicator.

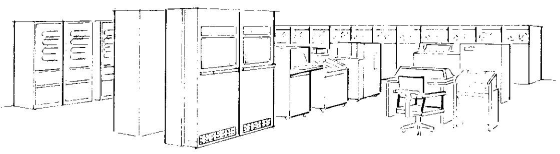
Relational Operators

Relation	Mnemonic	Execution Time (μsec)	Condition	Action
Greater Than	GTR	0.12	If $O_1 > O_2$	Place "1" in the least significant bit of the top of the stack. Otherwise clear the top of the stack.
Greater Than or Equal	GEQ		$\geq$	
Equal	EQL		$=$	
Not Equal	NEQ		$\neq$	
Less Than Equal	LEQ		$\leq$	
Less Than	LSS		$<$	

Logical Operators

Operation	Mnemonic	Execution Time (μsec)	Function
And	AND	0.067	Logical conjunction.
Or	OR		Logical disjunction.
Implies	IMP		Logical material implication.
Equivalent	EQV		Logical equivalence.
Negate	NOT		Logical negation.
Load Boolean Indicator	IF		If the operand is nonzero in the logical sense, set the Boolean Indicator.

Note: Logical shifts may be performed using the RLSH, ADEX, and FIX instructions. Since these operations are in general used during compilation and other data processing operations which are usually input limited, no great penalty is incurred. For instance,to unpack bits 35 through 30 of a word, the string RLSH; (Literal)-12; ADEX; FIX; PUS; (Literal)-6; ADEX; FIX; would suffice at a cost of two program words and two literals.



In addition to the standard arithmetic operations, three types of wired-in function evaluators are optionally provided. The first type, such as SINX, accepts a single argument and provides a functional value; the second type accepts a fixed number of arguments and provides a functional value; the third type accepts a parameter n and n arguments, and provides a functional value. Up to 32 functions of each type may be implemented.

The form of the operation is F_0, F_1, or F_2, corresponding to the above types. Each type operates upon the stack; all arguments and parameters must be in the stack when the operation begins.

As function evaluators are added to a system, the compiler must be regenerated to recognize these as intrinsic functions.

Typical functions of each type are:

F_0	:	SINX, COSX, E^X, LOGX
F_1	:	MOD (X, Y), A^B
F_2	:	MAX ($n, X_1, X_2, \ldots . X_n$)
		P_n ($n, X, C_0, C_1, \ldots C_n$) - the nth degree polynomial with argument X and coefficients $C_0 \ldots C_n$
		K_N ($n, X, C_0, C_1, \ldots C_n$) - the nth degree continued fraction with argument X and coefficients $C_0 \ldots C_n$ etc.

Note: There is an implied degree, n, in the F_0 and F_1 forms, the degree of the approximation used to evaluate these transcendental functions. Thus, a series of approximations may be included for a single function, each used to obtain a given precision at a saving in time when a lesser precision is acceptable.

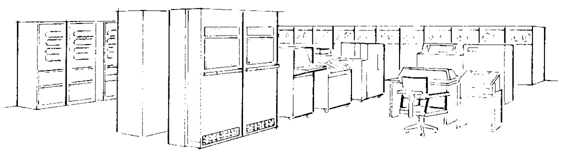
Address Operators

The address operators are processed in the address computer and are used performing all address arithmetic. They may address any register in the address computer plus the top of the stack (TOS). In particular, if the address register (AR) is addressed, the address operand finally obtained will depend upon the control bits in the word in the AR; e.g., if the AR word is a literal, it will be used directly, but if it is an indirect address, then the operand will be the least significant half of the word so referenced. Operation times quoted assume that the true operand is available in the AR register and give the time required to place the final result in the specified register.

Operation	Mnemonic	Execution Time (μsec)	Action
Load Register	LDR	0.067	Place the word in the AR in the register specified.
Load Address Accumulator	LDA	0.067	Place the word in the register specified in the address accumulator.
Store Address Accumulator	STA	0.067	Place the word in the address accumulator in the register specified.
Add to Register	ADR	0.100	Add the contents of the AR to the register specified.
Add to Address Accumulator	ADA	0.100	Add the contents of the register specified to the address accumulator.
Test for Equality	TEQ	0.100	If the word in the register specified equals the the word in the AR, set the address Boolean indicator (ABI).
Test for Inequality	TNE	0.100	If the word in the register specified is not equal to the word in the AR, set the ABI.
Test for Zero	TZE	0.067	If the word in the register specified is equal to ZERO, set the ABI.
Test for Nonzero	TNZ	0.067	If the word in the register specified is not equal to ZERO, set the ABI.
Increment by One	INC	0.100	Add ONE to the contents of the register specified.
Decrement by One	DEC	0.100	Subtract ONE from the contents of the register specified.
Set Register Flag	SRF	0.067	Set the flag of the register specified; this will cause any address requiring indexing to be modified by this register; only one register may be flagged at any given time.
Reset Register Flag	RRF	0.067	Reset the flag of the register specified.

For address control the four control bits of the address syllable specify the following:

Bit 6	Indirect Address
Bit 7	Relative
Bit 8	Index
Bit 9	Literal



Stack Operators

Stack operators are housekeeping functions which control the state of the stack in the D850 processor. They are supplemental to the stack control implied in each data operator.

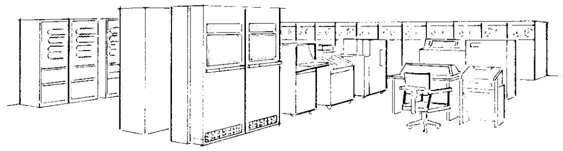
Operation	Mnemonic	Exection Time (μsec)	Action
Transfer Least Significant Half to Stack	LSH	0.067	Place the least significant half of the double-length accumulator on the top of the stack.
Reverse Transfer Least Significant Half to Stack	RLSH	0.067	Place the top of the stack in the least significant half of the double-length accumulator.
Enter in Stack	ENT	0.067	Place the operand on top of the stack.
Push Down Stack	PDS	0.067	Push down the contents of the stack so that the top position is vacant.
Push Up Stack	PUS	0.067	Clear the top of the stack. (TOS-1 becomes TOS.)

Control Operators

Control operators, processed in the instruction preprocessor, cause an interruption in the sequential addressing of the memory. All addresses are assumed to be direct references to thin film memory. Simple address modifications require an additional 0.100 μsec. In any branch type control word, the least significant two bits of the word indicate the syllable within the word which is to assume control of the program.

Operation	Mnemonic	Execution Time (μsec)	Action
Branch Unconditionally	BUN	0.517	Transfer control to the address specified.
Branch on False	BOF	0.517	If the Boolean indicator is not set, transfer control to the address specified.
Branch on True	BOT	0.517	If the Boolean indicator is set, transfer control to the address specified.
Address Branch on False	ABF	0.517	If the address Boolean indicator is not set, transfer control to the address specified.
Address Branch on True	ABT	0.517	If the address Boolean indicator is set, transfer control to the address specified.
Enter Close & Subroutine	CSE	4.0	Store the appropriate registers, transfer control to the address specified, establish new values for the appropriate registers; save the present program counter in the subroutine stack.
Return From Closed Subroutine	CRE	4.0	Transfer control to the address at the top of the subroutine stack, reload the appropriate registers.
Change Program Block	CPB	4.0	Store or clear appropriate registers and prepare for a change of program blocks.
Store	STO	0.517	Store the top of the stack in the specified address.
Data Transfer	DTF	1.0	Transmit the following address to the I/O control module specified. The cell addressed contains a descriptor identifying the source and destination of the data transfer as well as the block length. Typical transfers: Close-coupled fast store (thin film) → main memory (core); Main modular memory → main modular memory; Main modular memory → magnetic tape; Card reader (via B280 satellite computer) → main modular memory card reader

V D850 Sample Problems



V D850 Sample Problems

In order to demonstrate the capability of the D850 Modular Data Processing System, two sample problems are presented as programmed, tested, and timed: "The Monte Carlo Calculation of Molecular Flow Rates through a Cylindrical Elbow and Pipes of Other Shapes" as described by D. H. Davis in the Journal of Applied Physics, Vol. 31, No. 7, pp. 1169-1176, July 1960, and a "Hydro" problem which is typical of a large class of scientific data processing.

The problems were programmed in the language of the Burrough Algebraic Compiler, a representation of ALGOL 58. Although the actual compiler for the D850 system will be considerably more efficient than this, it was felt that this version of ALGOL 58 is sufficiently representative of the class of formal programming languages proposed that it could be used to obtain reasonably accurate (though conservative) figures for program volume and speed. The programs were checked using a Burroughs 220 data processor and were found to give results in excellent agreement with those published and provided.

The ALGOL statements of the problems were then hand-compiled into the Polish program strings of the D850 system, and representative sections of the Hydro problem program strings were accurately timed in terms of D850 computation speeds. These precise times were used to estimate the total running time for the Hydro problem on the D850.

As a check, those sections which were accurately timed were isolated from the Burroughs 220 program and were run independently in a series of time trials. Using the results of the Burroughs 220 time trials and the calculated D850 running times, a "speed-up" factor was determined. The speed-up factor was applied to the actual Burroughs 220 running time to obtain a second estimate of D850 running time. Having assurance that the estimating method was reasonably accurate, the speed-up factor was then applied to the actual Burroughs 220 running time for the

Monte Carlo problem. The results of the estimates and calculations are given below. It can be seen that even with a single computer module, the D850 far exceeds any existing or presently announced data processing system in computing speed.

TABLE 5-1. D850 TEST PROBLEM RUNNING TIMES

Calculated D850 running time for Section 12 of Hydro problem	66.490 μsec.	
D850 running time for 1 cycle of complete Hydro problem based on extended Section 12 running time	1238.4 μsec.	
D850 running time for 1,000,000 cycles of complete Hydro problem based on extended Section 12 running time		20.64 min/10^6 cycles
Burroughs 220 running time for 1000 cycles of Section 12 of Hydro problem	56.73 sec	
Speed-up factor = 56.73/0.0665	853	
Burroughs 220 running time for 1000 cycles of complete Hydro problem	1213 sec	
D850 running time for 1000 cycles of complete Hydro problem, based on speed-up factor	1434 sec	
D850 running time for 1,000,000 cycles of complete Hydro problem, based on speed-up factor		23.93 min/10^6 cycles
Burroughs 220 running time for 10,000 cycles of Monte Carlo problem	9288 sec	
D850 running time for 10,000 cycles of Monte Carlo problem, based on speed-up factor	10.80 sec	
D850 running time for 1,000,000 cycles of Monte Carlo problem, based on speed-up factor		18.00 min/10^6 cycles

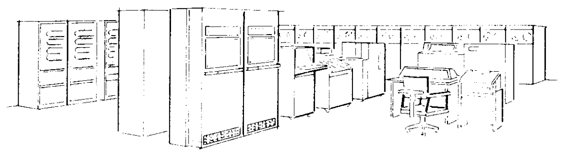


TABLE 5-2

ALGOL STATEMENT OF SECTION 12 OF HYDRO TEST PROBLEM

```
                SECTION 12$


          GPK3(X) = LZCK2(X)+LZCK3(X)+LZCK0(X)+LZCK4(X)$


          IF TEST18 EQL ZERO$
          GO OUT$


          LJA = LJK3(X).LJDN/(LJK3(X) +LJDN)$




          LJC = LJK4(X).LJUP/(LJK4(X) + LJUP)$
          LJB = LJK4(X).LJDN/(LJK4(X) + LJDN)$
          LJD = LJK3(X).LJUP/(LJK3(X) + LJUP)$


          IF TEST19 EQL ZERO$
          GO OUT$


          RH8 = RCK4(X)/8$
          KL2K3(X) = RH8.(LJA.(KLK4(X) + 2.KLK3(X) + KLK2(X))
             + LJC.(KLK4(X) + 2.KLK0(X) + KLK2(X)))$
          KL4K3(X) = RH8.(LJB.(KLK0(X) + 2.KLK4(X) + KLK3(X))
             +LJD.(KLK0(X) + 2.KLK2(X) + KLK3(X)))$


          END$
           STOP$
OUT..
                                STOP  111111$

                            FINISH$
COMPILED PROGRAM ENDS AT 0334
NEXT AVAILABLE CELL IS 3676
```

PROGRAMMING AND ESTIMATING TECHNIQUES

Presented in Table 5-2 are the ALGOL statements corresponding to Section 12 of the Hydro problem. They are written in the language of the Burroughs Algebraic Compiler (BAC), a formal language which is closely patterned after ALGOL 58.

The complete definition of the Burroughs Algebraic Computer System utilized with the Burroughs 220 data processing system is given in Reference 1. The sample problems in Appendices A-1 and B-1 are written in this system. This algebraic language is a hardware version of the international ALGOL 58 reference language defined in Reference 2. The primary operators available in this system are:

- Arithmetic

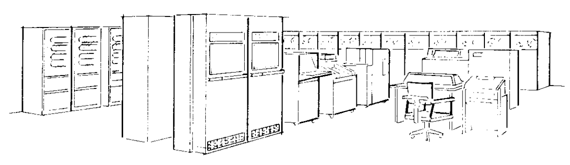
+	... addition
-	... subtraction
.	... multiplication
/	... division
*	... exponentiation

- Relational

GTR	... greater than
GEQ	... greater than or equal to
EQL	... equal to
LEQ	... less than or equal to
LSS	... less than
NEQ	... not equal to

- Boolean

NOT	... negative
AND	... conjunction
OR	... disjunction
IMPL	... implication
EQIV	... equivalence



DESCRIPTION OF THE HYDRO TEST PROBLEM

The Hydro problem consists of 13 blocks of equations, which are evaluated in order. The specific order in which the equations in each block are evaluated was altered to increase program efficiency; however, this did not destroy the intended computational flow.

All conditional branches not specifically defined by a test are treated as testing a sentinel for zero. A branch to an indicator stop occurs if a sentinel is zero. Otherwise, the program proceedes to the next line. All sentinels are set so that the prescribed flow will occurr.

Appendix A contains the ALGOL statement of the Hydro test problem, the corresponding D850 machine language program, and a tabulation of the results of the program as run on the Burroughs 220 data processing system.

The given input was used in computer runs, and the output was printed in a tabular format corresponding to that of the provided output. The floating-point values of the output have been printed for comparison with those provided.

In addition, a sequential printout of all variables as obtained from the ALGOL monitor is given. The names of the variables, along with their current values are printed in the order in which they are computed in a given block.

DESCRIPTION OF THE MONTE CARLO TEST PROBLEM

In the early 1930's, H. A. Lorentz and P. Clausing developed a set of formulas for determining the molecular flow rates through pipes utilized in near-vacuum systems, a problem of classical kinetic theory. The difficulty encountered in the practical application of their analysis is that it required the evaluation of complicated integrals. In order to avoid this problem, D. H. Davis developed a new method of solution which is amenable to calculation on an electronic digital computer. This method is a standard application of the Monte Carlo technique. It consists of generating a sequence of molecular histories from a set of random numbers utilizing the cosine law; this sequence of histories is then used to obtain an estimate of the flow probability. Figure 5-1 depicts these probabilities and their standard deviations as functions of the number of molecular histories as computed on the Burroughs 220 data processing system. The ALGOL program for the Monte Carlo problem, the D850 machine language program, and results as computed on the Burroughs 220 data processing system may be found in Appendix B.

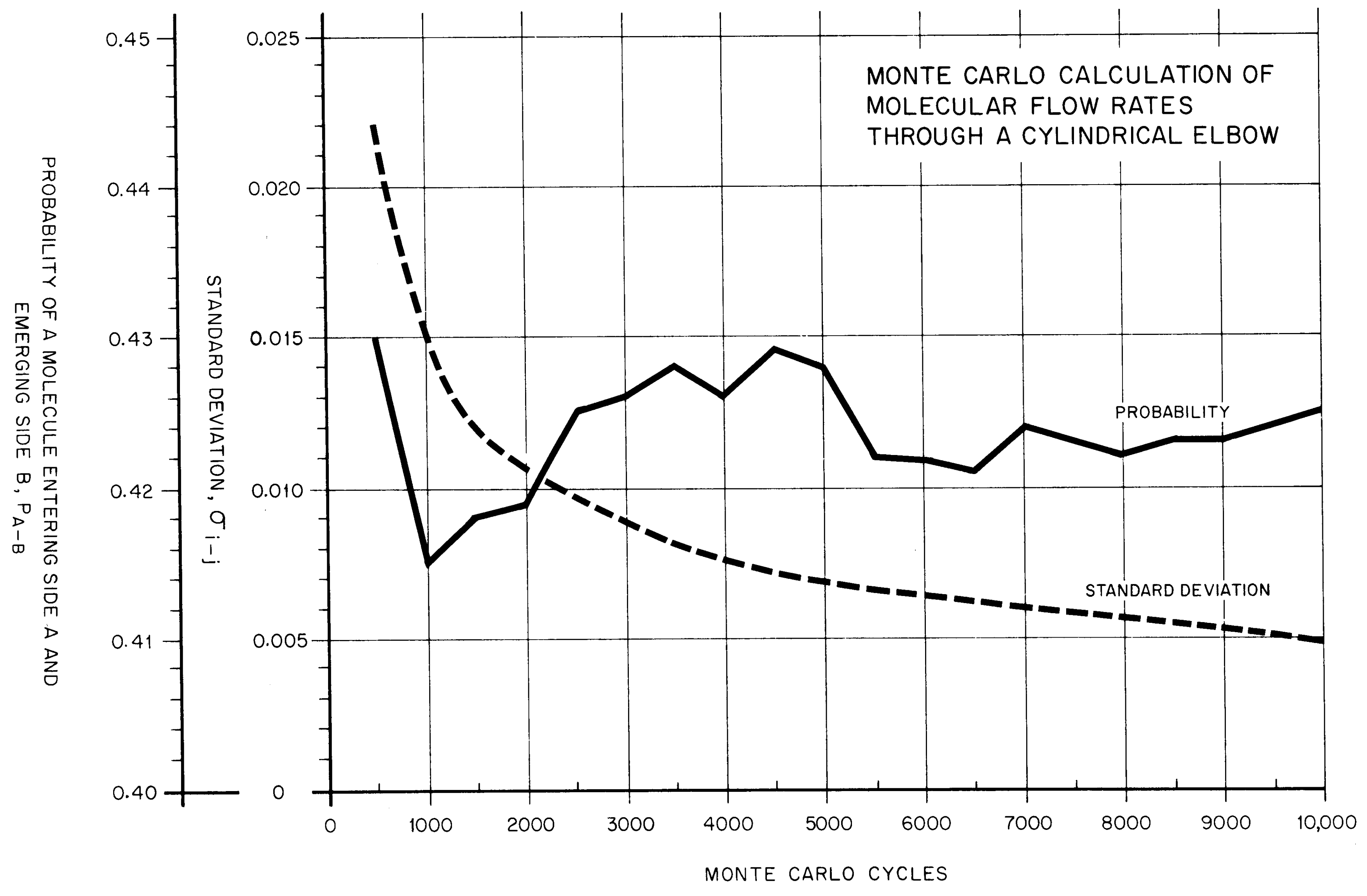
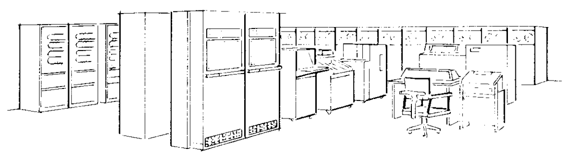


FIGURE 5-1. PLOT OF MONTE CARLO RESULTS



These operators in conjunction with the standard elementary procedures SIN, COS, TAN, SQRT, etc., may be combined in a generalized fashion to form a simple dynamic expression of an algorimthic process which is very close to the traditional mathematical nomenclature employed in the literature. The three central elements of expression are:

> The Arithmetic Statement e. g. Y=SQRT (A+B) for expressing standard relationships.
>
> The IF Clause e. g. IF (X CTR Y) AND (U LSS W) for expressing logical conditions.
>
> The FOR Clause e. g. FOR N = 1, 1, K for expressing iterative and sequential conditions.

The ALGOL language to be utilized in the D850 will be an improved version of the language described above. It will be patterned after ALGOL 60, the latest version of the international language which is fully described in Reference 3, and in Attachment 1 to this proposal.

Table 5-3 is a symbolic representation of the Polish program string corresponding to the ALGOL problem statement as it would appear in the memory of the D850. Operators are identified by an "O" in the first column of the syllable, literals by "L;" control words by "C," address operators by "AO," and addresses by "A;" when it is necessary to split an address between two words, the two halves are identified by "A1" and "A2." The remainder of each syllable contains a symbolic representation of the information. Operators and their variants are identified as described in Section 4; addresses are the symbolic addresses provided in the statement of the problem, with exceptions noted below; literals and other pure numbers are given in decimal form for convenience of reading - in particular, the form "X** -3" indicates a literal with "-3" in the exponent field and a zero integer part. Addresses followed by a lower case "i" indicate those which are to be indexed by the index register currently flagged. "TEMP" indicates a temporary storage location; addresses such as "ONE" refer to the constant 1; addresses such as "EXP2" refer to constants with 2 in the exponent field; the parenthesized numbers in branch addresses refer to the syllable within the word referenced.

The last two words of the program exemplify the manner in which a program is terminated. A program-identifying literal is placed in the stack, and control is transferred to the automatic operating and scheduling program (AOSP); by examining the identifier, the AOSP can then properly schedule the next program to be run.

TABLE 5-3

D 850 MACHINE LANGUAGE PROGRAM FOR SECTION 12 OF HYDRO PROBLEM

BITS 0 12 24 36 47

LOCATION		SYLLABLE 1		SYLLABLE 2		SYLLABLE 3		SYLLABLE 4	REMARKS
WD 0	A	LZCKO i			A	LZCK4i			
WD 1	O	ADD	A	LZCK3i			O	ADD	
WD 2	A	LZCK2i			O	ADD	O	STO	
WD 3	A	GPK3i			A	TEST18			
WD 4	O	IF	C	BOF	A	OUT			
WD 5	A	LJK3i			A	LJDN			
WD 6	O	MULT	A	LJK3i			A1	LJDN	
WD 7	A2	LJDN	O	ADD	O	RDIV-s	O	STO	
WD 8	A	LJA			A	LJK4i			
WD 9	A	LJUP			O	MULT	A1	LJK4i	
WD10	A2	LJK4i	A	LJUP			O	ADD	
WD 11	O	RDIV-s	O	STO	A	LJC			
WD 12	A	LJK4i			A	LJDN			
WD 13	O	MULT	A	LJK4i			A1	LJDN	
WD 14	A2	LJDN	O	ADD	O	RDIV-S	O	STO	
WD 15	A	LJB			A	LJK3i			
WD 16	A	LJUP			O	MULT	A1	LJK3i	
WD 17	A2	LJK3i	A	LJUP			O	ADD	
WD 18	O	RDIV-s	O	STO	A	LJD			
WD 19	L	X**-3							
WD 20	A	RCK4i			O	ADX	A1	KLK4i	
WD 21	A2	KLK4i	A	KLK2i			O	ADD	
WD 22	A	KLK3i			A	EXP2			
WD 23	O	ADX	O	ADD-sh	A	LJA			
WD 24	O	MULT	O	STO	A	TEMP			
WD 25	A	KLKOi			A	EXP2			
WD26	O	ADX	O	ADD-s	A	LJC			
WD 27	O	MULT	A	TEMP			O	ADD	
WD 28	O	MULT-sh	O	STO	A	KL2K3i			
WD 29	A	KLKOi			A	KLK3i			
WD 30	O	ADD	A	KLK4i			A1	EXP2	
WD 31	A2	EXP2	O	ADX	O	ADD-sh	A1	LJB	
WD 32	A2	LJB	O	MULT	O	STO	A1	TEMP	
WD 33	A2	TEMP	A	KLK2i			A1	EXP2	
WD 34	A2	EXP2	O	ADX	O	ADD-sh	A1	LJD	
WD 35	A2	LJD	O	MULT	A	TEMP			
WD 36	O	ADD	O	MULT-s	O	STO	A1	KL4K3i	
WD37	A2	KL4K3i							
WD 38	L	HYDRO		PROBLEM		SECTION12		IDENTIFIER	
WD 39	O	ENT	C	BUN	A	MCP			

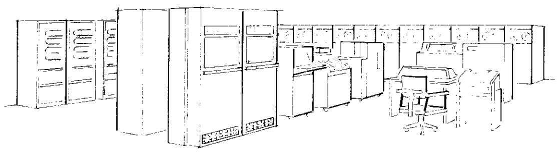


Table 5-4 illustrates the flow of data through the processor and allows accurate calculation of running time. The column headed "time" gives the elapsed time in microseconds from the start of the run. The remaining columns identify specific registers of the machine.

IPP	-	instruction preprocessor buffer
AAC	-	address accumulator
MAR	-	memory address register
DQL	-	the next empty location of the data queue
OQL	-	the next empty location of the operator queue
TOS	-	the top of the stack

Thus, an entry with "41. 993" in the time column and "WD22" in the IPP column indicates that 41. 993 microseconds after start, WD22 is in the instruction preprocessor buffer register. A data label parenthe-ized and preceded by "A" denotes the address of that data. The appear-ance of "SUM," "PROD," etc. in the TOS column indicates the end of the current arithmetic operation.

Operations consume the times specified in Section 4; transfer from IPP to AAC requires 0. 1 μsec; from AAC to MAR 0. 033 μsec; and from IPP to the queues, 0. 067 μsec. Access to the close-coupled fast store requires 0. 250 μsec, but the memory may not again be referenced for another 0. 150 μsec.

REFERENCES

1. Burroughs Algebraic Complier for the 220 Data-Processing System, Bulletin 220-21011-P, January, 1960.

2. ALGOL 58, Communications of the ACM, 1. No. 12, (1958), 8.

3. ALGOL 60, Communications of the ACM, May, 1960, p. 299-314.

TABLE 5-4

D-850 PROCESSING SEQUENCE FOR SECTION 12 OF HYDRO PROBLEM

Time (μsec)	IPP	AAC	MAR	DQL	OQL	TOS
0.000						
0.033			A(WD0)			
0.283	WD0					
0.383		A(LZCK0i)				
0.433			A(LZCK0i)			
0.533		A(LZCK4i)				
0.683	LZCK4i					
0.650				LZCK0i		
0.833			A(LZCK4i)			
1.083	LZCK4i			LZCK4i		LZCK0i
1.150						
1.233			A(WD1)			
1.483	WD1					
1.550					ADD	
1.650		A(LZCK3i)				
1.657					ADD	
1.670						SUM
1.683			A(LZCK3i)			
1.933	LZCK3i					
2.000				LZCK3i		
2.117			A(WD2)			
2.120						SUM
2.367	WD2					
2.400					ADD	
2.433					STO	
2.467		A(LZCK2i)				
2.540			A(LZCK2i)			
2.790	LZCK2i					
2.857				LZCK2i		
2.973			A(WD3)			
2.977						SUM
3.223	WD3					
3.323		A(GPK3i)				
3.407			A(GPK3i)			
3.507		A(TEST18)				
3.840			A(TEST18)			
4.090	TEST18					
4.123			A(WD4)			
4.157				TEST18		
4.353	WD4					
4.420					IF	
4.487					BOF	
4.587		A(OUT)				
4.687			A(WD5)			
4.937	WD5					
5.037		A(LJK3i)				
5.370	LJK3i					
5.437				LJK3i		
5.470		A(LJDN)				
5.803	LJDN					
5.870				LJDN		LJK3i
5.867			A(WD6)			
6.117	WD6					
6.123					MULT	
6.223		A(LJK3i)				
6.300			A(LJK3i)			
6.550	LJK3i					
6.583			A(WD7)			
6.617				LJK3i		

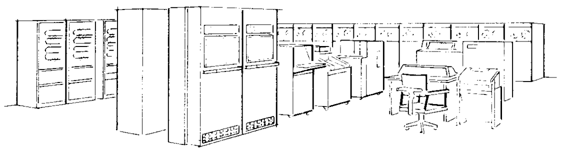


TABLE 5-4 (Cont'd.)

Time (μsec)	IPP	AAC	MAR	DQL	OQL	TOS
6.833	WD7					
6.900					ADD	
6.933		A(LJDN)				
6.967					RDIV-s	
7.017			A(LJDN)			
7.033					STO	
7.267	LJDN					
7.333				LJDN		
7.450			A(WD8)			
7.700	WD8					
7.800		A(LJA)				
8.203						PROD
8.323						SUM
13.003						QUOT
13.403						QUOT in Memory
13.437			A(LJK4i)			
13.587	LJK4i					
13.653				LJK4i		
13.870			A(WD9)			
14.120	WD9					
14.220		A(LJUP)			MULT	
14.303			A(LJUP)			
14.583	LJUP					
14.650				LJUP		LJK4i
14.737			A(WD10)			
14.987	WD10					
15.087		A(LJK4i)				
15.170			A(LJK4i)			
15.237					ADD	
15.270		A(LJUP)				
15.420	LJK4i					
15.453				LJK4i		
15.703			A(LJUP)			
15.953	LJUP					
16.020				LJUP		
16.137			A(WD11)			
16.387	WD11					
16.453					RDIV-s	
16.520					STO	
16.620		A(LJL)				
16.653			A(LJL)			
16.730						PROD
16.850						SUM
21.530						QUOT
21.930						QUOT in Memory
21.963			A(WD12)			
22.213	WD12					
22.313		A(LJK4i)				
22.397			A(LJK4i)			
22.497		A(LJDN)				
22.647	LJK4i					
22.713				LJK4i		
22.830			A(LJDN)			
23.080	LJDN					
23.147				LJDN		LJK4i
23.213			A(WD13)			
23.463	WD13					
23.530					MULT	
23.630		A(LJK4i)				
23.667			A(LJK4i)			
23.717	LJK4i					
23.783				LJK4i		

TABLE 5-4 (Cont'd.)

Time (μsec)	IPP	AAC	MAR	DQL	OQL	TOS
24.100			A(WD14)			
24.350	WD14					
24.450		A(LJDN)			ADD	
24.517					RDIV-s	
24.533			A(LJDN)			
24.567					STO	
24.783	LJDN					
24.850				LJDN		
24.967			A(WD15)			
25.217	WD15					
25.317		A(LJB)				
25.350			A(LJB)			
25.610						PROD
25.450		A(LJK3i)				
25.730						SUM
30.410						QUOT
30.810						QUOT in Memory
30.843			A(LJK3i)			
31.093	LJK3i					
31.160				LJK3i		
31.477			A(WD16)			
31.727	WD16					
31.827		A(LJUP)				MULT
31.943			A(LJUP)			
32.193	LJUP					
32.260				LJUP		LJK3i
32.377			A(WD17)			
32.627	WD17					
32.727		A(LJK3i)				
32.810			A(LJK3i)			
32.910		A(LJUP)				
32.977					ADD	
33.243	LJK3i					
33.277			A(LJUP)			
33.527	LJUP					
33.593				LJUP		
33.710			A(WD18)			
33.960	WD18					
34.027					RDIV-s	
34.093					STO	
34.193		A(LJD)				
34.227			A(LJD)			
34.340						PROD
34.460						SUM
39.140						QUOT
39.540						QUOT in Memory
39.573			A(WD19)			
39.823	WD19					
39.890				$X^{**} - 3$		
40.007			A(WD20)			
40.257	WD20					
40.357		A(RCK4i)				
40.423					ADX	
40.440			A(RCK4i)			
40.690	RCK4i					
40.757				RCK4i		
40.877			A(WD21)			SUM
41.127	WD21					
41.227		A(KLK4i)			ADD	
41.310			A(KLK4i)			
41.560	KLK4i					
41.627				KLK4i		

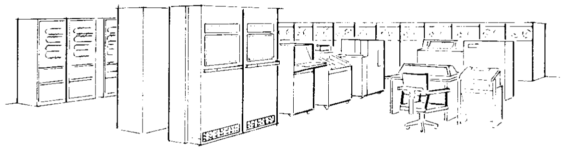


TABLE 5-4 (Cont'd.)

Time (μsec)	IPP	AAC	MAR	DQL	OQL	TOS
41.743			A(KLK4i)			
41.747						SUM
41.993	WD22					
42.093		A(KLK3i)				
42.177			A(KLK3i)			
42.277		A(EXP2)				
42.327	KLK3i			KLK3i		
42.610			A(EXP2)			
42.860	EXP2					
42.927				EXP2		
43.010			A(WD23)			
43.260	WD23					
43.327					ADX	
43.393					ADD-sh	
43.447						SUM
43.493		A(LJA)				
43.527			A(LJA)			
43.567						SUM
43.777	LJA					
43.843				LJA		
43.960			A(WD24)			
44.210	WD24					
44.277					MULT	
44.343					STO	
44.443		A(TEMP)				
44.477			A(TEMP)			
46.357						PROD
46.757						
46.790			A(WD25)			PROD in Memory
47.040	WD25		A(WD22)			
47.140		A(KLK0i)				
47.223			A(KLK0i)			
47.323		A(EXP2)				
47.473	KLK0i					
47.540				KLK0i		
47.653			A(EXP2)			
47.903	EXP2					
47.970				EXP2		KLK0i
48.087			A(WD26)			
48.337	WD26					
48.403					ADX	
48.470					ADD-s	
48.523						SUM
48.570		A(LJC)				
48.603			A(LJC)			
48.643						SUM
48.853	LJC					
48.920				LJC		
49.037			A(WD27)			
49.287	WD27					
49.353					MULT	
49.453		A(TEMP)			ADD	
49.487			A(TEMP)			
49.737	TEMP					
49.803				TEMP		
49.920			A(WD28)			
49.950	WD28					
49.987					MULT-sh	
50.053					STO	
50.153		A(KL2K3i)				
50.187			A(KL2K3i)			
51.433						PROD
51.553						SUM
53.633						PROD
54.033						PROD in Memory

TABLE 5-4 (Cont'd.)

Time (μsec)	IPP	AAC	MAR	DQL	OQL	TOS
54.067			A(WD29)			
54.317	WD29					
54.417		A(KLK0i)				
54.500			A(KLK0i)			
54.600		A(KLK3i)				
54.750	KLK0i					
54.817				KLK0i		
54.933			A(KLK3i)			
55.183	KLK3i					
55.250			A(WD30)	KLK3i		KLK0i
55.500	WD30					
55.567					ADD	
55.667		A(KLK4i)				
55.683			A(WD31)			
55.687						SUM
55.933	WD31					
56.033		A(EXP2)			ADX	
56.100					ADD-sh	
56.117			A(EXP2)			
56.367	EXP2					
56.433				EXP2		
56.550			A(WD32)			
56.553						SUM
56.673						SUM
56.800	WD32					
56.900		A(LJB)			MULT	
56.967					STO	
56.983			A(LJB)			
51.233	LJB					
57.300				LJB		
57.417			A(WD33)			
57.667	WD33					
57.767		A(TEMP)				
57.850			A(TEMP)			
57.950		A(KLK2i)				
59.380						PROD
59.780						PROD in Memory
59.813			A(KLK2i)			
60.063	KLK2i					
60.130				KLK2i		
60.243			A(WD34)			
60.493	WD34					
60.593		A(EXP2)			ADX	
60.660					ADD-sh	
60.677			A(EXP2)			
60.927	EXP2					
60.993				EXP2		
61.110			A(WD35)			
61.213						SUM
61.333						SUM
61.360	WD35					
61.460		A(LJD)			MULT	
61.493			A(LJD)			
61.593		A(TEMP)				
61.743	LJD					
61.810				LJD		
61.927			A(TEMP)			
62.177	TEMP					
62.443					TEMP	
62.360			A(WD36)			
62.610	WD36					
62.677					ADD	
62.743					MULT-s	
62.793			A(WD37)			
62.810					STO	

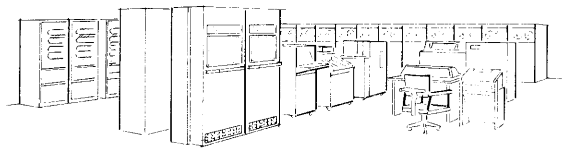
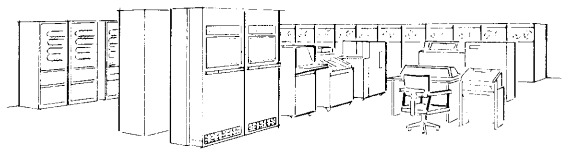


TABLE 5-4 (Cont'd.)

Time (μsec)	IPP	AAC	MAR	DQL	OQL	TOS
63.043	WD37					
63.143		A(KL4K3i)				
63.177			A(KL4K3i)			
63.890						PROD
64.010						SUM
66.090						PROD
66.490						PROD in Memory

Appendices



APPENDIX A
HYDRO TEST PROBLEM

HYDRO TEST PROBLEM STATEMENT IN ALGOL

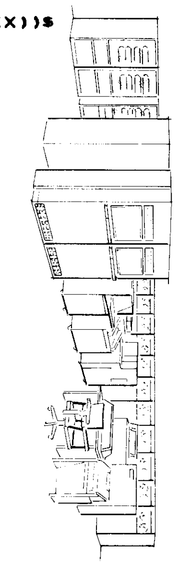
```
         PROCEDURE EOSEF(A.B.C)$
         BEGIN
            PANSWER = 0.1.A + 0.0137.A*4/B$
            EITHER IF A GEQ C$
               ANS = PANSWER$
            OTHERWISE$
               ANS = (A/C).PANSWER$
         EOSEF( ) = ANS$  RETURN
         END$

         PROCEDURE  EOSLF (A.B)$
         BEGIN
            ANS = 1.37.B/(0.15 + (5290.0/SQRT(A)*3)/
               (76.B/A + B))$
         EOSLF( ) = ANS$  RETURN
         END$

         INTEGER X$
         REAL OTHERWISE$

ARRAY    AK(20).AK1(20).ETO(20).GPK3(20).MTO(20).
         CHI(20).CFK(20).CHIO(20).CFT2(20).CFT4(20).
         DTHT(20).DTHTO(20).DTHO(20).DETO(20).Q4TO(20).
         FJT(20).FJ01T(20).FBK(20).FT(20).FAK(20).
         FJ02T(20).FDK(20).FAK1(20).FLT1(20).
         FDK2(20).FDKO(20).FAK4(20).FJTO(20).
         FTO(20).FJ1TO(20).FAKO(20).FLTO(20).
         FBKO(20).FBK2(20).FCKO(20).FBK1(20).
         FCK4(20).FBK3(20).FCK1(20).FBK4(20).
         JTK(20).JTK1(20).JTK2(20).KL2K3(20).
         KRMK2(20).KZMK2(20).KZMKO(20).KRMKO(20).
         KCTO(20).KCT4(20).KL2T2(20).KL2T4(20).
         KL4TO(20).KLK2(20).KCK3(20).KL4T3(20).
         KLK4(20).KLK3(20).KLKO(20).KL4K3(20).
         KL2K6(20).KCK6(20).KCK4(20).KL4K4(20).
         LRK(20).LZK(20).LRK1(20).LZK1(20).
         LCT2(20).LCTO(20).LZCK2(20).LRCK2(20).
         LJ2K2(20).LJ2KO(20).LCK2(20).KL4K7(20).
         LJK3(20).LZCK3(20).LZCKO(20).LZCK4(20).
         LJK4(20).LCK4(20).LCK3(20).PKO(20).
         PT(20).PK(20).P1K(20).P2K(20).P3K(20).
         P1K2(20).P1KO(20).P2K1(20).P2KO(20).P3K1(20).
         Q1T2(20).Q1TO(20).Q2T1(20).Q2KO(20).Q3T(20).
         Q3T1(20).Q4T(20).Q4T2(20).Q2TO(20).Q3TO(20).
         RT(20).RTO(20).RT1(20).RT2(20).RAK2(20).RAKO(20).
         RAK3(20).RBK2(20).RCK2(20).RBKO(20).RBK3(20).
         RBK4(20).RCK4(20).RCKO(20).RCK3(20).PTO(20).
         R1B5(20).R1F5(20).R2B5(20).R2F5(20).R3B5(20).
         R3F5(20).R4B5(20).R4F5(20).THATO(20).TT1(20).
         TT(20).THAT(20).THCT(20).TTO(20).THCTO(20).
         UTO(20).VTO(20).U2K2(20).U2KO(20).U2K4(20).
         UKO(20).UK2(20).UK3(20).UK4(20).U2K3(20).
         ZT(20).ZTO(20).ZT1(20).ZT2(20).ZAK2(20).
         VKO(20).VK2(20).VK3(20).VK4(20).
         ZAKO(20).ZAK3(20).ZBK2(20).ZCK2(20).
         ZBKO(20).ZBK3(20).ZBK4(20).ZCKO(20).
         ZCK4(20).ZCK3(20).WK2(20).WKO(20)          $
```

```
COMMENT     INITIALIZE TEST INDICATORS$

            TEST1=TEST2=TEST3=TEST4=TEST5=TEST6=
            TEST7=TEST8=TEST9=TEST10=TEST11=TEST12=
            TEST13=TEST14=TEST15=TEST16=TEST17=TEST18=
            TEST19=1$

            TEST9 = 0$

COMMENT          SECTION 1$
            FOR X = (1.1.20)$
            BEGIN
            READ($$DATA1)$
            READ($$DATA2)$

            TEMPO=0.25(RT(X)-RTO(X))$
            TEMP1=0.25(RT1(X)-RT2(X))$
            LRK(X)=TEMPO+TEMP1$
            KRUP=TEMPO-TEMP1$
            TEMPO=0.25(ZT(X)-ZTO(X))$
            TEMP1=0.25(ZT1(X)-ZT2(X))$
            LZK(X)=TEMPO+TEMP1$
            KZUP=TEMPO-TEMP1$
            JTK(X)=FJT(X)/TT(X)$

            IF TEST1 EQL ZERO$
            GO OUT$

            AK(X)=SQRT(LRK(X)*2+LZK(X)*2).(RTO(X)+RT2(X)
                  +RT(X)+RT1(X)  )$
            AIN=AK(X)+AIN$

            IF TEST2 EQL ZERO$
            GO OUT$

            THA=THAT(X)$
            THAT(X)=(CHI(X))*0.25+0.5(DTHT(X))$
            DTHL=0.5(THAT(X)+THA)$
            THCT(X)=0.5(CON.THAT(X)+C1N.THA+C2N.FJ01T(X))$
            FBK(X)=0.5(FT(X)+CHI(X))$
            FAK(X)=(THCT(X))*4$
            CFK(X)=PS+PT(X)$
            PS=(0.067.THAT(X)/TT(X))+0.00457.THAT(X)*4$
            DP=PS-PT(X)$
            PK(X)=DP.F1N+PS$
            PAL=DP.F2N+PS$
            PT(X)=PS$
            P1K(X)=P2K(X)=P3K(X)=P4UP=PAL $
            FJ02T(X)=FJ01T(X)$
            FJ01T(X)=THA$

COMMENT          SECTION 2$

            TS0=JTK(X)+JTK2(X)$
            TS1=KRUP+KRDN$
            KRMK2(X)=TS1/TS0$
            TS2=KZUP+KZDN$
```

D 850

```
                KZMK2(X)=TS2/TSO$
                WK2(X)=SQRT(TS1*2+TS2*2)$

COMMENT                  SECTION 3$

                TSO=JTK1(X)+JTK(X)$
                TS1=LRK1(X)+LRK(X)$
                TS2=LZK1(X)+LZK(X)$
                LRMUP=TS1/TSO$
                LZMUP=TS2/TSO$
                FDK(X)=FAK1(X)+FAK(X)$
                WUP=SQRT(TS1*2+TS2*2)$

COMMENT                  SECTION 4$

                RAK2(X)=RTO(X)$
                ZAK2(X)=ZTO(X)$
                KP1=P1K2(X)+Q1T2(X)-P1KO(X)-Q1TO(X)$
                LP2=P2K1(X)+Q2T1(X)-P2KO(X)-Q2TO(X)$
                KP3=P3K(X)+Q3T(X)-P3K1(X)-Q3T1(X)$
                LP4=P4UP+Q4T(X)-P4DN-Q4T2(X)$

COMMENT                  SECTION 5$

                TM1=ABS(RAK2(X)-RAKO(X))+ABS(ZAK2(X)
                      -ZAKO(X))$
                TM2=ABS(RAK2(X)-RAK3(X))+ABS(ZAK2(X)
                      -ZAK3(X))$

COMMENT                  SECTION 6$

                IF TEST3 EQL ZERO$
                GO OUT$

                WRK=2.WK2(X)/(WK2(X)+WKO(X))$
                WRL=2.WUP/(WUP+WDN)$
                TEMPO=WRL.KP1$
                TEMP1=WRK.LP2$
                TEMP2=(2-WRL).KP3$
                TEMP3=(2-WRK).LP4$
                DU=DTN.(TEMP1.KZMKO(X) - TEMPO.LZMDN
                       -TEMP2.LZMUP+TEMP3.KZMK2(X))$
                DV=DTN.(TEMPO.LRMDN-TEMP1.KRMKO(X)
                       +TEMP2.LRMUP-TEMP3.KRMK2(X))$

COMMENT                  SECTION 7$

                UTO(X)=DU+UTO(X)$
                VTO(X)=DV+VTO(X)$
                RBK2(X)=DTNUP.UTO(X)+RTO(X)$
                ZBK2(X)=DTNUP.VTO(X)+ZTO(X)$
                RCK2(X)=0.5(RTO(X)+RBK2(X))$
                ZCK2(X)=0.5.(ZTO(X)+ZBK2(X))$
                RTO(X)=RBK2(X)$
                ZTO(X)=ZBK2(X)$
                U2K2(X)=UTO(X)*2+VTO(X)*2$

                IF TEST4 EQL ZERO$
                GO OUT$
```

```
COMMENT                  SECTION 8$

                IF TEST5 EQL ZERO$
                GO OUT$

                AMN=0.5.AN$
                AM3DN=-0.25.AN$
                AM4DN=3.AM3DN$
                RAN=RBN$
                ZAN=ZBN$
                RBN=0.25.(RAKO(X)+2.RAK2(X)+RT2(X))$
                ZBN=0.25.(ZAKO(X)+2.ZAK2(X)+ZT2(X))$
                ALA=AK1(X).FLT1(X)$
                ALN=ALA+ALN$
                AM3UP = AK1(X)+AM3DN$
                AM4UP = AK1(X)+AM4DN$

                IF AM3UP LSS ZERO $  GO TO SECTION9$
                IF AM3DN LSS ZERO $  GO TO LINEE$
                IF AM4UP LSS ZERO $  GO TO LINE13$
                IF AM4DN LSS ZERO$
                   GO TO LINE14$
                   GO TO SECTION9$
LINEE..         IF AM4UP LSS ZERO$  GO TO LINE15$

                ALMN = AMN.FLT1(X)$
                GO TO SECTION9$
LINE13..        ALMN = ALA + ALMN$
                GO TO SECTION9$
LINE14..        ALMN = -AM4DN.FLT1(X) + ALMN$
                GO TO SECTION9$
LINE15..        ALMN = AM3UP.FLT1(X)$
                GO TO SECTION9$

COMMENT                  SECTION9$

SECTION9..      IF TEST6 EQL ZERO$
                GO OUT$

                FD = 0.25.(FDK2(X) + FDKO(X) + FAK4(X) + FAK1(X))$
                U2 = U2K2(X) + U2KO(X) + U2K4(X) + U2K3(X)$

                EITHER IF U2 + FD + ABS(DS) EQL ZERO$
                   TEST7 = 0$
                OTHERWISE$
                   TEST7 = 1$

                KRB1 = RBK2(X) - RBKO(X)$
                KZB1 = ZBK2(X) - ZBKO(X)$
                LRB2 = RBK2(X) - RBK3(X)$
                LZB2 = ZBK2(X) - ZBK3(X)$

                KRB3 = RBK3(X) - RBK4(X)$
                KZB3 = ZBK3(X) - ZBK4(X)$
                LRB4 = RBKO(X) - RBK4(X)$
                LZB4 = ZBKO(X) - ZBK4(X)$

                J41 = KRB1.LZB4 - LRB4.KZB1$
```

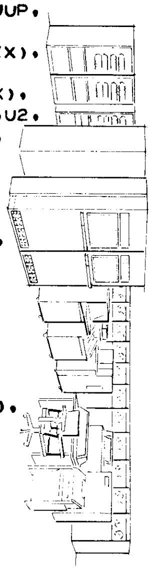


```
            J12 = KRB1.LZB2 - LRB2.KZB1$
            J23 = KRB3.LZB2 - LRB2.KZB3$
            J34 = KRB3.LZB4 - LRB4.KZB3$

            FJTO(X) = 0.25 . (J12 + J23 + J34 + J41)$
            RHA = 0.25.(RBK4(X)+RBKO(X)+RBK2(X)+RBK3(X))$
COMMENT    TEST ZERO BITS$
            RJ = FJTO(X) .RHA$
            TA = TTO(X)$
            TTO(X) = RJ/MTO(X)$
            TC = 0.5.(TTO(X) + TA)$

            KRB = 0.5.(KRB1 + KRB3)$
            KZB = 0.5.(KZB1 + KZB3)$
            LRB = 0.5.(LRB2 + LRB4)$
            LZB = 0.5.(LZB2 + LZB4)$

            KRKRB = KRB*2 + KZB*2$
            LRLRB = LRB*2 + LZB*2$

            KU1 = UK2(X) - UKO(X)$
            LU2 = UK2(X) - UK3(X)$
            KU3 = UK3(X) - UK4(X)$
            LU4 = UKO(X) - UK4(X)$

            KV1 = VK2(X) - VKO(X)$
            LV2 = VK2(X) - VK3(X)$
            KV3 = VK3(X) - VK4(X)$
            LV4 = VKO(X) - VK4(X)$

            UR1 = KU1.LZB - KV1.LRB$
            UR2 = -LU2.KZB + LV2.KRB$
            UR3 = KU3.LZB - KV3.LRB$
            UR4 = -LU4.KZB + LV4.KRB$

            RKR = 2.0/(TC.KRKRB)$
            RLR = 2.0/(TC.LRLRB)$

            G1 = RLR.UR1*2$
            G2 = RKR.UR2*2$
            G3 = RLR.UR3*2$
            G4 = RKR.UR4*2$

            Q = 0.5.(G1 + G2 + G3 + G4)$
            Q1TO(X) = G1 + 0.5.(G2 + G4)$
            Q2TO(X) = G2 + 0.5.(G1 + G3)$
            Q3TO(X) = G3 + 0.5.(G2 + G4)$
            Q4TO(X) = G4 + 0.5.(G1 + G3)$

            IF TEST8 EQL ZERO$
            GO OUT$

            MDT = TA - TTO(X)$

            IF TEST9 GTR ZERO$
            GO OUT$

            DE = MDT.(PKO(X) + Q) + DS$
            DTHE = 2.(THCTO(X) - THATO(X))$
```

```
THE = THATO(X) + DTHE$
DPHE = THE*4 - THATO(X)*4$
DTTE = TTO(X) - TA$

IF ABS(DTTE) GTR TA $
E = EOSEF(THATO(X),TA,THSTR)$
ED1 = EOSEF(THATO(X),TC,THSTR)$
ED2 = EOSEF(THE,TC,THSTR)$
ED3 = EOSEF(THCTO(X),TA,THSTR)$
ED4 = EOSEF(THCTO(X),TTO(X),THSTR)$

DD1 = (ED2 - ED1)/DTHE$
DD2 = (ED4 - ED3)/DTTE$
DD3 = (ED2 - ED1)/DPHE$

DTHTO(X) = (DE + MDT.DD2)/DD1$

IF CHIO(X) GTR ZERO$
DF = CHIO(X) - FTO(X)$
CHIO(X) = (THATO(X) +0.5.DTHO(X))*4 + CHIO(X)-CHI(X)$

DJ1 = E-ETO(X)/MTO(X)$
DJ2 = DETO(X) + (DTNON.DF/(2.MTO(X)))/CFT2(X)$
FJ1TO(X)=(DJ1-DJ2)/((1.0**-4)/E+ 0.02E + ABS(DJ1)
  + ABS(DJ2))$

IF PTO(X) GTR 1.0**-10$
IF E GTR 1.0**-10$
C2 = PTO(X).TA + PTO(X).TA/E$
DTC2 = FJTO(X)*2/(4.C2.(KRKRB + LRLRB))$

IF DTC2 GTR DTC2N$
IF Q NEQ ZERO$
DTQ = -FJTO(X)/(10(UR1 + UR2))$
ETO(X) = E.MTO(X)$
DEL = MTO(X).U2/8$

IF CHIO(X) GTR 1.0**-8$
BF1F = CFT2(X).LCT2(X).(FBK2(X) - FBKO(X))$
BF1B = CFT2(X).LCTO(X).(FCKO(X)-FBKO(X))$
BF3F = CFT2(X).KCTO(X).(FBK1(X)-FBKO(X))$
BF3B = CFT2(X).KCT4(X).(FBK4(X)-FBKO(X))$
BF2F = CFT2(X).KL2T2(X).(FBKO(X)-FBK(X))$
BF2B = CFT2(X).KL2T4(X).(FBKO(X)-FCK4(X))$
BF4F = CFT2(X).KL4T3(X).(FBK3(X)-FBKO(X))$
BF4B = CFT2(X).KL4TO(X).(FCK1(X)-FBKO(X))$
LZCK2(X) = (BFL - DF)/ (1.0**-8 + 0.01.CHIO(X) +
ABS(DF) + ABS(BF1F) + ABS(BF1B) + ABS(BF3F) + ABS(BF3B) +
4.( ABS(BF2F) + ABS(BF2B) + ABS(BF4F) + ABS(BF4B)))     $

CFT2(X) = DTNUP/(2.DD3.MTO(X))$
LRC = 0.5.(RCK2(X)-RCK4(X)+RCKO(X)-RCK3(X))$
KRCUP = 0.5.(RCK2(X)-RCK4(X)-RCKO(X)+RCK3(X))$
LZC = 0.5.(ZCK2(X)-ZCK4(X)+ZCKO(X)-ZCK3(X))$
KZCUP = 0.5.(ZCK2(X)-ZCK4(X)-ZCKO(X)+ZCK3(X))$
KLK2(X) = KRCUP.LRC + KZCUP.LZC$
JC = 0.5.(JL+FJTO(X))$
LRCK2(X) = LRC*2 + LZC*2$
LP2 = 0.5.FDK2(X)$
```

```
            LP4 = 0.5.FDKO(X)$

            IF TT1(X) NEQ ZEROS
            KP1 = 0.5.(FAK1(X) + FAKO(X))$
            KP3 = 0.5.(FAK4(X) + FAKO(X))$
            P1 = EOSLF(KP1.TC)$
            P2 = EOSLF(LP2.TC)$
            P3 = EOSLF(KP3.TC)$
            P4 = EOSLF(LP4.TC)$
            FLTO(X) = 4.EOSLF(FAKO(X).TC)$

            IF TEST16 EQL ZEROS
            GO OUTS
            LJ2K2(X) = LJ2K2(X).GNS
            LJ4 = LJ4.GNS
COMMENT            SECTION 10$

            IF LJ2KO(X) NEQ ZEROS
            LJUP = 2.LJ2K2(X).LJ4/(LJ2K2(X) + LJ4)$

            IF TEST17 EQL ZEROS
            GO OUTS

            RH1 = 0.5.(RCK4(X) + RCKO(X))$
            LRLRC = 0.5.(LRCK2(X) + LRCKO(X))$
            LCK2(X) = LRLRC.LJUP.RH1$
COMMENT            SECTION 11$

            LJK3(X) = 2.LJ1DN.LJ3/(LJ1DN + LJ3)$
            RH3 = 0.5.(RCK3(X) + RCK4(X))$
            KRKRC = 0.5.(KRCUP*2+KRCDN*2+KZCUP*2+KZCDN*2)$
            KCK3(X) = KRKRC.LJK3(X).RH3$
COMMENT            SECTION 12$

            GPK3(X) = LZCK2(X)+LZCK3(X)+LZCKO(X)+LZCK4(X)$

            IF TEST18 EQL ZEROS
            GO OUTS
            LJA = LJK3(X).LJDN/(LJK3(X) +LJDN)$
            LJC = LJK4(X).LJUP/(LJK4(X) + LJUP)$
            LJB = LJK4(X).LJDN/(LJK4(X) + LJDN)$
            LJD = LJK3(X).LJUP/(LJK3(X) + LJUP)$

            IF TEST19 EQL ZEROS
            GO OUTS
            RH8 = RCK4(X)/8$
            KL2K3(X) = RH8.(LJA.(KLK4(X) + 2.KLK3(X) + KLK2(X))
               + LJC.(KLK4(X) + 2.KLKO(X) + KLK2(X)))$
            KL4K3(X) = RH8.(LJB.(KLKO(X) + 2.KLK4(X) + KLK3(X))
               +LJD.(KLKO(X) + 2.KLK2(X) + KLK3(X)))$
COMMENT            SECTION 13$

            R1B5(X) = CFT4(X).LCK4(X)$
            R1F5(X) = CFT4(X).LCK3(X)$
            R2B5(X) = -CFT4(X).KL2K6(X)*2$
            R2F5(X) = -CFT4(X).KL2K3(X)*2$
            R3B5(X) = CFT4(X).KCK6(X)$
            R3F5(X) = CFT4(X).KCK4(X)$
            R4B5(X) = CFT4(X).KL4K4(X)*4$
            R4F5(X) = CFT4(X).KL4K7(X)*4$
   OUT..
```

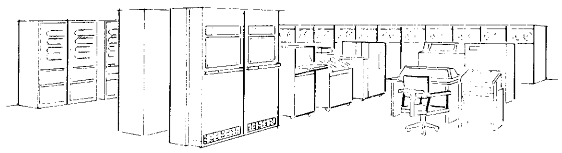
```
COMMENT     PRINT RESULTS OF CALCULATIONS$
            WRITE ($$FTITLE)$
            WRITE($$ANS1.F1)$
            WRITE($$FTITLE1)$
            WRITE($$ANS2.F2)$

            ENDS
             STOPS
INPUT       DATA1
            (AK1(X).AN.BFL.CON.C1N.C2N.CFT4(X).CHI(X).DETO(X).DS.
            DTC2N .DTHT(X).DTN.DTNON.DTNUP.F1N.F2N.FAKO(X).FAK1(X).
            FAK4(X).FBKO(X).FBK1(X).FBK2(X).FBK3(X).FBK4(X).FCKO(X).
            FCK1(X).FCK4(X).FDKO(X).FDK2(X).FJT(X).FLT1(X).FT(X).
            FTO(X).GN.JL.JTK1(X).JTK2(X).KCK4(X).KCK6(X).KCTO(X).
            KCT4(X).KL2K6(X).KL2T2(X).KL2T4(X).KL4K4(X).KL4K7(X).
            KL4TO(X).KL4T3(X).KLKO(X).KLK3(X).KLK4(X).KRCDN.KRDN.
            KRMKO(X).KZCDN.KZDN.KZMKO(X).LCK3(X).LCK4(X).LCTO(X).
            LCT2(X).LJ1DN.LJ2KO(X).LJ3.LJDN.LJK4(X).LRCKO.LRK1(X).
            LRMDN.LZCKO(X).LZCK3(X).LZCK4(X).LZK1(X).LZMDN.MTO(X).
            P1KO(X).P1K2(X).P2KO(X).P2K1(X).P3K1(X).P4DN.PKO(X).
            PTO(X).Q1T2(X).Q2T1(X).Q3T(X).Q3T1(X).Q4T(X).Q4T2(X).
            RAKO(X).RAK3(X).RBKO(X).RBK3(X).RBK4(X).RCKO(X).RCK3(X).
            RCK4(X).RT(X).RT1(X).RT2(X).THATO(X).THCTO(X).THSTR.
            TT(X).TT1(X).U2KO(X).U2K3(X).U2K4(X).UKO(X).UK2(X).
            UK3(X).UK4(X).VKO(X).VK2(X).VK3(X).VK4(X).WDN.WKO(X).
            ZAKO(X).ZAK3(X).ZBKO(X).ZBK3(X).ZBK4(X).ZCKO(X).ZCK3(X).
            ZCK4(X).ZT(X).ZT1(X).ZT2(X))$
INPUT       DATA2

            (AIN.ALN.CFT2(X).CHIO(X).ETO(X).FJO1T(X).LJ2K2(X).LJ4.
            PT(X).Q1TO(X).Q2TO(X).RBN.RTO(X).THAT(X).TM2.TTO(X).
            UTO(X).VTO(X).ZBN.ZTO(X))$
OUTPUT      ANS1
            (AK(X).ALA.ALMN.AM3DN.AM3UP.AM4DN.AM4UP.AMN.C2.CFK(X).DD1.
            DD2.DD3.DE.DEL.DF.DP.DPHE.DTC2.DTHE.DTHL.DTHTO(X).DTQ.
            DTTE.DU.DV.FAK(X).FBK(X).FD.FDK(X).FJO2T(X).FJ1TO(X).
            FJTO(X).FLTO(X).GPK3(X).JC.JTK(X).KCK3(X).KL2K3(X).
            KL4K3(X).KLK2(X).KP1.KP3.KRCUP.KRKRB.KRKRC.KRMK2(X).KRUP.
            KZCUP.KZMK2(X).KZUP.LCK2(X).LJA.LJB.LJC.LJD.LJK3(X).LJUP.
            LP2.LP4.LRC.LRCK2(X).LRK(X).LRLRB.LRLRC.LRMUP.LZC.
          LZCK2(X).LZK(X).LZMUP.MDT.P1.P2.P3.P4.P1K(X).P2K(X).P3K(X).
            P4UP.PAL.PK(X).PS.Q.Q3TO(X).Q4TO(X).R1B5(X).R1F5(X).
            R2B5(X).R2F5(X).R3B5(X).R3F5(X).R4B5(X).R4F5(X).RAK2(X).
            RAN.RBK2(X).RCK2(X).RH8.RHA.RJ.TC.THA.THCT(X).THE.TM1.U2.
            U2K2(X).UR1.UR2.UR3.UR4.WK2(X).WRK(X).WRL.WUP.ZAK2(X).
            ZAN,ZBK2(X).ZCK2(X))$

OUTPUT      ANS2

            (AIN.ALN.CFT2(X).CHIO(X).ETO(X).FJO1T(X).LJ2K2(X).LJ4.
            PT(X).Q1TO(X).Q2TO(X).RBN.RTO(X).THAT(X).TM2.TTO(X).
            UTO(X).VTO(X).ZBN.ZTO(X))$

FORMAT      F1(15(8F15.8.W2).W2).
            F2(4(5F24.8.W2).W2)$

FORMAT
FTITLE     (B5.*VARIABLES WHICH ARE NOT GIVEN AND ARE COMPUTED*.W2).
FTITLE1   (B5.*VARIABLES WHICH ARE GIVEN AND ARE RECOMPUTED*.W2)$

    FINISHS
```



D 850 MACHINE LANGUAGE PROGRAM FOR HYDRO TEST PROBLEM

BITS 0 12 24 36 47

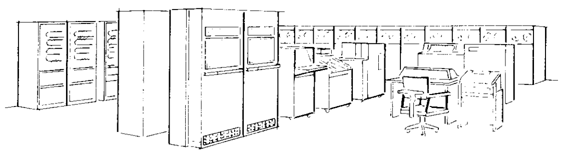
LOC	LABEL		SYLLABLE 1		SYLLABLE 2		SYLLABLE 3		SYLLABLE 4	REMARKS
		L	0.1							
		A	A			O	ENT-h	O	MULT-s	
		O	ENT	O	ENT-s	O	MULT-s	O	ENT-s	
		O	MULT-s	O	MULT	A	B			
		L	0.0137							
		O	DIV	O	ADD-s	A	A			
		A	C			O	GEQ	O	IF-s	
		C	BOT	A	LAB1(2)			O	STO	
		A	ANS			C	BUN	A1	LAB25(3)	
	LAB1(2)	A2	LAB25(3)	A	A			A1	C	
		A2	C	O	DIV	O	MULT-s	O	STO-h	
	LAB25 (3)	A	ANS			A	ANS			
		O	STO	A	EOSEF			C	CRE	
		L	1.37							
		A	B			A	A			
		L	0.15							
		L	5290.0							
		O	ENT	O	ENT-s	O	ENT-s	O	MULT-s	
		O	MULT-s	O	SQRT-s	O	DIV-s	A1	B	
		A2	B	O	ENT	O	ENT-s	A1	A	
		L	76.0							
		A2	A	O	MULT-s	O	DIV	O	ADD-s	
		O	DIV-s	O	ADD-s	O	DIV-s	O	STO	
		A	EOSLF			C	CRE	O	ENT	
		L	2							
		O	STO	A	TWO			O	ENT	
		L	1							
		O	STO	A	ONE			O	ENT	
		L	0							
		O	STO	A	ZERO			O	ENT	
		L	-1							
		O	STO	A	M1			O	ENT	
		L	-2							
		O	STO	A	M2			O	ENT	
		L	X**3							
		O	STO	A	EXP3			O	ENT	
		L	X**2							
		O	STO	A	EXP2			O	ENT	
		L	X**1							
		O	STO	A	EXP1			O	ENT	
		L	X**-1							
		O	STO	A	EXPM1			O	ENT	
		L	X**-2							
		O	STO	A	EXPM2			O	ENT	
		L	X**-3							
		O	STO	A	EXPM3					



BITS		0		12		24		36	47	
LOC	LABEL		SYLLABLE 1		SYLLABLE 2		SYLLABLE 3		SYLLABLE 4	REMARKS
		A	ONE			O	ENT	O	STO-h	
		A	TEST 1			O	STO-h	A1	TEST 2	
		A2	TEST 2	O	STO-h	A	TEST 3			
		O	STO-h	A	TEST 4			O	STO-h	
		A	TEST 5			O	STO-h	A1	TEST 6	
		A2	TEST 6	O	STO-h	A	TEST 7			
		O	STO-h	A	TEST 8			O	STO-h	
		A	TEST 10			O	STO-h	A1	TEST 11	
		A2	TEST 11	O	STO-h	A	TEST 12			
		O	STO-h	A	TEST 13			O	STO-h	
		A	TEST 14			O	STO-h	A1	TEST 15	
		A2	TEST 15	O	STO-h	A	TEST 16			
		O	STO-h	A	TEST 17			O	STO-h	
		A	TEST 18			O	STO-h	A1	TEST 19	
		A2	TEST 19	O	PDS	O	STO	A1	TEST 9	
		A2	TEST 9	A0	LDRIXR15	A	ZERO			
	LAB3 (2)	A0	SRFIXR15	A0	INCIXR 15	A	RT1(X)			
		A	RT2(X)			O	SUB	A1	EXPM2	
		A2	EXPM2	O	ADX	O	STO-h	A1	TEMP1	
		A2	TEMP1	A	RT(X)			A1	RTO(X)	
		A2	RTO(X)	O	SUB	A	EXPM2			
		O	ENT-s	A	TEMP1			O	ADD	
		O	STO	A	LRK(X)			O	RSUB-s	
		O	STO	A	KRUP			A1	ZT1(X)	
		A2	ZT1(X)	A	ZT2(X)			O	SUB	
		A	EXPM2			O	ADX	O	STO-h	
		A	TEMP1			A	ZT(X)			
		A	ZTO(X)			O	SUB	A1	EXPM2	
		A2	EXPM2	O	ENT-s	A	TEMP1			
		O	ADD	O	STO	A	LZK(X)			
		O	RSUB-s	O	STO	A	KZUP			
		A	FJT(X)			A	TT(X)			
		O	DIV	O	STO	A	JTK(X)			
		A	TEST1			A	ZERO			
		O	EQL	O	IF-s	C	BOT	A1	OUT	
		A2	OUT	A	LRK(X)			O	ENT	
		O	ENT-s	O	MULT-s	A	LZK(X)			
		O	ENT	O	ENT-s	O	MULT-s	O	ADD-s	
		O	SQRT-s	A	RTO(X)			A1	RT2(X)	
		A2	RT2(X)	O	ADD	A	RT(X)			
		O	ADD	A	RT1(X)			O	ADD	
		O	MULT-s	O	STO-h	A	AK(X)			
		A	AIN			O	ADD	O	STO	
		A	AIN			A	TEST 2			
		A	ZERO			O	EQL	C	BOT	
		A	OUT			A	THAT(X)			
		O	ENT	O	STO-h	A	THA			

BITS 0 12 24 36 47

LOC	LABEL		SYLLABLE 1		SYLLABLE 2		SYLLABLE 3		SYLLABLE 4	REMARKS
		L	0.25							
		A	CHI(X)			O	P_1(EXP)	A1	DTHT(X)	
		A2	DTHT(X)	A	EXPM1			O	ADX-s	
		O	ADD-s	O	STO-h	A	THAT(X)			
		O	ADD-s	A	EXPM1			O	ADX-s	
		A	CON			A	THAT(X)			
		O	MULT	A	C1N			A1	THA	
		A2	THA	O	MULT	O	ADD-s	A1	C2N	
		A2	C2N	A	FJ01T(X)			O	MULT	
		O	ADD-s	A	EXPM1			O	ADX-s	
		O	STO	A	THCT(X)			A1	FT(X)	
		A2	FT(X)	A	CHI(X)			O	ADD	
		A	EXPM1			O	ADX-s	O	STO	
		A	FBK(X)			A	THCT(X)			
		O	ENT	O	ENT-s	O	MULT-s	O	ENT-s	
		O	MULT-s	O	STO	A	FAK(X)			
		A	PS			A	PT(X)			
		O	ADD	O	STO	A	CFK(X)			
		L	0.067							
		A	THAT(X)			O	MULT	A1	TT(X)	
		A2	TT(X)	O	DIV	A	THAT(X)			
		L	0.00457							
		O	ENT	O	ENT-s	O	MULT-s	O	ENT-s	
		O	MULT-s	O	MULT-s	O	ADD-s	O	STO-h	
		A	PS			A	PT(X)			
		O	SUB	O	STO-h	A	DP			
		O	ENT-s	A	F1N			O	MULT	
		A	PS			O	ADD	O	STO	
		A	PK(X)			A	F2N			
		O	MULT	A	PS			O	ADD	
		O	STO-h	A	PAL			A1	PS	
		A2	PS	O	ENT	O	STO	A1	PT(X)	
		A2	PT(X)	O	STO-h	A	P4UP			
		O	STO-h	A	P3K(X)			O	STO-h	
		A	P2K(X)			O	STO-h	A1	P1K(X)	
		A2	P1K(X)	A	FJ01T(X)			O	ENT	
		O	STO	A	FJ02T(X)			A1	THA	
		A2	THA	O	ENT	O	STO	A1	FJ01T(X)	
		A2	FJ01T(X)	A	JTK(X)			A1	JTK2(X)	
		A2	JTK2(X)	O	ADD	O	ENT-s	A1	KRUP	
		A2	KRUP	A	KRDN			O	ADD	
		O	STO-h	A	TS1			O	RDIV-s	
		A	KRMK2			A	KZUP			
		A	KZDN			O	ADD	O	RDIV-s	
		O	ENT-s	O	MULT-s	A	TS1			
		O	ENT	O	ENT-s	O	MULT-s	O	ADD-s	
		O	SQRT-s	O	STO	A	WK2(X)			

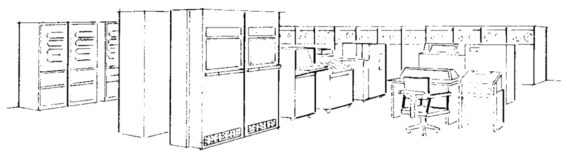


BITS 0 12 24 36 47

LOC	LABEL		SYLLABLE 1		SYLLABLE 2		SYLLABLE 3		SYLLABLE 4	REMARKS
		A	JTK1(X)			A	JTK(x)			
		O	ADD	O	ENT-s	A	LRK1(X)			
		A	LRK(x)			O	ADD	O	STO-h	
		A	TS1			O	RDIV-s	O	STO	
		A	LRMUP			A	LZK1(X)			
		A	LZK(X)			O	ADD	O	STO-h	
		A	TS2			O	RDIV-s	O	STO	
		A	LZMUP			A	FAK1(X)			
		A	FAK(X)			O	ADD	O	STO	
		A	FDK(x)			A	TS1			
		O	ENT	O	ENT-s	O	MULT-s	A1	TS2	
		A2	TS2	O	ENT	O	ENT-s	O	MULT-s	
		O	ADD-s	O	SQRT-s	O	STO	A1	WUP	
		A2	WUP	A	RTO(X)			O	ENT	
		O	STO	A	RAK2(X)			A1	ZTO(X)	
		A2	ZTO(X)	O	ENT	O	STO	A1	ZAK2(X)	
		A2	ZAK2(X)	A	P1K2(X)			A1	Q1T2(x)	
		A2	Q1T2(x)	O	ADD	A	P1K0(X)			
		O	SUB	A	Q1T0(X)			O	SUB	
		O	STO	A	KP1			A1	P2K1(X)	
		A2	P2K1(X)	A	Q2T1(X)			O	ADD	
		A	P2K0(X)			O	SUB	A1	Q2T0(X)	
		A2	Q2T0(X)	O	SUB	O	STO	A1	LP2	
		A2	LP2	A	P3K(X)			A1	Q3T(X)	
		A2	Q3T(X)	O	ADD	A	P3K1(X)			
		O	SUB	A	Q3T1(X)			O	SUB	
		O	STO	A	KP3			A1	P4UP	
		A2	P4UP	A	Q4T(X)			O	ADD	
		A	P4DN			O	SUB	A1	Q4T2(X)	
		A2	Q4T2(x)	O	SUB	O	STO	A1	LP4	
		A2	LP4	A	RAK2(X)			A1	RAK0(x)	
		A2	RAK0(x)	O	SUB	O	ABS-s	A1	ZAK2(x)	
		A2	ZAK2(X)	A	ZAK0(X)			O	SUB	
		O	ABS-s	O	ADD-s	O	STO	A1	TM1	
		A2	TM1	A	RAK2(X)			A1	RAK3(X)	
		A2	RAK3(X)	O	SUB	O	ABS-s	A1	ZAK2(X)	
		A2	ZAK2(X)	A	ZAK3(x)			O	SUB	
		O	ABS-s	O	ADD-s	O	STO	A1	TM2	
		A2	TM2	A	TEST 3			A1	ZERO	
		A2	ZERO	O	EQL	O	IF-s	C	BOT	
		A	OUT			A	WK2(X)			
		O	ENT	O	ENT-s	A	WK0(x)			
		O	ADD	O	DIV-s	A	EXP1			
		O	ADX-s	O	STO-h	A	WRK			
		A	WUP			O	ENT	O	ENT-s	
		A	WDN			O	ADD	O	DIV-s	
		A	EXP1			O	ADX-s	O	STO-h	

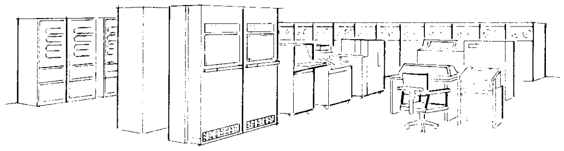
BITS 0 12 24 36 47

LOC	LABEL		SYLLABLE 1		SYLLABLE 2		SYLLABLE 3		SYLLABLE 4	REMARKS
		A	WRL			A	KP1			
		O	MULT	O	STO	A	TEMPO			
		A	LP2			O	MULT	O	STO	
		A	TEMP1			A	TWO			
		A	WRL			O	SUB	A1	KP3	
		A2	KP3	O	MULT	O	STO	A1	TEMP2	
		A2	TEMP 2	A	TWO			A1	WRK	
		A2	WRK	O	SUB	A	LP4			
		O	MULT	O	STO	A	TEMP3			
		A	DTN			A	TEMP1			
		A	KZMKO(x)			O	MULT	A1	TEMP O	
		A2	TEMP O	A	LZMDN			O	MULT	
		O	SUB-s	A	TEMP2			A1	LZMUP	
		A2	LZMUP	O	MULT	O	SUB-s	A1	TEMP 3	
		A2	TEMP3	A	KZMK2(x)			O	MULT	
		O	ADD-s	O	MULT-s	O	STO-h	A1	DU	
		A2	DU	A	DTN			A1	TEMPO	
		A2	TEMPO	A	LRMDN			O	MULT	
		A	TEMP1			A	KRMKO (x)			
		O	MULT	O	SUB-s	A	TEMP 2			
		A	LRMUP			O	MULT	O	ADD-s	
		A	TEMP3			A	KRMK2(x)			
		O	MULT	O	SUB-s	O	MULT-s	O	STO-h	
		A	DV			A	VTO (x)			
		O	ADD-s	O	STO	A	VTO (x)			
		A	UTO(x)			O	ADD-s	O	STO-h	
		A	UTO(x)			A	DTNUP			
		O	MULT-s	A	RTO(x)			O	ADD	
		O	STO	A	RBK2(x)			A1	DTNUP	
		A2	DTNUP	A	VTO (x)			O	MULT	
		A	ZTO(x)			O	ADD	O	STO	
		A	ZCK2(x)			A	RTO (x)			
		A	RBK2(x)			O	ADD	A1	EXP1	
		A2	EXP1	O	ADX-s	O	STO	A1	RCK2 (x)	
		A2	RCK2 (x)	A	ZTO (x)			A1	ZBK2 (x)	
		A2	ZBK2 (x)	O	ADD	A	EXP1			
		O	ADX-s	O	STO	A	ZCK2 (x)			
		A	RBK2(x)			O	ENT	O	STO	
		A	RTO(x)			A	ZBK2(x)			
		O	ENT	O	STO	A	ZTO(x)			
		A	UTO (x)			O	ENT	O	ENT-s	
		O	MULT-s	A	VTO (x)			O	ENT	
		O	ENT-s	O	MULT-s	O	ADD-s	O	STO	
		A	U2K2(x)			A	TEST 4			
		A	ZERO			O	EQL	O	IF-s	
		C	BOT	A	OUT			A1	TEST 5	
		A2	TEST 5	A	ZERO			O	EQL	



BITS		0		12		24		36	47	
LOC	LABEL		SYLLABLE 1		SYLLABLE 2		SYLLABLE 3		SYLLABLE 4	REMARKS
		O	IF-s	C	BOT	A	OUT			
		A	AN			A	EXPM1			
		O	ADX	O	STO	A	AMN			
		A	AN			O	CHS	A1	EXPM2	
		A2	EXPM2	O	ADX	O	STO-h	A1	AM3DN	
		A2	AM3DN	O	MULT	O	STO	A1	AM4DN	
		L	3.0							
		A2	AM4DN	A	RBN			O	ENT	
		O	STO	A	RAN			A1	ZBN	
		A2	ZBN	O	ENT	O	STO	A1	ZAN	
		A2	ZAN	A	EXPM2			A1	RAKO(X)	
		A2	RAKO(X)	A	RAK2(X)			A1	EXP1	
		A2	EXP1	O	ADX	O	ADD-s	A1	RT2(X)	
		A2	RT2(X)	O	ADD	O	ADX-s	O	STO	
		A	RBN			A	EXPM2			
		A	ZAKO(X)			A	EXP1			
		A	ZAK2(X)			O	ADX	O	ADD-s	
		A	ZT2(X)			O	ADD	O	ADX-s	
		O	STO	A	ZBN			A1	AK1(X)	
		A2	AK1(X)	A	FLT1(X)			O	MULT	
		O	STO-h	A	ALA			A1	ALN	
		A2	ALN	O	ADD	O	STO	A1	ALN	
		A2	ALN	A	AK1(X)			A1	AM3DN	
		A2	AM3DN	O	ADD-h	O	STO	A1	AM3UP	
		A2	AM3UP	A	AM4DN			O	ADD	
		O	STO	A	AM4UP			A1	AM3UP	
		A2	AM3UP	O	PDS	O	LSS-h	O	IF-s	
		C	BOF	A	SEC9(1)			A1	AM3DN	
		A2	AM3DN	O	LSS-h	O	IF-s	C	BOF	
		A	LINEE(4)			A	AM4UP			
		O	LSS-h	O	IF-s	C	BOF	A1	LINE13(2)	
		A2	LINE13(2)	A	AM4DN			O	LSS-h	
		O	IF-s	C	BOF	A	LINE14(1)			
	LINEE(4)	C	BUN	A	SEC9(1)			A1	AM4UP	
		A2	AM4UP	O	LSS-h	O	IF-s	C	BOF	
		A	LINE15(4)			A	AMN			
		A	FLT1(X)			O	MULT	O	STO	
		A	ALMN			C	BUN	A1	SEC9(1)	
	LINE13(2)	A2	SEC9(1)	A	ALA			A1	ALMN	
		A2	ALMN	O	ADD	O	STO	A1	ALMN	
		A2	ALMN	C	BUN	A	SEC9(1)			
	LINE14(1)	A	AM4DN			O	CHS	A1	FLT1(X)	
		A2	FLT1(X)	O	MULT	A	ALMN			
		O	ADD	O	STO	A	ALMN			
	LINE15(4)	C	BUN	A	SEC9(1)			A1	AM3UP	
		A2	AM3UP	A	FLT1(X)			O	MULT	
		O	STO	A	ALMN					

BITS		0		12		24		36	47	
LOC	LABEL		SYLLABLE 1		SYLLABLE 2		SYLLABLE 3		SYLLABLE 4	REMARKS
	SEC9 (1)	A	TEST6			A	ZERO			
		O	EQL	O	IF-s	C	BOT	A1	OUT	
		A2	OUT	A	FDK2(X)			A1	FDKO(X)	
		A2	FDKO(X)	O	ADD	A	FAK4(X)			
		O	ADD	A	FAK1(X)			O	ADD	
		A	EXPM2			O	ADX	O	STO-h	
		A	FD			A	U2K2(X)			
		A	U2KO(X)			O	ADD	A1	U2K4(X)	
		A2	U2K4(X)	O	ADD	A	U2K3(X)			
		O	ADD	O	STO-h	A	U2			
		O	ADD-s	A	DS			O	ABS	
		O	ADD-s	A	ZERO			O	EQL	
		O	IF-s	C	BOF	A	LAB2(4)			
		O	PDS	O	STO	A	TEST7			
	LAB2(4)	C	BUN	A	LAB21(2)			A1	ONE	
		A2	ONE	O	ENT	O	STO	A1	TEST7	
	LAB21(2)	A2	TEST7	A	RBK2(X)			A1	RBKO(X)	
		A2	RBKO(X)	O	SUB			A1	ZBKO(X)	
		A2	ZBKO(X)	A	ZBK4(X)			O	SUB	
		O	MULT-s	A	RBKO(X)			A1	RBK4(X)	
		A2	RBK4(X)	O	SUB	A	ZBK2(X)			
		A	ZBKO(X)			O	SUB	O	MULT-s	
		O	SUB-s	A	RBK2(X)			A1	RBKO(X)	
		A2	RBKO(X)	O	SUB	A	ZBK2(X)			
		A	ZBK3(X)			O	SUB	O	MULT-s	
		A	RBK2(X)			A	RBK3(X)			
		O	SUB	A	ZBK2(X)			A1	ZBKO(X)	
		A2	ZBKO(X)	O	SUB	O	MULT-s	O	SUB-s	
		A	RBK3(X)			A	RBK4(X)			
		O	SUB	A	ZBK2(X)			A1	ZBK3(X)	
		A2	ZBK3(X)	O	SUB	O	MULT-s	A1	RBK2(X)	
		A2	RBK2(X)	A	RBK3(X)			O	SUB	
		A	ZBK3(X)			A	ZBK4(X)			
		O	SUB	O	MULT-s	O	SUB-s	A1	RBK3(X)	
		A2	RBK3(X)	A	RBK4(X)			O	SUB	
		A	ZBKO(X)			A	ZBK4(X)			
		O	SUB	O	MULT-s	A	RBKO(X)			
		A	RBK4(X)			O	SUB	A1	ZBK3(X)	
		A2	ZBK3(X)	A	ZBK4(X)			O	SUB	
		O	MULT-s	O	SUB-s	O	ADD-s	O	ADD-s	
		O	ADD-s	A	EXPM2			O	ADX-s	
		O	STO-h	A	FJTO(X)			A1	RBK4(X)	
		A2	RBK4(X)	A	RBKO(X)			O	ADD	
		A	RBK2(X)			O	ADD	A1	RBK3(X)	
		A2	RBK3(X)	O	ADD	A	EXPM2			
		O	ADX-s	O	STO-h	A	RHA			
		O	MULT-s	O	STO	A	RJ			

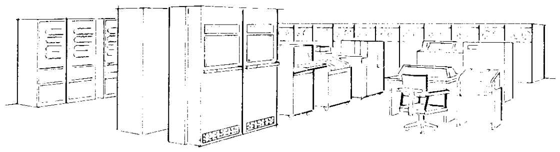


BITS 0 12 24 36 47

LOC	LABEL		SYLLABLE 1		SYLLABLE 2		SYLLABLE 3		SYLLABLE 4	REMARKS
		A	TTO(x)			A	RJ			
		A	MTO(X)			0	DIV	0	STO-h	
		0	ADD-s	A	EXPM1			0	ADX-s	
		0	STO	A	TC			A1	EXPM1	
		A2	EXPM1	A	RBK2(x)			A1	RBK0(x)	
		A2	RBK0(x)	0	SUB	A	RBK3(x)			
		A	RBK4(x)			0	SUB	0	ADD-s	
		0	ADX-s	0	STO-h	A	KRB			
		0	ENT-s	0	MULT-s	A	EXPM1			
		A	ZBK2(x)			A	ZBK0(x)			
		0	SUB	A	ZBK3(x)			A1	ZBK4(x)	
		A2	ZBK4(x)	0	SUB	0	ADD-s	0	ADX-s	
		0	STO-h	A	KZB			0	ENT-s	
		0	MULT-s	A	EXPM1			A1	RBK2 (x)	
		A2	RBK2(x)	A	RBK3(x)			0	SUB	
		A	RBK0(X)			A	RBK4 (x)			
		0	SUB	0	ADD-s	0	ADX-s	0	STO-h	
		A	LRB			0	ENT-s	0	MULT-s	
		A	EXPM1			A	ZBK2(x)			
		A	ZBK3(x)			0	SUB	A1	ZBK0(X)	
		A2	ZBK0(x)	A	ZBK4(x)			0	SUB	
		0	ADD-s	0	ADX-s	0	STO-h	A1	LZB	
		A2	LZB	0	ENT-s	0	MULT-s	0	ADD-s	
		0	STO	A	LRLRB			0	ADD-s	
		0	STO	A	KRKRB			A1	UK2 (x)	
		A2	UK2(x)	A	UK0(x)			0	SUB	
		A	LZB			0	MULT	A1	VK2 (x)	
		A2	VK2(x)	A	VK0(x)			0	SUB	
		A	LRB			0	MULT	0	SUB-s	
		0	STO	A	UR1			A1	VK2 (x)	
		A2	VK2(x)	A	VK3(x)			0	SUB	
		A	KRB			0	MULT	A1	UK2(x)	
		A2	UK2(x)	A	UK3(x)			0	SUB	
		A	KZB			0	MULT	0	SUB-s	
		0	STO	A	UR2			A1	UK3 (x)	
		A2	UK3(x)	A	UK4 (x)			0	SUB	
		A	LZB			0	MULT	A1	VK3 (x)	
		A2	VK3 (x)	A	VK4 (x)			0	SUB	
		A	LRB			0	MULT	0	SUB-s	
		0	STO	A	UR3			A1	VK0 (x)	
		A2	VK0(x)	A	VK4(x)			0	SUB	
		A	KRB			0	MULT	A1	UK0 (x)	
		A2	UK0(x)	A	UK4(x)			0	SUB	
		A	KZB			0	MULT	0	SUB-s	
		0	STO	A	UR4			A1	UR1	
		A2	UR1	0	ENT	0	ENT-s	0	MULT-s	
		A	EXP1			0	ADX-s	A1	TC	

BITS 0 12 24 36 47

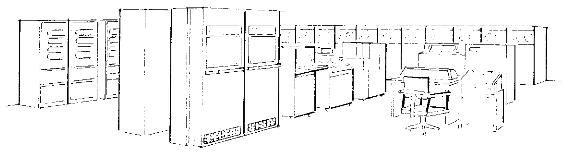
LOC	LABEL		SYLLABLE 1		SYLLABLE 2		SYLLABLE 3		SYLLABLE 4	REMARKS
		A2	TC	A	LRLRB			O	MULT	
		O	DIV-s	O	STO-h	A	G1			
		A	UR2			O	ENT	O	ENT-s	
		O	MULT-s	A	EXP1			O	ADX-s	
		A	TC			A	KRKRB			
		O	MULT	O	DIV-s	O	STO-h	A1	G2	
		A2	G2	A	UR3			O	ENT	
		O	ENT-s	O	MULT-s	A	EXP1			
		O	ADX-s	A	TC			A1	LRLRB	
		A2	LRLRB	O	MULT	O	DIV-s	O	STO-h	
		A	G3			A	UR4			
		O	ENT	O	ENT-s	O	MULT-s	A1	EXP1	
		A2	EXP1	O	ADX-s	A	TC			
		A	KRKRB			O	MULT	O	DIV-s	
		O	STO-h	A	G4			O	ADD-s	
		O	ADD-s	O	ADD-s	A	EXPM1			
		O	ADX-s	O	STO	A	Q			
		A	G2			A	G4			
		O	ADD	A	EXPM1			O	ADX-s	
		A	G1			O	ADD	O	STO	
		A	Q1TO(x)			A	G1			
		A	G3			O	ADD	A1	EXPM1	
		A2	EXPM1	O	ADX-s	A	G2			
		O	ADD	O	STO	A	Q2TO(x)			
		A	G2			A	G4			
		O	ADD	A	EXPM1			O	ADX-s	
		A	G3			O	ADD	O	STO	
		A	Q3TO(x)			A	G1			
		A	G3			O	ADD	A1	EXPM1	
		A2	EXPM1	O	ADX-s	A	G4			
		O	ADD	O	STO	A	Q4TO(x)			
		A	TEST8			A	ZERO			
		O	EQL	O	IF-s	C	BOT	A1	OUT	
		A2	OUT	A	TA			A1	TTO(x)	
		A2	TTO(x)	O	SUB	O	STO-h	A1	MDT	
		A2	MDT	O	PDS	O	LSS	O	IF-s	
		C	BOT	A	OUT			A1	MDT	
		A2	MDT	A	PKO(x)			A1	Q	
		A2	Q	O	ADD	O	MULT-s	A1	DS	
		A2	DS	O	ADD	O	STO	A1	DE	
		A2	DE	A	THCTO(x)			A1	THATO(x)	
		A2	THATO(x)	O	SUB	A	EXP1			
		O	ADX-s	O	STO-h	A	DTHE			
		A	THATO(x)			O	ADD	O	STO-h	
		A	THE			O	ENT-s	O	MULT-s	
		O	ENT-s	O	MULT-s	A	THATO(x)			
		O	ENT	O	ENT-s	O	MULT-s	O	ENT-s	



BITS		0		12		24		36	47	
LOC	LABEL		SYLLABLE 1		SYLLABLE 2		SYLLABLE 3		SYLLABLE 4	REMARKS
		O	MULT-s	O	SUB-s	O	STO	A1	DPHE	
		A2	DPHE	A	TTO(x)			A1	TA	
		A2	TA	O	SUB	O	STO-h	A1	DTTE	
		A2	DTTE	O	ABS	A	TA			
		O	GTR	O	IF-s	C	BOF	A1	OUT	
		A2	OUT	A	THATO(X)			A1	TA	
		A2	TA	A	THSTR			C	CSE	
		A	EOSEF			O	STO	A1	E	
		A2	E	A	THATO(X)			A1	TC	
		A2	TC	A	THSTR			C	CSE	
		A	EOSEF			O	STO	A1	ED1	
		A2	ED1	A	THE			A1	TC	
		A2	TC	A	THSTR			C	CSE	
		A	EOSEF			O	STO	A1	ED2	
		A2	ED2	A	THCTO(X)			A1	TA	
		A2	TA	A	THSTR			C	CSE	
		A	EOSEF			O	STO	A1	ED3	
		A2	ED3	A	THCTO(X)			A1	TTO(X)	
		A2	TTO(X)	A	THSTR			C	CSE	
		A	EOSEF			O	STO	A1	ED4	
		A2	ED4	A	ED2			A1	ED1	
		A2	ED1	O	SUB	A	DTHE			
		O	DIV	O	STO	A	DD1			
		A	ED4			A	ED3			
		O	SUB	A	DTTE			O	DIV	
		O	STO	A	DD2			A1	ED2	
		A2	ED2	A	ED1			O	SUB	
		A	DPHE			O	DIV	O	STO	
		A	DD3			A	DE			
		A	MDT			A	DD2			
		O	MULT	O	ADD-s	A	DD1			
		O	DIV	O	STO	A	DTHTO(X)			
		A	CHIO(X)			A	ZERO			
		O	GTR	O	IF-s	C	BOF	A1	OUT	
		A2	OUT	A	CHIO(X)			A1	FTO(x)	
		A2	FTO(X)	O	SUB	O	STO	A1	DF	
		A2	DF	A	DTHQ(X)			A1	EXPM1	
		A2	EXPM1	O	ADX	A	THATO(X)			
		O	ADD	O	ENT-s	O	MULT-s	O	ENT-s	
		O	MULT-s	A	CHIO(X)			O	ADD	
		A	CHI(X)			O	SUB	O	STO	
		A	CHIO(X)			A	THATO(X)			
		L	0.02							
		O	GTR	O	IF-s	C	BOF	A1	OUT	
		A2	OUT	A	E			A1	ETO(X)	
		A2	ETO(X)	A	MTO(X)			O	DIV	
		O	SUB-s	O	STO	A	DJ1			

BITS 0 12 24 36 47

LOC	LABEL		SYLLABLE 1		SYLLABLE 2		SYLLABLE 3		SYLLABLE 4	REMARKS
		A	DETO (x)			A	DTNON			
		A	DF			O	MULT	A1	MTO (x)	
		A2	MTO (x)	A	EXP1			O	ADX-s	
		O	DIV-s	A	CFT2(x)			O	DIV	
		O	ADD-s	O	STO-h	A	DJ2			
		A	DJ1			O	RSUB			
		L	10**-4							
		A	E			O	DIV	A1	E	
		L	0.02							
		A2	E	O	MULT	O	ADD-s	A1	DJ1	
		A2	DJ1	O	ABS	O	ADD-s	A1	DJ2	
		A2	DJ2	O	ABS	O	ADD-s	O	DIV-s	
		O	STO	A	FJ1TO(X)			A1	PTO (x)	
		L	10**-10							
		A2	PTO (x)	O	LEQ	O	IF-s	C	BOF	
		A	OUT			A	E			
		L	10**-10							
		O	GTR	O	IF-s	C	BOF	A1	OUT	
		A2	OUT	A	PTO (x)			A1	TA	
		A2	TA	O	MULT	O	ENT-s	A1	E	
		A2	E	O	DIV	O	ADD-s	O	STO-h	
		A	C2			A	EXP2			
		O	ADX-s	A	KRKRB			A1	LRLRB	
		A2	LRLRB	O	ADD	O	MULT-s	A1	FJTO (X)	
		A2	FJTO(X)	O	ENT	O	ENT-s	O	MULT-s	
		O	RDIV-s	O	STO-h	A	DTC2			
		A	DTC2N			O	GTR	O	IF-s	
		C	BOT	A	OUT			A1	Q	
		A2	Q	O	IF-s	C	BOF	A1	OUT	
		A2	OUT	A	FJTO(x)			O	CHS	
		L	10.0							
		A	UR1			A	UR2			
		O	ADD	O	MULT-s	O	DIV-s	O	STO	
		A	DTQ			A	E			
		A	MTO (X)			O	MULT	O	STO	
		A	ETO (X)			A	U2			
		A	MTO(X)			O	MULT	A1	EXPM3	
		A2	EXPM3	O	ADX-s	O	STO	A1	DEL	
		A2	DEL	A	CHIO(X)			O	GTR	
		L	10**-8							
		O	IF-s	C	BOF	A	OUT			
		A	BFL			A	DF			
		O	SUB	A	CHIO(X)					
		L	0.01							
		O	MULT	A	DF			O	ABS	
		L	10**-8							
		O	ADD	O	ADD-s					

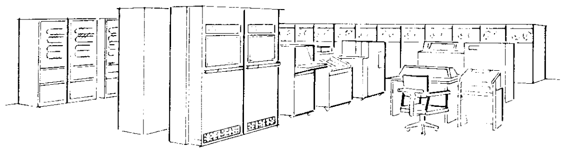


BITS 0 12 24 36 47

LOC	LABEL		SYLLABLE 1		SYLLABLE 2		SYLLABLE 3		SYLLABLE 4	REMARKS
		A	CFT2(x)			A	LCT2 (x)			
		A	FBK2(x)			A	FBK0(x)			
		0	SUB	0	MULT-s	0	MULT-s	0	ABS-s	
		0	ADD-s	A	CFT2(x)			A1	LCT0(x)	
		A2	LCT0(x)	A	FCK0(x)			A1	FBK0(x)	
		A2	FBK0(x)	0	SUB	0	MULT-s	0	MULT-s	
		0	ABS-s	0	ADD-s	A	CFT2(x)			
		A	KCT0(x)			A	FBK1(x)			
		A	FBK0(x)			0	SUB	0	MULT-s	
		0	MULT-s	0	ABS-s	0	ADD-s	A1	CFT2(x)	
		A2	CFT2(x)	A	KCT4(x)			A1	FBK4(x)	
		A2	FBK4(x)	A	FBK0(x)			0	SUB	
		0	MULT-s	0	MULT-s	0	ABS-s	0	ADD-s	
		L	4.0							
		A	CFT2(x)			A	KL2T2 (x)			
		A	FBK0(x)			A	FBK(x)			
		0	SUB	0	MULT-s	0	MULT-s	0	ABS-s	
		A	CFT2 (x)			A	KL2T4(x)			
		A	FBK0(x)			A	FCK4(x)			
		0	SUB	0	MULT-s	0	MULT-s	0	ABS-s	
		A	CFT2(x)			A	KL4T3(x)			
		A	FBK3(x)			A	FBK0(x)			
		0	SUB	0	MULT-s	0	MULT-s	0	ABS-s	
		A	CFT2(x)			A	KL4T0(x)			
		A	FCK1(x)			A	FBK0(x)			
		0	SUB	0	MULT-s	0	MULT-s	0	ABS-s	
		0	ADD-s	0	ADD-s	0	ADD-s	0	MULT-s	
		0	ADD-s	0	DIV-s	0	STO	A1	LZCK2(x)	
		A2	LZCK2(x)	A	DTNUP			A1	DD3	
		A2	DD3	A	MT0(x)			0	MULT	
		A	EXP1			0	ADX-s	0	DIV-s	
		0	STO	A	CFT2(x)			A1	RCK2(x)	
		A2	RCK2(x)	A	RCK4(x)			0	SUB	
		A	RCK0(x)			0	ADD	A1	RCK3(x)	
		A2	RCK3(x)	0	SUB	A	EXPM1			
		0	ADX-s	0	STO	A	LRC			
		A	RCK2(x)			A	RCK4(x)			
		0	SUB	A	RCK0(x)			0	SUB	
		A	RCK3(x)			0	ADD	A1	EXPM1	
		A2	EXPM1	0	ADX-s	0	STO	A1	KRCUP	
		A2	KRCUP	A	ZCK2(x)			A1	ZCK4(x)	
		A2	ZCK4(x)	0	SUB	A	ZCK0(x)			
		0	ADD	A	ZCK3(x)			0	SUB	
		A	EXPM1			0	ADX-s	0	STO-h	
		A	LZC			A	ZCK2(x)			
		A	ZCK4(x)			0	SUB	A1	ZCK0(x)	
		A2	ZCK0(x)	0	SUB	A	ZCK3(x)			

BITS 0 12 24 36 47

LOC	LABEL		SYLLABLE 1		SYLLABLE 2		SYLLABLE 3		SYLLABLE 4	REMARKS
		O	ADD	A	EXPM1			O	ADX-s	
		O	STO-h	A	KZCUP			O	MULT-s	
		A	KRCUP			A	LRC			
		O	MULT	O	ADD-s	O	STO	A1	KLK2(X)	
		A2	KLK2(X)	A	JL			A1	FJTO(X)	
		A2	FJTO(X)	O	ADD	A	EXPM1			
		O	ADX-s	O	STO	A	JC			
		A	LRC			O	ENT	O	ENT-s	
		O	MULT-S	A	LZC			O	ENT	
		O	ENT-S	O	MULT-s	O	ADD-s	O	STO	
		A	LRCK2(X)			A	FDK2(X)			
		A	EXPM1			O	ADX-S	O	STO	
		A	LP2			A	FDKO(X)			
		A	EXPM1			O	ADX-S	O	STO	
		A	LP4			A	TT1(X)			
		A	ZERO			O	EQL	O	IF-S	
		C	BOT	A	OUT			A1	FAK1(X)	
		A2	FAK1(X)	A	FAKO(X)			O	ADD	
		A	EXPM1			O	ADX-S	O	STO	
		A	KP1			A	FAK4(X)			
		A	FAKO(X)			O	ADD	A1	EXPM1	
		A2	EXPM1	O	ADX-S	O	STO	A1	KP3	
		A2	KP3	A	KP1			A1	TC	
		A2	TC	C	CSE	A	EOSLF			
		O	STO	A	P1			A1	LP2	
		A2	LP2	A	TC			C	CSE	
		A	EOSLF			O	STO	A1	P2	
		A2	P2	A	KP3			A1	TC	
		A2	TC	C	CSE	A	EOSLF			
		O	STO	A	P3			A1	LP4	
		A2	LP4	A	TC			C	CSE	
		A	EOSLF			O	STO	A1	P4	
		A2	P4	A	FAKO(X)			A1	TC	
		A2	TC	C	CSE	A	EOSLF			
		A	EXP2			O	ADX-S	O	STO	
		A	FLTO(X)			A	TEST16			
		A	ZERO			O	EQL	O	IF-S	
		C	BOT	A	OUT			A1	LJ2K2(X)	
		A2	LJ2K2(X)	A	GN			O	MULT	
		O	STO	A	LJ2K2(X)			A1	LJ4	
		A2	LJ4	A	GN			O	MULT	
		O	STO	A	LJ4			A1	LJ2KO(X)	
		A2	LJ2KO(X)	A	ZERO			O	NEQ	
		O	IF-S	C	BOF	A	LJ2K2(X)			
		A	LJ4			O	MULT	A1	EXPM1	
		A2	EXPM1	O	ADX-S	A	LJ2K2(X)			
		A	LJ4			O	ADD	O	DIV-S	



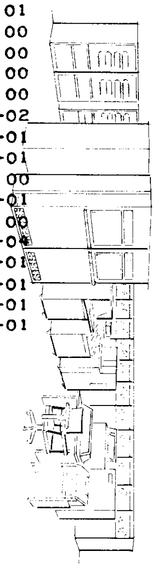
BITS 0 12 24 36 47

LOC	LABEL		SYLLABLE 1		SYLLABLE 2		SYLLABLE 3		SYLLABLE 4	REMARKS
		O	STO-h	A	LJUP			A1	TEST17	
		A2	TEST17	A	ZERO			O	EQL	
		O	IF-s	C	BOT	A	OUT			
		A	LRCK2(x)			A	LRCKO(x)			
		O	ADD	A	EXPM1			O	ADX-s	
		O	STO-h	A	LRLRC			O	MULT-s	
		A	RCK4(x)			A	RCKO(x)			
		O	ADD	A	EXPM1			O	ADX-s	
		O	MULT-s	O	STO	A	LCK2(x)			
		A	LJ1DN			A	LJ3			
		O	MULT	A	LJ1DN			A1	LJ3	
		A2	LJ3	O	ADD	O	DIV-s	A1	EXP1	
		A2	EXP1	O	ADX-s	O	STO-h	A1	LJK3(x)	
		A2	LJK3(x)	A	RCK3(x)			A1	RCK4(x)	
		A2	RCK4(x)	O	ADD	A	EXPM1			
		O	ADX-s	A	KRCUP			O	ENT	
		O	ENT-s	O	MULT-s	A	KRCDN			
		O	ENT	O	ENT-s	O	MULT-s	O	ADD-s	
		A	KZCUP			O	ENT	O	ENT-s	
		O	MULT-s	O	ADD-s	A	KZCDN			
		O	ENT	O	ENT-s	O	MULT-s	O	ADD-s	
		A	EXPM1			O	ADX-s	O	STO-h	
		A	KRKRC			O	MULT-s	O	MULT-s	
		O	STO	A	KCK3(x)			A1	LZCK4(x)	
		A2	LZCK4(x)	A	LZCKO(x)			O	ADD	
		A	LZCK2(x)			O	ADD	A1	LZCK3(x)	
		A2	LZCK3(x)	O	ADD	O	STORE	A1	GPK3(x)	
		A2	GPK3(x)	A	TEST18			O	IF-s	
		C	BOF	A	OUT			A1	LJK3(x)	
		A2	LJK3(x)	A	LJDN			O	MULT	
		A	LJK3(x)			A	LJDN	O	ADD	
		O	RDIV-s	O	STO	A	LJA			
		A	LJK4(x)			A	LJUP			
		O	MULT	A	LJK4(x)			A1	LJUP	
		A2	LJUP	O	ADD	O	RDIV-s	O	STO	
		A	LJC			A	LJK4(x)			
		A	LJDN			O	MULT	A1	LJK4(x)	
		A2	LJK4(x)	A	LJDN			O	ADD	
		O	RDIV-s	O	STO	A	LJB			
		A	LJK3(x)			A	LJUP			
		O	MULT	A	LJK3(x)			A1	LJUP	
		A2	LJUP	O	ADD	O	RDIV-s	O	STO	
		A	LJD			A	RCK4(x)			
		A	EXPM3			O	ADX	A1	KLK4(x)	
		A2	KLK4(x)	A	KLK2(x)			O	ADD	
		A	KLK3(x)			A	EXP2			
		O	ADX	O	ADD-sh	A	LJA			

BITS		0		12		24		36	47	
LOC	LABEL		SYLLABLE 1		SYLLABLE 2		SYLLABLE 3		SYLLABLE 4	REMARKS
		O	MULT	O	STO	A	TEMP			
		A	KLKO(x)			A	EXP2			
		O	ADX	O	ADD-s	A	LJC			
		O	MULT	A	TEMP			O	ADD	
		O	MULT-sh	O	STO	A	KL2K3(x)			
		A	KLKO(x)			A	KLK3 (x)			
		O	ADD	A	KLK4(x)			A1	EXP2	
		A2	EXP2	O	ADX	O	ADD-sh	A1	LJB	
		A2	LJB	O	MULT	O	STO	A1	TEMP	
		A2	TEMP	A	KLK2(x)			A1	EXP2	
		A2	EXP2	O	ADX	O	ADD-s	A1	LJD	
		A2	LJD	O	MULT	A	TEMP			
		O	ADD	O	MULT-s	O	STO	A1	KL4K3(x)	
		A2	KL4K3(x)	A	CFT4(x)			A1	LCK4(x)	
		A2	LCK4(x)	O	MULT-h	O	STO	A1	R1B5(x)	
		A2	R1B5(x)	A	LCK3(x)			O	MULT-h	
		O	STO	A	R1F5(x)			A1	KL2K6(x)	
		A2	KL2K6(x)	O	ENT	O	ENT-s	O	MULT-s	
		O	MULT-sh	O	CHS-s	O	STO	A1	R2B5(x)	
		A2	R2B5(x)	A	KL2K3(x)			O	ENT	
		O	ENT-s	O	MULT-s	O	MULT-sh	O	CHS-s	
		O	STO	A	R2F5(x)			A1	KCK6(x)	
		A2	KCK6(x)	O	MULT-h	O	STO	A1	R3B5(x)	
		A2	R3B5(x)	A	KCK4(x)			O	MULT-h	
		O	STO	A	R3F5(x)			A1	KL4K4(x)	
		A2	KL4K4(x)	O	ENT	O	ENT-s	O	MULT-s	
		O	ENT-s	O	MULT-s	O	MULT-sh	O	STO	
		A	R4B5(x)			A	KL4K7(x)			
		O	ENT	O	ENT-s	O	MULT-s	O	ENT-s	
		O	MULT-s	O	MULT-s	O	STO	A1	R4F5(x)	
		A2	R4F5(x)	C	DTF	A	ANS1			
		C	DTF	A	ANS2			C	DTF	
		A	DATA1	C		C	DTF	A1	DATA1	
		A2	DATA1	C	BUN	A	LAB3(2)			
		L	HYDRO		PROGRAM		IDENTIFIER			
		O	ENT	C	BUN	A	MCP			

SEQUENTIAL PRINT-OUT OF RESULTS AS COMPUTED

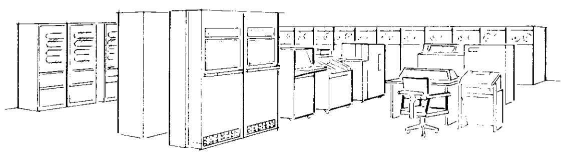
```
TEST1=   .1000000000, 01
TEST1=   .1000000000, 01
TEST1=   .1000000000, 01
TEST1=   .1000000000, 01
TEST1=   .1000000000, 01
TEST1=   .1000000000, 01
TEST1=   .1000000000, 01
TEST1=   .1000000000, 01
TEST1=   .1000000000, 01
TEST1=   .1000000000, 01
TEST9=   .1000000000, 01
TEST8=   .1000000000, 01
TEST7=   .1000000000, 01
TEST6=   .1000000000, 01
TEST5=   .1000000000, 01
TEST4=   .1000000000, 01
TEST3=   .1000000000, 01
TEST2=   .1000000000, 01
TEST1=   .1000000000, 01
TEST9=   .0000000000, 00
X     =   0000000001
TEMPO=   .3625250000,-01
TEMP1=   .8598500000,-01
LRK   =  .1222375000, 00
KRUP  = -.4973250000,-01
TEMPO= -.1585625000, 00
TEMP1=   .2148775000, 00
LZK   =  .5631500000,-01
KZUP  = -.3734400000, 00
JTK   =  .1655067100, 01
AK    =  .2563257000, 00
AIN   =  .9828257000, 00
THA   =  .3445600000, 00
THAT  =  .9902580300, 00
DTHL  =  .6674090000, 00
THCT  =  .6563870000, 00
FBK   =  .3359650000, 00
FAK   =  .1856264500, 00
CFK   =  .9683600000, 00
PS    =  .3259535900, 00
DP    = -.6424064100, 00
PK    = -.2000680500, 00
PAL   =  .1745127100, 00
PT    =  .3259535900, 00
P4UP  =  .1745127100, 00
P3K   =  .1745127100, 00
P2K   =  .1745127100, 00
P1K   =  .1745127100, 00
FJO2T=   .8210000000,-01

FJO1T=   .3445600000, 00
TSO   =  .2519287100, 01
TS1   =  .4094275000, 00
KRMK2=   .1625172000, 00
TS2   =  .4900200000, 00
KZMK2=   .1945074000, 00
WK2   =  .6385534000, 00
TSO   =  .1733567100, 01
TS1   =  .8304575000, 00
TS2   =  .6281050000, 00
LRMUP=   .4790454800, 00
LZMUP=   .3623194000, 00
FDK   =  .6358864500, 00
WUP   =  .1041237500, 01
RAK2  =  .4623300000, 00
ZAK2  =  .7270000000, 00
KP1   = -.2568300000, 00
LP2   =  .5621100000, 00
KP3   = -.7003372900, 00
LP4   = -.5654272900, 00
TM1   =  .8861800000, 00
TM2   =  .6178800000, 00
WRK   =  .1332494000, 01
WRL   =  .1031128500, 01
TEMPO= -.2648247300, 00
TEMP1=   .7490082000, 00
TEMP2= -.6785368400, 00
TEMP3= -.3774261000, 00
DU    =  .3149644400, 00
DV    = -.3777723900, 00
UTO   =  .1298844400, 01
VTO   =  .2273976100, 00
RBK2  =  .1259248900, 01
ZBK2  =  .8665220700, 00
RCK2  =  .8607894500, 00
ZCK2  =  .7967610000, 00
RTO   =  .1259248900, 01
ZTO   =  .8665220700, 00
U2K2  =  .1738706300, 01
AMN   =  .4831500000,-01
AM3DN= -.2415750000,-01
AM4DN= -.7247250000,-01
RAN   =  .9442700000, 00
ZAN   =  .3485300000, 00
RBN   =  .3200600000, 00
ZBN   =  .4353750000, 00
ALA   =  .5364249400, 00
ALN   =  .7752249400, 00
AM3UP=   .9203025000, 00

AM4UP=   .8719875000, 00
ALMN  =  .2744147000,-01
FD    =  .3416375000, 00
U2    =  .2910396300, 01
TEST7=   .1000000000, 01
KRB1  =  .3944589000, 00
KZB1  =  .7885720700, 00
LRB2  =  .1150568900, 01
LZB2  =  .3026520700, 00
KRB3  = -.5864200000, 00
KZB3  = -.1111000000,-01
LRB4  =  .1696900000, 00
LZB4  = -.4970300000, 00
J41   = -.3298706900, 00
J12   = -.7879226900, 00
J23   = -.1646984000, 00
J34   =  .2933535800, 00
FJTO  = -.2472845500, 00
RHA   =  .7319547200, 00
RJ    = -.1810010900, 00
TA    =  .7342200000, 00
TTO   = -.1855507900, 00
TC    =  .2743346000, 00
KRB   = -.9598055000,-01
KZB   =  .3887310300, 00
LRB   =  .6601294500, 00
LZB   = -.9718896500,-01
KRKRB=   .1603240700, 00
LRLRB=   .4452165800, 00
KU1   =  .1174900000, 00
LU2   = -.6156500000, 00
KU3   =  .2797800000, 00
LU4   = -.4533600000, 00
KV1   =  .6600000000,-03
LV2   =  .6228100000, 00
KV3   = -.3179000000,-01
LV4   =  .5903600000, 00
UR1   = -.1185441600,-01
UR2   =  .1795446100, 00
UR3   = -.6206013000,-02
UR4   =  .1195720200, 00
RKR   =  .4547269400, 02
RLR   =  .1637487800, 02
G1    =  .2301115200,-02
G2    =  .1465869800, 01
G3    =  .6306718200,-03
G4    =  .6501443400, 00
Q     =  .1059472900, 01

Q1TO  =  .1060308100, 01
Q2TO  =  .1467335600, 01
Q3TO  =  .1058637600, 01
Q4TO  =  .6516102300, 00
MDT   =  .9197707900, 00
DE    =  .2506045700, 01
DTHE  = -.7253200000, 00
THE   =  .9027000000,-01
DPHE  = -.4424073900, 00
DTTE  = -.9197707900, 00
PANSW=   .8981523200,-01
ANS   =  .8981523200,-01
E     =  .8981523200,-01
PANSW=   .1036557000, 00
ANS   =  .1036557000, 00
ED1   =  .1036557000, 00
PANSW=   .9030315900,-02
ANS   =  .1449185000,-02
ED2   =  .1449185000,-02
PANSW=   .4607826900,-01
ANS   =  .3710263100,-01
ED3   =  .3710263100,-01
PANSW=   .4218570800,-01
ANS   =  .3396830600,-01
ED4   =  .3396830600,-01
DD1   =  .1409123100, 00
DD2   =  .3407724000,-02
DD3   =  .2310235300, 00
DTHTO=   .1780667700, 02
DF    =  .5345000000, 00
CHIO  =  .8222937000, 00
DJ1   =  .4554643000,-02
DJ2   =  .1223619900, 01
FJ1TO= -.9902371400, 00
C2    =  .4327280600, 01
DTC2  =  .5834123500,-02
DTQ   =  .1474651100, 00
ETO   =  .8761296200,-01
DEL   =  .3548791600, 00
BF1F  =  .3987673800,-01
BF1B  =  .1205421300, 00
BF3F  =  .3827137100,-01
BF3B  =  .2185770700, 00
BF2F  = -.9608145600,-02
BF2B  =  .3456238800,-01
BF4F  =  .8135182900,-01
BF4B  =  .9782520200,-01
LZCK2=   .9745435900,-01

CFT2  =  .1361295600, 01
LRC   =  .1253997200, 00
KRCUP=   .6655297000, 00
LZC   =  .3326905000, 00
KZCUP= -.3011950000,-01
KLK2  =  .7343676700,-01
JC    =  .1058027200, 00
LRCK2=   .1264080400, 00
LP2   =  .4929500000,-01
LP4   =  .2918300000, 00
KP1   =  .5096350000, 00
KP3   =  .4015250000, 00
ANS   =  .1064111800,-02
P1    =  .1064111800,-02
ANS   =  .3290528100,-03
P2    =  .3290528100,-03
ANS   =  .9432369600,-03
P3    =  .9432369600,-03
ANS   =  .8030263500,-03
P4    =  .8030263500,-03
ANS   =  .1125236700,-02
FLTO  =  .4500946800,-02
LJ2K2=   .5389631900, 00
LJ4   =  .3336211300, 00
LJUP  =  .4121309400, 00
RH1   =  .2636600000, 00
LRLRC=   .5428290000, 00
LCK2  =  .5898512300,-01
LJK3  =  .4270828700, 00
RH3   =  .5337250000, 00
KRKRC=   .4865759700, 00
KCK3  =  .1109124600, 00
GPK3  =  .2078964300, 01
LJA   =  .2849564500, 00
LJC   =  .2407331700, 00
LJB   =  .3453747500, 00
LJD   =  .2097368400, 00
RH8   =  .8732500000,-02
KL2K3=   .1203709300,-01
KL4K3=   .1361504000,-01
R1B5  =  .2025434500, 00
R1F5  =  .6756749500,-01
R2B5  = -.1994230500, 00
R2F5  = -.2992591100,-04
R3B5  =  .9083835700,-01
R3F5  =  .9494643800,-01
R4B5  =  .2240521100,-01
R4F5  =  .2417961700,-01
```



FINAL RESULTS OF HYDRO PROBLEM COMPUTATIONS

```
  VARIABLES WHICH ARE NOT GIVEN AND ARE COMPUTED

 .25632570, 00  .53642494, 00  .27441470,-01 -.24157500,-01  .92030250, 00 -.72472500,-01  .87198750, 00  .48315000,-01
 .43272806, 01  .96836000, 00  .14091231, 00  .34077240,-02  .23102353, 00  .25060457, 01  .35487916, 00  .53450000, 00
-.64240641, 00 -.44240739, 00  .58341235,-02 -.72532000, 00  .66740900, 00  .17806677, 02  .14746511, 00 -.91977079, 00
 .31496444, 00 -.37777239, 00  .18562645, 00  .33596500, 00  .34163750, 00  .63588645, 00  .82100000,-01 -.99023714, 00
-.24728455, 00  .45009468,-02  .20789643, 01  .10580272, 00  .16550671, 01  .11091246, 00  .12037093,-01  .13615040,-01
 .73436767,-01  .50963500, 00  .40152500, 00  .66552970, 00  .16032407, 00  .48657597, 00  .16251720, 00 -.49732500,-01
-.30119500,-01  .19450740, 00 -.37344000, 00  .58985123,-01  .28495645, 00  .34537475, 00  .24073317, 00  .20973684, 00
 .42708287, 00  .41213094, 00  .49295000,-01  .29183000, 00  .12539972, 00  .12640804, 00  .12223750, 00  .44521658, 00
 .54282900, 00  .47904548, 00  .33269050, 00  .97454359,-01  .56315000,-01  .36231940, 00  .91977079, 00  .10641118,-02
 .32905281,-03  .94323696,-03  .80302635,-03  .17451271, 00  .17451271, 00  .17451271, 00  .17451271, 00  .17451271, 00
-.20006805, 00  .32595359, 00  .10594729, 01  .10586376, 01  .65161023, 00  .20254345, 00  .67567495,-01 -.19942305, 00
-.29925911,-04  .90838357,-01  .94946438,-01  .22405211,-01  .24179617,-01  .46233000, 00  .94427000, 00  .12592489, 01
 .86078945, 00  .87325000,-02  .73195472, 00 -.18100109, 00  .27433460, 00  .34456000, 00  .65638700, 00  .90270000,-01
 .88618000, 00  .29103963, 01  .17387063, 01 -.11854416,-01  .17954461, 00 -.62060130,-02  .11957202, 00  .63855340, 00
 .13324940, 01  .10311285, 01  .10412375, 01  .72700000, 00  .34853000, 00  .86652207, 00  .79676100, 00

  VARIABLES WHICH ARE GIVEN AND ARE RECOMPUTED

        .98282570, 00         .77522494, 00         .13612956, 01         .82229370, 00         .87612962,-01
        .34456000, 00         .53896319, 00         .33362113, 00         .32595359, 00         .10603081, 01
        .14673356, 01         .32006000, 00         .12592489, 01         .99025803, 00         .61788000, 00
       -.18555079, 00         .12988444, 01         .22739761, 00         .43537500, 00         .86652207, 00
```

D 850



APPENDIX B

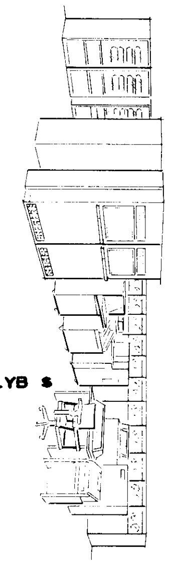
MONTE CARLO TEST PROBLEM

MONTE CARLO TEST PROBLEM STATEMENT IN ALGOL

```
                REFERENCE..MONTE CARLO CALCULATION OF MOLECULAR FLOW
                         RATES THROUGH A CYCLINDRICAL ELBOW AND PIPES
                         OF OTHER SHAPES BY D. H. DAVIS.  JOURNAL OF
                         APPLIED PHYSICS. VOL. 31. NO. 7. 1169-1176
                         JULY. 1960 $

INTEGER         I.J.K.L.N.NMAX.M.U.TALLYA.TALLYB $   REAL OTHERWISE $

ARRAY           NI(8).NIJ(8).MIJ(4.8).SUBPIJ(4.8).IDCHARS(15) $

COMMENT         ALGOL PROCEDURE FOR GENERATING A RANDOM NUMBER

                REFERENCE..A NEW PSEUDO-RANDOM NUMBER GENERATOR
                         BY A. ROTENBERG. JOURNAL OF THE
                         ASSOCIATION FOR COMPUTING MACHINERY.
                         VOL. 7. NO. 1. P. 75. JANUARY 1960 $

PROCEDURE       RANDOM ($U) $
                BEGIN
INTEGER         U $   REAL OTHERWISE $
                U = 1000.U + U + 1357975317 $
                RANDOM() = U . 10**-11 $
                RETURN
                END $


COMMENT         THE FOLLOWING CLOSED SUBROUTINE IS FOR COMPUTING
                          1) SIN PHI. COS PHI
                          2) COS THETA. SIN THETA
                          3) MU(1). MU(2). MU(3) $

SUBROUTINE      MU $
                BEGIN

LOOP1..         U1 = RANDOM($U) $
                U2 = RANDOM($U) $

                TEMP = 2.U1 - 1 $
                TEMP1 = TEMP*2 + U2*2 $

                IF TEMP1 GEQ 1 $    GO TO LOOP1 $

                SINPHI = ( 2.TEMP.U2 ) / TEMP1 $
                COSPHI = ( TEMP*2 - U2*2 ) / TEMP1 $

LOOP2..         U3 = RANDOM($U) $
                U4 = RANDOM($U) $

                IF U3 LSS U4 $    GO TO LOOP2 $

                COSTHETA = U3 $
                SINTHETA = SQRT (1 - COSTHETA*2) $

                MU1 = COSPHI . SINTHETA $
                MU2 = SINPHI . SINTHETA $
                MU3 = COSTHETA $

                RETURN
                END $


START..         WRITE ($$TITLE) $

                FOR I = (1.1.3) $
                BEGIN READ($$INLABELS) $ WRITE($$OUTLABELS.FLABELS) END $

                READ ($$INPUTDATA) $
                WRITE ($$OUTDATA.FDATA) $
                WRITE ($$FLOUTPUT) $

COMMENT         INITIALIZE THE PROGRAM VARIABLES $

                FOR L = (1.1.8) $    NI(L) = NIJ(L) = 0 $

                FOR I = (1.1.4) $
                FOR J = (1.1.8) $
                MIJ(I.J) = 0 $

                N = TALLYA = TALLYB = 0 $
                U = 2438367893 $


COMMENT         COMPUTE THE STARTING COORDINATES X. Y. Z $

NEWMOL..        X = A $

LOOP3..         Y = 2.RANDOM($U) - 1 $
                Z = 2.RANDOM($U) - 1 $
                Y2PLUSZ2 = Y*2 + Z*2 $

                IF Y2PLUSZ2 GEQ 1 $    GO TO LOOP3 $

COMMENT         COMPUTE THE ENTRANCE AREA INDEX I $

                L = 2 . Y2PLUSZ2 $

                EITHER IF Y GTR 0 $
                   M = 1 $
                OTHERWISE $
                   M = 0 $

                I = 2.L + M $


COMMENT         COMPUTE THE INITIAL DIRECTION ANGLES $

                ENTER MU $

                GAMMA = MU1 $
                BETA = MU2 $
                ALPHA = -MU3 $
```

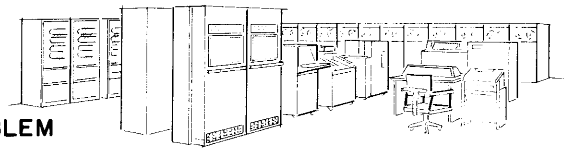
```
COMMENT      COMPUTE THE FIRST DISTANCE S1 AND S2
                 WHERE.. S1 = THE DISTANCE TO WALL A
                 AND..    S2 = THE DISTANCE TO THE X=Y PLANE $

             P = BETA*2 + GAMMA*2 $
             Q = BETA.Y + GAMMA.Z $
             R = Y*2 + Z*2 - 1 $
             TEMP = Q / P $

             S1 = -TEMP + SQRT (TEMP*2 - R/P) $

             TEMP = (X-Y) / (BETA-ALPHA) $

             EITHER IF TEMP GEQ 0 $
                S2 = TEMP $
             OTHERWISE $
                S2 = SMAX $


             IF S2 LSS S1 $   GO TO CYLINDERB $


COMMENT      THE MOLECULE HAS COLLIDED WITH WALL A
                 COMPUTE 1) THE COORDINATES OF THE COLLISION
                         2) NEW DIRECTION COSINES
                         3) THE DISTANCES S0.S1. AND S2
                                 WHERE..S0 = THE DISTANCE TO OPENING A $


CYLINDERA..  X = X + ALPHA . S1 $
             Y = Y + BETA . S1 $
             Z = Z + GAMMA . S1 $

             ENTER MU $

             ALPHA = -MU2 $
             TEMP = SQRT (Y*2 + Z*2) $
             BETA = (-Z.MU1 - Y.MU3) / TEMP $
             GAMMA = (Y.MU1 - Z.MU3) / TEMP $

             TEMP = (A-X) / ALPHA $

             EITHER IF TEMP GEQ 0 $
                S0 = TEMP $
             OTHERWISE $
                S0 = SMAX $

             P = BETA*2 + GAMMA*2 $
             Q = BETA.Y + GAMMA.Z $
             S1 = -2.Q / P $

             TEMP = (X-Y) / (BETA-ALPHA) $

             EITHER IF TEMP GEQ 0 $
                S2 = TEMP $
             OTHERWISE $
                S2 = SMAX $

             IF (S0 LSS S1) AND (S0 LSS S2) $   GO TO LOCTALLYA $
```

```
             IF S1 LSS S2 $   GO TO CYLINDERA $

COMMENT      THE MOLECULE HAS ENTERED CYLINDER B
                 COMPUTE THE DISTANCES S3 AND S4
                      WHERE.. S3 = THE DISTANCE TO WALL B
                      AND..   S4 = THE DISTANCE TO OPENING B $

CYLINDERB..  P = ALPHA*2 + GAMMA*2 $
             Q = ALPHA.X + GAMMA.Z $
             R = X*2 + Z*2 - 1 $
             TEMP = Q / P $

             S3 = -TEMP + SQRT (TEMP*2 - R/P) $

             TEMP = (B-Y) / BETA $

             EITHER IF TEMP GEQ 0 $
                S4 = TEMP $
             OTHERWISE $
                S4 = SMAX $

             IF S4 LSS S3 $   GO TO LOCTALLYB $

COMMENT      THE MOLECULE HAS COLLIDED WITH WALL B
                 COMPUTE 1) THE COORDINATES OF THE COLLISION
                         2) NEW DIRECTION COSINES
                         3) THE DISTANCES S2.S3 AND S4 $

LOOP4..      X = X + ALPHA . S3 $
             Y = Y + BETA . S3 $
             Z = Z + GAMMA . S3 $

             ENTER MU $

             TEMP = SQRT (X*2 + Z*2) $
             ALPHA = (-Z.MU1 - X.MU3) / TEMP $
             BETA = MU2 $
             GAMMA = (X.MU1 - Z.MU3) / TEMP $

             TEMP = (X-Y) / (BETA-ALPHA) $

             EITHER IF TEMP GTR 0 $
                S2 = TEMP $
             OTHERWISE $
                S2 = SMAX $

             P = ALPHA*2 + GAMMA*2 $
             Q = ALPHA.X + GAMMA.Z $
             S3 = -2.Q / P $

             TEMP = (B-Y) / BETA $

             EITHER IF TEMP GEQ 0 $
                S4 = TEMP $
             OTHERWISE $
                S4 = SMAX $

             IF (S4 LSS S2) AND (S4 LSS S3) $   GO TO LOCTALLYB $

             IF S3 LSS S2 $   GO TO LOOP4 $
```



```
COMMENT       THE MOLECULE HAS ENTERED CYLINDER A
                  COMPUTE THE DISTANCES SO AND S1 $

              TEMP = (A-X) / ALPHA $

              EITHER IF TEMP GEQ 0 $
                 SO = TEMP $
              OTHERWISE $
                 SO = SMAX $

              P = BETA*2 + GAMMA*2 $
              Q = BETA.Y + GAMMA.Z $
              R = Y*2 + Z*2 - 1 $
              TEMP = Q / P $

              S1 = -TEMP + SQRT (TEMP*2 - R/P) $

              IF S1 LSS SO $   GO TO CYLINDERA $


COMMENT       COMPUTE THE EXIT COORDINATES AND CORRESPONDING AREA INDEXS

LOCTALLYA..   TALLYA = TALLYA + 1 $

              X = X + ALPHA . SO $
              Y = Y + BETA . SO $
              Z = Z + GAMMA . SO $

              L = 2.(Y*2 + Z*2) $

              EITHER IF Y GTR 0 $
                 M = 1 $
              OTHERWISE $
                 M = 0 $

              J = 2.L + M $
              GO TO UPDATE $

LOCTALLYB..   TALLYB = TALLYB + 1 $

              X = X + ALPHA . S4 $
              Y = Y + BETA . S4 $
              Z = Z + GAMMA . S4 $

              L = 2.(X*2 + Z*2) $

              EITHER IF X GTR 0 $
                 M = 1 $
              OTHERWISE $
                 M = 0 $

              J = 4 + 2.L + M $

UPDATE..      NI(I+1) = NI(I+1) + 1 $
              NIJ(J+1) = NIJ(J+1) + 1 $
              MIJ(I+1,J+1) = MIJ(I+1,J+1) + 1 $

              N = N + 1 $
```

```
              IF MOD(N,K) NEQ 0 $   GO TO CONTINUE $

              TEMP = TALLYB $
              TEMP1 = TALLYA + TALLYB $
              PIJ = TEMP / TEMP1 $
              SIGMAPIJ = SQRT (PIJ . (1 - PIJ) / TEMP1) $

              FOR I = (1,1,4) $
              FOR J = (1,1,8) $
              SUBPIJ(I,J) = MIJ(I,J) / NI(I) $

              WRITE ($$OUTANSWERS,FANSWERS) $
              WRITE ($$OUTNI,FNI) $
              WRITE ($$OUTNIJ,FNIJ) $
              WRITE ($$FLSUBPIJ) $
              FOR I = (1,1,4) $   WRITE ($$OUTSUBPIJ,FSUBPIJ) $
              WRITE ($$NEWPAGE) $

CONTINUE..    IF N LEQ NMAX $   GO TO NEWMOL $

              GO TO START $

INPUT
INLABELS      (FOR J = (1,1,15) $ IDCHARS(J) ),
INPUTDATA     (A,B,SMAX,NMAX,K) $

OUTPUT
OUTLABELS     (FOR J = (1,1,15) $ IDCHARS(J) ),
OUTDATA       (A,B,SMAX,K,NMAX),
OUTANSWERS    (N,PIJ,SIGMAPIJ,TALLYA,TALLYB),
OUTNI         (FOR L = (1,1,4) $ NI(L)),
OUTNIJ        (FOR L = (1,1,8) $ NIJ(L)),
OUTSUBPIJ     (FOR J = (1,1,8) $ SUBPIJ(I,J)) $

FORMAT
TITLE         (B35,*MONTE CARLO TEST PROBLEM FOR THE AEC AT *,
                  *LIVERMORE*,W3,W4),
FLABELS       (15A5,W0),
FDATA         (W4,B35,*INPUT DATA.. RATIO A =*,X5.1,W4,
                  B48,*RATIO B =*,X5.1,W4,
                  B48,*MAXIMUM LENGTH SM =*,X6.1,W4,
                  B48,*PRINT CYCLE =*,I6,W4,
                  B48,*NUMBER OF ITERATIONS =*,I7,W4),
FLOUTPUT      (W4,B34,*OUTPUT DATA...*,W4),
FANSWERS      (W4,B49,*CYCLE NUMBER =*,I7,W4,
               W4,B44,*PROBABILITY  P (A-B) =*,X10.8,W4,
               B38,*STANDARD DEVIATION SIGMA P(I-J) =*,X10.8,W4,W4,
               B34,*NUMBER OF MOLECULES EXITING VIA OPENING A =*,I7,W4,
               B34,*NUMBER OF MOLECULES EXITING VIA OPENING B =*,I7,W4),
FNI           (W4,B10,*SYMETRY CHECK OF INPUT MOLECULES*,W4,
               B10,*A(I) I=0,1,3  =*,4X10.0,W0),
FNIJ          (B10,*SYMETRY CHECK OF OUTPUT MOLECULES*,W4,
               B10,*A(J) J=0,1,7  =*,8X10.0,W0),
FLSUBPIJ      (W4,B27,*PROBABILITY FOR AREA SUBDIVISIONS P(I-J)   *,
                 *I=0,1,3   AND   J=0,1,7*,W0,W0),
FSUBPIJ       (X23.4,7X11.4,W4),
NEWPAGE       (W3,W4,W4,) $


FINISH $
```



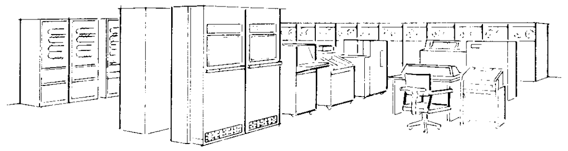
D 850 MACHINE LANGUAGE PROGRAM FOR MONTE CARLO TEST PROBLEM

BITS		0		12		24		36	47	
LOC	LABEL		SYLLABLE 1		SYLLABLE 2		SYLLABLE 3		SYLLABLE 4	REMARKS
	RANDOM	A	U			O	MULT-h	O	ADD-S	
		L	1,00000							
		L	1357975317							
		O	ADD	O	STO-h	A	U			
		L	10**-11							
		O	MULT	O	STO	A	RANDOM			
		C	CRE	O	STO	A	ONE			
		L	1							
		L	2							
		O	STO	A	TWO			O	STO	
		L	O							
		A	ZERO			O	STO	A1	M1	
		L	-1							
		A2	M1	O	STO	A	M2			
		L	-2							
	MU, LOOP 1(1)	A	U			C	CBE	A1	RANDOM	
		A2	RANDOM	O	STO	A	U1			
		A	U			C	CBE	A1	RANDOM	
		A2	RANDOM	O	STO	A	U2			
		A	TWO			A	U1			
		O	MULT	A	ONE			O	SUB	
		O	STO-h	A	TEMP			O	ENT	
		O	ENT-s	O	MULT-s	A	U2			
		O	ENT	O	ENT-s	O	MULT-s	O	ADD-s	
		O	STO-h	A	TEMP 1			A1	ONE	
		A2	ONE	O	GEQ	O	IF-s	C	BOF	
		A	LOOP 1 (1)			A	TWO			
		A	TEMP			O	MULT	A1	U2	
		A2	U2	O	MULT	A	TEMP 1			
		O	DIV	O	STO	A	SINPHI			
		A	TEMP			O	ENT	O	ENT-s	
		O	MULT-s	A	U2			O	ENT	
		O	ENT-s	O	MULT-s	O	SUB-s	A1	TEMP 1	
		A2	TEMP-1	O	DIV	O	STO	A1	COSPHI	
		A2	COSPHI							
	LOOP 2 (1)	A	U			C	CSE	A1	RANDOM	
		A2	RANDOM	O	STO	A	U3			
		A	U			C	CSE	A1	RANDOM	
		A2	RANDOM	O	STO	A	U4			
		A	U3			A	U4			
		O	LSS	O	IF	C	BOF	A1	LOOP 2(1)	
		A2	LOOP 2(1)	A	U3			O	STO-h	
		A	COSTHETA			O	ENT	O	ENT-s	
		O	MULT-s	A	ONE			O	RSUB	
		O	SQRT-s	O	STO-h	A	SINTHETA			
		A	COSPHI			O	MULT	O	STO	
		A	MU 1			A	SINPHI			

D 850

BITS 0 12 24 36 47

LOC	LABEL		SYLLABLE 1		SYLLABLE 2		SYLLABLE 3		SYLLABLE 4	REMARKS
		A	SINTHETA			O	MULT	O	STO	
		A	MU 2			A	COSTHETA			
		O	STO	A	MU3			C	CRE	
	START (1)	B	DTF	A	TITLE			A0	LDR1	
		L	O							
	LAB 1 (1)	A0	INC 1	C	DTF	A	OUTLABELS			
		A0	TEQ1	L	20			A0	ABT	
		A	LAB 1(1)			C	DTF	A1	INPUTDATA	
		A2	INPUTDATA	A0	LRD 2	L	O			
	LAB 2 (2)	A0	SRF 2	A0	INC 2	O	STO-h	A1	NI-i	
		L	O							
		A2	NI-i	O	STO-h	A	NIJ-i			
		A0	TEQ 2	L	8			A0	ABT	
		A	LAB 2(2)			O	STO-h	A1	TALLY B	
		A2	TALLYB			O	STO-h	A1	TALLYA	
		A2	TALLYA			O	STO	A1	N	
		A2	N			O	ENT	O	STO	
		L	2438367893							
	NEWMOL (3)	A	U			A	A			
		O	ENT	O	STO	A	X			
	LOOP3 (1)	A	TWO			A	U			
		O	ENT	C	CSE	A	RANDOM			
		O	MULT-s	A	ONE			O	SUB	
		O	STO	A	Y			A1	TWO	
		A2	TWO	A	U			O	ENT	
		C	CSE	A	RANDOM			O	MULT-s	
		A	ONE			O	SUB	O	STO-h	
		A	Z			O	ENT-s	O	MULT-s	
		A	Y			O	ENT	O	ENT-s	
		O	MULT-s	O	ADD-s	O	STO-h	A1	YSQPZSQ	
		A2	YSQPZSQ	A	ONE			O	GEQ	
		O	IF	C	BOF	A	LOOP3(1)			
		A	YSQPZSQ			A	EXP1			
		O	ADX-s	A	Y			A1	ZERO	
		A2	ZERO	O	GTR	O	IF	C	BOT	
		A	LAB3(4)			A	ONE			
		O	ENT	O	STO	A	M			
	LAB3(4)	C	BUN	A	LAB31(2)			A1	ZERO	
		A2	ZERO	O	ENT	O	STO	A1	M	
	LAB31(2)	A2	M	A	TWO			A1	L	
		A2	L	O	MULT	A	M			
		O	ADD	O	STO	A	I			
		C	CSE	A	MU			A1	MU1	
		A2	MU1	O	ENT	O	STO	A1	GAMMA	
		A2	GAMMA	A	MU2			O	ENT	
		O	STO	A	BETA			A1	MU3	
		A2	MU3	O	CHS	O	ENT	O	STO	
		A	ALPHA			A	BETA			

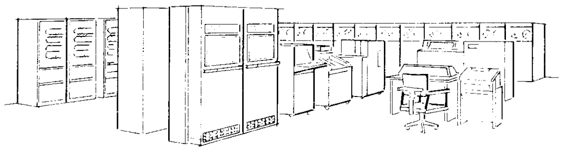


BITS 0 12 24 36 47

LOC	LABEL		SYLLABLE 1		SYLLABLE 2		SYLLABLE 3		SYLLABLE 4	REMARKS
		0	ENT	0	ENT-s	0	MULT-s	A1	GAMMA	
		A2	GAMMA	0	ENT	0	ENT-s	0	MULT-s	
		0	ADD-s	0	STO	A	P			
		A	BETA			A	Y			
		0	ADD	A	GAMMA			A1	Z	
		A2	Z	0	MULT	0	ADD-s	0	STO	
		A	Q			A	Y			
		0	ENT	0	ENT-s	0	MULT-s	A1	Z	
		A2	Z	0	ENT	0	ENT-s	0	MULT-s	
		0	ADD-s	A	ONE			0	SUB	
		0	STO	A	R			A1	Q	
		A2	Q	A	P			0	DIV	
		0	STO-h	A	TEMP			0	CHS-s	
		0	ENT-s	0	MULT-hs	A	R			
		A	P			0	DIV	0	SUB-s	
		0	SQRT-s	0	ADD-s	0	STO	A1	S1	
		A2	S2	A	X			A1	Y	
		A2	Y	0	SUB	A	BETA			
		A	ALPHA			0	SUB	0	DIV-s	
		0	STO-h	A	TEMP			A1	ZERO	
		A2	ZERO	0	GEQ	0	IF	C	BOT	
		A	LAB4(3)			A	TEMP			
		0	STO	A	S2			C	BUN	
	LAB4(3)	A	LAB41(1)			A	SMAX			
		0	ENT	A	S2			0	STO	
	LAB41 (1)	A	S2			A	S1			
		0	LSS	0	IF	C	BOF	A1	CYLB(3)	
	CYLA (2)	A2	CYLB(3)	A	X			A1	ALPHA	
		A2	ALPHA	A	S1			0	MULT	
		0	ADD-s	0	STO	A	X			
		A	Y			A	BETA			
		A	S1			0	MULT	0	ADD-s	
		0	STO	A	Y			A1	Z	
		A2	Z	A	GAMMA			A1	S1	
		A2	S1			0	MULT	0	ADD-s	
		0	STO	A	Z			C	CSE	
		A	MU			A	MU2			
		0	ENT	0	CHS-s	0	STO	A1	ALPHA	
		A2	ALPHA	A	Y			0	ENT	
		0	ENT-s	0	MULT-s	A	Z			
		0	ENT	0	ENT-s	0	MULT-s	0	ADD-s	
		0	SQRT-s	0	STO	A	TEMP			
		A	Z			0	CHS	A1	MU1	
		A2	MU1	0	MULT	A	Y			
		A	MU3			0	MULT	0	SUB-s	
		A	TEMP			0	DIV	0	STO	
		A	BETA			A	Y			

BITS 0 12 24 36 47

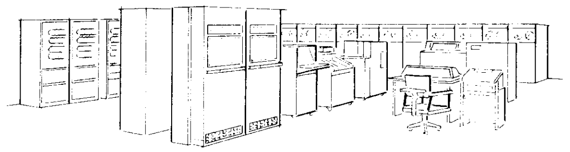
LOC	LABEL		SYLLABLE 1		SYLLABLE 2		SYLLABLE 3		SYLLABLE 4	REMARKS
		A	MU1			O	MULT	A1	Z	
		A2	Z	A	MU3			O	MULT	
		O	SUB-s	A	TEMP			O	DIV	
		O	STO	A	GAMMA			A1	A	
		A2	A	A	X			O	SUB	
		A	ALPHA			O	DIV	O	STO-h	
		A	TEMP			A	ZERO			
		O	GEQ	O	IF	C	BOT	A1	LAB5(3)	
		A2	LAB5(3)	A	TEMP			O	ENT	
		O	STO	A	SO			C	BUN	
	LAB5(3)	A	LAB51(1)			A	SMAX			
		O	ENT	O	STO	A	SO			
	LAB51(1)	A	BETA			O	ENT	O	ENT-s	
		O	MULT-s	A	GAMMA			O	ENT	
		O	ENT-s	O	MULT-s	O	ADD-s	O	STO	
		A	P			A	BETA			
		A	Y			O	MULT	A1	GAMMA	
		A2	GAMMA	A	Z			O	MULT	
		O	ADD-s	O	STO-h	A	Q	A1	TWO	
		A2	TWO	O	CHS	O	MULT	A1	P	
		A2	P	O	DIV	O	STO	A1	S1	
		A2	S1	A	X			A1	Y	
		A2	Y	O	SUB	A	BETA			
		A	ALPHA	O	SUB	O	DIV-s	O	STO-h	
		O	PDS	O	LSS-sh	O	IF	C	BOF	
		A	LAB6(1)			O	STO	A1	S2	
		A2	S2	C	BUN	A	LAB61(3)			
	LAB6(1)	A	SMAX			O	ENT	O	STO	
	LAB61(3)	A	S2			A	SO			
		A	S1			O	LSS	A1	SO	
		A2	SO	A	S2			O	LSS	
		O	AND-s	O	IF-s	C	BOT	A1	LOCTALLYA(2)	
		A2	LOCTALLYA(2)	A	S1			A1	S2	
		A2	S2	O	LSS	O	IF-s	C	BOT	
	CYLB(3)	A	CYLA(2)			A	ALPHA			
		O	ENT	O	MULT-h	A	GAMMA			
		O	ENT	O	MULT	O	ADD-s	O	STO	
		A	P			A	X			
		O	MULT	A	GAMMA			A1	Z	
		A2	Z	O	MULT	O	ADD-s	O	STO	
		A	Q			A	X			
		O	ENT	O	MULT	A	Z			
		O	ENT	O	MULT	O	ADD-s	A1	ONE	
		A2	ONE	O	SUB	O	STO	A1	R	
		A2	R	A	Q			A1	P	
		A2	P	O	DIV	O	STO-h	A1	TEMP	
		A2	TEMP	O	CHS-s	O	ENT-s	O	MULT-sh	



BITS 0 12 24 36 47

LOC	LABEL		SYLLABLE 1		SYLLABLE 2		SYLLABLE 3		SYLLABLE 4	REMARKS
		A	R			A	P			
		O	DIV	O	RSUB-s	O	SQRT-s	O	ADD-s	
		O	STO-h	A	S3			A1	B	
		A2	B	A	Y			O	SUB	
		A	BETA			O	DIV	O	STO-h	
		A	TEMP			O	PDS	O	LSS-h	
		O	IF-s	C	BOF	A	LAB7 (3)			
		O	STO	A	S4			C	BUN	
	LAB7 (3)	A	LAB71(4)			A	SMAX			
	LAB71 (4)	O	STO	A	S4			A1	S4	
		A2	S4	O	LSS	O	IF-s	C	BOT	
	LOOP4 (3)	A	LOCTALLYB(4)			A	X			
		A	ALPHA			A	S3			
		O	MULT	O	ADD-s	O	STO	A1	X	
		A2	X	A	Y			A1	BETA	
		A2	BETA	A	S3			O	MULT	
		O	ADD-s	O	STO	A	Y			
		A	Z			A	GAMMA			
		A	S3			O	MULT	O	ADD-s	
		O	STO	A	Z			C	CSE	
		A	MU			A	X			
		O	ENT	O	ENT-s	O	MULT-s	A1	Z	
		A2	Z	O	ENT	O	ENT-s	O	ADD-s	
		O	SQRT-s	O	STO-h:	A	TEMP			
		A	Z			O	CHS	A1	MU1	
		A2	MU1	O	MULT	A	X			
		A	MU3			O	MULT	O	SUB-s	
		O	DIV	O	STO	A	ALPHA			
		A	MU2			O	ENT	O	STO	
		A	BETA			A	X			
		A	MU1			O	MULT	A1	Z	
		A2	Z	A	MU3			O	MULT	
		O	SUB-s	A	TEMP			O	DIV	
		O	STO	A	GAMMA			A1	X	
		A2	X	A	Y			O	SUB	
		A	BETA			A	ALPHA			
		O	SUB	O	DIV-s	O	STO-h	A1	TEMP	
		A2	TEMP	O	PDS	O	LEQ-h	O	IF-s	
		C	BOF	A	LAB8 (3)			O	STO	
		A	S2			C	BUN	A1	LAB81 (4)	
	LAB8 (3)	A2	LAB81(4)	A	SMAX			O	ENT	
	LAB81 (4)	O	STO	A	S2			A1	ALPHA	
		A2	ALPHA	O	ENT	O	ENT-s	O	MULT-s	
		A	GAMMA			O	ENT	O	ENT-s	
		O	MULT-s	O	ADD-s	O	STO	A1	P	
		A2	P	A	ALPHA			A1	X	
		A2	X	O	MULT			A1	GAMMA	

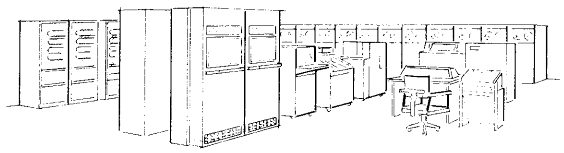
LOC	LABEL		SYLLABLE 1		SYLLABLE 2		SYLLABLE 3		SYLLABLE 4	REMARKS
BITS		0		12		24		36	47	
		A2	GAMMA	A	Z			O	MULT	
		O	ADD-s	O	STO-h	A	Q			
		A	M2			O	MULT	A1	P	
		A2	P	O	STO	A	S3			
		A	B			A	Y			
		O	SUB	A	BETA			O	DIV	
		O	STO-h	A	TEMP			O	PDS	
		O	LSS-h	O	IF-s	C	BOF	A1	LAB9(4)	
		A2	LAB9(4)	O	STO	A	S4			
	LAB9 (4)	C	BUN	A	LAB91(2)			A1	SMAX	
		A2	SMAX	O	ENT	O	STO-h	A1	S4	
	LAB91(2)	A2	S4	A	S2			O	LSS	
		A	S4			A	S3			
		O	AND-s	O	IF-s	C	BOT	A1	LOCTALLYB(4)	
		A2	LOCTALLYB(4)	A	S3			A1	S2	
		A2	S2	O	LSS-s	O	IF-s	C	BOT	
		A	LOOP4(3)			A	A			
		A	X			O	SUB	A1	ALPHA	
		A2	ALPHA	O	DIV	O	STO-h	A1	TEMP	
		A2	TEMP	O	PDS	O	LSS-s	O	IF-s	
		C	BOF	A	LAB10(2)			O	STO	
		A	SO			C	BUN	A1	LAB101(4)	
	LAB10 (2)	A2	LAB101(4)	A	SMAX			O	ENT	
	LAB101(4)	O	STO	A	SO			A1	BETA	
		A2	BETA	O	ENT	O	ENT-s	O	MULT-s	
		A	GAMMA			O	ENT	O	ENT-S	
		O	MULT-s	O	ADD-s	O	STO-h	A1	P	
		A2	P	A	BETA			A1	Y	
		A2	Y	O	MULT	A	GAMMA			
		A	Z			O	MULT	O	ADD-s	
		O	STO-h	A	Q			O	RDIV	
		O	STO	A	TEMP			A1	Y	
		A2	Y	O	ENT	O	ENT-s	O	MULT-s	
		A	Z			O	ENT	O	ENT-s	
		O	MULT-s	O	ADD-s	A	ONE			
		O	SUB	O	STO	A	R			
		A	TEMP			O	ENT	O	ENT-S	
		O	MULT-s	A	R			A1	P	
		A2	P	O	DIV	O	SUB-s	O	SQRT-s	
		A	TEMP			O	SUB	O	STO-h	
		A	S1			A	SO			
		O	LSS-s	O	IF-S	C	BOF	A1	CYLA(2)	
	LOCTALLYA (2)	A2	CYLA(2)	A	TALLYA			A1	ONE	
		A2	ONE	O	ADD	O	STO	A1	TALLYA	
		A2	TALLYA	A	X			A1	ALPHA	
		A2	ALPHA	A	SO			O	MULT	
		O	ADD-s	O	STO	A	X			



BITS 0 12 24 36 47

LOC	LABEL		SYLLABLE 1		SYLLABLE 2		SYLLABLE 3		SYLLABLE 4	REMARKS
		A	Y			A	BETA			
		A	SO			O	MULT	O	ADD-s	
		O	STO	A	Y			A1	Z	
		A2	Z	A	GAMMA			A1	SO	
		A2	SO	O	MULT	O	ADD-s	O	STO-h	
		A	Z			O	ENT-s	O	MULT-s	
		A	Y			O	ENT	O	ENT-s	
		O	MULT-s	O	ADD-s	A	TWO			
		O	MULT	O	STO	A	L			
		A	Y			O	PDS	O	GTR-s	
		O	IF-s	C	BOF	A	LAB11(2)			
		A	ONE			O	ENT	O	STO	
		A	M			C	BUN	A1	LAB111 (2)	
	LAB11 (2)	A2	LAB111 (2)	O	PDS	O	STO	A1	M	
	LAB111 (2)	A2	M	A	TWO			A1	L	
		A2	L	O	MULT	A	M			
		O	ADD	O	STO	A	J			
	LOCTALLYB (4)	C	BUN	A	UPDATE(4)			A1	TALLYB	
		A2	TALLYB	A	ONE			O	ADD	
		O	STO	A	TALLYB			A1	X	
		A2	X	A	ALPHA			A1	S4	
		A2	S4	O	MULT	O	ADD-s	O	STO	
		A	X			A	Y			
		A	BETA			A	S4			
		O	MULT	O	ADD-s	O	STO	A1	Y	
		A2	Y	A	Z			A1	GAMMA	
		A2	GAMMA	A	S4			O	MULT	
		O	ADD-s	O	STO-h	A	Z			
		O	ENT-s	O	MULT-s	A	X			
		O	ENT	O	ENT-s	O	MULT-s	O	ADD-s	
		A	TWO			O	MULT	O	STO	
		A	L			A	X			
		O	PDS	O	GTR-s	O	IF-s	C	BOF	
		A	LAB12(4)			A	ONE			
		O	ENT	O	STO	A	M			
	LAB12 (4)	C	BUN	A	LAB121(4)			O	PDS	
	LAB121 (4)	O	STO	A	M			A1	TWO	
		L	4							
		A2	TWO	A	L			O	MULT	
		A	M			O	ADD	O	ADD-s	
	UPDATE (4)	O	STO	A	J			A0	RRF	
		A0	LDR IXR3	A	I			A0	INC IXR3	
		A0	LDA IXR3	A	NI-i			A1	ONE	
		A2	ONE	O	ADD	O	STO	A1	NI -i	
		A1	NI-i	A0	RRF	A0	LDR IXR4	A1	J	
		A2	J	A0	INC IXR4	A0	LDA IXR4	A1	NIJ-i	
		A2	NIJ -i	A	ONE			O	ADD	

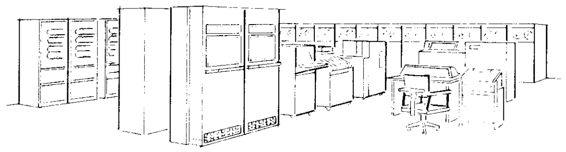
BITS		0		12		24		36	47	
LOC	LABEL		SYLLABLE 1		SYLLABLE 2		SYLLABLE 3		SYLLABLE 4	REMARKS
		O	STO	A	NIJ-i			A1	N	
		A2	N	A	ONE			O	ADD	
		O	STO-h	A	N			A1	K	
		A2	K	O	ENT	O	P_1 (MOD)	O	PDS	
		O	NEQ-s	O	IF-s	C	BOT	A1	CONTINUE	
		A2	CONTINUE	A	TALLYB			O	ENT	
		O	ENT-s	A	TALLYA			O	ADD	
		O	DIV	O	STO-h	A	PIJ			
		O	ENT-s	A	ONE			O	RSUB	
		O	MULT-s	A	TALLYB			O	DIV	
		O	SQRT-s	O	STO			A1	SIGMAIJ	
		A2	SIGMAIJ	C	DTF	A	OUTANSWERS			
		C	DTF	A	OUTNI			C	DTF	
		A	OUTNIJ			C	DTF	A1	OUTSUBPIJ	
	CONTINUE(2)	A2	OUTSUBPIJ	A	N			A1	NMAX	
		A2	NMAX	O	LEQ	O	IF-s	C	BOF	
		A	NEWMOL( )			C	BUN	A1	START( )	
		A2	START( )	C	DTF	A	INPUT DATA			
		L	MONTE		CARLO		PROGRAM		IDENTIFIER	
		O	ENT	C	BUN	A	MCP			



RESULTS OF MONTE CARLO CALCULATIONS AT 500 CYCLE INTERVALS

```
                         INPUT DATA.. RATIO A =  1.0

                                      RATIO B =  2.0

                                      MAXIMUM LENGTH SM =  10.0

                                      PRINT CYCLE =   500

                                      NUMBER OF ITERATIONS =  10000


                        OUTPUT DATA...


                                       CYCLE NUMBER =     500


                                 PROBABILITY  P (A-B) = .43000000

                          STANDARD DEVIATION SIGMA P(I-J) = .02214046


                        NUMBER OF MOLECULES EXITING VIA OPENING A =    285

                        NUMBER OF MOLECULES EXITING VIA OPENING B =    215


SYMMETRY CHECK OF INPUT MOLECULES
A(I) I=0,1,3  =         124.      115.      129.      132.

SYMMETRY CHECK OF OUTPUT MOLECULES
A(J) J=0,1,7  =          69.       60.       78.       78.       60.       54.       47.       54.

                 PROBABILITY FOR AREA SUBDIVISIONS P(I-J)    I=0,1,3    AND    J=0,1,7

     .1612      .1370      .1854      .1532      .0806      .1048      .0806      .0967

     .1826      .1130      .0608      .1565      .1478      .1391      .0956      .1043

     .1395      .1007      .2248      .1317      .1085      .0930      .1085      .0930

     .0757      .1287      .1439      .1818      .1439      .0984      .0909      .1363

                                 ____________

                                       CYCLE NUMBER =    1000


                                 PROBABILITY  P (A-B) = .41500000

                          STANDARD DEVIATION SIGMA P(I-J) = .01558123


                        NUMBER OF MOLECULES EXITING VIA OPENING A =    585

                        NUMBER OF MOLECULES EXITING VIA OPENING B =    415


SYMMETRY CHECK OF INPUT MOLECULES
A(I) I=0,1,3  =         269.      232.      241.      258.

SYMMETRY CHECK OF OUTPUT MOLECULES
A(J) J=0,1,7  =         153.      135.      164.      133.      118.       97.       93.      107.

                 PROBABILITY FOR AREA SUBDIVISIONS P(I-J)    I=0,1,3    AND    J=0,1,7

     .1672      .1338      .1821      .1115      .1078      .0966      .0929      .1078

     .1637      .1465      .1077      .1465      .1120      .1163      .0991      .1077

     .1452      .1120      .2531      .0954      .1037      .0912      .1078      .0912

     .1356      .1472      .1124      .1782      .1472      .0852      .0736      .1201
```

```
                              CYCLE NUMBER =    1500
                         PROBABILITY  P (A-B) = •41800000
                   STANDARD DEVIATION SIGMA P(I-J) = •01273514


                  NUMBER OF MOLECULES EXITING VIA OPENING A =    873
                  NUMBER OF MOLECULES EXITING VIA OPENING B =    627


SYMMETRY CHECK OF INPUT MOLECULES
A(I) I=0•1•3  =      391•      350•      369•      390•
SYMMETRY CHECK OF OUTPUT MOLECULES
A(J) J=0•1•7  =      214•      189•      251•      219•      178•      148•      147•      154•

              PROBABILITY FOR AREA SUBDIVISIONS P(I-J)   I=0•1•3   AND   J=0•1•7

      •1713     •1202     •1713     •1278     •1125     •0920     •0920     •1125
      •1571     •1400     •1142     •1342     •1200     •1171     •1142     •1028
      •1327     •1002     •2655     •1111     •0948     •0921     •1084     •0948
      •1102     •1435     •1179     •2076     •1461     •0948     •0794     •1000
                                  ___________________
                              CYCLE NUMBER =    2000
                         PROBABILITY  P (A-B) = •41900000
                   STANDARD DEVIATION SIGMA P(I-J) = •01103265


                  NUMBER OF MOLECULES EXITING VIA OPENING A =   1162
                  NUMBER OF MOLECULES EXITING VIA OPENING B =    838


SYMMETRY CHECK OF INPUT MOLECULES
A(I) I=0•1•3  =      525•      453•      497•      525•
SYMMETRY CHECK OF OUTPUT MOLECULES
A(J) J=0•1•7  =      287•      245•      334•      296•      243•      200•      192•      203•

              PROBABILITY FOR AREA SUBDIVISIONS P(I-J)   I=0•1•3   AND   J=0•1•7

      •1733     •1161     •1676     •1371     •1066     •0990     •0914     •1085
      •1589     •1258     •1081     •1280     •1368     •1214     •1081     •1125
      •1368     •1126     •2776     •1106     •0925     •0824     •1046     •0824
      •1066     •1352     •1123     •2114     •1504     •0990     •0819     •1028
                                  ___________________
                              CYCLE NUMBER =    2500
                         PROBABILITY  P (A-B) = •42560000
                   STANDARD DEVIATION SIGMA P(I-J) = •00988867


                  NUMBER OF MOLECULES EXITING VIA OPENING A =   1436
                  NUMBER OF MOLECULES EXITING VIA OPENING B =   1064


SYMMETRY CHECK OF INPUT MOLECULES
A(I) I=0•1•3  =      645•      559•      637•      659•
SYMMETRY CHECK OF OUTPUT MOLECULES
A(J) J=0•1•7  =      359•      302•      406•      369•      306•      254•      243•      261•

              PROBABILITY FOR AREA SUBDIVISIONS P(I-J)   I=0•1•3   AND   J=0•1•7

      •1674     •1240     •1674     •1286     •1100     •0976     •0930     •1116
      •1520     •1162     •1144     •1359     •1341     •1234     •1037     •1198
      •1412     •1177     •2574     •1098     •0989     •0863     •1067     •0816
      •1153     •1244     •1062     •2124     •1471     •1016     •0864     •1062
```



```
                         CYCLE NUMBER =    3000

                       PROBABILITY  P (A-B) = .42633333

                 STANDARD DEVIATION SIGMA P(I-J) = .00902908


              NUMBER OF MOLECULES EXITING VIA OPENING A =   1721

              NUMBER OF MOLECULES EXITING VIA OPENING B =   1279


SYMMETRY CHECK OF INPUT MOLECULES
A(I) I=0,1,3  =     754.     675.     771.     800.

SYMMETRY CHECK OF OUTPUT MOLECULES
A(J) J=0,1,7  =     431.     364.     487.     439.     357.     319.     290.     313.

           PROBABILITY FOR AREA SUBDIVISIONS P(I-J)   I=0,1,3   AND   J=0,1,7

     .1657     .1273     .1684     .1220     .1074     .1061     .0954     .1074

     .1570     .1214     .1096     .1303     .1274     .1229     .1125     .1185

     .1387     .1193     .2581     .1141     .1024     .0894     .0933     .0843

     .1162     .1175     .1087     .2137     .1387     .1087     .0875     .1087
```

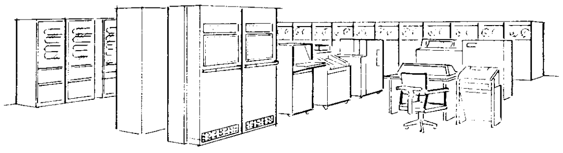
```
                         CYCLE NUMBER =    3500

                       PROBABILITY  P (A-B) = .42828571

                 STANDARD DEVIATION SIGMA P(I-J) = .00836415


              NUMBER OF MOLECULES EXITING VIA OPENING A =   2001

              NUMBER OF MOLECULES EXITING VIA OPENING B =   1499


SYMMETRY CHECK OF INPUT MOLECULES
A(I) I=0,1,3  =     894.     795.     882.     929.

SYMMETRY CHECK OF OUTPUT MOLECULES
A(J) J=0,1,7  =     507.     423.     565.     506.     424.     379.     338.     358.

           PROBABILITY FOR AREA SUBDIVISIONS P(I-J)   I=0,1,3   AND   J=0,1,7

     .1655     .1241     .1610     .1252     .1129     .1085     .0928     .1096

     .1484     .1245     .1207     .1270     .1320     .1232     .1119     .1119

     .1451     .1167     .2494     .1133     .1043     .0952     .0941     .0816

     .1216     .1184     .1130     .2077     .1356     .1076     .0893     .1065
```

```
                         CYCLE NUMBER =    4000

                       PROBABILITY  P (A-B) = .42675000

                 STANDARD DEVIATION SIGMA P(I-J) = .00782039


              NUMBER OF MOLECULES EXITING VIA OPENING A =   2293

              NUMBER OF MOLECULES EXITING VIA OPENING B =   1707


SYMMETRY CHECK OF INPUT MOLECULES
A(I) I=0,1,3  =    1026.     906.    1015.    1053.

SYMMETRY CHECK OF OUTPUT MOLECULES
A(J) J=0,1,7  =     574.     497.     650.     572.     491.     433.     383.     400.

           PROBABILITY FOR AREA SUBDIVISIONS P(I-J)   I=0,1,3   AND   J=0,1,7

     .1578     .1257     .1656     .1247     .1189     .1072     .0964     .1033

     .1412     .1280     .1214     .1280     .1357     .1280     .1081     .1092

     .1487     .1172     .2472     .1172     .1073     .0926     .0896     .0798

     .1263     .1263     .1130     .1984     .1301     .1073     .0902     .1082
```

```
                              CYCLE NUMBER =    4500
                         PROBABILITY  P (A-B) = .42911111
                    STANDARD DEVIATION SIGMA P(I-J) = .00737826


                    NUMBER OF MOLECULES EXITING VIA OPENING A =   2569
                    NUMBER OF MOLECULES EXITING VIA OPENING B =   1931


SYMMETRY CHECK OF INPUT MOLECULES
A(I) I=0,1,3  =     1142.     1031.     1141.     1186.

SYMMETRY CHECK OF OUTPUT MOLECULES
A(J) J=0,1,7  =      651.      555.      730.      633.      562.      477.      436.      456.

            PROBABILITY FOR AREA SUBDIVISIONS P(I-J)   I=0,1,3   AND   J=0,1,7

     .1628     .1243     .1672     .1243     .1217     .1050     .0945     .0998

     .1377     .1280     .1193     .1231     .1406     .1222     .1183     .1105

     .1533     .1156     .2445     .1165     .1051     .0893     .0885     .0867

     .1247     .1256     .1155     .1947     .1332     .1087     .0885     .1087
```

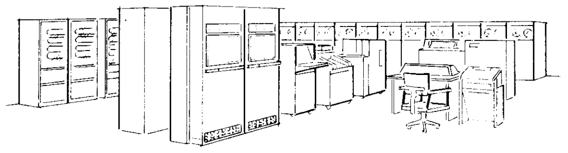
```
                              CYCLE NUMBER =    5000
                         PROBABILITY  P (A-B) = .42860000
                    STANDARD DEVIATION SIGMA P(I-J) = .00699860


                    NUMBER OF MOLECULES EXITING VIA OPENING A =   2857
                    NUMBER OF MOLECULES EXITING VIA OPENING B =   2143


SYMMETRY CHECK OF INPUT MOLECULES
A(I) I=0,1,3  =     1264.     1145.     1287.     1304.

SYMMETRY CHECK OF OUTPUT MOLECULES
A(J) J=0,1,7  =      728.      617.      818.      694.      622.      538.      470.      513.

            PROBABILITY FOR AREA SUBDIVISIONS P(I-J)   I=0,1,3   AND   J=0,1,7

     .1677     .1242     .1629     .1226     .1250     .1075     .0893     .1004

     .1406     .1301     .1152     .1240     .1362     .1213     .1161     .1161

     .1538     .1157     .2470     .1134     .1017     .0940     .0854     .0885

     .1203     .1242     .1242     .1924     .1357     .1088     .0874     .1065
```

```
                              CYCLE NUMBER =    5500
                         PROBABILITY  P (A-B) = .42218181
                    STANDARD DEVIATION SIGMA P(I-J) = .00665984


                    NUMBER OF MOLECULES EXITING VIA OPENING A =   3178
                    NUMBER OF MOLECULES EXITING VIA OPENING B =   2322


SYMMETRY CHECK OF INPUT MOLECULES
A(I) I=0,1,3  =     1381.     1280.     1420.     1419.

SYMMETRY CHECK OF OUTPUT MOLECULES
A(J) J=0,1,7  =      817.      696.      895.      770.      662.      590.      520.      550.

            PROBABILITY FOR AREA SUBDIVISIONS P(I-J)   I=0,1,3   AND   J=0,1,7

     .1665     .1303     .1600     .1216     .1216     .1107     .0926     .0963

     .1460     .1328     .1195     .1250     .1320     .1187     .1125     .1132

     .1577     .1211     .2422     .1147     .0957     .0943     .0866     .0873

     .1240     .1226     .1247     .1966     .1331     .1064     .0880     .1042
```



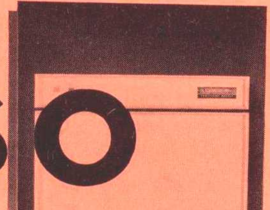
```
                              CYCLE NUMBER =    6000
                          PROBABILITY  P (A-B) = .42216666
                    STANDARD DEVIATION SIGMA P(I-J) = .00637628


                 NUMBER OF MOLECULES EXITING VIA OPENING A =   3467
                 NUMBER OF MOLECULES EXITING VIA OPENING B =   2533


SYMMETRY CHECK OF INPUT MOLECULES
A(I) I=0,1,3  =      1505.     1396.     1540.     1559.

SYMMETRY CHECK OF OUTPUT MOLECULES
A(J) J=0,1,7  =       883.      750.      989.      845.      725.      644.      559.      605.

             PROBABILITY FOR AREA SUBDIVISIONS P(I-J)   I=0,1,3   AND   J=0,1,7

       .1647     .1308     .1614     .1189     .1229     .1102     .0923     .0983

       .1404     .1303     .1232     .1296     .1332     .1189     .1117     .1124

       .1603     .1181     .2428     .1162     .0974     .0948     .0818     .0883

       .1231     .1212     .1282     .1962     .1308     .1064     .0885     .1051
                                   _____________________
                              CYCLE NUMBER =    6500
                          PROBABILITY  P (A-B) = .42153846
                    STANDARD DEVIATION SIGMA P(I-J) = .00612490


                 NUMBER OF MOLECULES EXITING VIA OPENING A =   3760
                 NUMBER OF MOLECULES EXITING VIA OPENING B =   2740


SYMMETRY CHECK OF INPUT MOLECULES
A(I) I=0,1,3  =      1627.     1520.     1662.     1691.

SYMMETRY CHECK OF OUTPUT MOLECULES
A(J) J=0,1,7  =       955.      823.     1067.      915.      786.      709.      597.      648.

             PROBABILITY FOR AREA SUBDIVISIONS P(I-J)   I=0,1,3   AND   J=0,1,7

       .1665     .1333     .1604     .1186     .1223     .1124     .0903     .0958

       .1368     .1335     .1230     .1309     .1335     .1203     .1131     .1085

       .1600     .1185     .2460     .1155     .0974     .0950     .0788     .0884

       .1241     .1218     .1241     .1957     .1312     .1094     .0869     .1064
                                   _____________________
                              CYCLE NUMBER =    7000
                          PROBABILITY  P (A-B) = .42400000
                    STANDARD DEVIATION SIGMA P(I-J) = .00590670


                 NUMBER OF MOLECULES EXITING VIA OPENING A =   4032
                 NUMBER OF MOLECULES EXITING VIA OPENING B =   2968


SYMMETRY CHECK OF INPUT MOLECULES
A(I) I=0,1,3  =      1748.     1655.     1787.     1810.

SYMMETRY CHECK OF OUTPUT MOLECULES
A(J) J=0,1,7  =      1019.      890.     1144.      979.      853.      767.      647.      701.

             PROBABILITY FOR AREA SUBDIVISIONS P(I-J)   I=0,1,3   AND   J=0,1,7

       .1659     .1298     .1613     .1195     .1201     .1172     .0903     .0955

       .1341     .1377     .1220     .1280     .1341     .1184     .1135     .1117

       .1611     .1186     .2439     .1141     .1007     .0940     .0777     .0895

       .1209     .1232     .1237     .1955     .1331     .1093     .0895     .1044
```

```
                                   CYCLE NUMBER =    7500

                              PROBABILITY  P (A-B) = .42320000

                         STANDARD DEVIATION SIGMA P(I-J) = .00570498


                      NUMBER OF MOLECULES EXITING VIA OPENING A =    4326

                      NUMBER OF MOLECULES EXITING VIA OPENING B =    3174


SYMMETRY CHECK OF INPUT MOLECULES
A(I) I=0,1,3  =      1875.     1767.     1920.     1938.

SYMMETRY CHECK OF OUTPUT MOLECULES
A(J) J=0,1,7  =      1109.     947.      1226.     1044.     895.      827.      704.      748.

               PROBABILITY FOR AREA SUBDIVISIONS P(I-J)   I=0,1,3   AND   J=0,1,7

      .1680     .1274     .1632     .1189     .1178     .1157     .0912     .0976

      .1369     .1375     .1216     .1290     .1312     .1205     .1137     .1092

      .1651     .1166     .2421     .1125     .0979     .0968     .0796     .0890

      .1212     .1243     .1238     .1945     .1310     .1088     .0923     .1037
                                  ______________________
                                   CYCLE NUMBER =    8000

                              PROBABILITY  P (A-B) = .42250000

                         STANDARD DEVIATION SIGMA P(I-J) = .00552260


                      NUMBER OF MOLECULES EXITING VIA OPENING A =    4620

                      NUMBER OF MOLECULES EXITING VIA OPENING B =    3380


SYMMETRY CHECK OF INPUT MOLECULES
A(I) I=0,1,3  =      1989.     1910.     2032.     2069.

SYMMETRY CHECK OF OUTPUT MOLECULES
A(J) J=0,1,7  =      1187.     1020.     1301.     1112.     945.      881.      757.      797.

               PROBABILITY FOR AREA SUBDIVISIONS P(I-J)   I=0,1,3   AND   J=0,1,7

      .1679     .1317     .1598     .1186     .1141     .1171     .0910     .0995

      .1382     .1387     .1219     .1282     .1298     .1209     .1130     .1089

      .1712     .1161     .2411     .1136     .0974     .0939     .0797     .0866

      .1164     .1242     .1256     .1933     .1314     .1092     .0956     .1039
                                  ______________________
                                   CYCLE NUMBER =    8500

                              PROBABILITY  P (A-B) = .42329411

                         STANDARD DEVIATION SIGMA P(I-J) = .00535906


                      NUMBER OF MOLECULES EXITING VIA OPENING A =    4902

                      NUMBER OF MOLECULES EXITING VIA OPENING B =    3598


SYMMETRY CHECK OF INPUT MOLECULES
A(I) I=0,1,3  =      2102.     2041.     2158.     2199.

SYMMETRY CHECK OF OUTPUT MOLECULES
A(J) J=0,1,7  =      1268.     1078.     1377.     1179.     992.      941.      811.      854.

               PROBABILITY FOR AREA SUBDIVISIONS P(I-J)   I=0,1,3   AND   J=0,1,7

      .1679     .1317     .1569     .1160     .1165     .1165     .0908     .1032

      .1371     .1357     .1234     .1308     .1254     .1219     .1151     .1102

      .1728     .1153     .2405     .1130     .0954     .0945     .0824     .0857

      .1191     .1250     .1255     .1928     .1296     .1105     .0941     .1032
```



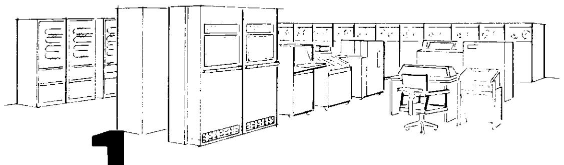
```
                              CYCLE NUMBER =   9000

                          PROBABILITY  P (A-B) = •42355555

                     STANDARD DEVIATION SIGMA P(I-J) = •00520849


                    NUMBER OF MOLECULES EXITING VIA OPENING A =   5188

                    NUMBER OF MOLECULES EXITING VIA OPENING B =   3812


SYMMETRY CHECK OF INPUT MOLECULES
A(I) I=0•1•3  =     2239•     2167•     2274•     2320•

SYMMETRY CHECK OF OUTPUT MOLECULES
A(J) J=0•1•7  =     1336•     1142•     1466•     1244•     1047•      992•      872•      901•

               PROBABILITY FOR AREA SUBDIVISIONS P(I-J)    I=0•1•3   AND   J=0•1•7

      •1688     •1330     •1576     •1156     •1165     •1143     •0915     •1022

      •1365     •1352     •1245     •1287     •1273     •1218     •1158     •1098

      •1723     •1152     •2414     •1130     •0949     •0923     •0844     •0861

      •1163     •1245     •1267     •1935     •1267     •1129     •0965     •1025
                                  ___________________
                              CYCLE NUMBER =   9500

                          PROBABILITY  P (A-B) = •42452631

                     STANDARD DEVIATION SIGMA P(I-J) = •00507111


                    NUMBER OF MOLECULES EXITING VIA OPENING A =   5467

                    NUMBER OF MOLECULES EXITING VIA OPENING B =   4033


SYMMETRY CHECK OF INPUT MOLECULES
A(I) I=0•1•3  =     2351•     2303•     2386•     2460•

SYMMETRY CHECK OF OUTPUT MOLECULES
A(J) J=0•1•7  =     1401•     1216•     1539•     1311•     1111•     1062•      929•      931•

               PROBABILITY FOR AREA SUBDIVISIONS P(I-J)    I=0•1•3   AND   J=0•1•7

      •1684     •1335     •1582     •1148     •1169     •1156     •0923     •0999

      •1350     •1346     •1254     •1289     •1267     •1224     •1172     •1094

      •1705     •1190     •2388     •1106     •0963     •0959     •0846     •0838

      •1166     •1252     •1252     •1951     •1276     •1134     •0975     •0991
                                  ___________________
                              CYCLE NUMBER =  10000

                          PROBABILITY  P (A-B) = •42520000

                     STANDARD DEVIATION SIGMA P(I-J) = •00494373


                    NUMBER OF MOLECULES EXITING VIA OPENING A =   5748

                    NUMBER OF MOLECULES EXITING VIA OPENING B =   4252


SYMMETRY CHECK OF INPUT MOLECULES
A(I) I=0•1•3  =     2477•     2448•     2502•     2573•

SYMMETRY CHECK OF OUTPUT MOLECULES
A(J) J=0•1•7  =     1477•     1270•     1619•     1382•     1178•     1121•      980•      973•

               PROBABILITY FOR AREA SUBDIVISIONS P(I-J)    I=0•1•3   AND   J=0•1•7

      •1671     •1332     •1590     •1150     •1174     •1146     •0916     •1017

      •1360     •1335     •1262     •1282     •1286     •1229     •1164     •1078

      •1686     •1167     •2378     •1115     •0999     •0975     •0851     •0827

      •1197     •1247     •1247     •1958     •1251     •1134     •0991     •0971
```

ATTACHMENT 1



ALGOL 60

for the D850 modular data processing system



SECTION 1

JUST WHAT IS ALGOL?

Someone has said, "Mathematics is the science of computing, but of computing as little as possible." Undoubtedly he was thinking of the enormous reduction of labor that is realized, for instance, in solving a simple expression for the area under a parabola, instead of tediously counting little squares. The same thought applies to the use of a digital computer. We often use a computer to perform highly repetitious calculations which could not be done in any other practical way.

For a long time, however, the use of these accurate, high-speed electronic devices has been bogged down in the difficult job of instructing them—the process known as programing. Since computers had to be instructed step by step on the most basic level, each programer found himself repeating in a general way many things that another programer using the same machine had to do *for himself:* getting information into the machine, operating on it in some way, and printing out the results. In spite of many similarities in their programs, there was virtually no possibility of one programer making use of another's work.

ALGOL changes this situation. It provides a person who needs to use a computer with a means of expressing himself in easy-to-understand, common-sense terms and relieves him of the need to understand the details of the computer's operation. The programing methods used previously, all of which were different from machine to machine, now become the concern only of the automatic system which turns the programer's directions into a set of instructions which the machine can "understand." Although ALGOL is not the first of such languages, the earlier ones were so closely associated with particular machines that they were not applicable to other computers, even those of the same manufacturer. ALGOL, on the other hand, is intended to be completely general and thus independent of any particular computer.

In preparing any program, there are stages of planning. The first is adapting the problem to the demands of a digital computer. This problem is much the same whether a program is written for a particular machine or in more general terms.

The second part of preparing a program is much more dependent on the nature of the machine. For any computer, however, if the programer must use the machine instructions directly to prepare his program, he must exercise the most scrupulous care to insure that he has not committed any of many possible errors: computing with values obtained from a wrong location in memory, setting up the wrong qualifications to enter or leave a portion of the program which is used repeatedly, ad infinitum. This part of the programer's job is called "housekeeping," an apt name because of its connotations of repetition and drudgery.

When the special program called a compiler (built into the D850) transforms an ALGOL program into one which the computer can employ, it assumes these responsibilities; housekeeping conventions have been previously established — prepackaged, if you will — and the compiler performs virtually all of these duties, relying on the dependable performance of the computer to maintain accuracy.

With this indication of some of the advantages of using ALGOL to program a digital computer, let us examine the role of the language itself in the program.

Dr. Herriot of Stanford University has made an interesting distinction between a mathematical statement of a problem and its ALGOL counterpart. The mathematical statement is static; the ALGOL statement, on the other hand, is dynamic. The ALGOL statement, however complex it may be, describes an actual step-by-step procedure for obtaining a solution to a problem, using actual numbers and achieving a number as a result. For example, when we write the equation which gives the hypotenuse of a right triangle

$$C = \sqrt{A^2 + B^2}$$

we do not qualify in any way the order in which we will arrive at the sum of the squares, the units we will use in performing the calculation, or the means of determining the square root of the sum $A^2 + B^2$. The ALGOL equivalent for this would be written as

C ← **SQRT** (A*2 + B*2)

Now there are many questions which may occur to you; after all, what we have written here in ALGOL *resembles* the algebraic equation which we wrote, but it is certainly not altogether like it.

We can easily recognize the C, A, and B from the algebraic equation; in ALGOL they belong to a class of symbols called *identifiers*. Identifiers are used in ALGOL much as in algebra, where a symbol is used to denote a value which has yet to be calculated, so that it can be distinguished from others with which we are working. A large number of such identifiers can be written in this symbolic way, drawing on letters of the alphabet, and numerals if desired; it is necessary that the first character of an identifier always be a letter, and that there be no spaces to break up the identifier.

Examples of identifiers:

A
B10
PRESSURE
A1B2C3D4E5

To go back to our example

C ← **SQRT** (A*2 + B*2)

the symbol + is used in ALGOL in an easily understandable way; it has the same significance as the plus sign in the algebraic statement

$$C = \sqrt{A^2 + B^2}$$

that is, addition. The class to which the plus sign belongs is that of *arithmetic operators*. There are, of course, other operators in ALGOL, the arithmetic operators being:

subtraction	−	(minus sign)
multiplication	×	(cross-product sign)
division	/	(solidus)
exponentiation	*	(asterisk)
division	**DIV**	

(The last division operator is a special one which will be described after the discussion of the number systems employed in the D 850)

From our list, we now can see the significance of the asterisk in our ALGOL equivalent. Where in mathematical notation we wrote A^2, in the D 850 version of ALGOL we write A*2. The 2 here is called the *exponent* of A; if we were using an expression such as X^Y, we would write its ALGOL equivalent as X*Y, and so forth. More complex expressions are possible and will be discussed later.

SQRT in our ALGOL equivalent

C ← **SQRT** (A*2 + B*2)

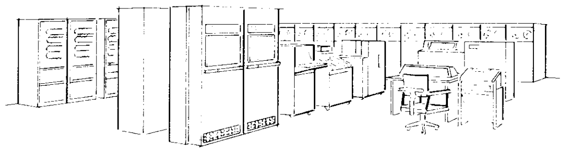
means "square root," and is equivalent to $\surd$ or to the power of 1/2 in mathematical notation. When we use the symbol $\surd$ in mathematics, we indicate its extent by adding a vinculum (overbar) $\sqrt{\quad}$; when we use $^{1/2}$, we indicate the extent by parentheses. The parentheses in our ALGOL statement thus perform the same task as when they enclose any mathematical expression.

We are left with one symbol to be explained, the left-pointing arrow:

C ← **SQRT** (A*2 + B*2)

This arrow is called the *replacement operator*. It replaces the current value of the identifier C with the value of the expression to its right, allowing the programer to use C whenever he has need to refer to the value of $\sqrt{A^2 + B^2}$. Although the left-pointing arrow is often placed where we would find an equal sign (=) in algebra, it is important to emphasize (for reasons which will become apparent later) that *the replacement operator is not entirely equivalent to the equal sign in mathematical notation.*

To extend the discussion of **SQRT**, it is necessary to say that the method used to perform the calculation is made available from a section of the compiler called the library, and is thus termed a *library function*. The programer need only write the name of a library function in order to make use of it. There are several other such functions, some of which compute trigonometric functions, others the logarithms, and so forth. To avoid confusion, names of these functions naturally cannot be used for any other purpose, so care must be taken not to use their names to designate anything else in the program. The ALGOL class to which **SQRT**, the names of the other library functions, and certain other



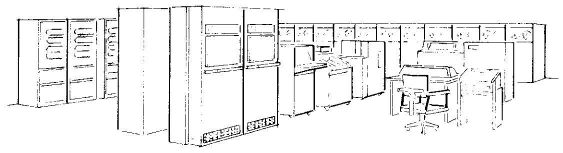
words belong is called *reserved words*, a list of which appears in APPENDIX D. (Reserved words appear in this text in boldface.)

Writing a program in ALGOL closely resembles the detailed problem definition required if one were to give a problem to a machine-language programer for computer solution. The language and format of the problem definition have been standardized. The computer itself produces the machine-language program, as well as doing the processing. Therefore, it is particularly important to remember that the machine-language programer (the compiler which is in the computer) is not familiar with the problem and knows about it *only* what is contained in the problem (the ALGOL program). The problem must, therefore, be complete, unambiguous, and expressed in an acceptable form and terminology. The form will approximate the well-known: "Given; To Find; Calculations; Solution." The terminology will be ALGOL.

It is hoped that this very short treatment will serve to indicate some of the properties of ALGOL, but it must be understood that the example we have discussed is quite rudimentary. As the reader continues in this manual and finds his proficiency increasing, he will likely be struck again and again by the similarity of ALGOL to a natural language such as English. Just as English serves to write either

"See the cat. The cat is on the table."

or to discuss highly abstruse subjects, so ALGOL lets us express problems ranging from the example above to problems of great complexity. When we begin learning to write in ALGOL, though, we can just as well begin on the level of "See the cat."



SECTION 2 BASIC DEFINITIONS

SYMBOLS

The symbol set which is used in this manual consists of:

- the capital letters A through Z
- the digits 0 through 9
- punctuation symbols , : ; ( ) [ ]
- operational symbols (to be listed as introduced)

The complete list of symbols used in BURROUGHS ALGOL 60 appears in APPENDIX A.

NUMBERS

A number in BURROUGHS ALGOL 60 is written as a string of from one to eleven decimal digits.[2]

ALGOL allows the use of two types of numbers, called type **REAL** and type **INTEGER.** Type **REAL** numbers are those which include a decimal point. Type **INTEGER** numbers are the whole numbers, that is, those which do not include a decimal point. Both types include positive numbers, negative numbers, and zero.

The numbers which result from calculations are of one or the other type, depending upon the type of the numbers which went into the calculation and upon the nature of the calculation itself. The rules which determine the type of these results appear in the discussion of arithmetic expressions.

IDENTIFIERS

Identifiers are used in ALGOL programs as names, for purposes of reference.

Identifiers consist of at least one letter, followed by any letters, digits, or combination of letters and digits. No spaces may appear as part of an identifier. Some examples of identifiers are:

X	X1	SUMX	A	T
Y	Y2	AVERAGEX	ALT	T1
M	M53	LC	ALTITUDE	TIME
N	N4AND5	LIFTCOEFF	B3JFG29Z	TIME1

[2]For problems requiring greater precision, the number size can be extended to 23 decimal digits.

Identifiers are commonly used to name the variables and constants which appear in mathematical formulas. Identifiers are also combined with numbers, punctuation, and operational symbols to form the statements, expressions, declarations, etc., of a complete ALGOL program.

REPLACEMENT OPERATOR

The replacement operator symbol is a left-pointing arrow (←). It indicates that the value of whatever stands to the right of the arrow is to replace the value of the variable to the left of the arrow. Thus the statement

X ← Y

tells the computer to replace the value of X with the value of Y. The complete construction—the replacement operator with its left- and right-hand parts—is called an *assignment statement;* when executed by the computer it assigns the value as indicated above.

If a problem solution requires frequent reference to a constant (such as π), the programer may wish to use an identifier (such as PI) in his calculations. To do so, he might write the assignment statement

PI ← 3.14159

after which he may employ π in his program by simply using the identifier PI, which has been assigned the value 3.14159.

A common requirement in many problems is the setting of initial values. For example, if sums (used for tallies or totals) are calculated, it is necessary to set the sums to zero before the first pass through the calculation. The assignment statement

SUMX ← 0

replaces the value of the variable identified by SUMX with zero. If several variables must be set to a common value, they may be strung together; for example, the assignment statement

SUMX ← SUMY ← TALLYT ← K ← 0

sets the values of the variables whose identifiers are SUMX, SUMY, TALLYT, and K to zero.

ARITHMETIC OPERATORS AND EXPRESSIONS

The BURROUGHS ALGOL system for the D 850 uses the following arithmetic operators and symbols:

OPERATOR	SYMBOL
Add	+
Subtract	−
Multiply	×
Divide	/ or **DIV** (two different results)
Exponentiate	*

These operators are used with numbers or identifiers to form arithmetic expressions, such as:

T + 9.4	N − M	B/2	D*3
Z + Y	2 × A	180/PI	2*N
N − 1	BASE × HEIGHT	VEL/TIME	A*B

The sequence of performing a series of arithmetic operations is normally from left to right. However, this process is interrupted in accordance with the following priorities:

First:	operations enclosed by parentheses
Second:	exponentiation
Third:	multiplication and division
Fourth:	addition and subtraction

Since ALGOL 60 *defines* division by a term as multiplication by the inverse of the term (that is: A/B is the same as A × B^{-1}), equal priorities are assigned to multiplication and division, with these operations taking place from left to right as they occur in the program.

The foregoing definition and rule give the single exception from the priorities commonly used in ordinary algebra. For instance, in ordinary algebra the expression A/B × C is usually interpreted as A/(B × C); but in ALGOL 60 it means (A/B) × C. Parentheses must be used in ALGOL 60 to indicate denominators with more than one factor, and may be used as desired to indicate sequencing of operations.

Arithmetic expressions can constitute the right-hand member of assignment statements. Thus we can write

```
 DIST ← VEL/TIME
 AREA ← (BASE × HEIGHT)/2
    Y ← A*2 + B*2
DELTA ← PREVT − PREST
```

In each case, the number resulting from the evaluation of the expression on the right of the replacement operator is assigned as the value of the identifier on the left.

Remember that only identifiers may occupy the left-hand position in an assignment statement. It would be incorrect to write

C*2 ← **SQRT**(A*2 + B*2)

since C*2 is an arithmetic expression. The programer could write instead

CSQUARE ← **SQRT**(A*2 + B*2)

and the identifier CSQUARE may then be used to refer to

$\sqrt{A^2 + B^2}$.

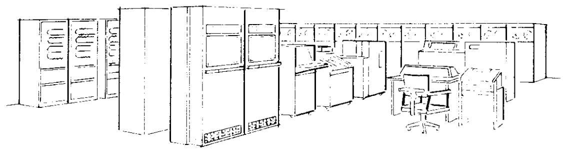
TYPE OF ARITHMETIC EXPRESSION

The type (**REAL** or **INTEGER**) of an arithmetic expression is automatically determined by the types of its components. Adding, subtracting, or multiplying two operands of type **INTEGER** gives a result of type **INTEGER.** The result will be of type **REAL** if either component is **REAL** or if both components are **REAL.**

As indicated in the list above, ALGOL provides two kinds of division. The operation indicated by the solidus (/), more commonly called the "slash" or "slant," gives a result of type **REAL** for any combination of **INTEGER** or **REAL** components, or both. This operation yields a conventional quotient which may have a fractional part, with the sign of the quotient plus or minus as in ordinary algebra.

The other division symbol, **DIV**, is used only where both operands are of type **INTEGER;** the result is always of type **INTEGER.** The sign of the result is plus or minus as in algebra. The result is always truncated (cut off) to be an **INTEGER.** No rounding is done in this cutting-off process; thus, if the normal quotient would be + 12.83, it is cut off to + 12; similarly, − 12.83 is truncated to − 12. (**DIV** is one of the reserved words. See APPENDIX D.)

For exponentiation, raising a number of type **INTEGER** to a positive power which is also of type



INTEGER will give a result of type **INTEGER.** Any other defined result will be of type **REAL.**

Because a negative number raised to a fractional power may be undefined in mathematics, there are certain invalid combinations for exponentiation. To define all situations completely, a table of valid and invalid combinations is given in APPENDIX B.

It is especially important to realize that although the type (**REAL** or **INTEGER**) of an arithmetic expression is determined by the foregoing rules, the type of the result following insertion of the expression into a statement (after the replacement operator) can change.

Example:

$Z \leftarrow (A + B)/C$

Because division with the slant sign is defined as always giving a result of type **REAL,** the expression $(A + B)/C$ is of type **REAL.** If the type of Z were **INTEGER,** then *this result would be automatically converted* to type **INTEGER,** and appropriately truncated. Conversely, an expression of type **INTEGER** would be converted to type **REAL** in such a statement if the variable to the left of the replacement operator were of type **REAL.**

EXAMPLES OF ARITHMETIC EXPRESSIONS

In the following examples, the left column gives an ALGOL assignment statement which contains an arithmetic expression. The right column gives the algebraic interpretation of that statement. Extra parentheses have been used in some of the interpretations to indicate the complete meanings.

ADDITION

$Z \leftarrow A + B$	$Z = A + B$
$Z \leftarrow E + F + G$	$Z = (E + F) + G$

SUBTRACTION

$Z \leftarrow W - R$	$Z = W - R$
$Z \leftarrow W - R - S$	$Z = (W - R) - S$

MULTIPLICATION

$Z \leftarrow A \times B$	$Z = A \times B$
$Z \leftarrow A \times B \times C \times D$	$Z = \{(A \times B) \times C\} \times D$

DIVISION

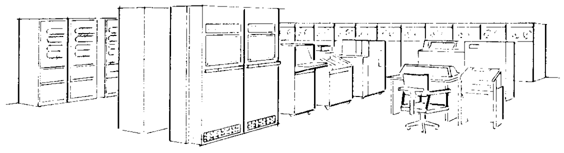
$Z \leftarrow A/B$	$Z = \frac{A}{B}$
$Z \leftarrow A/B/C$	$Z = (A/B)/C$

EXPONENTIATION

$Z \leftarrow A*B$	$Z = A^{B}$
$Z \leftarrow A*(-B)$	$Z = A^{(-B)}$

COMBINATIONS OF OPERATIONS

$Z \leftarrow A+B-C+D$	$Z = \{(A+B)-C\}+D$
$R \leftarrow Z\times Y+A\times B$	$R = (Z\times Y)+(A\times B)$
$S \leftarrow A+B\times C\times D$	$S = A+\{(B\times C)\times D\}$
$T \leftarrow A\times B/C\times D$	$T = \frac{A\times B}{C} \times D$
$M \leftarrow A/B\times C-D+E/F$	$M = \{\left(\frac{A}{B}\right)\times C\}-D+\frac{E}{F}$
$Z \leftarrow A*B\times C$	$Z = (A^{B})\times C$
$P \leftarrow A\times B*C$	$P = A(B^{C})$
$V \leftarrow A*B+C$	$V = (A^{B})+C$
$W \leftarrow A*(B+C)$	$W = A^{(B+C)}$
$Z \leftarrow A*(B\times C)-D*(E/F)$	$Z = A^{(B\times C)}-(D)^{\left(\frac{E}{F}\right)}$



SECTION 3 BOOLEAN EXPRESSIONS

Boolean expressions are rules for computing the logical values **TRUE** and **FALSE.** If the condition stated is satisfied, the result is **TRUE,** otherwise the result is **FALSE.** Just as the value of an arithmetic expression is of type **REAL** or **INTEGER,** the value of a Boolean expression is of type **BOOLEAN.**

Boolean expressions employ two kinds of operators: relational and logical.

The relational operators are:

$<$ less than	$>$ greater than
$\leq$ less than or equal to	$\geq$ greater than or equal to
$=$ equal to	$\neq$ not equal to

Relational operators specify a comparison between two terms which may have any arithmetic value.

Examples:

$R > 0$	$R < +.46$
$R + 3 \times Y \leq 2 \times Z$	$Y \geq 12.34$
$R = Y$	$100 \neq Z$

In each example, the entire expression has either the value **TRUE** or the value **FALSE,** depending on whether or not the specified relation holds.

The logical operators used in ALGOL 60 for the D 850 are:

$\wedge$	and
$\vee$	or
$\neg$	not
EQV	equivalent to
IMP	implies

Evaluating a Boolean expression containing a logical operator involves application of the rule stated for that operator to the operands, which are restricted to the values **TRUE** and **FALSE.** (**EQV** and **IMP** are, of course, reserved words.)

Logical operands must be:

(a) the result of a relational operation, or
(b) the result of a logical operation, or
(c) a declared **BOOLEAN** variable. (Declarations of type are described in SECTION 6.)

The logical operators have the following meanings. (Where the conditions stated are *not* met, the result is **FALSE.**)

$\wedge$	(and)	If *both* operands connected by this operator have the value **TRUE,** the result is **TRUE.**
$\vee$	(or)	If *either* operand connected by this operator is **TRUE,** the *result* is **TRUE.**
$\neg$	(not)	If the operand following this operator is **FALSE,** the *result* is **TRUE.**
EQV	(equivalent to)	If the operands connected by this operator have the same logical value, the *result* is **TRUE.**
IMP	(implies)	If the operands connected by this operator have the same logical value, or if the first (left-hand) operand is **FALSE,** then the *result* is **TRUE.**

A tabular form of the above definitions follows.

Let A and B be logical operands, then:

A	TRUE	TRUE	FALSE	FALSE
B	TRUE	FALSE	TRUE	FALSE
A $\wedge$ B	TRUE	FALSE	FALSE	FALSE
A $\vee$ B	TRUE	TRUE	TRUE	FALSE
$\neg$ B	FALSE	TRUE	FALSE	TRUE
A EQV B	TRUE	FALSE	FALSE	TRUE
A IMP B	TRUE	FALSE	TRUE	TRUE

Examples using the logical operators are given

EXAMPLE	QUESTION POSED
$\neg$ (B < 100)	Is it **FALSE** that B is less than 100?
(A = 10) $\wedge$ (B < 100)	Is it **TRUE** that A is equal to 10 *and* that B is less than 100?
(X > Y) $\vee$ (Z > W)	Is it **TRUE** *either* that X is greater than Y *or* that Z is greater than W?
(A = B) **EQV** (C $\neq$ 0)	Is it **TRUE** that both relational tests give the same result; i.e., are *both* **TRUE** or *both* **FALSE**?
(M $\neq$ N) **IMP** (R < S)	Is it **TRUE** that the first (left-hand) relational value is equivalent to the second, *or* that the first is **FALSE**?

The interpretations of the examples are stated in the form of questions because that is the way the programer is likely to use these expressions in his programs. (See Control Statements, SECTION 5.)

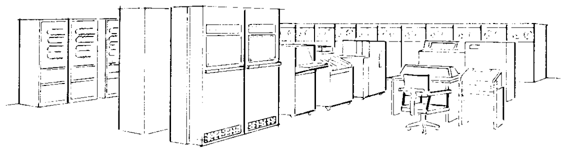
In the following examples, A, B, C, and D represent logical values or complete logical operations (such as the results of the preceding examples).

EXAMPLE	QUESTION POSED
A $\vee$ B	Is it **TRUE** that *either* A *or* B is **TRUE**?
$\neg$ (A $\vee$ B)	Is it **FALSE** that *either* A *or* B is **TRUE**?
(A $\wedge$ B) $\vee$ (C $\wedge$ D)	Is it **TRUE** *either* that *both* A and B are **TRUE,** or that *both* C and D are **TRUE?**
(A $\vee$ B) $\vee$ ($\neg$ C)	Is it **TRUE** *either* that A or B is **TRUE,** *or* that C is **FALSE**?

There are standard rules for the priority of performing relational and logical operations. While the general rule for performing these operations is to proceed from left to right, this rule is subordinated to the following priority list.

First priority:	Parenthesized operations
Second priority:	Evaluation of arithmetic expressions
Third priority:	Relational operations as met from left to right
Fourth priority:	$\neg$ (not)
Fifth priority:	$\wedge$ (and)
Sixth priority:	$\vee$ (or)
Seventh priority:	**IMP** (implies)
Eighth priority:	**EQV** (equivalent to)

This priority list indicates that in many cases parentheses may be omitted. However, to avoid errors in interpretation and for easier reading, it is recommended that parentheses be used in compound expressions.



SECTION 4 STANDARD FUNCTIONS

A function designator defines a single value which is the result of a specific set of operations on given parameters. Certain frequently used functions have been designated standard functions and incorporated in ALGOL so that the programer need not write the detailed steps to compute these values.

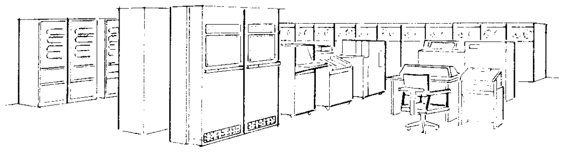
The arguments upon which these standard functions are to operate *must be enclosed with parentheses.* A list of the standard function designators, all of which are reserved words, follows:

STANDARD FUNCTION DESIGNATOR	FUNCTION DESIGNATED
SIN	Sine
COS	Cosine
ARCTAN	Arctangent
SQRT	Square root
LN	Natural logarithm
LOG	Logarithm, base ten
EXP	Exponential function
ABS	Absolute value
SIGN	Sign. According to whether the value of E is greater than zero, equal to zero, or less than zero, **SIGN** (E) is + 1, 0, or − 1.
ENTIER	Transfers from type **REAL** to type **INTEGER** by assigning the largest integer not greater than the value.

Examples:

(1) R← (− B + **SQRT** (B*2 − 4 × A × C))/ (2 × A)
(2) TANY ← **SIN** (Y)/**COS** (Y)
(3) R ← **ENTIER** (Z)

In example (2), **SIN** and **COS** are standard function designators, but TANY is an identifier; the angle Y must be expressed in radians, and TANY will, of course, be a type **REAL** number such as 2.3456789. In example (3), the value of R is replaced by the largest integer which is not greater than the value of the type **REAL** number Z; the sign of R will be the same as the sign of Z.



SECTION 5 OPERATIONAL STATEMENTS

Statements are the sentences of this algebraic language; as in ordinary written English, the order in which they appear is very important. Statements are separated by semicolons. A group of statements may be combined to form a compound statement by preceding the first statement of the group with the word **BEGIN** and following the last statement with the word **END.** It is sometimes necessary to identify a particular statement so that it may be referenced in other statements. To do this, a statement is given a label, which is an identifier followed by a colon. Operational statements fall into one of two general categories: assignment statements and control statements.

ASSIGNMENT STATEMENTS: SUMMARY

Assignment statements have been mentioned in preceding sections. Here these references are summarized for easier comparison with control statements.

Assignment statements contain the replacement operator ← denoting the substitution of

a number, or
the value of an identifier, or
the value of an expression

on the right for the identifier on the left.

Examples:

SUMB ← 0; [Zero replaces SUMB.]

M ← N; [The value of N replaces the value of M.]

A ← B/C − V − Q × S;
[The quantity $\frac{B}{C} - V - QS$ replaces A.]

HYPTNS: C ← **SQRT** (A*2 + B*2);
[HYPTNS is a statement label. The quantity $\sqrt{A^2 + B^2}$ replaces C.]

ROOT: Y ← (− B + **SQRT** (B*2 − 4 × A × C))/(2 × A);
[ROOT is the statement label. The quantity $\frac{-B + \sqrt{B^2 - 4AC}}{2A}$ replaces Y.]

CONTROL STATEMENTS

In the normal sequence of operations, the successive statements are executed as they are encountered. It is sometimes desirable to interrupt this normal sequence, as when one or more statements are to be repeated several times, or are to be executed only under specific conditions. The interruption of the normal sequence is called *transfer of control* since, once the transfer has taken place, successive statement sequencing continues from the new point of reference.

Transfer of control in ALGOL is accomplished through use of the control statement, which may be *unconditional, conditional,* or *iterative.*

Unconditional Control Statements

Unconditional transfer of control statements are formed by following the words **GO TO** with a label which specifies the point in the program where control is to be resumed; some examples are shown below. More general **GO TO** statements are possible, but are not considered in this section.

GO TO BILL;
GO TO COEFLIFT;
GO TO SECONDSTOP;
GO TO START;
GO TO M3L75;

Conditional Control Statements

Conditional control statements cause other statements to be executed or skipped depending on the current values of specified Boolean expressions. Conditional control statements provide the ability to make decisions necessary for the completely automatic solution of a problem.

The conditional control statement may have either of the following formats:

IF Boolean expression **THEN** statement; next statement

IF Boolean expression **THEN** statement **ELSE** statement; next statement

NOTE: The statement following **THEN** may not begin with the word **IF.**

In either case, when the relational or logical expression following **IF** (the **IF** clause) is **TRUE,** the statement following **THEN** is executed and control is transferred to the beginning of the next statement, unless the **THEN** statement contains a change of control operation (as shown in examples 2 and 4 which follow). When the **IF** clause is **FALSE,** the **THEN** statement is skipped. In the first case above (**IF** ... **THEN**), control is transferred to the beginning of the next statement. In the second case above **(IF ... THEN ... ELSE),** control is transferred to the statement following **ELSE;** after that statement has been executed, control is transferred to the beginning of the next statement.

Examples:

(1) **IF** $P \leq 0$ **THEN** $P \leftarrow .5; Y \leftarrow 2 \times P + 3;$ ...

(2) **IF** $Y \leq .0001$ **THEN** **GO TO** OUT; $N \leftarrow N/2;$ ...

(3) **IF** $A = B$ **THEN** $C \leftarrow 1$ **ELSE** $C \leftarrow 1 - A/B;$ $D \leftarrow C \times M;$ ...

(4) **IF** $(Y \geq 0) \wedge (Y < .0001)$ **THEN** **GO TO** OUT **ELSE GO TO** CONT;

The conditional control statement may contain more than one **IF** clause. In this case, the **IF** clauses are evaluated one after the other in sequence from left to right until one yielding the value **TRUE** is found. Only with a **TRUE** condition is the associated **THEN** clause executed, after which control is transferred to the beginning of the next statement.

Example:

```
KEN: IF A = B THEN C ← 1
     ELSE IF A < B THEN
     C ← 1 − A/B ELSE GO TO REVERSE;
```

Iterative Control Statements

The purpose of the **FOR** statement in ALGOL 60 is to facilitate writing an iterative operation. An operation is said to be iterative when the same statement is to be executed repeatedly a specified number of times or is to be executed for each one of a designated set of values assigned to a variable. The **FOR** statement contains a **FOR** clause and a **DO** statement. The **FOR** clause gives the conditions under which the **DO** statement is to be executed repeatedly zero or more times. (The **DO** statement would be executed zero times—i.e., would not be executed—if the conditions of the **FOR** clause were not satisfied.) The **FOR** statement has the following format:

FOR (variable) ← (**FOR** list) **DO** (statement)

The **FOR** list is composed of one or more elements separated by commas, and gives a rule for obtaining the values which are consecutively assigned to the variable. This sequence of values is obtained from the **FOR** list elements by taking these one by one in the order in which they are written, left to right. There are three kinds of **FOR** list elements: arithmetic expression element, **STEP-UNTIL** element, and **WHILE** element. In defining these, only one-element **FOR** lists will be considered.

ARITHMETIC EXPRESSION ELEMENT

An arithmetic expression alone may be a **FOR** list element and as such indicates that the variable will take on the value of the expression prior to the execution of the **DO** statement.

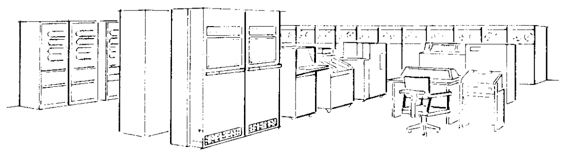
FOR variable ← $\begin{array}{c}\text{arithmetic}\\\text{expression}\end{array}$ **DO** statement; $\begin{array}{c}\text{next}\\\text{statement}\end{array}$

When the **DO** statement has been executed, control is transferred to the beginning of the next statement.

```
FOR J ← 3 DO Z ← 2 × J + J*3;
next statement

FOR S ← C + D DO BEGIN M ← S*2;
N ← M + 5;
V ← R/S + L + M × N END; next statement
```

STEP-UNTIL ELEMENT

The effect of evaluating the **FOR** list element **STEP-UNTIL** is similar to the result obtained from counting when given a starting point, a limit, and



the increment by which to count. (For example: Count by 2's from 10 through 90.) This element has the form:

starting point **STEP** increment **UNTIL** limit

The starting point, increment, and limit are arithmetic expressions. The statement following the word **DO** in the example shown below is executed once for each value computed by stepping *from* the starting point *through* the limit.

> **FOR** variable ← starting point **STEP** increment
> **UNTIL** limit
> **DO** statement; next statement;

In the above form, the following sequence takes place:

(1) The variable is replaced with the value of the starting point.
(2) The variable is compared with the limit. If the variable has *passed* the limit, control is transferred to the beginning of the next statement. If the variable has *not* passed the limit, the statement following **DO** is executed, then the variable is altered by the amount of the increment, and the sequence continues at (2) above.

Note that since the increment may be either positive or negative the limit may be approached from either direction.

Example:

> **FOR** A ← 1 **STEP** 1 **UNTIL** 10
> **DO** statement; next statement

In this example, the **DO** statement will be executed ten times, after which control will be transferred to the beginning of the next statement.

> **FOR** Z ← 3 × Y + 2 **STEP** 2 × B **UNTIL**
> B*2 + 1 **DO** statement; next statement

In this example, the **DO** statement will be executed repeatedly until the value Z, which is increased by 2B after each execution, exceeds the value $B^2 + 1$. At that time, control is transferred to the beginning of the next statement. Unlike the previous example, in which the **DO** statement is always executed exactly ten times, the number of times the **DO** statement in this example is executed is not fixed since it is dependent on the current values of B and Y.

Example:

> APPROX ← 0; **FOR** M ← Z1 **STEP** 0.005
> **UNTIL** Z2 **DO** APPROX ← APPROX +
> 0.005 × (C × M*2 + D × M + E);

WHILE ELEMENT

The **FOR** list element **WHILE** implies duration and may be thought of as representing "as long as." This element has the form:

arithmetic expression **WHILE** Boolean expression

A **FOR** statement containing the **WHILE** element has the following form:

> **FOR** variable ← arithmetic expression **WHILE** Boolean expression
>
> **DO** statement; next statement

The statement following **DO** will be executed repeatedly as long as the Boolean expression following **WHILE** is true.

The following events take place:

(1) The variable is replaced with the value of the arithmetic expression.
(2) The Boolean expression is evaluated. If the result is *not* **TRUE,** control is transferred to the beginning of the next statement. If the result is **TRUE,** execute the **DO** statement, then continue from (1).

Example:

> **FOR** Q ← 2 × V **WHILE** V < 10
> **DO** statement; next statement

In this example the value computed from the expression 2 × V replaces Q and the **DO** statement is executed as long as the value of V is less than 10. Note that an exit—i.e., change of control to another statement—from this **FOR** statement is dependent upon either a change in the value of V as a result of the **DO** statement (see the third example at the end of SECTION 6) or the presence of a change-of-control operation within the **DO** statement.

Example:

```
FOR Q ← 2 × V WHILE V < 10
DO BEGIN M ← (Q + 5 × R) × Q;
GO TO APRIL END; next statement
```

NOTE: In this example, Q is first set equal to 2V. Then, if V is less than 10, the statement following **BEGIN** is executed, and control is then transferred to APRIL. If V becomes equal to or greater than 10, control skips to the next statement instead of being transferred to APRIL.

This series of statements is completely equivalent to:

```
FOR Q ← 2 × V DO IF V < 10
THEN BEGIN M ← (Q + 5 × R) × Q;
GO TO APRIL END; next statement
```

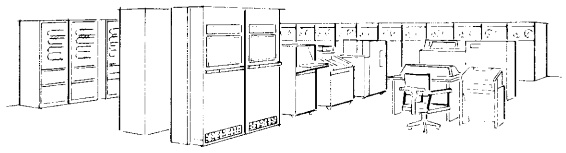
THE FOR LIST

As stated previously, the **FOR** list may contain several elements separated by commas. The elements within a single **FOR** list may be all of the same kind or of different kinds. They are completely evaluated, individually, as they are met from left to right, and the statement following **DO** is executed repeatedly until the **FOR** list is exhausted or until control is transferred to another point in the program as a result of the execution of the **DO** statement. (See the last example in SECTION 6.)

Example:

```
FOR L ← 1, 3, 7, 11 DO statement;
next statement
```

To exit from any **FOR** statement, either of two methods is used, depending on the operation involved. When the final (rightmost) **FOR** list element has been completely evaluated, control is transferred automatically to the beginning of the next statement in sequence. The alternate method is an exit resulting from the execution of a **GO TO** control statement from within the **DO** statement. (In this case, the **DO** statement is probably a compound statement.) If the exit is caused by a **GO TO** statement, the variable retains the value which it had immediately before the exit took place. Otherwise the value of the variable after exit is considered to be "unknown," according to the rules of ALGOL 60, and should not be used.



SECTION 6 DECLARATIONS AND BLOCKS

The main body of an ALGOL program is an ordered list of statements which, when executed, produces the specified solution. As was stated at the beginning of SECTION 5, successive statements may be combined to form a compound statement by preceding the first statement with the word **BEGIN** and following the last statement with the word **END**. The statements between **BEGIN** and **END** are treated as a whole rather than as separate units.

Statements are composed of identifiers, operators, numbers, punctuation marks, and reserved words (e.g., **IF**, **GO TO**) combined according to the rules of ALGOL. With the exception of identifiers, the individual items represented by each of these terms have specific meanings. Identifiers, because they are arbitrarily selected for and used in one program, have meaning only within that program. Therefore, all identifiers used in a program, except those employed as labels or as the formal parameters of a procedure declaration (see SECTION 7), must be introduced prior to their use; this is done with a declaration which defines certain properties of the identifiers.

TYPE DECLARATIONS

The type declaration defines the type of the variable named by an identifier. The type may be **REAL, INTEGER,** or **BOOLEAN.** The type declaration specifies that *all values which the identifier takes on* must be of the designated type.

The type declaration has the following form:

type identifier, identifier, ..., identifier;

Examples:

REAL M, Y, Z;
INTEGER C;
BOOLEAN A, B;

BLOCKS

A logical segment of a program is a section of coding which is considered by the programer to be a complete and primarily independent unit. Its independence derives from the fact that a section's elements may have meaning only within that particular section. An entire program is a logical segment and it may contain subprograms which are also logical segments. In ALGOL the logical segment is called a block. A block is defined as a program section which is preceded by the word **BEGIN,** includes at least one declaration and one statement, and is followed by the word **END.** A block has the following form:

BEGIN declaration; statement; . . . ;
statement **END**

A declaration is valid only for the block in which it appears, and has effect throughout that block. All declarations for a block must immediately follow the word **BEGIN,** and any entry to that block must be made at the word **BEGIN.**

Exit from the block, as a result of encountering the word **END** or a transfer-of-control statement (**GO TO**), cancels the declarations made within the block. The identifiers declared in the block, then, have no significance outside the block and may be used for other purposes. If an identifier is further declared with the word **OWN**, upon re-entry to the block it will assume the value it had at the last exit. Except for the values of these **OWN**-declared identifiers upon re-entry to a block, the value of each variable declared within a block is unknown until the variable has appeared to the left of the replacement operator in an assignment statement.

Blocks may be labeled by preceding the word **BEGIN** with an identifier followed by a colon.

Blocks written within blocks are allowed as long as all the preceding rules are strictly followed. For example:

Label: **BEGIN** declaration; statement; statement; ... **BEGIN** declaration; statement **END**; statement **END**

In this example, the *inner* **BEGIN-END** pair establishes a block within a larger block.

SWITCH DECLARATIONS

A **SWITCH** declaration names a group of alternative points in a program to which control may be transferred as the result of a single **GO TO** statement. The selection of the actual point to which control is transferred depends on conditions existing at the time of the transfer. The declaration contains a list of the labels of the statements to which control may be transferred, a replacement operator, and a separate label by which the **SWITCH** declaration may be referenced. The **SWITCH** declaration has the following form:

SWITCH name ← Label*1*, Label*2*, ..., Label*n*;

Example:

SWITCH BENNY ← ERRORLOOP, GOODRESULT, ALLTHRU;

In order to transfer control to one of these three points by means of the switch, the program must encounter a change-of-control statement such as:

GO TO BENNY [Y − 2]; next statement

The expression in brackets is evaluated. If the value is 1, control is transferred to ERRORLOOP; if the value is 2, control is transferred to GOODRESULT; and if 3, to ALLTHRU.

If the value of the expression is not a whole number, it will be rounded and then truncated to a whole number for purposes of selecting the transfer. If the value is either zero or outside the range of the number of labels given in the list, no transfer of control results and the program continues with the statement following the **GO TO** statement (indicated above by "next statement").

ARRAY DECLARATIONS

An array is a group or set of items arranged in such a manner that each item may be identified by its position within the group. A familiar array is a standard classroom seating plan with desks arranged in uniform rows and columns. Each desk in the room may be uniquely named in terms of its row and column. Thus, classroom A, row 5, column 3 would locate one particular desk in the designated room. Also, just as different students could occupy one particular desk from time to time, different values can be assigned to one position in an array.

In ALGOL an array declaration is used to define a fixed arrangement of items; it names the array, specifies its dimensions, and states the range within each dimension. The form of the array declaration follows.

ARRAY name [dimension*1*, dimension*2*, ..., dimension*n*]

Each dimension has the form:

lower limit: upper limit

Classroom A, with five rows and six columns, would be defined in ALGOL as follows:

ARRAY A [1:5, 1:6]

VARIABLES WITH SUBSCRIPTS

To name a single item within an array, the programer uses the identifier of the array, followed by the appropriate list of subscripts. The subscripts in a list are separated by commas, and the entire list is enclosed in brackets. Thus, to refer to the value of the variable in the fifth row and third column of array A, the programer would write:

A [5, 3]

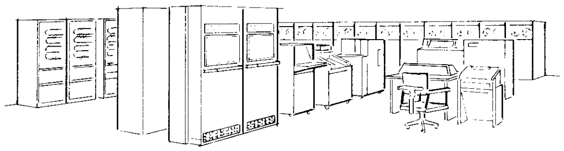
The subscript list may contain arithmetic expressions, variables, and variables with subscripts. Some examples are:

V [J, K]
RATE [2 + X]
V [J, K [3, N]]

TYPES OF ARRAYS

In mathematical problems the items in an array normally are numerical values. Therefore, in ALGOL it is necessary to precede the **ARRAY** declaration with a type declaration—**REAL, INTEGER,** or **BOOLEAN.** If the type declaration is absent, type **REAL** is understood.

More than one array may be defined within one **ARRAY** declaration, and, if several have the same number of dimensions and the same ranges within each dimension, these need only be given once.



Every array defined within a single declaration must be of the same type. Example:

```
REAL ARRAY M, N, Q [1:10, 3:7],
S [1:5, 1:30, 2:19], T[1:4]
```

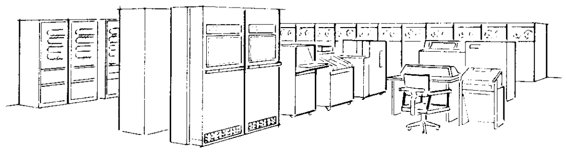
Five arrays of type **REAL** are defined by the above **ARRAY** declaration. Three of the arrays—M, N, and Q—have the same dimensions and ranges. The terms of the array dimension (lower limit: upper limit) may be arithmetic expressions involving identifiers if those identifiers have been declared and given a value in a block that contains the block in which the **ARRAY** declaration appears.

```
INTEGER ARRAY MAC [1:P + 2, K:L]
```

The identifiers P, K, and L must have values at the time the **ARRAY** declaration is encountered, since otherwise the declaration is meaningless.

The subscripts used with the **ARRAY** name, to indicate a single item within the set, follow the ALGOL conventions for subscripts and may be defined within the block in which they appear. Examples:

```
MAC [2, R]

COM3 [I, J]

FOR Q ← 2 × V WHILE V < 10 DO BEGIN
M [V] ← (Q + 5 × R) × Q;
V ← V + 1 END; next statement

FOR M ← 1 STEP 3 UNTIL 19, 20, 3 × N × A
WHILE A > 1 DO BEGIN
F [M] ← Z [M] + 5 × G; A ← A − 5;

R ← A × F [M] END; next statement
```



SECTION 7 PROCEDURES

GENERAL NATURE OF PROCEDURES

A procedure is a section of coding which is to be executed at several points throughout the same program, or used without alteration in more than one program. It is because of this multiple usage that a section of coding is made into a procedure; as such, it can be incorporated into any program exactly as it was first written. Also, for multiple use in one program, it need be written only once, with each execution being called for by a simple notation. It is characteristic of a procedure that the operations to be executed are fixed, while the values of the variables, or the variables themselves, may be different when each point is reached from which the procedure is entered.

The section of coding to be used as a procedure is written once in a procedure declaration which has the format shown below. The parts of the procedure declaration must appear in the order indicated here.

> NOTE: Words and symbols enclosed by parentheses *in the examples* of this section represent quantities within actual parentheses, and are not merely remarks to the reader.

PROCEDURE HEADING

PROCEDURE name (list of formal parameters) ;

- **VALUE** list of formal parameters to be replaced by the values of the actual parameters;
- Specifications giving information about the formal parameters;

PROCEDURE BODY

BEGIN declarations for identifiers which have meaning only within this procedure;
statement; ... ; last statement **END**

The procedure heading begins with the reserved word **PROCEDURE** which indicates that what follows is a procedure declaration. Each procedure is given an identifier (name) by which it may be referenced. Formal parameters are names (identifiers) given to the variables of the procedure which obtain actual values when the procedure is used in a program. They represent quantities obtained from the main program which may be used in calculations of the procedure or which may be assigned new values through the execution of the procedure. These names are chosen when the procedure is first written and have no connection with a particular program. They take on actual meaning only when the procedure is called upon for execution by a program.

When a program makes use of a procedure (i.e., calls for its execution) the formal parameters are replaced by actual values or names actually being used in the main program.

The **VALUE** part is used if one or more formal parameters are to be replaced by an actual value before a procedure is executed. Since formal parameters which are to be replaced by different names are not contained in the **VALUE** list, a procedure declaration need not have a **VALUE** part.

Specifications for formal parameters are optional and their inclusion is suggested as an aid to persons using the procedure.

The procedure body may be a compound statement, a block, or even a single statement. All the rules pertaining to those segments of an ALGOL program apply. The following example will help clarify the concept of a procedure.

Assume that a procedure is required which will calculate the factorial of any whole number. The problem to be solved is: For any whole number N, compute $N! = 1 \times 2 \times 3 \times \ldots \times N$. For example, if $N = 6$, the problem becomes $6! = 1 \times 2 \times 3 \times 4 \times 5 \times 6$.

The following procedure accomplishes this:

```
PROCEDURE FACTORIAL (N, F) ;
VALUE N; INTEGER N;
BEGIN INTEGER I; F ← 1;
FOR I ← 2 STEP 1 UNTIL N DO
F ← I × F END
```

The effect of this procedure is that F is replaced by N!; it can be seen then that when this procedure is actually used, N represents some number which has been previously determined by some other portion of the program. The factorial of N is to be calculated in the procedure; therefore, N is listed after **VALUE.** On the other hand, F represents a variable which is, as a result of the procedure, to be assigned a new value; therefore, F is not listed after **VALUE.** It can be seen that the procedure body is a block, hence the variable I has meaning only within the procedure.

At this point it is well to remember that a **PROCEDURE** declaration is merely another declaration like type or **ARRAY** and, when incorporated in a program, appears in the head of a block.

When a program requires the fixed set of steps outlined in some **PROCEDURE** declaration, a procedure statement is written which supplies the values of the variable in the procedure, or assigns new variables to replace those named in the **PROCEDURE** declaration, and causes the procedure to be executed.

A procedure statement has the following format:

Procedure identifier (list of actual parameters)

The procedure identifier is the name assigned to the procedure in the **PROCEDURE** declaration. The list of actual parameters must contain all the identifiers, expressions, and constants which are to be substituted for the corresponding items in the list of formal parameters found in the **PROCEDURE** declaration. *The number of actual parameters must thus agree with the number of formal parameters.*

A procedure statement which would call for the execution of the above example (**PROCEDURE** FACTORIAL) is:

FACTORIAL (BOB, JOE)

When a procedure is called upon for execution, three operations take place, in effect:

(1) All formal parameters of the procedure which are listed after **VALUE** are replaced by the values of the corresponding actual parameters when the procedure statement is encountered in the program.

(2) The other formal parameters are replaced by the names of the corresponding actual parameters.

(3) The procedure body is then inserted into the program, taking the place of the procedure statement.

The example discussed above will be used to illustrate the process.

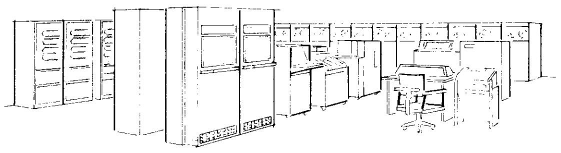
Given: A program which includes the **PROCEDURE** declaration FACTORIAL and the associated procedure statement, as well as other declarations and statements.

```
BEGIN   DECLARATION;
        DECLARATION;
        PROCEDURE FACTORIAL (F, N) ;
        VALUE N; INTEGER N;
        BEGIN INTEGER I; F ← 1;
        FOR I ← 2 STEP 1 UNTIL
        N DO F ← I × F END;
        DECLARATION;
        STATEMENT;
        STATEMENT;
        . . .
        STATEMENT;
        FACTORIAL (BOB, JOE);
        STATEMENT;
        . . .
        STATEMENT

END
```

Assume: JOE = 5 when the procedure statement is encountered.

Operations preparatory to execution: The first operation inserts the value of JOE, 5, in place of N in the procedure body. The latter then looks like this, in effect:

The next operation inserts the identifier BOB in place of F throughout the procedure body:



```
        BEGIN INTEGER I; BOB ← 1;
        FOR I ← 2 STEP 1 UNTIL 5 DO
        BOB ← I × BOB  END
```

Finally the altered procedure body replaces the procedure statement and is executed as if the following were the program:

```
BEGIN   DECLARATION;
        DECLARATION;
        PROCEDURE FACTORIAL (F, N);
        VALUE N; INTEGER N;
        BEGIN INTEGER I; F ← 1;
        FOR I ← 2 STEP 1 UNTIL N
        DO F ← I × F END;
        DECLARATION;
        STATEMENT;
        . . .
        STATEMENT;
        BEGIN INTEGER I; BOB ← 1;
        FOR I ← 2 STEP 1 UNTIL 5
        DO BOB ← I × BOB END;
        STATEMENT;
        . . .
        STATEMENT

END
```

Another example of a **PROCEDURE** declaration and possible procedure statements making use of it is given below.

PROCEDURE declaration example:

```
    PROCEDURE FALLINGBODY (T, V, S);
    VALUE T; REAL T, V, S;
    BEGIN REAL G; G ← 32.172;
    S ← G × T*2/2; V ← G × T  END
```

The name of this procedure is FALLINGBODY. The variables T, V, and S are the formal parameters which correspond to identifiers appearing outside the procedure. The procedure body is a block. The variable G is declared within the body of the **PROCEDURE** declaration and therefore *has meaning only within the procedure.*

Procedure statement examples:

(1) FALLINGBODY (FINALTIME, SPEED, DISTANCE)
(2) FALLINGBODY (5.88, VEL2, DIST2)

Either of these two example procedure statements could be used to call out the procedure named FALLINGBODY. The first example causes the value of the variable FINALTIME to be used in place of T, and the results to be SPEED and DISTANCE instead of the variables (formal parameters) V and S. The second example causes the numerical value 5.88 to be used for T; in this case the actual parameter is the number itself since it is not an identifier. The results are to be VEL2 and DIST2 in place of the parameters V and S.

PROCEDURES AS FUNCTIONS

In SECTION 5 function designators were discussed, and a list was given of those which are considered standard in ALGOL. It is possible for the ALGOL programer to create more functions. Two additional things are required of a **PROCEDURE** declaration in order for it to define the value of a function designator:

(1) An assignment statement must appear, in the procedure body, which has the procedure identifier to the left of the replacement operator.
(2) Since the **PROCEDURE** declaration is defining a single value, its type must be declared by preceding the word **PROCEDURE** by one of the three reserved words (**REAL, INTEGER,** or **BOOLEAN**) which indicate type.

Earlier in this section an example of a **PROCEDURE** declaration was shown which used the identifier FACTORIAL. It was constructed in the normal way. The following shows how this same **PROCEDURE** declaration might look if it were made into a function which (provided it has first been declared) can be used just as the standard functions can be used.

```
    INTEGER PROCEDURE FACTORIAL (N);
    VALUE N; INTEGER N;
    BEGIN INTEGER I, F; F ← 1;
    FOR I ← 2 STEP 1 UNTIL N DO
    F ← I × F; FACTORIAL ← F END
```

In order to make use of this procedure in a program, a notation called "function designator" is used. The function designator and the procedure statement have the same format:

Procedure identifier (list of actual parameters)

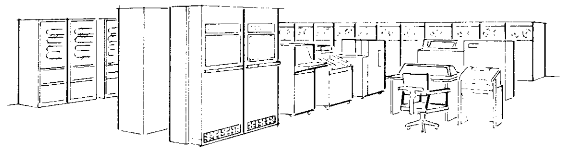
They differ in use only. The procedure statement stands alone, whereas the function designator is used as part of either an arithmetic or a logical expression. For instance, the following statements include function designators which refer to the above procedure.

Assignment statement:

BOB ← FACTORIAL (JOE)

Conditional statement:

IF P > FACTORIAL (Q) **THEN** **GO TO** TOWN; next statement



SECTION 8 SAMPLE PROBLEMS

For computer solution of a problem, it is necessary to have data (parameters of the problem) entered into the computer by some data input process, and to have results given back by means of a data output process. Because the formal ALGOL 60 language does not deal with input or output, no formal input or output symbolism will be given here. Sentences in the example programs which follow indicate which items are to be read into the computer or written out by the computer, to illustrate the sequence of such processes in a complete computer program.

The formal definitions for data input and data output may be found in the manual describing the extensions to ALGOL which form a part of the programing language for the D 850.

EXAMPLE 1

Given:

A curved line described by the equation:

(1) $Y = X^5 - \frac{X^3}{4} - 4X^2 + 21X - 5.238$

The equation for the slope (Y′) of the same curve:

(2) $Y' = 5X^4 - \frac{3}{4} X^2 - 8X + 21$

Find:

(a) The height of the curve above the X axis (value of Y) from equation (1) for values of X equal to 0, ½, 1, 1½, 2, 2½, . . . , 11½, 12.

(b) The values of Y′ for those same values of X.

ALGOL PROGRAM / EXPLANATORY REMARKS

BEGIN

The entire program is a block, made up of a type declaration and a **FOR** statement. The **FOR** statement includes an output statement.

REAL X, Y, YPRIME;

This type declaration indicates that the variables X, Y, and YPRIME may be assigned any value within the set of real numbers.

FOR X ← 0 **STEP** 1/2 **UNTIL** 12

The rest of the program (except for the final **END**) is a **FOR** statement. This is the **STEP-UNTIL** element with:

0 as the starting point,
1/2 as the increment, and
12 as the limit.

DO BEGIN	This begins the **DO** statement portion of the **FOR** statement. It is in itself a compound statement, made up of two assignment statements and an output statement.
Y ← X*5 − (X*3)/4 − 4 × X*2 + 21 × X − 5.238;	This is the first assignment statement, which assigns to Y the value of the arithmetic expression on the right.
YPRIME ← 5 × X*4 − (3 × X*2)/4 − 8 × X + 21;	This is the second assignment statement, which assigns to YPRIME the value of the arithmetic expression on the right.
{Print out the values of X, Y, and YPRIME}	This is where the output statement would be placed. It would result in printing three values each time the **DO** statement portion of the **FOR** statement is executed, or 25 times altogether.
END	End of the **DO** statement.
END	End of the block.

EXAMPLE 2

Given: (a) The series of numbers 17, 24, 31, 38, 45, 52, ..., where each successive term is found by adding 7 to the previous term, and where 17 is defined as the first term;

(b) Equations

(1) $L = A + (N - 1)D$,

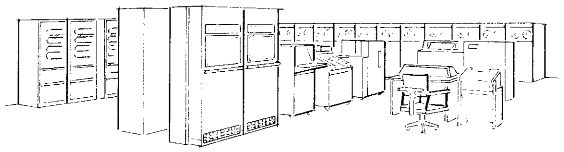
(2) $S = \frac{N}{2}(A + L)$

The following are the symbols and meanings:

A	is the first term of such an arithmetic series.
D	is the difference between two successive terms.
N	is the number of terms in the series up to the point in question.
L	is the Nth term of the series.
S	is the sum of the first N terms of the series.

Find: the value of the 50th term of the series by equation (1), and the sum of the first 50 terms by equation (2). Similarly, find the values of the 75th, 100th, 125th, 150th, etc., to the 300th term, and the sums of the terms to each of these points.

ALGOL PROGRAM	EXPLANATORY REMARKS
BEGIN	The entire program is a block made up of two type declarations, an output statement, and a **FOR** statement which includes another output statement.
REAL L, S;	This type declaration indicates that the variables L and S may be assigned any value within the set of real numbers.



INTEGER N;	This type declaration indicates that values assigned to the variable N must always be restricted to those numbers in the set of integers.
{Print column headings "N," "L," and "S"}	This represents an output statement which would result in printing the letters N, L, and S at the top of a page.
FOR N ← 50 **STEP** 25 **UNTIL** 300	The rest of the program (except for the final **END**) is a **FOR** statement. This is the **STEP-UNTIL** element, with: 50 as the starting point, 25 as the increment, and 300 as the limit.
DO BEGIN	The **DO** statement portion of the **FOR** statement starts here. It is in itself a compound statement, made up of two assignment statements and an output statement.
L ← 17 + (N − 1) × 7;	The first assignment statement.
S ← N/2 × (17 + L);	The second assignment statement.
{Print out the values of N, L, and S}	This represents an output statement which would print the values of N, L, and S under the appropriate column headings. Eleven such printouts would result.
END	End of **DO** statement.
END	End of block.

EXAMPLE 3

Given: (1) $A = 2 \times W \times L + 2 \times W \times H + 2 \times L \times H$

(2) $R = \dfrac{A}{4\pi}$

(3) $r = \dfrac{A}{4R\pi^2}$

Find: (a) The total outside surface area of a rectangular box, where the width (W), length (L), and height (H) are 12 inches, 27 inches, and 14 inches, respectively. Use equation (1).

(b) The radius of a sphere which has the same surface area as the box. Use equation (2).

(c) The radius (r) of the cross section of a torus (doughnut) such that the torus will have the same surface area as the box and sphere, using the radius of the sphere (R) as the major radius (R) of the torus. Use equation (3).

ALGOL PROGRAM	EXPLANATORY REMARKS
BEGIN	The entire program is a block made up of two type declarations, six assignment statements, and an output statement.

```
REAL A, RS, RT;
INTEGER W, L, H;
  W ← 12;
  L ← 27;
  H ← 14;
  A ← 2 × W × L + 2 × W × H + 2 × L × H;
  RS ← SQRT (A/(4 × 3.14159) ) ;
  RT ← A/(4 × RS × 3.14159 * 2) ;
{Print out values of A, RS, and RT}
END
```

{Print out values of A, RS, and RT}: This represents the output statement. This program would result in one printout.

END: End of block.

EXAMPLE 4

Given: (1) Mean value = A = $(X_1 + X_2 + X_3 + \ldots + X_n)/n$

(2) Standard Deviation (Σ) = $\sqrt{\dfrac{X^2_1 + X^2_2 + \ldots + X^2_n}{n} - A^2}$

where $X_1, X_2, \ldots, X_n$ is a set of values, and n is the number of values.

Find:
(a) The solution of equation (1) for the mean value of the set.
(b) The solution of equation (2) for the standard deviation.
(c) The identification and value of the term which differs most from the mean value. (Identify by subscript number.)
(d) Which (if any) of the terms (values) are exactly equal to the mean value. (Identify by subscript numbers.)

Method:
1. Solve the first equation for the value of A.
2. Solve the second equation for the value of the standard deviation.
3. Find the maximum of the absolute values of the following expressions, and record the subscript number.

   $(X_1 - A), (X_2 - A), (X_3 - A), \ldots, (X_n - A)$.
4. Compare $X_1, X_2, \ldots, X_n$ successively with A, and record the subscript numbers of the terms which are equal to A.

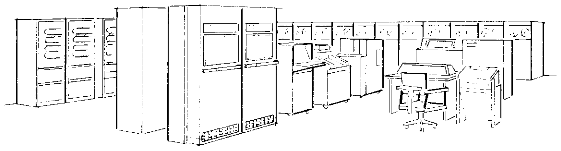
The ALGOL statements shown below result.

(This section of coding computes the mean value.)

```
SUM ← 0
FOR I ← 1 STEP 1 UNTIL N DO SUM ← SUM + X [I] ;
A ← SUM/N;
```

(This section computes the standard deviation.)

```
SUMSQ ← 0;
FOR I ← 1 STEP 1 UNTIL N DO SUMSQ ← SUMSQ + X [I] * 2;
STANDEV ← SQRT (SUMSQ/N - A * 2) ;
```



(This section computes the maximum deviation.)

```
MAXDEV ← ABS (X [I] − A) ; INDEX ← 1;
FOR I ← 2 STEP 1 UNTIL N DO BEGIN
Z ← ABS (X [I] − A) ; IF Z > MAXDEV THEN BEGIN MAXDEV ← Z; INDEX ← I END END
```

(This section finds the subscript numbers of the terms equal to A.)

```
J ← 1;
FOR I ← 1 STEP 1 UNTIL N DO
IF X [I] = A THEN BEGIN Y [J] ← I; J ← J + 1 END
```

This program in ALGOL as presented is almost complete, but a few additional constructions are required.

A. Because every ALGOL program is considered a block in itself, it is necessary to place **BEGIN** and **END** around the program.

B. Because all identifiers of a program must be declared, the following declarations must be included, and placed at the beginning of the block:

```
INTEGER N, I, J, INDEX;
INTEGER ARRAY Y [1:1000];
REAL A, SUM, SUMSQ, STANDEV, MAXDEV, Z;
REAL ARRAY X [1:1000];
```

Note that these declarations of arrays with a subscript range of 1 to 1000 allow up to 1000 values of X to be used, and similarly up to 1000 values of Y (in case all values of X were equal and therefore all equal to A).

C. As pointed out earlier, data input and data output statements are also required to perform the following:

(1) Read data input values of $X_1, X_2, \ldots, X_n$, and n.

(2) Print an output page heading which reads:

COMPUTATION OF MEAN VALUE AND STANDARD DEVIATION.

(3) Print the values found for A, STANDEV, MAXDEV, and INDEX.

(4) Print a heading which reads:

SEQUENCE NUMBERS OF THE TERMS EQUAL TO THE MEAN VALUE.

(5) Print out the (J − 1) values in the set Y [J] (these being the sequence numbers of (4) above).

(6) If there are no terms equal to the mean value, then, instead of (4) and (5) above, print out a line which reads:

NO VALUES OF X [I] ARE EQUAL TO A.

ALGOL PROGRAM

```
BEGIN
```

EXPLANATORY REMARKS

The entire program is a block made up of two type declarations, two array declarations, an input statement, seven assignment statements, four **FOR** statements, an **IF** statement, and two output statements.

```
  REAL A, SUM, SUMSQ, STANDEV, MAXDEV, Z;
  INTEGER N, I, J, INDEX;
  INTEGER ARRAY Y [1:1000];
  REAL ARRAY X [1:1000];
```

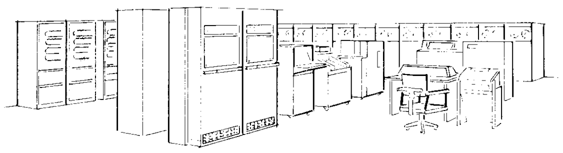
{Read data input values of X_1, X_2, X_3, . . . , X_n, and n};	This represents the input statement which would cause n + 1 values to be read.
SUM ← 0;	
FOR I ← 1 **STEP** 1 **UNTIL** N **DO** SUM ← SUM + X [I];	The first **FOR** statement.

A ← SUM/N;
SUMSQ ← 0;
FOR I ← 1 **STEP** 1 **UNTIL** N **DO**
SUMSQ ← SUMSQ + X [I]*2;
	The second **FOR** statement.

STANDEV ← **SQRT** (SUMSQ/N − A*2);	
MAXDEV ← **ABS** (X [I] − A);	
INDEX ← 1;	
FOR I ← 2 **STEP** 1 **UNTIL** N	The third **FOR** statement starts here.
DO BEGIN Z ← **ABS** (X [I] − A);	This is the **DO** statement portion, which is itself a compound statement made up of an assignment statement and a conditional statement.
IF Z > MAXDEV **THEN**	This is the conditional statement, of which another compound statement is a part.

BEGIN MAXDEV ← Z;
INDEX ← I **END**	

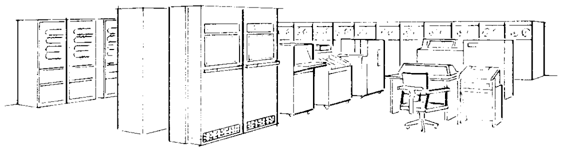
END;	This terminates the third **FOR** statement.
{Print heading which reads: COMPUTATION OF MEAN VALUE, STANDARD DEVIATION, AND GREATEST DEVIATION}	The first output statement, which results in printing a heading.
{Print values of A, STANDEV, MAXDEV, and INDEX}	The second output statement, which results in printing four values.
J ← 1;	
FOR I ← 1 **STEP** 1 **UNTIL** N	The fourth **FOR** statement.
DO IF X [I] = A **THEN**	The **DO** statement portion.
BEGIN Y [J] ← 1; J ← J + 1 **END**;	
IF J = 1 **THEN** {Print out a line which reads: NO VALUES OF X [I] ARE EQUAL TO A.}	This is the conditional statement, and the output statement which will be executed if J = 1.
ELSE BEGIN	Compound statement (made up of two output statements), which will be executed if J ≠ 1.
{Print a heading line which reads: SEQUENCE NUMBERS OF TERMS EQUAL TO THE MEAN VALUE}	
{Print out the sequence numbers, which are the (J − 1) terms of the Y array.}	
END	End of conditional statement.
END	End of block.



APPENDIX A

BURROUGHS D 850 ALGOL 60 SYMBOLS

		BURROUGHS D 850 ALGOL 60 HARDWARE REPRESENTATION	FORMAL ALGOL 60 REFERENCE LANGUAGE SYMBOLS
	Blank		
	Decimal Point	.	.
	Comma	,	,
	Colon	:	:
	Semicolon	;	;
	Left Parentheses	(	(
	Right Parentheses	)	)
	Left Bracket	[	[
	Right Bracket	]	]
	Replacement Operator	←	:=
Relational Operators	Less	<	<
	Less or Equal	≤	≦
	Equal	=	=
	Greater or Equal	≥	≧
	Greater	>	>
	Not Equal	≠	≠
Logical Operators	And	∧	∧
	Or	∨	∨
	Not	¬	¬
	Equivalent	EQV	≡
	Implies	IMP	⊃
Arithmetic Operators	Add	+	+
	Subtract	−	−
	Multiply	×	×
	Divide	/	/
	Integer Divide	DIV	÷
	Exponentiate	*	↑
Alphabetic Characters	A through Z		A, a through Z, z
Numeric Characters	0 through 9		0 through 9



APPENDIX B

DEFINITIONS OF EXPONENTIATION OPERATIONS

These definitions of results are for the operation A^n, where A is to be raised to the nth power. A represents a number of either type **REAL** or type **INTEGER**.

(I) Where n is a number of type **INTEGER**:

(1) If n is greater than zero, and A is not equal to zero, then $A^n = A \times A \times A \times \ldots \times A$ (n times). The result is of type **REAL** if A is type **REAL**, or type **INTEGER** if A is type **INTEGER**.

(2) If n is greater than zero, and A equals zero, the result is zero.

(3) If n equals zero, and A is not equal to zero, then A^n equals 1, and the type of the answer is the same as the type of A.

(4) If n equals zero, and A also equals zero, then the result is undefined.

(5) If n is less than zero, and A equals zero, then the result is undefined.

(6) If n is less than zero, and A is not equal to zero, then $A^n = 1/(A \times A \times A \times \ldots \times A)$ where the denominator has n factors, and the result is always of type **REAL**.

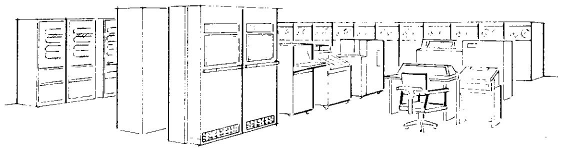
(II) Where n is a number of type **REAL**:

(1) If A is greater than zero, then $A^n = (e)^{n \times \ln(A)}$, and the result is always of type **REAL**.

(2) If A equals zero and n is greater than zero, the result is zero.

(3) If A equals zero, and n is equal to zero or less than zero, the result is undefined.

(4) If A is less than zero, the result is undefined.

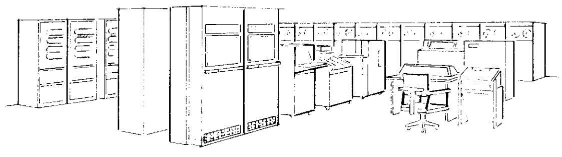


APPENDIX C

GLOSSARY

Algorithm:	A statement of the steps to be followed in the solution of a problem.
Argument:	One of the independent variables upon whose value that of the function depends.
ARRAY:	An ordered arrangement of items, such as a determinant, matrix, or vector.
Function:	(1) An element whose value is so related to some other element (or elements) that it is dependent upon those secondary elements (arguments).
	(2) The relationship which defines the value of a dependent variable based on the value(s) of the independent variable(s) (arguments).
Instruction:	In conventional computer usage, an instruction is a symbol (or group of symbols) recognized by the computer as an order to perform an operation. Typical instructions include: add, multiply, read a punched card, write on magnetic tape, store in memory, and change control to another part of the program if a certain condition (such as a minus sign) exists. Under older methods, a programer had to learn from 30 to 150 different instruction symbols, and all their individual variants and peculiarities.
INTEGER:	An ALGOL reserved word indicating that the numbers or variables so designated may have only numerical values which are whole numbers or zero.
Integer:	A whole number, either positive or negative, containing no fractional or decimal part. Zero may be an integer.
Integral:	An adjective indicating the integer or whole-number portion of the modified term.
Inverse:	The inverse of N is 1/N; the inverse of B/A is A/B. "Reciprocal" is used with the same meaning.
Machine Language:	The system of codes by which instructions and data are represented internally within a particular data processing system.
Operand:	Any of the quantities entering into an operation.
Operator:	A symbol which specifies that a defined action is to take place.
Power:	Raising a number to the power of n means multiplying the number by itself n times; thus, in ALGOL notation, 10*3 means 10 × 10 × 10, or 1000. This process is also called exponentiation; in the above example, 3 is the *exponent* of 10.
Problem language:	The words and symbols used in the original formal statement of a problem, as for hand computation.
Program:	An ordered list of instructions which directs the computer to perform certain operations in a specified sequence to solve a problem.

Programer:	One who designs a set of computer operations to solve a problem (see Program).
Pseudo-Language:	An arbitrary system of codes, independent of the hardware of a computer, which is used to express the steps in a computer program. It usually will be converted (translated) into an equivalent set of machine language codes which are actually used to perform the program on a computer.
REAL:	An ALGOL reserved word indicating that the numbers or variables so designated may take on any real-number value.
Real Numbers:	The set of *all* positive and negative numbers, including integers and zero, but excluding any imaginary or complex numbers.
Reciprocal:	See Inverse.
Scale Factor:	A factor by which a quantity in a problem is multiplied or divided to determine the location of the decimal point.
Subscript:	In algebra, a subscript is a number or letter, appearing below the center of a line of other symbols, which represents the position of an element in an array. For example: A_5 is the fifth element in an array called A. In ALGOL, subscripts are enclosed in brackets; the above example would be written A[5] in ALGOL. In a two-dimensional array (i.e., one with rows and columns), two subscripts are used to select the row and column. For example: $B_{5,3}$ names the element in the fifth row, third column, of array B; in ALGOL we would write B[5,3].



APPENDIX D

RESERVED WORD LIST

The following reserved words, which form a major portion of the D 850 ALGOL vocabulary, may be used *only as shown in this manual.*

ABS
ARCTAN
ARRAY
BEGIN
BOOLEAN
COS
DIV
DO
ELSE
END
ENTIER
EQV
EXP
FALSE
FOR
GO TO
IF
IMP
INTEGER
LN
LOG
OWN
PROCEDURE
REAL
SIGN
SIN
SQRT
STEP
SWITCH
THEN
TRUE
UNTIL
VALUE
WHILE

Burroughs

Burroughs Corporation

Detroit 32, Michigan

NEW DIMENSIONS / in computation for military systems