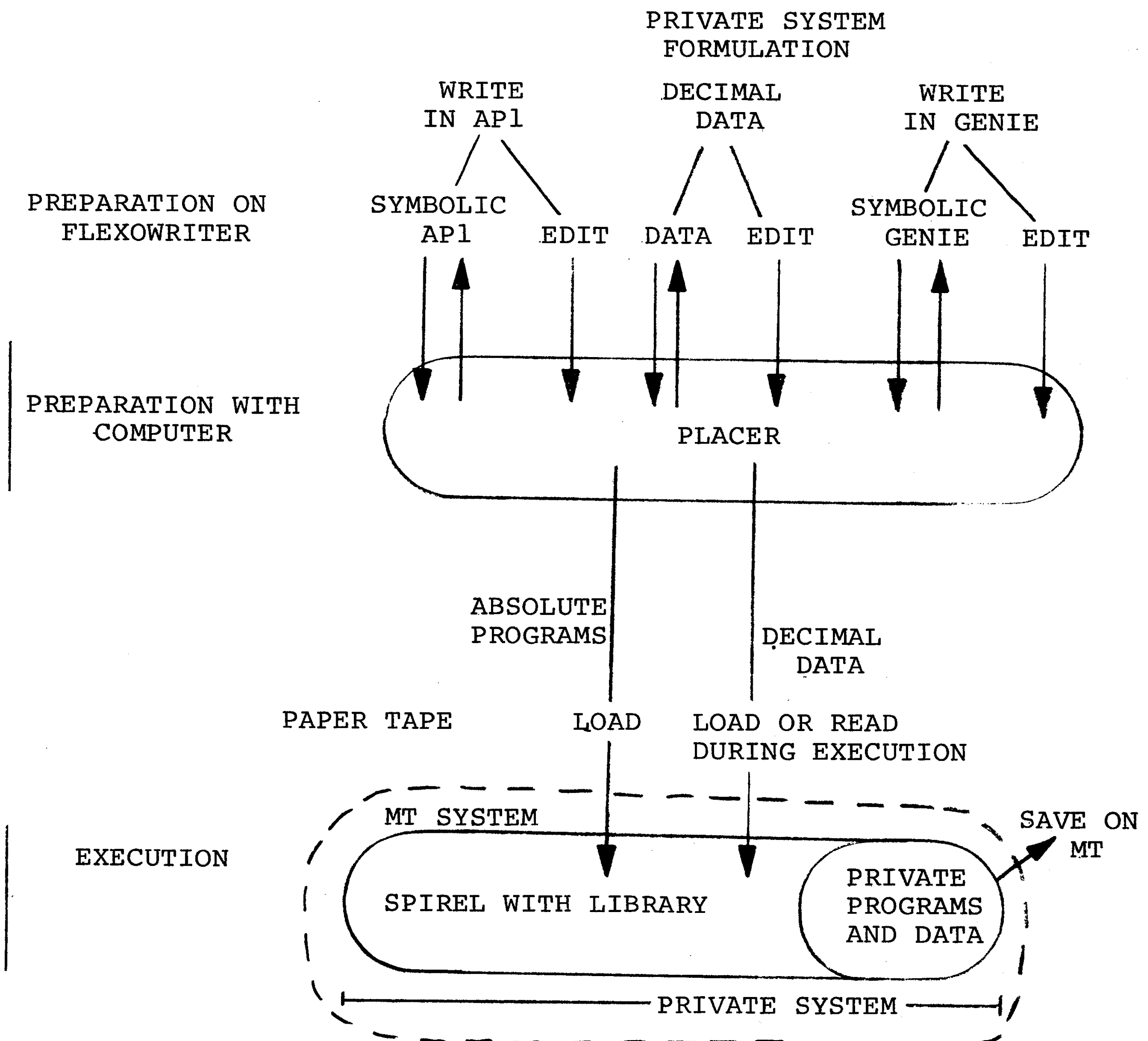
PROGRAMMING SYSTEMS

How does one use the programming systems for the Rice Computer?

The solution to a private computing problem is a private system of programs and data. The diagram illustrates use of the Rice Computer programming systems in the construction and execution of a private system.



May, 1967

A private system consists of:

the operating system with library programs + private data and programs

So, private programs and data are added to SPIREL to form a private system.

SPIREL is the operating system for the Rice Computer. It may be used from the console or dynamically by private programs for such operations as:

reading, correcting, printing, punching } of programs and data

execution of programs

Dynamic storage control is provided automatically by SPIREL, and various debugging aids are available in the system.

The SPIREL system is composed of SPIREL and a library of programs which perform functions likely to be useful to private programs. Any private program in a private system built on SPIREL may use any library program as a subroutine. The library provides functions such as:

trigonometric functions

vector and matrix operations

formatted printing and plotting

The construction of a private system entails the preparation of private programs and data. PLACER is the programming system used for preparation of system elements; it has nothing to do with system execution. It works on one program or one data tape at a time, and provides the following operations:

editing, listing on the printer, punching } of programs or data

translation of programs from symbolic to executable form and vice versa

Programs are written in a symbolic langauge:

AP1, the assembly language in which statements correspond one-to-one with instructions to be executed by the computer

or Genie, the formula language in which mathematical notation is employed and which contains as a subset the AP2 assembly language

PLACER contains the AP1 assembly program and the Genie compiler. AP1 translates programs written in the AP1 language to executable form. The Genie compiler translates programs written in the Genie formula language to executable form.

The magnetic tape system provides

filling of memory from a block on magnetic tape

dumping of the contents of memory onto magnetic tape

SPIREL and PLACER are on magnetic tape and are read into memory by use of the magnetic tape system. Once a private system has been composed in memory, the magnetic tape system may be used to write the system onto tape. Subsequent use does not require loading of private items from paper tape.

PLACER

PLACER

Function of PLACER . . . .

PLACER Operation . . . .

Read . . . .

Edit . . . .

Punch . . . .

List . . . .

Check . . . .

Translate . . . .

Back-Translate . . . .

Run . . . .

The PLACER system is designed to facilitate preparation of programs for the Rice Computer.

Programs exist on paper tape in two forms:

symbolic -- in the assembly language or Genie language.

absolute -- in machine language, as translated from symbolic form from assembly or Genie language, ready to be loaded for execution.

PLACER operations which may be applied to program tapes are:

<u>read</u> symbolic tape -- forming "tape image" in memory

<u>edit</u> image -- change, insert, or delete lines in image, per edit instructions on paper tape

<u>punch</u> image -- on paper tape

<u>list</u> image -- on printer

<u>check</u> symbolic tape -- against image

<u>translate</u> image -- assemble or compile as appropriate, and produce absolute tape

<u>back-translate</u> absolute tape -- read absolute tape, "back translate" into symbolic assembly language, forming image as if symbolic tape had been read.

PLACER operations may also be applied to data tapes which are manually prepared on the flexowriter. In this case, the translation operations would not be meaningful.

When PLACER is read into memory from magnetic tape program *340 is executed, and the main PLACER stop

(I): 00 HTR CC

occurs. One or several PLACER operations may then be designated in the sense lights:

SL^1 --- read symbolic tape
SL^2 --- edit
SL^3 --- punch (edited) symbolic tape
SL^4 --- list (edited) symbolic tape
SL^5 --- check (edited) symbolic tape punched
SL^6 --- translate (edited) symbolic tape
SL^7 --- back-translate absolute tape
SL^{15} -- run with SPIREL
SL^{10}-SL^{15} --- run with specified system tape block

The original tape to be processed should be placed in the reader. SL^7 is used if this tape is absolute, and SL^1 is used if it is symbolic. It is not meaningful to elect both SL^7 and SL^1 operations in PLACER.

Pushing CONTINUE at the main PLACER stop with more than one operation designated causes the operations to be carried out in the following order:

SL^1 or SL^7 --- read symbolic tape or read absolute tape and back-translate to AP1 symbolic, forming symbolic image in the machine
SL^2 --- wait for edit tape, then edit image
SL^3 --- punch (edited) image, generating symbolic tape
SL^4 --- list (edited) image on the printer
SL^5 --- check symbolic tape against edited image if tape is ready in the reader; if tape is not ready and translation operation is designated, go on to translation and

return to check after translation; if tape is not ready and translation is complete or not designated, wait for tape; if tape does not check, do not exercise run option

SL^6 --- translate edited image, generating output on the printer and absolute tape

SL^{10}-SL^{15} --- obtain designated block from magnetic tape for running, SPIREL if only SL^{15}

If only one operation is designated in SL^1 through SL^7 at the main PLACER stop, pushing CONTINUE will cause a stop for that operation:

(I): 0i HTR CC

for SL^i on. Then options for the particular operation may be designated in the sense lights before pushing CONTINUE to cause the operation to be carried out.

The PLACER operations and the options available for each are explained in detail in the succeeding sections.

SL[1], READ

The symbolic tape to be read must begin with a carriage return. All characters beyond the last cr on the tape are ignored by the system. When the reading is complete, the system has in the machine a tape image.

Options

If *only* SL[1] is turned on at the main PLACER stop so that only the READ operation is designated, the stop

(I): 01 HTR CC

occurs. READ options may then be designated in the sense lights as follows:

SL[15] causes reading to terminate at the first double carriage return punch. A double carriage return is any two carriage returns not separated by a printable character. Here printable characters include those represented by a backward arrow; nonprintable characters include only the space, the tab, case punches, and the carriage return itself.

Pressing CONTINUE with *no* tape in the reader will cause exit to the console typewriter. A program may then be typed in, using exactly the format used on the flexowriter. The backspace key will *not* properly backspace over the characters "=⃒, <⃒, ≤⃒". It will *not* backspace beyond carriage returns. To *erase* a line, type a question mark (?). To erase the entire input text, type the sequence ???. Depressing the "index" key will cause exit of the read option.

SL^2, EDIT

The stop

(I): 02 HTR CC

occurs. The edit tape is placed in the reader.† Pushing CONTINUE causes this tape, which must contain only the corrections for the tape image in the machine, to be read. When reading is complete, PLACER's tape image in the machine is edited.

Each edit of a tape image requires specification of a range of lines in the tape image to be affected by the edit. The carriage return numbers for the original image are used for this purpose. A line in a symbolic tape is terminated by a carriage return, these being numbered from 1 on the listing. The edit range is specified by initial carriage return i and final carriage return f, interpreted as from and not including carriage return i through carriage return f. Such a range will be denoted (i,f) here.

Each edit is one of the following:

- replacement of lines (i,f) in the image with n (octal) symbolic lines read from the edit tape. The specification is punched

(l.c.)i(sp)f(sp)n(cr)

The n lines of the replacement follow the specification on the edit tape, and each line is terminated by a carriage return.

- deletion of lines (i,f) in the image. The specification is punched

(l.c.)i(sp)f(sp)0(cr)

No symbolic lines accompany this specification on the edit tape. Deletion is just the case of replacement with n = 0.

- insertion in the image after carriage return i of n (octal) symbolic lines read from the edit tape. The specification is punched

(l.c.)i(sp)i(sp)n(cr)

Insertion is just the case of replacement with i = f, a null range to designate position only.

- "move" for replacement of lines (i_1,f_1) in the image with lines (i_2,f_2) in the image. The specification is punched

(l.c.)i_1(sp)f_1(sp)i_2(sp)f_2(cr)

No symbolic lines accompany this specification on the edit tape. The lines (i_2, f_2) are deleted from their former position.

o "move" for insertion in the image after carriage return i_1 of lines (i_2, f_2) in the image. The specification is punched

$$(l.c.)i_1(sp)i_1(sp)i_2(sp)f_2(cr)$$

No symbolic lines accompany this specification on the edit tape. The lines (i_2, f_2) are deleted from their former position.

Edit specifications may overlap since the carriage return numbers are preserved in all edit operations internally. Numerous edits of the tape image are possible using the latest carriage return numbers.

There are no EDIT options.

† Pressing CONTINUE with no tape in the reader at the edit halt will cause exit to the console typewriter. Edits may then be typed exactly in the format used on the flexowriter, being certain to type an initial carriage return before the first edit specification. If this option is reached accidently, depressing the index key immediately will cause the edit halt to reappear. The index key is used also to exit the edit mode. As in the read option a question mark (?) will erase a line. Do not, however, use the 3 question mark sequence.

SL^3, PUNCH

The tape image (symbolic version) is punched out on paper with corrections if editing was done. It is advised that the CHECK option always be used with this operation.

There are no PUNCH options.

SL^4, LIST

The tape image in the machine is listed on the printer with carriage return numbers. If the tape begins with 'DEFINE' (as do Genie program tapes), superscript and subscript lines will be printed above and below the base line. Other tapes will be listed with more lines per page, one line per carriage return number with superscript printed as the character '↑' and subscript as '↓'. A lower case Roman letter after f is printed as '. upper case letter'.

If a line ends in a superscript or subscript position, it is followed by a message noting the displacement.

Options

If only SL^4 is turned on at the main PLACER stop so that only the LIST operation is designated, the stop

(I): 04 HTR CC

occurs. LIST options may then be designated in the sense lights as follows:

SL^{13} forces printing of separate superscript and subscript lines.
SL^{15} causes double spacing on the listing.

SL[5], CHECK

If the tape to be checked is not in the reader, the stop

(I): 05 HTR CC

occurs. Pushing CONTINUE causes the tape that is read to be compared to the tape image in the machine. An error print is given if the comparison fails.

There are no CHECK options.

SL^6, TRANSLATE

The tape image in the machine is translated, by the Genie compiler if it begins with 'DEFINE', by the assembly program otherwise.

Both translation procedures produce output on the printer and absolute program tapes. Details are given in the literature on the assembly and Genie languages.

Assembly Options

If only SL^6 is turned on at the main PLACER stop so that only the TRANSLATE operation is designated, the stop

(I): 06 HTR CC

occurs. TRANSLATE options for assembly may then be designated in the sense lights as follows:

SL^9 causes assembly output on the printer to be double spaced.
SL^{11} suppresses punching of the absolute tape.
SL^{13} causes punching of a self-loading absolute tape. Such a tape will load by using the LOAD switch on the console. See assembly language literature for more details.

Note that SL^{14} and SL^{15} are turned on automatically and should be left on.

Also, if TRANSLATE is selected with other operations at the main PLACER stop, assembly options may be designated in the indicator lights (IL^9, IL^{11}, IL^{13} as above).

Compilation Options

If only SL^6 is turned on at the main PLACER stop so that only the TRANSLATE operation is designated, the stop

(I): 06 HTR CC

occurs. TRANSLATE options for compilation may then be designated

in the sense lights as follows:

$\underline{SL^{12}}$ punches the internal portion of the Symbol Table, including internal names and statement labels, as a program tail. See SPIREL-CONSOL COMMUNICATION for details on use.

$\underline{SL^{13}}$ suppresses output of absolute tape.

$\underline{SL^{14}}$ provides condensed compilation on the printer -- only the first instruction of each command sequence and no Symbol Table.

$\underline{SL^{15}}$ causes output during compilation of intermediate code forms -- sets and phase 1 code. This is rarely of interest to the general user.

Also, if TRANSLATE is selected with other operations at the main PLACER stop, compilation options may be designated in the indicator lights (IL^{12}, IL^{13}, IL^{14}, IL^{15} as above).

SL[7], BACK-TRANSLATE

If the absolute tape to be translated is not in the reader, the stop

(I): 07 HTR CC

occurs. Pushing CONTINUE causes the absolute tape to be read, and a symbolic tape image of an equivalent AP1 program is constructed in memory. This image is no different from one generated by the READ operation.

Details of Back-Translation

Several types of program tapes are recognized by the back-translator which generates an appropriate ORG pseudo-order in each case:

program to be loaded by SPIREL at a fixed location

program to be loaded by SPIREL with numbered or named codeword

program to be loaded by SPIREL as an element of a numbered or named array

program to be loaded with the console LOAD switch at a fixed location or at the setting of B6

In normal use, the process of back-translation takes place in two phases:

1) flow analysis from word 1 of the program to determine which words may be executed as instructions and which are internal data words or constants

2) construction of a symbolic tape image to represent the program, with OCT pseudo-orders for constants and symbolic labels only on lines which are referenced by instructions within the program

Information is passed from the first phase to the second by tagging the words of the program as they are classified. The tag conventions are:

no tag on data words not explicitly referenced in the program

tag 1 on data words explicitly referenced in the program and on all cross-reference words

tag 2 on instructions not explicitly referenced in the program

tag 3 on instructions referenced in the program

Tag 0 may also indicate an instruction which cannot be identified as such.

It is possible for a program to be written in such a way that the flow analysis will not distinguish properly between instructions and constants. Three of the most common programming situations which cause analysis problems are:

- entry points at other than the first instruction of a program
- use of transfer vectors or computed transfers within a program (e.g., TRA CC+B3)
- use of the X register, as in JMP in the operation field or CC+X in the auxiliary field

BACK-TRANSLATE options (discussed below) make it possible to specify as executable instructions those words which would not otherwise be identified as such.

Options

If only SL^7 is turned on at the main PLACER stop so that only the BACK-TRANSLATE operation is designated, the stop

(I): 07 HTR CC

occurs. BACK-TRANSLATE options may then be designated in the sense lights as follows:

SL^{12} suppresses flow analysis; tape image construction is performed on the basis of the tags on the program as read.

SL^{13} causes the back-translator to accept a list of words which must be identified as instructions. Immediately after the program tape is read, the stop

(I): 13 HTR CC

occurs. A tape listing words to be identified as instructions is read. The format is

[cr] AAAAA [cr] BBBBB [cr] CCCCC ......

where [cr] is a 'carriage return' punch and AAAAA, BBBBB, CCCCC, ... are five-digit (octal) relative locations in the program. Note that it is only necessary to specify the first word of a block of instructions and analysis will find the others; a block is ended by an unconditional transfer instruction, either explicit or implicit.

$\underline{SL^{14}}$ causes punching of the program with tags after flow analysis.

$\underline{SL^{15}}$ suppresses construction of the symbolic tape image.

SL^{10}-SL^{15}, RUN

After options designated by SL^{1} through SL^{7} are complete, the octal number NN in SL^{10}-SL^{15} designates that block NN is to be loaded from the system tape. The following is a special case:

NN = 1, or SL^{15} only, for the closest SPIREL

In any case, the system obtained from magnetic tape is "fresh"; the program operated on by other PLACER options is not loaded.

ASSEMBLY LANGUAGE

ASSEMBLY LANGUAGE

Symbolic Coding . . . .

Instruction Form . . . .

Types of Symbols . . . .

Instruction Content . . . .

Operation Codes . . . .
- Class 0, Tests and Transfers
- Class 1, Arithmetic
- Class 2, Fetch, Store, Tags
- Class 4, B-Registers, Lights, Special Registers, Shifts
- Class 5, Logic and Fast Registers
- Class 6, Input-Output
- Class 7, Analog Input, Shifts, Delays
- Summary of Operation Codes

Pseudo-Orders . . . .
- ORG and END
- EQU
- BSS and BES
- BCD, FLX, REM
- DEC and OCT
- REF

Macro-Orders . . . .
- Application
- Definition
- Call
- Examples

Assembly Procedure . . . .

Coding Examples . . . .

SYMBOLIC CODING

The absolute machine language of the Rice Computer is described in detail in the Rice Computer Manual. In practice, programs are not written in the absolute language of the computer but in a symbolic language. A language which provides symbolic notation for instructions, or commands, that correspond one-for-one with absolute machine instructions is called an assembly language. The program which translates assembly language into machine language is called an assembly program.

Use of the assembly language for the Rice Computer depends on a knowledge of the absolute machine instruction format, a familiarity with the registers of the computer, and a general acquaintance with the instruction repertoire -- all explained in the Rice Computer Manual. Two forms of the Rice Computer assembly language are available:

AP1, for independent use

AP2, for use within Genie programs

The corresponding assembly programs have the same names:

AP1, an independent assembly program

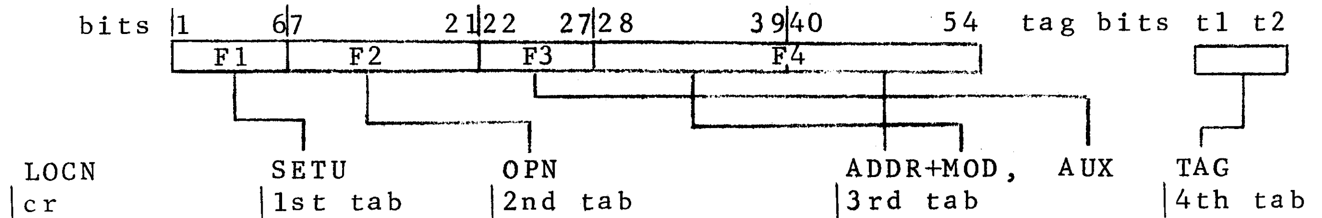
AP2, a subset of the Genie compiler

The two assembly languages are very similar. The major distinction concerns octal and decimal numerals. In AP1, all numeric constants are assumed to be octal unless immediately preceded by the special symbol "d", meaning decimal. In AP2, all numeric constants are assumed to be decimal, except when octal form is indicated by a plus sign immediately preceding the octal number.

In the following discussions, M stands for the final number formed in the last 15 bits of I (the instruction register) after all specified indirect addressing and B-modification has taken place; and if Q is any machine location, then (Q) stands for the contents of location Q.

INSTRUCTION FORM

The general form of an AP1 or AP2 instruction and its correspondence to a machine-language instruction as explained in the Rice Computer Manual is



Here "cr" denotes "carriage return", and "tab" denotes "tabulate" on the flexowriter used for preparation of input to the assembly programs.

LOCN gives the symbolic label (if any) on the instruction.

SETU corresponds to Field 1: bring a "fast" register to U; then inflect (U).

OPN corresponds to a Field 2 operation chosen from one of seven classes.

AUX corresponds to Field 3: alter a B-register, send (U) or (R) to a "fast" register, send the M portion of I to a B-register, or clear R.

ADDR+MOD corresponds to Field 4: compute the final address M, sending M to the last 15 bits of I; load S with M or (M); then inflect (S).

All fields may be symbolically coded. All fields but MOD and TAG may be coded numerically.

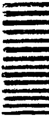
If no TAG is to be specified, the 4th tab may be omitted. If no AUX operation is to be specified, the preceding comma may be omitted.

Precise definitions of the allowed symbols are as follows:

Register names. The following symbols are used as names of "fast" registers:

A-series Z, U, R, S, T4, T5, T6, T7

B-series CC, B1, B2, B3, B4, B5, B6, PF



These may appear in SETU, ADDR+MOD, and AUX fields. The symbol I may be used in SETU and AUX. The special register names may be used in ADDR; these are

SL	sense lights
IL	indicator lights
ML	mode lights
TL	trapping lights
P2	second pathfinder
X	increment register
TT	"to-tape" register
FT	"from-tape" register

These symbols may be used only as register names.

Special characters. *, a(AP1) or #(AP2), d(AP1), +, -, |, →, (,), "tab", "cr", and , (comma).

Operation codes. These include the mnemonic operation codes in the assembly vocabulary, pseudo-operation codes (AP1 only), macro-operations (AP1 only), and general symbols defined by the user as operation codes with a LET (in Genie for AP2) or an EQU (in AP1). All of these areas are covered in later discussions.

General names. In AP2, a private name may be

a single lower case Roman letter

or an upper case Roman letter, followed by upper case Roman letters, followed by lower case Roman letters, followed by numerals.

In AP1, a private name may be

an upper case Roman letter, followed by upper case Roman letters, followed by numerals.

Spaces may not appear in names. Any number of characters may form

a name; AP2 will retain the first four if lower case Roman letters are used, the first five otherwise; AP1 will retain the first six. The following are general names in AP1 and AP2: B, M3, COMM, ZETA2. The following are general names in AP2, but not in AP1: b, Comm, Zeta2. General names may appear only in the LOCN and ADDR fields.

Each field of the symbolic instruction has a well-defined form. If this form is not recognized by the assembly system, a message is printed during assembly. The acceptable contents of each field are as follows:

<u>LOCN</u>. This field may be blank or absolute or symbolic. Absolute LOCN fields are permitted only when an AP1 program is being assembled in absolute form (see the ORG pseudo-order discussion). A symbolic LOCN field may contain any general name. A name may not appear in LOCN more than once in any one program.

<u>SETU</u>. This field may be blank, absolute, or F, where F is an A- or B-series register name or "I", or any of the forms -F, |F|, or -|F|. If SETU is blank, "U" is understood and the octal equivalent 01 is inserted into the machine instruction. I sets U to the integer +1; -I sets U to the integer -1. Note that Z sets U to all zeroes; -Z sets U exponent to zero and U mantissa to minus zero, or all ones.

Examples: B1 |T4| -PF -|R| Z -I

If the T-flag is on for register Ti (i=4,5,6,7), indirect addressing through Ti will occur when Ti is addressed in the SETU field. To denote this mode of addressing the * may be used before the register name:

*Ti -*Ti |*Ti| -|*Ti|

This is a symbolic convenience only, and these will be translated as:

Ti -Ti |Ti| -|Ti|

<u>OPN</u>. This field may be absolute or an operation code. In the case of conditional transfers, a symbolic operation has the form IF(CCC)TTT where CCC represents test conditions and TTT is a mnemonic for a transfer order. Other symbolic operation codes consist of

one or more 3-letter mnemonics. Special symbols such as →, +, -, ",", and +i (where i is an octal integer) are sometimes permitted (see the section on operation codes).

AUX. This field may be blank, absolute, or one of the forms U→F, R→F, I→Bi, Bi+1, Bi-1, or Bi+X. Bi stands for one of the B-series register names; F is any A- or B-series register name; I refers to the last 15 bits of the instruction register; and X is the increment register. As a special case, R→Z causes R to be cleared to zero.

Example: U→T4 R→PF I→B1 B2+1 B3-1 B4+X

If the T-flag is on for register Ti (i=4,5,6,7), indirect addressing through Ti will occur when Ti is addressed in the AUX field. To denote this mode of addressing the * may be used before the register name:

U→*Ti R→*Ti

This is a symbolic convenience only, and these will be translated exactly as:

U→Ti R→Ti

ADDR+MOD. ADDR may be blank or absolute or symbolic, or the ADDR+MOD field may consist of an octal or decimal number to be used as an operand. MOD is either blank or one or more of the B-series register names, connected to ADDR by + signs. Special inflections control the IM and IA bits as follows: IM bit 1 is set to 1 (to load S with M instead of (M)) whenever the symbol "a" (AP1) or "#" (AP2) appears, or whenever certain OPN mnemonics are used (see the section on operation codes). IM bits 2 (absolute value) and 3 (minus) are controlled by the special forms -Q, |Q|, and -|Q|, where Q is an allowed ADDR+MOD symbol. The IA (indirect addressing) bit is set to 1 whenever the symbol "*" appears in this field.

If ADDR is symbolic, any A-series register name, any special register name, or any general name is acceptable. A general name may be followed by a relative part consisting of an integer preceded

by a + or - sign.

If ADDR is absolute, any octal integer of not more than 5 digits, or any decimal integer of absolute value not larger than 32,767, is permissible. Any octal or decimal integer above these limits or any floating point decimal number is treated as the name of a location containing that number; storage space is reserved for it at the end of the program. In this case, no MODs are allowed, and only the absolute value and - inflections are meaningful.

All characters appearing within parentheses in this field are ignored, so that an address field which is modified by the program may be conveniently noted. For example, (FWA)+B1+B2 is treated as Z+B1+B2. If a symbol appears in ADDR but never in LOCN, a blank location will be reserved at the end of the program. ADDR+MOD should not be blank; the Z character may always be used to produce a zero field.

Examples of equivalent AP1 and AP2 ADDR+MOD fields are:

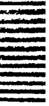
AP1	AP2
COMM+10 or COMM+d8	COMM+8 or COMM++10
-|A+B1-d12| or -|A+B1-14|	-|A+B1-12| or -|A+B1-+14|
a*ZETA	#*ZETA
d48	48
-ad122+B1	-#122+B1
B4+B5	B4+B5
00500	+00500
d2.009027	2.009027
777700000	+777700000
30	24

The only field which may be continued onto another line is ADDR+MOD, AUX by punching a "cr" followed immediately by three "tab" characters, so that continuation lines will follow under ADDR+MOD, AUX.

<u>TAG</u>. This field may be blank or symbolic. If no tag is desired, the 4th tab punch may be omitted. If a tag is desired, the TAG field must contain one of the mnemonics TG1, TG2, or TG3. The corresponding tag will be placed on the assembled instruction, printed on the octal listing, and punched with the instruction in checksum format.

OPERATION CODES

The most common Field 2 operations have been given names in the vocabulary of AP1 and AP2 for convenience in coding. All Field 2 operations are fully explained in the machine manual. The mnemonics defined in this section are summarized in a chart at the end of the section. These operation code symbols may not be used for any other purpose. Other Field 2 operations may be given general names by use of LET (in Genie for AP2) or EQU (AP1), and such symbols are then treated as operation codes throughout the program in which they have been defined.



• Class 0, Tests and Transfers

In the list below, the symbols are followed by their octal equivalents and a brief explanation of their meanings; the indication "a,#" means that the operation symbol automatically causes IM bit 1 to be set to 1 (to load S with M instead of (M)), since the operation indicated deals with M rather than with (S).

The four unconditional transfers are represented by:

		octal codes	
a,#	HTR	00000	Halt and transfer. Halt, setting CC to M when CONTINUE is pressed.
a,#	TRA	01000	Transfer. Set CC to M.
	SKP	02000	Skip. Subtract (S) from (U); then increment CC by 1, skipping the next order.
	JMP	03000	Jump. Subtract (S) from (U); then increment CC by (X), the increment register.

Conditional transfers have the form IF(CCC)TTT where TTT is one of the above transfer mnemonics, and CCC represent one, two, or three test conditions joined by + or × signs. Use of the + sign indicates that the specified transfer is to occur if any of the conditions listed is satisfied; use of the × sign indicates that the specified transfer occurs only when all of the conditions listed are satisfied simultaneously. A single order may not contain both + and × signs. One condition from each of the first three groups may be specified; or a Group IV mnemonic may be combined with a Group III test as noted. If a TRA or HTR is used, the specified test is made on (U). If a SKP or JMP is used, the specified test is normally performed on (U)-(S). The exceptions to this rule are noted below Group II.

Group I

	octal code	
PSN	00100	Positive sign. Is the sign bit of U equal to 0?
MOV	00200*	Mantissa overflow. Is Indicator Light #4 on?
EOV	00300*	Exponent overflow. Is Indicator Light #5 on?
NSN	00500	Negative sign. Is the sign bit of U equal to 1?
NMO	00600*	No mantissa overflow. Is Indicator Light #4 off?
NEO	00700*	No exponent overflow. Is Indicator Light #5 off?

*Note that indicator lights are turned off when tested.

Group II

		octal code	
	ZER	00010	Zero. Is (U) mantissa all 1's or all 0's?
	EVN	00020	Even. Is bit 54 of U equal to zero?
a,#	SLN	00030*	Sense light on. Are all the sense lights corresponding to 1's in M on?
	NUL	00040**	Null. Are all 54 bits of U zero?
	NZE	00050	Non-zero. Is (U) mantissa different from zero?
	ODD	00060	Odd. Is bit 54 of U equal to 1?
a,#	SLF	00070*	Sense light off. Are all the sense lights corresponding to 1's in M off?

*Note that sense lights are not altered when tested. SLN and SLF tests are meaningful only with SKP or JMP orders, and in these cases no subtraction takes place.

**If the NUL test is used with a SKP or JMP order, a logical comparison is made as follows: wherever a bit of R is equal to zero, the

bits in corresponding positions of U and S are compared. If (U) is identical with (S) in each of these positions, the resulting (U) is null and the NUL portion of the test is satisfied. If the NUL comparison is not satisfied, the resulting (U) is meaningless.

<u>Group III</u>

	octal code	
TG1	00001*	Tag 1. Is Indicator Light #1 on?
TG2	00002*	Tag 2. Is Indicator Light #2 on?
TG3	00003*	Tag 3. Is Indicator Light #3 on?
NTG	00004*	No tag. Are Indicator Lights #1, #2, #3 all off?
NT1	00005*	No tag 1. Is Indicator Light #1 off?
NT2	00006*	No tag 2. Is Indicator Light #2 off?
NT3	00007*	No tag 3. Is Indicator Light #3 off?

*Note that indicator lights are turned off when tested.

<u>Group IV</u>

	octal code	
POS	00110	Positive <u>or</u> zero. Is (U) mantissa greater than or equal to zero?
NEG	00510	Negative <u>or</u> zero. Is (U) mantissa less than or equal to zero?

A + sign must be used when combining either of these mnemonics with a Group III test.

	octal code	
PNZ	04150	Positive <u>and</u> non-zero. Is (U) mantissa strictly greater than zero?
NNZ	04550	Negative <u>and</u> non-zero. Is (U) mantissa strictly less than zero?

A × sign must be used when combining either of these mnemonics with a Group III test.

- <u>Class 1, Arithmetic</u>

In the list below, the symbols are followed by their octal equivalents and a brief explanation of their meanings.

Any Class 1 mnemonic may be followed by → or +1, to cause storing of the final (U) in the location addressed by M; by +2, storing (U) at location (B6); or by +3, storing (U) at location M+(B6). Octal codes may be joined by a '+' to Class 1 mnemonics for various special operations. If n is such an octal code, the combination appears as

mnemonic +n	in AP1
mnemonic ++n	in AP2

Any floating point mnemonic may be followed by +1j (j=0, 1, 2, or 3), causing the last bit of (U) to be set to 1 (rounded) after the operation but before storing. After floating point mnemonics +4j suppresses normalization of the result, +5j rounds and suppresses normalization. Other options are given in the machine manual.

The Class 1 mnemonics are as follows:

<u>Fixed point</u>

	octal code	
ADD	10000	Add. (U)+(S)→U.
SUB	10100	Subtract. (U)-(S)→U.
BUS	14100	Reverse subtract. (S)-(U)→U.
MPY	10200	Multiply. (U)×(S)→U,R (double length).
IMP	10220	Integer multiply. (U)×(S)→U.
DIV	10300	Divide. Double length (U,R)÷(S)→U, 2^{47}× remainder →R.
VID	16300	Reverse divide. (S)÷(U)→U, 2^{47}× remainder →R.
IDV	13300	Integer divide. (U)÷(S)→U, remainder →R.
VDI	17300	Reverse integer divide. (S)÷(U)→U, remainder →R.

Floating Point

	octal code	
FAD	10400	Floating add. $(U)+(S)\rightarrow U$.
FSB	10500	Floating subtract. $(U)-(S)\rightarrow U$.
BSF	14500	Reverse floating subtract. $(S)-(U)\rightarrow U$.
FMP	10600	Floating multiply. $(U)\times(S)\rightarrow U,R$ (double length).
FDV	10700	Floating divide. Double length $(U,R)\div(S)\rightarrow U$, $2^{47}\times$ remainder $\rightarrow R$.
VDF	16700	Reverse floating divide. $(S)\div(U)\rightarrow U$, $2^{47}\times$ remainder $\rightarrow R$.

Class 2, Fetch, Store, Tags

In the list below, the symbols are followed by their octal equivalents and a brief explanation of their meanings; the indication "a,#" means that the operation symbol automatically causes IM bit 1 to be set to 1 (to load S with M instead of (M)), since the operation indicated deals with M rather than with (S).

Any Group I or Group II mnemonic may be followed by a comma and any Group III mnemonic. In addition, any Group I or Group III mnemonic may be followed by → or +1, storing (U) with (ATR) at location M; or by +2, storing (U) with (ATR) at location (B6); or any Group I, II, or III mnemonic may be followed by +3, storing (U) with (ATR) at location M+(B6). Note that all Group I and Group II mnemonics clear (ATR) unless followed by a Group III mnemonic.

The Class 2 mnemonics are as follows:

Group I

	octal code	
CLA	21700	Clear and add. Bring (S) to U.
BEU	21000*	Bring exponent to U. Exponent portion of (S) replaces exponent portion of (U).
BMU	20700*	Bring mantissa to U. Mantissa portion of (S) replaces mantissa portion of (U).
BLU	21400*	Bring left half to U. Left half of (S) replaces left half of (U).
BRU	20300*	Bring right half to U. Right half of (S) replaces right half of (U).
BIU	20200*	Bring inflections to U. Inflection portion of (S) replaces inflection portion of (U).
BAU	20100*	Bring address to U. Address portion of (S) replaces address portion of (U).
BNA	21600*	Bring all except address to U. Inflection and left portions of (S) replace inflection and left portions of (U).

*The "bring" mnemonics may be joined by commas to fetch more than one portion of a word.

Group II

		octal code	
	RPE	20701*	Replace exponent. Exponent portion of (U) replaces exponent portion of word at location M.
	RPM	21001*	Replace mantissa. Mantissa portion of (U) replaces mantissa portion of word at location M.
	RPL	20301*	Replace left half. Left half of (U) replaces left half of word at location M.
	RPR	21401*	Replace right half. Right half of (U) replaces right half of word at location M.
	RPA	21601*	Replace address. Address portion of (U) replaces address portion of word at location M.
	RPI	21501*	Replace inflections. Inflection portion of (U) replaces inflection portion of word at location M.
a,#	STO	20001	Store. Store (U) at location M.

*The "replace" mnemonics may not be combined with each other.

Group III

	octal code	
ST1	20010	Set Tag 1. Set ATR to 1.
ST2	20020	Set Tag 2. Set ATR to 2.
ST3	20030	Set Tag 3. Set ATR to 3.
WTG	20040	With Tag. Do not change ATR.

Group IV

	octal code	
NOP	30000	No operation. Do not alter (U) or (ATR).
FST	20041	Fetch and store. Bring contents of location M to S; then store (U) with (ATR) at location M.
RWT	21641	Replace address, with tag. Address portion of (U) replaces address portion of word at location M, without changing the tag on the word at location M.

Double Option

Any Class 2 operation applied to U with original F4 address N may also be applied to R with origianl F4 address N+1 by use of the mnemonic:

	octal code	
DBL	20004	Double. After operating on U with original F4 address N, apply same operation to R with original F4 address N+1.

Examples:

```
BAU,DBL      DATA
```

loads the address portion of U from the location DATA and loads the address portion of R from the location DATA +1.

```
STO,DBL      *ANS
```

stores (U) through the codeword at location ANS and stores (R) through the codeword at location ANS +1.

Use of the +2 store option with DBL stores (U) with (ATR) at location (B6), stores (R) with (ATR) at location (B6+1), and increments (B6) by 1. The +3 store option with DBL uses (B6) for both stores and does not increment (B6).

After a double operation, the M portion of (I) contains the final address used with R.

• Class 4, B-Registers, Lights, Special Registers, Shifts

In the list below, the symbols are followed by their octal equivalents and a brief explanation of their meanings; the indication "a,#" means that the operation symbol automatically causes IM bit 1 to be set to 1 (to load S with M instead of (M)), since the operation indicated deals with M rather than with (S).

The Class 4 mnemonics are as follows:

B-registers

		octal code	
a,#	TSR	40000	Transfer to subroutine. Set PF to (CC); then set CC to M.
a,#	SBi	4000i	Set Bi. Set Bi to M, for i=1, 2, ..., 6.
a,#	SPF	40007	Set PF. Set PF to M.
a,#	ACC	41000	Add to CC. (CC)+M→CC.
a,#	ABi	4100i	Add to Bi. (Bi)+M→Bi, for i=1, 2, ..., 6.
a,#	APF	41007	Add to PF. (PF)+M→PF.
	ERM	00020	Enter repeat mode. Turn on mode light #2.

The ERM mnemonic is meaningful only when joined by a comma to one of the above Class 4 mnemonics.

Lights

		octal code	
a,#	SLN	42000	Sense lights on. Turn on sense lights corresponding to 1's in M.
a,#	ILN	42001	Indicator lights on. Turn on indicator lights corresponding to 1's in M.
a,#	MLN	42002	Mode lights on. Turn on mode lights corresponding to 1's in M.
a,#	TLN	42003	Trap lights on. Turn on trapping lights corresponding to 1's in M.
a,#	SLF	42004	Sense lights off. Turn off sense lights corresponding to 1's in M.
a,#	ILF	42005	Indicator lights off. Turn off indicator lights corresponding to 1's in M.

		octal code	
a,#	MLF	42006	Mode lights off. Turn off mode lights corresponding to 1's in M.
a,#	TLF	42007	Trap lights off. Turn off trapping lights corresponding to 1's in M.

Note that lights corresponding to 0's in M are not affected by the above orders.

<u>Special registers</u>

		octal code	
a,#	STX	43005	Set X. Set the increment register to M.
a,#	STT	43006	Set TT. Set the to-tape register to M.
a,#	SFT	43007	Set FT. Set the from-tape register to M.

<u>Shifts</u>

		octal code	
a,#	DMR	44000	Double mantissa right. Arithmetic right shift of (U,R) mantissa M places.
a,#	DML	44010	Double mantissa left. Arithmetic left shift of (U,R) mantissa M places.
a,#	LUR	45010	Logical U right. Shift (U) right M places, shifting zeros into left end of U.
a,#	LUL	45020	Logical U left. Shift (U) left M places, shifting zeros into right end of U.
a,#	LRR	45001	Logical R right. Shift (R) right M places, shifting zeros into left end of R.
a,#	LRL	45002	Logical R left. Shift (R) left M places, shifting zeros into right end of R.
a,#	LRS	45015	Long right shift. Shift (U,R) right M places, shifting (U) into R and zeros into left end of U.
a,#	LLS	45062	Long left shift. Shift (U,R) left M places, shifting (R) into U and zeros into right end of R.

		octal code	
a,#	CRR	45055	Circle right. Shift (U,R) right M places, shifting (U) into R and right end of (R) into left end of U.
a,#	CRL	45066	Circle left. Shift (U,R) left M places, shifting (R) into U and left end of (U) into right end of R.
a,#	BCT	46000	Bit count. Clear U; shift R right M places; add each 1 which spills from R one at a time into U.

<u>T-flags</u>

	TFU	47000	T-flags and ITR to U. Clear U, then bring two ITR and four T-flag bits to U: ITR in octal (0,1,2, or 3) → bits 49 and 50, TF4→bit 51, TF5→bit 52, TF6→bit 53, TF7→bit 54.

• Class 5, Logic and Fast Registers

In the list below, the symbols are followed by their octal equivalents and a brief explanation of their meanings.

Any Class 5 mnemonic may be followed by → or +1, to cause storing of the final (U) at location M; by +2, storing (U) at location (B6); or by +3, storing (U) at location M+(B6). In addition, any Class 5 mnemonic may be preceded by a - sign, causing the final result in U to be complemented (before storing). The Class 5 mnemonics are as follows:

	octal code	
CPL	50100	Complement. Change all 1's in U to 0's and all 0's to 1's.
XUR	54000	Exchange (U) and (R). (U)→R as (R)→U.
LDU	50410	Load U. (S)→U.
LDR	50400	Load R. (S)→R without disturbing (U).
LTi	504i0	Load Ti. (S)→Ti without disturbing (U) or (R), for i=4, 5, 6, 7.
STF	50540	Set T-flag. Turn on flag bit for the T-register being loaded to cause indirect addressing in F1 and F3. Meaningful only if adjoined to LTi by comma.
SUR	53000	Shuffle S, U, and R. (U)→R then (S)→U.
ORU	50010	Or to U. Logical or for each bit position: (U)=0 and (S)=0 results in (U)=0; otherwise, (U)=1 as result.
AND	50314	And. Logical and for each bit position: (U)=1 and (S)=1 results in (U)=1; otherwise, (U)=0 as result.
XTR	50020	Extract. For each bit position: (S)→U if (R)=1, (U) unchanged if (R)=0.
SYD	53220	Symmetric difference. For each bit position: (U)=(S) results in (U)=0; (U)≠(S) results in (U)=1.
SYS	53120	Symmetric sum. For each bit position: (U)=(S) results in (U)=1; (U)≠(S) results in (U)=0.

• <u>Class 6, Input-Output</u>

In the list below, the symbols are followed by their octal equivalents and a brief explanation of their meanings; the indication "a,#" means that the operation symbol automatically causes IM bit 1 to be set to 1 (to load S with M instead of (M)), since the operation indicated deals with M rather than with (S).

For detailed explanations of reading, printing, punching, plotting, and magnetic tape operation, see the Rice Computer Manual.

The Class 6 mnemonics are as follows:

Paper tape

		octal code	
a,#	RTR	60000*	Read triads. Read 1 to 18 triads from paper tape into U.
a,#	RHX	60100*	Read hexads. Read 1 to 9 hexads from paper tape into U.
	PHX	60400	Punch hexads. Punch 1 to 9 hexads from (S) onto paper tape.
	PH7	60500	Punch hexads with 7th hole. Punch 1 to 9 hexads, each with a 7th hole, from (S) onto paper tape.
	PTR	60600	Punch triads. Punch 1 to 18 triads from (S) onto paper tape.

*Either "Read" mnemonic may be followed by → or +1, storing (U) at location M; by +2, storing (U) at location (B6); by +3, storing (U) at location M+(B6); by +40 to turn on IL4 (mantissa overflow) if there is no tape in the reader.

Console typewriter

	octal code	
TYP	60700	Type. Type (S) as 18 octal digits on console typewriter.

Printer

		octal code	
a,#	PRN	61110	Print numeric. Print, using first 32 characters of print wheel, from print matrix beginning at location M; space one line after printing.

		octal code	
a,#	PRA	61210	Print alphanumeric. Print as above, using all characters.
a,#	PRO	61310	Print octal. Print as above, using characters 0-7 only.
	SPA	61010	Space. Advance printer paper one line.
	SP2	61020	Space, format 2. Advance printer paper to next 1/22 page mark.
	SP3	61030	Space, format 3. Advance printer paper to next 1/11 page mark.
	SP4	61040	Space, format 4. Advance printer paper to next 1/6 page mark.
	SP5	61050	Space, format 5. Advance printer paper to next 1/3 page mark.
	SP6	61060	Space, format 6. Advance printer paper to next 1/2 page mark.
	PAG	61070	Page restore. Advance printer paper to next new page.
	DLY	61000	Printer delay. n successive executions of DLY will delay the machine for at least n-1 tenths of a second and not more than n tenths of a second.

Magnetic tape

		octal code	
a,#	WDi	64i00	Write data on MT unit i; i=Z(for 0), 1, 2, 3.
	WMi	64i20	Write marker from last 8 bits of (S) on MT unit i; i=Z(for 0), 1, 2, 3.
a,#	RDi	65i00	Read data from MT unit i; i=Z(for 0), 1, 2, 3.
	SMi	66i00*	Search for marker in last 8 bits of (S) on MT unit i; i=Z(for 0), 1, 2, 3.
	RWi	66i01	Rewind tape on MT unit i; i=Z(for 0), 1, 2, 3.
	BCK	60040	Backward. Perform operation in backward direction.
	NST	65004	No store. Do not store to memory. This is meaningful only for read MT orders.

*Search is overlapped with computer operation, but next order to searching transport will hang until search is complete.

Oscilloscope and strip chart plot

	octal code	
PLT	67000	Plot on oscilloscope or strip chart.
ADV	67700	Advance movie film.

• Class 7, Analog Input, Shifts, Delays

Any Class 7 mnemonic may be followed by → or +1, to cause storing of the final (U) at location M; by +2, storing (U) at location (B6); or by +3, storing (U) at M+(B6). This class deals with various instructions used in conjunction with operation of the analog-to-digital converter.

The Class 7 mnemonics are as follows:

	octal code	
WAT	71100	Wait. Machine will wait until the next pulse from a crystal-controlled 1 kc. pulse generator before exiting Field 2.
LS1	72010	Special fast arithmetic shifts of double-length (U,R), left if S exponent positive, right if S exponent negative. Shifts are 8 bits at a time. LSi indicates i shifts of 8 bits. These shifts are principally used in unpacking converted data. The mnemonics may by combined to get different length shifts: LS4,LS1 would give 5 shifts of 8 bits (total: 40 bits). These shifts do not pass through the exponents of U or R nor through the sign of R, but do shift into the sign of U.
LS2	72020	
LS4	72040	
MCN	72110	Manual conversion. An A-to-D conversion of the channel specified by (S) will be performed.
ACN	72364	Automatic conversion. Six conversions from channels 1 through 6 will be performed.

Conversion results will be packed into U as follows: The 8 bits (sign plus 7 bits) resulting from each conversion will be packed into the mantissa with the bits resulting from the first conversion farthest to the left and the bits resulting from last conversion in the right-most 8 bits of U. The U exponent will be set to 77. The R mantissa is used.

There are sixteen channels into the converter. The channel to be converted is specified by the right-most 16 bits of S. Channel 1 corresponds to S_{m47}, Channel 2 to S_{m46}, etc.

In addition to the formal store options, operations may be performed with the 72xxx orders as follows:

72xxx + 400	(S) will be sent to U before performing any other operation.
72xxx + 200	(S) will be cleared and a 1 sent to S_{m47}.
72xxx + 4	(S) will be logically shifted 1 to the left each time (U,R) is shifted 8 to the left. Notice that this feature can be used to sample consecutively numbered channels automatically.

• Summary of Operation Codes

The accompanying chart summarizes the Field 2 mnemonics available in AP1 and AP2. If an operation code is followed by the symbol "@", the corresponding mnemonic causes IM bit 1 to be set to 1.

The symbol "→" following an operation mnemonic of class 1, 2, 5, 6, 7 causes a final store of U to M.

The symbol "-" preceding a class 5 operation mnemonic causes a final logical complement of U.

For more than one operation mnemonic in an instruction, the octal codes will be combined by a logical OR. In most cases, mnemonics are separated by commas. In class 0, the tests are separated by "+" for "ANY", by "x" for "ALL". The mnemonics "POS" and "NEG" are compound "ANY" tests and the mnemonics "PNZ" and "NNZ" are compound "ALL" tests.

SUMMARY OF OPERATION CODES

CLASS 0

- HTR 00000@
- TRA 01000@
- SKP 02000
- JMP 03000

- PSN 00100
- MOV 00200
- EOV 00300
- NSN 00500
- NMO 00600
- NEO 00700

- IF(ANY)HTR 00000@
- IF(ANY)TRA 01000@
- IF(ANY)SKP 02000
- IF(ANY)JMP 03000

- ZER 00010
- EVN 00020
- SLN 00030@
- NUL 00040
- NZE 00050
- ODD 00060
- SLF 00070@

- IF(ALL)HTR 04000@
- IF(ALL)TRA 05000@
- IF(ALL)SKP 06000
- IF(ALL)JMP 07000

- TGi 0000i
- NTG 00004
- NTi 00004+i
  i=1,2,3

- POS 00110
- PNZ 00150
- NEG 00510
- NNZ 00550

CLASS 1

- ADD 10000
- SUB 10100
- MPY 10200
- DIV 10300
- BUS 14100
- IMP 10220
- IDV 13300
- VID 16300
- VDI 17300

- FAD 10400
- FSB 10500
- FMP 10600
- FDV 10700
- BSF 14500
- VDF 16700

CLASS 2

- STO 20001@
- FST 20041

- BEU 21000
- BLU 21400
- BAU 20100
- BRU 20300
- BMU 20700
- BIU 20200
- CLA 21700

- DBL 20004

- RPL 20301
- RPE 20701
- RPM 21001
- RPR 21401
- RPA 21601
- RPI 21501
- RWT 21641

- STi 200i0
  i=1,2,3
- WTG 20040
- NOP 30000

CLASS 4

- TSR 40000@
- SBi 4000i@
- SPF 40007@
- ACC 41000@
- ABi 4100i@
- APF 41007@
- ERM 40020
  i=1,...6

- BCT 46000@

- TFU 47000

- SLN 42000@
- ILN 42001@
- MLN 42002@
- TLN 42003@
- SLF 42004@
- ILF 42005@
- MLF 42006@
- TLF 42007@

- STX 43005@
- STT 43006@
- SFT 43007@

- DMR 44000@
- DML 44010@

- LUR 45010@
- LUL 45020@
- LRR 45001@
- LRL 45002@
- LRS 45015@
- LLS 45062@
- CRR 45055@
- CRL 45066@

CLASS 5

- LDR 50400
- LDU 50410
- LTi 504i0
  i=4,5,6,7

- STF 50540
- SUR 53000
- XUR 54000

- CPL 50100
- ORU 50010
- AND 50314
- SYD 53220
- SYS 53120
- XTR 50020

CLASS 6

- RTR 60000@
- RHX 60100@
- PHX 60400
- PH7 60500
- PTR 60600

- TYP 60700

- PRN 61110@
- PRA 61210@
- PRO 61310@
- SPA 61010
- SPi 610i0
  i=2,...,6
- PAG 61070

- PLT 67000
- ADV 67700

- WDi 64i00
- WMi 64i20
- RDi 65i00
- NST 65004
- SMi 66i00
- RWi 66i01
- BCK 60040
  i=Z,1,2,3

CLASS 7

- WAT 71100
- ACN 72364
- MCN 72110
- LSi 720i0
  i=1,2,4

The tables on this page summarize the options available in SETU (Field 1), AUX (Field 3), and ADDR+MOD (Field 4). In the tables

A indicates the full length special registers Z, U, R, S, T4, T5, T6, T7 specified in the second triad by 0, 1, 2, 3, 4, 5, 6, 7.

B and Bi indicate the short index registers CC, B1, B2, B3, B4, B5, B6, PF specified in the second triad by 0, 1, 2, 3, 4, 5, 6, 7.

I and M indicate the number formed in the address field of the instruction. (M) indicates the contents of the memory location numbered M.

Exceptions are R→Z, 10 in field 3 and I or |Z|, 20 and -I or -|Z|, 30 in field 1. R→Z has the result that R is cleared to Z. I or |Z| has the result that an integer 1 goes to U. -I or -|Z| has the result that an integer -1 goes to U.

1st Triad		Field 1 (SETU)	
A	0	B	4
-A	1	-B	5
|A|	2	|B|	6
-|A|	3	-|B|	7

1st Triad		Field 3 (AUX)	
U→A	0	U→Bi	4
R→A	1	R→Bi	5
Bi+1	2	Bi-1	6
Bi+X	3	I→Bi	7

1st Triad		Field 4 (ADDR+MOD)	
(M)	0	M	4
-(M)	1	-M	5
|(M)|	2	|M|	6
-|(M)|	3	-|M|	7

Pseudo-orders govern the process of AP1 assembly and facilitate the handling of blocks of various types of data within AP1 programs. Pseudo-orders do not exist in AP2.

- ORG and END

All programs to be assembled by AP1 must be started by an ORG (origin) order and terminated by an END order.

The function of ORG is to initialize the assembly process, to identify the program which follows, and to determine whether it is to be assembled in relative or absolute final form. The ORG order is preceded by a "cr" and an "uc" or "lc" punch (upper or lower case).

A relativized program will run anywhere in memory. If an order in location P refers in Field 4 to location Q, it is through a Control Counter reference of the form CC+(Q-P)-1. A relativized program that will load under SPIREL control is generated if the LOCN field of the ORG pseudo-order is not blank; the ADDR field must be blank or zero in this case. To assemble a program to load with codeword at address N (octal) the ORG pseudo-order has the form

```
N                           ORG
| cr          | 1st tab     | 2nd tab
```

To assemble a program to load symbolically with name S (5 or fewer characters) the ORG pseudo-order has the form

```
S                           ORG
| cr          | 1st tab     | 2nd tab
```

To assemble a program to load as the Ath element of the Bth element ... of array K the ORG pseudo-order has the form

```
K,...,B,A                   ORG
| cr          | 1st tab     | 2nd tab
```

Here A,B,... are octal numbers; K is the codeword address or name

(as above) of the array to which the program belongs. As many as five levels may be specified. All control words are provided for the loading of the program as the designated array element.

A relativized program is also produced if the ORG pseudo-order has zero ADDR field and blank LOCN field. This form is only appropriate if the self-loading option is to be used during assembly. The self-loading tape produced will load with the LOAD switch beginning at the address in B6.

An absolute program will run only at the specified memory location. Field 4 reference to location Q is made directly. An absolute program is generated if the ADDR field is not blank or zero; the LOCN field must be blank or zero. To assemble a program to load at address M (octal) the ORG pseudo-order has the form

cr	1st tab	2nd tab	3rd tab
	ORG	M	

The program will load with the LOAD switch if the self-loading option is used during assembly; otherwise it will load under SPIREL control.

The END order has the form

cr	1st tab	2nd tab
	END	cr cr

where "END" must be immediately followed by two (or more) carriage returns.

Neither ORG nor END cause any words to be generated in a program.

- EQU

The EQU (equivalence) order gives a numeric equivalent for a symbol or equates one symbol to another. The order has the form

L		EQU	M
cr	1st tab	2nd tab	3rd tab

where L (in LOCN) is the symbol defined by the pseudo-order, SETU is blank, and M (in ADDR) is either absolute or a symbol whose value has previously been defined through its appearance in the LOCN field of another order. L is assigned the value M. If M is a 5-digit octal code, the symbol L may appear in the OPN field of any order following the EQU order; L will be treated as an operation code and will be replaced during assembly by the octal code for which it stands. No words are added to the program being assembled due to an EQU.

• <u>BSS and BES</u>

Either of these orders inserts a block of zero words into the body of the program. BSS (block started by symbol) and BES (block ended by symbol) have the form

```
L                          XXX        M
| cr        | 1st tab  | 2nd tab  | 3rd tab
```

where L (in LOCN) is blank or symbolic, SETU is blank, and M (in ADDR) is absolute. M is the number of zero words to be inserted. If L is symbolic, it is assigned as if the LOCN field had been associated with the first (BSS) or last (BES) word in the block.

- BCD, FLX, REM

These orders deal with alphanumeric data and have the form

```
 L                         XXX       M
| cr        | 1st tab   | 2nd tab | 3rd tab
```

where SETU is always blank. The operation mnemonic must be followed by a "tab" character, and after that all characters (in the ADDR field M) are retained, 9 characters per word. Any occurrence of the "cr tab tab tab" sequence to continue the character string is replaced by a "space". For BCD (binary coded decimal), each character is converted to a corresponding printer hexad and the words are stored into the program being assembled; if L (in LOCN) is symbolic, it is assigned as if associated with the first word stored. For FLX (flexowriter), all codes (including case shifts, etc.) are preserved without conversion and the words are stored into the program being assembled; L (in LOCN) may be symbolic as for BCD. For REM (remarks), L (in LOCN) must be blank; this order is used only to obtain printed comments in the program listing, and no words are stored into the program being assembled.

- <u>DEC, OCT, and HDC</u>

The DEC (decimal), OCT (octal), and HDC (hexadecimal, i.e. base 16) orders are used for inserting numeric data into the body of the program. They have the form

L		XXX	M
cr	1st tab	2nd tab	3rd tab

where L (in LOCN) is blank or symbolic, SETU is blank, and M (in ADDR) consists of a list of one or more octal or decimal numbers. If L is symbolic, it is assigned as if associated with the first number in the list. Each number must be separated from its successor by a comma, and each will be stored into a separate word in the program being assembled. Continuation lines should not be used; for long lists of numbers, several DEC or OCT pseudo-orders in succession may be used to produce a continuous block of data. An octal number consists of one to 18 octal digits. A decimal integer consists of one to 14 decimal digits; a floating point decimal number, of one to 14 significant figures and a decimal point. A hexadecimal number consists of one to 13 hexadecimal digits (0, 1,...,9, a, b, c, d, e, f). It may be 14 hexadecimal digits if its value is less than or equal to 3ffffffffffff.

• <u>REF</u>

The REF (reference) order defines a single cross-reference word in the program being assembled. All REFs for a program must appear immediately after the ORG order, before any code for the program. The form of a REF order is

NAME		REF	CONTENT
cr	1st tab	2nd tab	3rd tab

or

NAME		REF	*CONTENT
cr	1st tab	2nd tab	3rd tab

Each REF must contain a location symbol, the name used to address it in the code for the program. The ADDR field of the REF specifies the content of the cross-reference word: a string of characters containing <u>only</u> upper case letters and numbers which will be converted to printer hexads, filled to 5 with '25' hexads or truncated to 5 as appropriate. If the cross-reference word is to contain an indirect addressing bit (for a vector, matrix or program), this is denoted by '*' before the hexad string, with no intervening spaces or punches. If k REFs appear in a program, the first will be at location -(k-1) of the final program, ..., the kth at location 77777 (-0). The punched output of the final program will be followed by a control word to set the initial index of the program to -(k-1). When the program is loaded, execution of the control word to set initial index to -(k-1) will cause SPIREL to operate on each of the k cross-reference words as follows:

1) make an entry in the Symbol Table (ST) of the 5 hexads in the cross-reference word;
2) insert the corresponding Value Table (VT) address in the address field of the cross-reference word.

Indirect reference in the assembled program through the REF then causes addressing of the item with name in ST, the value in VT for a scalar or the codeword in VT for a vector, matrix, or program.

For a double operand, such as a complex scalar or non-scalar, two cross-references must be used and these must appear in the order of the parts of the operand. The name of the operand is associated with the first part, and the second part is named "ditto", which is printed '←←←←←' but typed '#####'. If A is a complex scalar its cross-references might appear as

cr	1st tab	2nd tab	3rd tab
AREAL		REF	A
AIMAG		REF	←←←←←

where '←←←←←' is typed '#####'. It may be that one of the cross-references is never referred to in the code; this is the only case where an unlabelled REF may be used, but two REFs must be given.

• Application

Macro-orders are available in the AP1 assembly language. This facility allows the coder to define parameterized sequences of code and have these substituted in his program during assembly. Since a code pattern may thus be written only once for more than one occurrence in the program, a number of advantages are offered:

-- Symbolic code for the program is shorter;
-- code for the program is less prone to error because fewer instructions are prepared;
-- the program is more easily changed because a single change in a macro definition will take effect in all occurrences at assembly;
-- the program is more readable because single macro names appear in the code for operations which actually require sequences of machine instructions.

A macro-order is a general name which has been defined by the programmer to represent one or more valid AP1 instructions. Then, at each subsequent call of the macro-order, these instructions are inserted into the assembled program. Any order included in the macro-definition may contain a parameter in one or more fields; such a field may be changed each time a macro-order is called by specifying a different value for the parameter at each call.

Example. Suppose in an AP1 program there existed the following code:

```
CLA        ALPHA
FAD        B6+1,U→T4
STO        GAMMA
 :
CLA        B6
FAD        BETA,B6+1
STO        B6
 :
CLA        ALPHA
FAD        BETA,U→R
STO        GAMMA
```

The programmer could have saved himself the effort of writing the repetitious sequences of instructions by defining a macro-order called SUM with four parameters as follows:

```
SUM     MACRO     ADONE+ADTWO→TOTAL,AUX
        CLA       ADONE
        FAD       ADTWO,AUX
        STO       TOTAL
        MEND
```

Then, having defined the macro-order SUM, the programmer could call it in his AP1 code, using different parameter values at each call:

```
        SUM       ALPHA,B6+1, GAMMA,U→T4
         .
         .
         .
        SUM       B6,BETA,B6,B6+1
         .
         .
         .
        SUM       ALPHA,BETA,GAMMA,U→R
```

The instructions assembled would be identical with those originally written by the programmer, but the repetitious code would not appear in the program.

• <u>Definition</u>

A macro-definition specifies a set of instructions, gives the set a name, and determines which fields (if any) are to contain parameters. The macro-definition consists of three parts: (1) the MACRO pseudo-order, in which the LOCN field gives the name of the macro-order and the ADDR field gives the list of parameters; (2) the set of instructions to be represented by the macro-name; (3) the MEND pseudo-order, ending the macro-definition.

(1) The MACRO pseudo-order may or may not include a list of parameters and must be one of the following forms:

```
NAME                    MACRO        PARA,PARB,...,PARZ
| cr         | 1st tab  | 2nd tab    | 3rd tab

NAME                    MACRO
| cr         | 1st tab  | 2nd tab
```

The <u>name</u> of the macro-order may be any valid AP1 general name. This is its only appearance in the LOCN field; it is written in the OPN field at each <u>call</u> of the macro. If the macro-order has parameters, they are listed in the ADDR field of the MACRO pseudo-order. A parameter name is any valid AP1 general name, and is separated from the next parameter name by one of the following special characters:

, = → + - x / ( )

The last parameter is followed by a carriage return; if more than one line is required, the 'cr tab tab tab' sequence follows (but does not replace) the separating character at the end of the first line. Note that if parentheses are used, they must be used in pairs. In this way meaningful notation may be employed in the list of parameter names; for example,

```
COMP           MACRO          RATE,TIME,DIST,TOTAL
```

could also be written

```
COMP        MACRO     RATE(TIME)→DIST,TOTAL
```

or

```
COMP        MACRO     RATE×TIME=DIST→TOTAL
```

(2) Any reasonable number of instructions may be represented by the macro-name; generally, a lengthy set of instructions will best be coded in closed subroutine form rather than in the open form generated by a macro-order. Any valid AP1 instructions except pseudo-orders may be included. Symbols which have appeared in the ADDR field of the MACRO pseudo-order are parameters and are subject to the special rules described below; all other symbols are treated in accordance with the usual AP1 conventions. Orders within a macro-definition may conform to the rules for instruction content, or they may include parameter names which are then subject to the rules below.

LOCN: Symbolic LOCN fields which are not parameters may be used within a macro-definition, but such symbols are not meaningful outside the set of instructions comprising the macro-definition; they may be referenced only by other orders within the set. A symbolic LOCN field which is a parameter name must be given a different value at each call of the macro-order; these values may then be addressed by orders outside the macro-definition. Note, however, that orders within the macro-definition may reference LOCN symbols which appear elsewhere in the program, including those defined by pseudo-orders.

SETU: A single parameter name may appear in SETU, with or without the minus and absolute value signs normally permitted in this field. All values taken by this parameter at subsequent calls of the macro must then be valid SETU symbols or octal equivalents. Note that if a - or | | sign is included, it is effective regardless of whether another - or | | sign is used with a SETU

symbol as a parameter value at a subsequent call; such inflection signs are combined by a logical 'or'. If, at a given call, a SETU parameter value is omitted, it is replaced by the octal code '01' (do not change U).

OPN: Multiple parameter names are permitted in OPN to allow flexible coding of Class 0 tests, Class 2 tag orders, etc. These parameter names may be combined with the special symbols such as →, +, ×, etc., normally permitted in this field. In the case of multiple parameters, values need not be specified for all parameters at every call if the resulting code is valid. Parameter values for OPN may include any valid OPN symbols or octal codes; the special symbols →, +, ×, etc. may also be used as part of parameter values.

ADDR+MOD: This field may consist of a single parameter name, which is to assume a value equivalent to any valid ADDR+MOD form (e.g., *ZETA, B1+B2+1, M+B6); or the field may include several parameters, provided the values they assume at any given call result in valid code (for example, SYMB+BREG+NUMB might become BETA+PF+3 or *ALPHA+B2+1); or one or more parameter names may be combined with other symbols and/or numbers which are to remain the same at each call (such as NAME+B1+1, which might become ABC+B1+1 or XYZ+B1+1). A parameter value may be omitted entirely at a given call if such an omission does not destroy the validity of the remaining code. The special symbols such as *, a, -, and | | may appear either with the parameter name or as part of the parameter value, and are combined by a logical 'or'.

AUX: This field may consist entirely of a single parameter name; if so, the value assumed by this parameter must be a valid AUX octal code or symbolic equivalent (e.g. U→T4, B1-1, etc.). Alternatively, either or both of the fast register symbols (and also I and X) may be represented by parameter names, provided that only valid combinations are used for parameter values (for example, B1-X and I→T4 are not permitted).

<u>TAG</u>: The customary TAG symbols (TG1, TG2, TG3) may appear within a macro-definition, or this field may contain a parameter name for which one of the above symbolic values will be substituted when the macro-order is called.

(3) The MEND pseudo-order which terminates the macro-definition is as follows:

MEND

| cr | 1st tab | 2nd tab

More than one macro-definition may appear within a given program, provided each is bracketed by its own MACRO and MEND pseudo-orders. The same parameter names may be used in separate macro-definitions without causing confusion, but they must not be used as symbols elsewhere in the program. A macro-definition may appear at any point in a program; it generates no code at this point, and transfers around the macro-definition are not needed. The only restriction is that a macro-order must be <u>defined</u> before it is <u>called</u>. One macro-definition may not appear within another, but a previously defined macro-order may be called within the definition of another macro-order.

- Call

After a macro-order has been defined, it may be called by writing the name of the macro-order in the OPN field of an instruction; if the macro-order uses parameters, their values for this particular call are listed in the ADDR field of the same instruction. Parameter values for a macro-order are listed in the same order as the list of parameter names in the MACRO pseudo-order of the corresponding macro-definition. Parameter values are separated by commas; the list is terminated by a cr, and the 'cr tab tab tab' sequence following a comma may be used to continue the list onto a second line. Certain parameters may be omitted at a given call; in this case, two adjacent commas (with or without spaces between them) or a comma followed by a cr indicate an omitted parameter. A macro-order will usually be called at several different points in a program. Any call may have a symbolic LOCN field, but no two calls may have the same symbolic LOCN field. The LOCN symbol is assigned to the first order of the set of instructions represented by the macro-order, unless the LOCN field of this order contains a parameter name for which a value is specified at the current call; in this case, the parameter value takes precedence. Note that several orders may replace a single macro-order; hence relative addressing around a call must be used with care.

At each call, the sequence of parameter values must correspond to the sequence of parameter names which appeared in the macro-definition, but the values assumed by the parameters will usually differ from one call to another. A parameter value may consist of any string of characters which, when substituted into the macro-definition at each occurence of the corresponding parameter name, will produce valid AP1 code for the field in which it occurs. If the call lies within another macro-definition, a parameter name from the outer macro-definition may be used as a parameter value for the inner macro-call.

- Examples

Suppose an AP1 program contains the following code:

```
B1        SB1       B2,U→B2
          LT4       *MATR1
B1        SB1       B2,U→B2
           :
B3        SB3       B4,U→B4
          LT5       *MATR2
B3        SB3       B4,U→B4
```

This could be written by defining a macro-order such as

```
BREGS               MACRO     BA,BB,SBA,LTJ,MATRI
          BA        SBA       BB,U→BB
                    LTJ       *MATRI
          BA        SBA       BB,U→BB
                    MEND
```

and calling it as follows:

```
                    BREGS     B1,B2,SB1,LT4,MATR1
                      :
                    BREGS     B3,B4,SB3,LT5,MATR2
```

Another example of a macro-definition might be:

```
STORE               MACRO     TREG,OPN,TAG,SYMB,BMOD
          |TREG|    OPN,TAG   SYMB+BMOD,I→BMOD
          BMOD      RPA,WTG   SYMB-1
                    MEND
```

where the call

```
                    STORE     T4,STO,ST2,ALPHA,B3
```

would produce

```
          |T4|      STO,ST2   ALPHA+B3,I→B3
          B3        RPA,WTG   ALPHA-1
```

and the call

```
                    STORE       -T6,FST,B6,B1
```

would produce

```
        -|T6|       FST         B6+B1,I→B1
        B1          RPA,WTG     B6-1
```

All of the preceding examples are crowded with parameters in order to demonstrate the versatility and flexibility of macro-orders. In actual practice, many instances will be found where only one or two symbols vary at each repetition of otherwise identical blocks of code. Here the saving in programming time and in reducing the likelihood of introducing errors when copying lengthy sections of code will prove substantial. For example, the following block of code might occur repeatedly in a control program linking various subroutines:

```
LITES       MACRO       SUBR
            CLA         SL
            RWT         RESET
            SLF         77777
            TSR         *SUBR
            SLF         77777
RESET       SLN         (Z)
            MEND
```

Once defined, the macro-order "LITES" could be called at each point in the program where a transfer to a subroutine occurs. By specifying the particular subroutine as a parameter value of the macro-order, one order could be written in place of six each time.

A macro-order using no parameters at all would be useful, for example, in reversing the indexing of a matrix:

```
TRANS          MACRO
        B1     SB1         B2,U→B2
               LT4         *MATR
        B1     SB1         B2,U→B2
               MEND
```

At each call, the macro-order "TRANS" would cause T4 to be loaded with the desired element of the transposed matrix MATR.

As noted above, one previously defined macro-order may be called within the definition of another, producing a set of "nested" macro-orders. In the following example, such a set of nested macro-orders is used to multiply two matrices and store their product as a third matrix.

The outermost macro-order MULT has as parameters the codeword addresses and dimensions of the matrices involved; MATA has NROW rows and L columns, MATB has L rows and MCOL columns, and the product matrix MATC has NROW rows and MCOL columns. Within the initialization and storage operations performed by MULT, a second macro-order PROD is called; its definition uses two of the same parameters used by MULT and it performs the actual arithmetic and indexing operations required for the matrix multiplication. Both these macro-definitions are assumed to be embedded in a larger program in which numerous matrices of varying dimensions must be multiplied together.

```
                   230                    ORG
                                           :
                                  AP1 instructions
                                           :
 ┌  PROD                                  MACRO          MATA,MATB
 │  LOOP              B2                  SB2            B3,U→B3
 │                                        LT4            *MATA
 │                    B2                  SB2            B3,U→B3
 definition           B1                  SB1            B3,U→B3
    of                T4                  FMP            *MATB,B1-1
 inner macro                              FAD            T5,U→T5
 │                    B1                  SB1            B3,U→B3
 │                    B3                  IF(NZE)TRA     LOOP
 └                                        MEND
 ┌  MULT                                  MACRO          MATA,MATB,MATC,NROW,MCOL,L
 │                                        SB1            NROW
 │  OUTER                                 SB2            MCOL
 │  INNER             Z                   SB3            L,U→T5
 definition        ┌→                     PROD           MATA,MATB
    of             │  T5                  STO            *MATC,B2-1
 outer macro       │  B2                  IF(NZE)TRA     INNER
 │   call of  ─────┘                      SB1            B1-1
 │   inner macro      B1                  IF(NZE)TRA     OUTER
 └                                        MEND
                                           :
                                          MULT           A,B,C,5,3,7
                                           :
                                  AP1 instructions
                                           :
                                          MULT           M1,M2,M3,*P,*Q,*V
                                           :
                                          MULT           G,H,J,2,2,*N
                                           :
                                  AP1 instructions
                                           :
                                          END
```

An AP1 program is assembled by exercising option #6 in the PLACER system.

Assembly output on the printer consists of error messages, program listing, and symbol table. These are discussed below. Assembly also provides a punched paper tape which contains the assembled program to be loaded under SPIREL control or with the LOAD switch. Assembly options are also discussed below.

Error indications. An AP1 error indication is produced by apparent errors in syntax or sequencing. The type of error and its location are given by a message:

ERROR IN [F] AT CR NO [N]

where F is the name of the field in error
and N is the placer listing carriage return number of the line containing the error.

If a single instruction is continued onto more than one line, the carriage return number for the last line will pertain to the entire instruction.

Assembled program listing. Four columns are printed, giving:

(a) The symbolic location (if any exists).

(b) The location count, relative position of the word in the program, in octal.

(c) The instruction in octal, broken into fields, with tag.

(d) The symbolic address (if any exists).

Locations not assigned by the coder are assigned by the assembly program beyond the code for the program being assembled. These appear with their names below a row of asterisks in the program listing. A name may be one supplied by the coder, as 'A' in the case

STO A

where 'A' never appears in a LOCN field. A name may also be one supplied by the assembly program for long octal or full length decimal numbers referenced in ADDR, as in the cases

```
                 AND     77777 0000 7777 00000
or               CLA     d3.0
or               ADD     d412697
```

Specifically, the names assigned to numbers by the assembly program are '←0000A', '←0000B',... in order of occurrence in the program being assembled.

Symbol table. The table of symbols is printed out in seven columns giving information relevant to the symbols defined in the program:

(a) The relative position in the table.

(b) The symbol.

(c) A number (usually 0) which determines the type of object for which the symbol stands.

(d) The equivalent assigned to the symbol (5 octal digits), unless the symbol is a macro name or a macro parameter.

(e) A number (usually 1) which indicates reference in the program to the symbol. A number 3 denotes a symbol which appears in a LOCN field but not in an ADDR field, so this may be an unnecessarily defined location in the program. A number 0 appears on macro names and macro parameters and on symbols given a numeric equivalent.

(f) An 18 digit octal number. The first 5 digits indicate the line at which an equivalent was assigned.

(g) A number which indicates how (if at all) the equivalent was assigned:

0: by appearing in the LOCN field of an order.

1: by appearing in the LOCN field of an EQU pseudo-order in which the address was symbolic (see section on pseudo-orders).

2: by appearing in the LOCN field of an EQU pseudo-order in which the address was numeric (see section on pseudo-orders).

Assembly Options. If only option #6 of PLACER is requested, the stop

(I): 06 HTR CC

occurs. In addition to sense lights 14 and 15 which are turned on automatically, other sense lights may be turned on for special forms of output.

SL9 on: Print with double (instead of single) spacing.

SL11 on: Do not punch assembled program.

SL13 on: Punch self-loading tape. The tape produced will load by using the LOAD switch on the console. An absolute program will load to the origin specified. A relativized program will load to the setting of B6. These program forms are discussed under the ORG pseudo-order.

• Storage Exchange

This program STEX handles dynamic storage allocation in SPIREL. If B1 = codeword address of array and B2 = length of array upon execution of STEX, space is taken, and B1 = first word address of block upon exit. A more detailed explanation of the use of this program may be found in the SPIREL literature. The remarks in the program serve to explain the program's operation.

Lines	Comments
2	This program has codeword address 154.
6	+2, store to B6 option on class 5.
13	EQU'ed name in field 4; only the first 6-hexads of any name are retained.
25	Decimal integer constant in ADDR; 'a' bit is generated automatically due to shift order in OPN.
37	Simple store option '→' on class 1 arithmetic order; store is to fast register T6.
46	R is cleared to zero in AUX by R → Z, not Z → R.
60	Increment of CC in AUX causes a skip.
65	-I in field 1 sets U to the integer -1.
100	Only AUX is used here; no operation is performed in OPN.
101	I → B3 means final address to B3 in AUX.
110	More than two B-mods in field 4.
131	Store ATR to memory in OPN, compound mnemonic.
137	+3, store to B6 + M option in OPN.
155	Control counter is incremented by contents of X register in AUX, causing a jump.
174	Long octal constant is used in ADDR and is stored at bottom of program.

Lines	Comments
224	T7 is restored from value stored on the B6-list.
227-230	Labelled long octal constants out of code sequence. The first will be right-adjusted, filled with leading zeroes to 18 octal places.
231	Binary coded decimal psuedo-order generates two words of hexads here.
232-240	Equated symbolic names.

```
                                                                         1
154                ORG                                                   2
                                                                         3
                   REM           STEX FOR SPIRFL                         4
                                                                         5
         T7        LT7+2         B1,B6+1                                 6
         -Z        TRA           @*SAVE,U→R                              7
         7         BAU+2         X,B6+1                                 10
                   LDR           STORAG                                 11
         7         LLS           @15,U→T6                               12
         7         BAU           FIRSTEX,U→T5                           13
         7         BAU           @B1,U→T4                               14
         P1        IF(ZER)TRA    REORG,R→Z                              15
         T7        IF(NUL)TRA    TAKE                                   16
                                                                        17
                   REM           INACTIVATE SPACE ADDRESSED BY B1
                                                                        21
GIVE1              CLA           B1,U→T7                                22
                   IF(NUL)TRA    GIVE5,U→B4                             23
                   CRL           @15,R→B3                               24
         T7        LUR           @24,U→B5                               25
         P5        LUR           3,U→B5                                 26
                   IF(NUL)TRA    GIVE2                                  27
                   LDR           MASK2                                  30
         7         IF(NUL)SKP    T7                                     31
                   AB4           B5,CC+1                                32
                   AB4           B5-1                                   33
GIVE2    7         BAU           @B3+1,U→R                              34
         7         BAU           @B4                                    35
                   IF(NEG)SKP    T5                                     36
         F         ADD→          T6                                     37
         P4        RPA           B1                                     40
         T7        AND           MASK1                                  41
                   IF(NUL)TRA    GIVE3                                  42
         P1        STO           B6,B6+1                                43
         P4        ADD           @B3-1,U→B1                             44
         7         BAU+2         @B3,B6+1                               45
                   TRA           GIVE1,R→Z                              46
GIVE3    7         LDR→          B1,R→B4                                47
         7         LLS           @15,U→PF                               50
         7         BAU           @B4=1,PF+1                             51
                   IF(POS)SKP    T5,R→7                                 52
                   TRA           GIVE4                                  53
         FF        LRS           @15,B4-1                               54
         P         BAU           *STORAG,I→B3                           55
                   STO           B4                                     56
         P3        IF(NUL)TRA    CC+1                                   57
         P4        RPA           B3,CC+1                                60
         P4        RPA           STORAG                                 61
GIVE4    7         BAU           @B1                                    62
                   IF(NZE)SKP    T4                                     63
                   TRA           TAKE                                   64
GIVE5    -I        ADD→          B6-1,P1-1                              65
                   IF(NZE)TRA    GIVE1,R→Z                              66
                   CLA           B6-2,I→B6                              67
                   TRA           GIVE3,U→B1                             70
                                                                        71
                   REM           ACTIVATE BLOCK OF LENGTH B2+1
                                                                        73
```

```
TAKE      P2     IF(ZER)TRA      ATAKE,R+Z                     74
          7      BAU             a32+1,U+T7                    75
          T6     IF(POS)SKP      T7                            76
          7      TRA             ATAKE,U+B1                    77
                 NOP             a7,U+T6                       100
                 LDR             *STORAG,I+B3                  101
          7      LLS             a15                           102
                 IF(POS)SKP      T7,U+P5                       103
                 TRA             REORG                         104
TAKE1     T4     STO             B3,B3+1                       105
TAKE2     P5     LRS             a15,R+B4                      106
          P5     IF(ZER)TRA      TAKE3                         107
          P      STO             B3+B2,I+B4                    110
TAKE3     P4     RPA             STORAG                        111
          T4     IF(ZER)TRA      ATAKE,R+Z                     112
          P3     TRA             ATAKE,U+B1                    113
                                                               114
                 REM             WRITE ACTIVE BLOCKS TO LOW ADDRESSES
                                                               116
REORG     P6     MLN             04000,U+T7                    117
                 STX             Z,I+B6                        120
                 SB3             *FIRSTEX,I+B4                 121
                 TRA             REORG7                        122
REORG1           CLA             *B4,U+B5                      123
                 CRL             a15,R+B1                      124
          P6     IF(NZE)TRA      REORG2                        125
                 SB3             B3+B1+1,I+B4                  126
                 TRA             REORG6                        127
REORG2           CLA             a35+a6                        130
                 RPA,WTG         *B4                           131
                 AND             MASK1                         132
                 IF(NZE)TRA      REORG3,B3+1                   133
                 AB3             B1,B4-1                       134
          P4     RPA             CC+1,B1+1                     135
          I      AB4,ERM         B1+1,U+B5                     136
                 CLA,WTG+3       B3,B1-1                       137
                 TRA             REORG7                        140
REORG3           CLA,WTG+3       B4,B4+1                       141
REORG4           SUR,LDR+3       B4,B4+1                       142
                 IF(NUL)TRA      REORG5,B1-1                   143
                 50114           MASK2,R+B5                    144
                 IF(NUL)TRA      CC+1                          145
          P      LUL             a3,CC+1                       146
          P      LUL             a3,B5-1                       147
                 DMR             a36                           150
                 IF(PSNxZER)TRA  CC+1,B5+1                     151
                 ADD             a35-1,U+B5                    152
          P3     RWT             B5-1                          153
REORG5    P1     IF(NZE)TRA      REORG4,B3+1                   154
REORG6    7      BAU             a34,CC+X                      155
                 IF(NEG)SKP      T4                            156
          P6     STX             Z,U+PF                        157
REORG7    P4     IF(NZE)SKP      a*LASTEX                      160
                 TRA             REORG8                        161
                 LDR             B4                            162
          7      LLS             a15,U+B1                      163
                 IF(NUL)TRA      REORG1                        164
          P3     SUB             a31+B4,I+B4                   165
                 TRA             REORG7,U+B6                   166
```

```
REORG8    T7     MLF              04000,U+B6              167
          T4     IF(NNZ)SKP       T5                      170
          PF     ADD+             T4                      171
                 IF(SLF)SKP       00002                   172
                 TRA              REORG9                  173
                 CLA              0000240100000000000     174
                 BAU              @NOTE,U+T7              175
                 SLN              00002                   176
                 TSR              *XCWD                   177
                 SLF              00002                   200
REORG9           LDR              G,R+B4                  201
                 CLA              B4,U+P5                 202
          Z      LDR              MASK2,U+PF              203
REORF10   P5     IF(NZF)SKP       @B4+1,B4+1              204
          T6     TRA              REOR11,U+B5             205
                 CLA              B4+1                    206
                 LUR              @27,U+B1                207
                 CLA              B1+PF,U+PF              210
                 IF(NUL)SKP       Z,PF-1                  211
                 APF              1                       212
          PF     RPA              B4+1                    213
                 TRA              REOR10                  214
REOR11    T4     IF(NZF)TRA       TAKE1,R+Z               215
          Z      TRA              TAKE2,U+B2              216
                                                          217
ATAKE     1T61   LRS              @15                     220
          R      RPL              STORAG                  221
          P1     STX              @*36-1,U+T7             222
          Z      IF(ZFR,NTC)TRA   @*UNSAVE,B6-1           223
          T7     LT7              B5-1,U+B1               224
          PF     AB6              @77776,U+CC             225
                                                          226
MASK1            OCT              400000000               227
MASK2            OCT              77777777740077777       230
NOTE             BCD              REORGANIZATION          231
G                EQU              135                     232
XCWD             EQU              135                     233
SAVE             EQU              136                     234
UNSAVE           EQU              137                     235
STORAG           EQU              100                     236
FIRSTEX          EQU              101                     237
LASTEX           EQU              102                     240
                 END                                      241
                                                          242
                                                          243
```

PROGRAM   154   4/12/66 15.27

```
STEY FOR SPIREL
                              1       07 50472 26 0002 00000
                              2       10 01000 02 4400 00136        SAVE
                              3       00 20102 26 0000 77775
                              4       01 50400 00 0000 00100        STORAG
                              5       00 45062 06 4000 00017
                              6       00 20100 05 0000 00101        FIRSTE
                              7       00 20100 04 4002 00000
                             10       41 01010 10 4001 00070        REORG
                             11       07 01040 00 4001 00047        TAKE
INACTIVATE SPACE ADDRESSED BY B1
          GIVE1              12       01 21700 07 0002 00000
                             13       01 01040 44 4001 00041        GIVE5
                             14       01 45066 53 4000 00017
                             15       07 45010 45 4000 00030
                             16       45 45010 45 4000 00003
                             17       01 01040 00 4001 00004        GIVE2
                             20       01 50400 00 0001 00167        MASK2
                             21       00 02040 00 0000 00007
                             22       01 41004 20 4040 00000
                             23       01 41004 00 4040 77776
          GIVE2              24       00 20100 02 4010 00001
                             25       00 20100 00 4020 00000
                             26       01 02510 00 0000 00005
                             27       02 10001 00 0000 00006
                             30       44 21601 00 0002 00000
                             31       07 50314 00 0001 00155        MASK1
                             32       01 01040 00 4001 00004        GIVE3
                             33       41 20001 26 4100 00000
                             34       44 10000 41 4010 77776
                             35       00 20102 26 4010 00000
                             36       01 01000 10 4001 77752        GIVE1
          GIVE3              37       00 50401 54 0002 00000
                             40       00 45062 47 4000 00017
                             41       00 20100 27 4020 77776
                             42       01 02110 10 0000 00005
                             43       01 01000 00 4001 00006        GIVE4
                             44       47 4501F 64 4000 00017
                             45       02 20100 73 0400 00100        STORAG
                             46       01 20001 00 4020 00000
                             47       43 01040 00 4001 00001
                             50       44 21601 20 0010 00000
                             51       44 21601 00 0000 00100        STORAG
          GIVE4              52       00 20100 00 4002 00000
                             53       01 02050 00 0000 00004
                             54       01 01000 00 4001 00004        TAKE
          GIVE5              55       30 10001 61 0100 77776
                             56       01 01050 10 4001 77732        GIVE1
                             57       01 21700 76 0100 77775
                             60       01 01000 41 4001 77755        GIVE3
ACTIVATE BLOCK OF LENGTH B2+1
          TAKE               61       42 01010 10 4001 00117        ATAKE
                             62       00 20100 07 4004 00001
                             63       06 02110 00 0000 00007
                             64       00 01000 41 4001 00114        ATAKE
                             65       01 20000 06 4000 00000
                             66       01 50400 73 0400 00100        STORAG
                             67       00 45062 00 4000 00017
                             70       01 02110 45 0000 00007
                             71       01 01000 00 4001 00007        REORG
          TAKE1              72       04 20001 23 4010 00000
          TAKE2              73       45 4501F 54 4000 00017
                             74       45 01010 00 4001 00001        TAKE3
                             75       02 20001 74 4014 00000
```

```
        TAKE?      76      44 ?1601 00 0000 00100     STORAG
                   77      04 01010 10 4001 00101     ATAKE
                  100      43 01000 41 4001 00100     ATAKE
WRITE ACTIVE BLOCKS TO LOW ADDRESSES
        REORG     101      46 42002 07 4000 04000
                  102      01 4300E 76 4000 00000
                  103      01 40003 74 4400 00101     FIRSTE
                  104      01 01000 00 4001 00035     REORG7
        REORG1    105      01 21700 45 0420 00000
                  106      01 45066 51 4000 00017
                  107      46 01050 00 4001 00002     REORG2
                  110      01 40003 74 4012 00001
                  111      01 01000 00 4001 00025     REORG6
        REORG2    112      01 21700 00 4140 00000
                  113      01 21641 00 0420 00000
                  114      01 50214 00 0001 00072     MASK1
                  115      01 01050 23 4001 00005     REORG3
                  116      01 41003 64 4002 00000
                  117      44 21601 21 0001 00001
                  120      20 41024 45 4002 00001
                  121      01 21742 61 0040 00000
                  122      01 01000 00 4001 00017     REORG7
        REORG3    123      01 21743 24 0020 00000
        REORG4    124      01 53403 24 0020 00000
                  125      01 01040 61 4001 00010     REORG5
                  126      01 50114 55 0001 00061     MASK2
                  127      01 01040 00 4001 00001
                  130      02 45020 20 4000 00011
                  131      02 45020 65 4000 00011
                  132      01 44000 00 4000 00044
                  133      01 05110 25 4001 00001
                  134      01 10000 45 4040 77776
                  135      43 21641 00 0040 77776
        REORG5    136      41 01050 23 4001 77764     REORG4
        REORG6    137      00 20100 30 4020 00000
                  140      01 02510 00 0000 00004
                  141      46 4300E 47 4000 00002
        REORG7    142      44 02050 00 4400 00102     LASTEX
                  143      01 01000 00 4001 00005     REORG8
                  144      01 50400 00 0020 00000
                  145      00 45062 41 4000 00017
                  146      01 01040 00 4001 77735     REORG1
                  147      43 10100 74 4022 00000
                  150      01 01000 46 4001 77770     REORG7
        REORG8    151      07 42006 46 4000 04000
                  152      04 06550 00 0000 00005
                  153      47 10001 00 0000 00004
                  154      01 02070 00 4000 00002
                  155      01 01000 00 4001 00005     REORG9
                  156      01 21700 00 0001 00034     +0000A
                  157      01 20100 07 4001 00031     NOTE
                  160      01 42000 00 4000 00002
                  161      01 40000 00 4400 00126     XCWD
                  162      01 42004 00 4000 00002
        REORG9    163      01 50400 54 0000 00125     G
                  164      01 21700 45 0020 00000
                  165      00 50400 47 0001 00022     MASK2
        REOR10    166      45 02050 24 4020 00001
                  167      06 01000 45 4001 00007     REOR11
                  170      01 21700 00 0020 00001
                  171      01 45010 41 4000 00033
                  172      01 21700 47 0202 00000
                  173      01 02040 67 0000 00000
                  174      01 41007 00 4000 00001
                  175      47 21601 00 0020 00001
                  176      01 01000 00 4001 77766     REOR10
```

```
REOR11     177      04 01050 10 4001 77671     TAKE1
           200      00 01000 42 4001 77671     TAKE2
ATAKE      201      26 45015 00 4000 00017
           202      02 20301 00 0000 00100     STORAG
           203      41 43005 07 4500 77776
           204      00 01014 66 4400 00137     UNSAVE
           205      07 50470 41 0100 77776
           206      47 41006 40 4000 77776
MASK1      207      00 00000 00 4000 00000
MASK2      210      77 77777 77 7400 77777
NOTE       211      61 44566 14 6405 55071
           212      40 63505 65 5252 52525
*********
+0000A     213      00 00240 10 0000 00000
```

```
314   SAVE      0   136   0   243000000000000   0
315   STORAG    0   100   0   245000000000000   0
316   FIRSTE    0   101   0   246000000000000   0
317   REORG     0   101   1   124000000000000   0
320   TAKE      0    61   1    77000000000000   0
321   GIVE1     0    12   1    23000000000000   0
322   GIVE5     0    55   1    66000000000000   0
323   GIVE2     0    24   1    35000000000000   0
324   MASK2     0   210   1   235000000000000   0
325   MASK1     0   207   1   233000000000000   0
326   GIVE3     0    37   1    50000000000000   0
327   GIVE4     0    52   1    63000000000000   0
330   ATAKE     0   201   1   224000000000000   0
331   TAKE1     0    72   1   110000000000000   0
332   TAKE2     0    73   1   111000000000000   0
333   TAKE3     0    76   1   114000000000000   0
334   REORG7    0   142   1   165000000000000   0
335   REORG1    0   105   1   130000000000000   0
336   REORG2    0   112   1   135000000000000   0
337   REORG6    0   137   1   162000000000000   0
340   REORG3    0   123   1   146000000000000   0
341   REORG4    0   124   1   147000000000000   0
342   REORG5    0   136   1   161000000000000   0
343   LASTEX    0   102   0   247000000000000   0
344   REORG8    0   151   1   174000000000000   0
345   REORG9    0   163   1   206000000000000   0
346   +0000A    0   213   1   251000000000000   0
347   NOTE      0   211   1   237000000000000   0
350   XCWD      0   126   0   242000000000000   0
351   G         0   125   0   241000000000000   0
352   REOR10    0   166   1   211000000000000   0
353   REOR11    0   177   1   222000000000000   0
354   UNSAVE    0   137   0   244000000000000   0
```

- Matrix Inverse

This program computes the inverse and determinant of a real matrix and prints an error message if the matrix is singular. The method used is essentially in-place Gaussian reduction as described in "An Introduction to Numerical Mathematics", Stiefel, E.L., 1963, page 3. Each successive pivot element is the largest in absolute value of all the remaining choices in a given column. The result is a compromise between speed and accuracy. An n x n matrix is numerically singular if the ratio of any two pivot elements exceeds $10^6/n$. The codeword address of the matrix to be inverted is in T7 on entry, the inverse is stored as USTAR (codeword address 10), and the determinant is output in T7. If the matrix is singular, T7 = 0 on exit.

Lines 11 to 36:

The fast registers are saved, the input matrix is copied if necessary, internal constants are computed, the row codewords are labelled, and DET is initialized.

Lines 37 to 61:

The next column is scanned for the largest element, the largest and smallest pivot are stored and tested.

Lines 62 to 101:

The exchange algorithm is now applied to USTAR, the non-scalar accumulator in Genie and the pivot element is multiplied into DET.

Lines 102 to 113:

The two appropriate row codewords and their back references are exchanged if necessary.

Lines 114 to 151:

The columns of the final inverse matrix are now sorted as necessary due to non-diagonal pivoting.

Lines 152 to 157:

This section of code causes printing of an error message.

Lines	Comments
2	This is a symbolically named program, INV.
4-5	Cross-reference words for named items referred to by INV.
7	Extra carriage returns and a remark in the code sequence.
11	Use of +2 store option in operation field, store to B6.
12	Minus inflection in SETU, compound test in OPN, use of EQU'ed name in address field. The 'a' bit is not required since TRA gives this inflection automatically.
15-16	EQU'ed name in address field, and REF'ed name in address field.
35	Decimal constant in address field will be stored at the bottom of the program.
41	Absolute value inflections in SETU and ADDR, and indirect addressing specified by '*' in ADDR.
46	'→' codes as a store to M, here MAXP; '+1' in OPN is equivalent.
66	Enter repeat mode option on set or add to B-register orders.
106	Use of more than one B-modifier in field 4, B1 + PF + M (M = 0).
127	Reset X register from number originally stored on B6-list.
154	The address part of this instruction or M was replaced by the contents of PF at the instruction on line 13. Anything in ( ) is ignored in assembly.
160	A decimal constant is defined and is stored at EPSLN.
162-165	'Z' with OCT causes zero to be stored at these locations.

Lines	Comments
166-171	EQU psuedo-orders assign numeric values to names.
173-174	The END pseudo-order terminates the code but generates no instructions. It is followed by two carriage returns.

```
                                                                1
INV                 ORG                                         2
                                                                3
MCOPY               REF          *MCOPY                         4
ERRP                REF          *-ERRP                         5
                                                                6
                    REM          INV(T7) - USTAR                7
                                                               10
          Z         BAU+2        X,36+1                        11
          -Z        IF(ZER,EOV)TRA  a*SAVE,U-R                 12
          PF        RPA          PFSAVE,R-Z                    13
          Z         BAU          T7,R-B3                       14
                    IF(ZER)SKP   aUSTAR,I-B1                   15
          T7        TSR          a*MCOPY,U-B2                  16
          Z         STO          aMINP                         17
                    STO          aMAXP                         20
                    CLA          B1,U-B1                       21
                    CRL          a-15,R-B4                     22
          -B4       CPL          aZ                            23
                    FMP          TW047                         24
                    VDF          EPSLN                         25
                    STO          aERROR                        26
ROWSTO              LDR          B1+1,B3+1                     27
                    LLS          a-15,U-B5                     30
          B3        LRS          a-15                          31
          F         STO          aB1+1,B1+1                    32
          B3        IF(POS)SKP   a34                           33
                    TRA          aROWSTO                       34
          B4        LDR          a1.0,U-PF                     35
          F         STO          aDET                          36
INVLP     Z         STX          aZ,U-T6                       37
                    SB2          aPF,I-B1                      40
SCAN      |T6|      IF(POS)SKP   |*USTAR|                      41
          B1        LT6          *USTAR,U-B3                   42
          B3        AB1          a77776,U-T7                   43
          B1        IF(PNZ)TRA   aSCAN                         44
          |T6|      IF(ZER,EOV)TRA  aSINGLR                    45
                    LT5-         MAXP,U-T4                     46
          T5        IF(ZER)TRA   aFIRST                        47
          T4        IF(PNZ)SKP   T5                            50
          T5        STO          aMAXP                         51
FIRST     T4        LT6-         MINP                          52
          T6        IF(ZER)TRA   aSTEST                        53
          T4        IF(NNZ)SKP   T6                            54
          T6        STO          aMINP                         55
STEST               CLA          MAXP                          56
                    FDV          MINP                          57
                    IF(NEG)SKP   ERROR,R-Z                     60
                    TRA          aSINGLR                       61
          B3        LT4          -1.0,U-B1                     62
          T4        LT5-         *USTAR,R-B2                   63
                    FDV          T5,U-T4                       64
          T5        FMP-         DET,B2+1                      65
          B4        SB3,ERM      a35,U-T6                      66
          T4        FMP-         *USTAR,B3-1                   67
LOOP1     B4        SB2          aPF,U-B1                      70
          B1        IF(ZER)JMP   T7,B4-1                       71
          Z         LT5-         *USTAR                        72
                    SB2          a35                           73
```

```
LOOP2            SB1         a*T7                    74
        T5       FMP         *USTAR                  75
                 SB1         aB4+1                   76
                 FAD.        *USTAR,B2-1             77
        B2       IF(PNZ)TRA  aLOOP2                  100
        B4       IF(PNZ)TRA  aLOOP1                  101
        T7       SB4         a*T6,U+B3               102
                 CLA         USTAR,U+B1              103
        B3       IF(NZE)SKP  aPF                     104
                 TRA         aTEST,PF-1              105
                 CLA         B1+PF,U+B2              106
                 LDR.        B1+B3,I+B3              107
        B3       STO         aB2,R+B2                110
        R        STO         aB1+PF,I+B3             111
        B3       STO         aB2,PF-1                112
TEST    PF       IF(PNZ)TRA  aINVLP                  113
                 SB3         aB4,I+B2                114
HUNT             LDR         B1+B3                   115
        Z        LLS         aH15                    116
                 IF(ZER)SKP  aB2,B3-1                117
        B3       IF(PNZ)TRA  aHUNT                   120
        B2       IF(ZER)SKP  aB3+1                   121
                 TRA         aSWAP,B3+1              122
        B1       TRA         aFIX,U+PF               123
LAST             SB2         aB2-1,I+B3              124
        B2       IF(PNZ)TRA  aHUNT                   125
OUT              TRA         a*UNSAVE                126
                 STX         a*B6-1,B6-1             127
        PF       LT7         DET,U+CC                130
SWAP    B1       SB1         aB4,U+PF                131
EXLOOP           LDR         *USTAR                  132
        B2       SB2         aB3,U+B3                133
        R        LDR.        *USTAR                  134
        B2       SB2         aB3,U+B3                135
        R        STO         *USTAR,B1-1             136
        B1       IF(PNZ)TRA  aEXLOOP                 137
                 CLA         PF+B3                   140
                 LDR         PF+B2                   141
                 LLS         aH15,U+R                142
                 CRR         aH15                    143
                 STO         aPF+B3                  144
FIX              LDR         PF+B2                   145
                 LLS         aH15                    146
        B5       LRS         aH15                    147
        R        STO         aPF+B2                  150
        PF       TRA         aLAST,U+B1              151
SINGLR  Z        SB1         aUSTAR,U+B2             152
                 TSR         a*STEX                  153
PFSAVE  I        SPF         a(Z),U+B1               154
                 TRA         a*ERROR,B1+1            155
        Z        STO         aDET                    156
                 TRA         aOUT                    157
EPSLN            DEC         1000000.0               160
TWO47            OCT         062000000000000000      161
ERROR            OCT         Z                       162
MINP             OCT         Z                       163
MAXP             OCT         Z                       164
DET              OCT         Z                       165
USTAR            EQU         10                      166
```

```
STEX            EQU          135                       167
SAVE            EQU          136                       170
UNSAVE          EQU          137                       171
                                                       172
                END                                    173
                                                       174
```

```
PROGRAM       INV                                              4/11/66 16.14

        MCOPY  77776         54 42565 77 0400 00000
        ERROR  77777         75 44616 15 7400 00000
INV(T7) → USTAR
                    1        00 20102 26 0000 77775
                    2        10 01310 02 4400 00136     SAVE
                    3        47 21601 10 0001 00140     PFSAVE
                    4        00 20100 53 0000 00007
                    5        01 02010 71 4000 00010     USTAR
                    6        07 40000 42 4401 77767     MCOPY
                    7        00 20001 00 4001 00143     MINP
                   10        01 20001 00 4001 00143     MAXP
                   11        01 21700 41 0002 00000
                   12        01 45066 54 4000 00017
                   13        54 50100 00 4000 00000
                   14        01 10600 00 0001 00134     TW047
                   15        01 16700 00 0001 00132     EPSLN
                   16        01 20001 00 4001 00133     ERROR
        ROWSTO     17        01 50400 23 0002 00001
                   20        01 45062 45 4000 00017
                   21        43 45015 00 4000 00017
                   22        02 20001 21 4002 00001
                   23        43 02110 00 4020 00000
                   24        01 01000 00 4001 77771     ROWSTO
                   25        44 50400 47 0001 00130     +0000A
                   26        02 20001 00 4001 00126     DET
        INVLP      27        00 43005 06 4000 00007
                   30        01 40002 71 4200 00000
        SCAN       31        26 02110 00 2400 00010     USTAR
                   32        41 50460 43 0400 00010     USTAR
                   33        43 41001 07 4000 77776
                   34        41 05150 00 4001 77773     SCAN
                   35        26 01310 00 4001 00104     SINGLR
                   36        01 50451 04 0001 00115     MAXP
                   37        05 01010 00 4001 00002     FIRST
                   40        04 06150 00 0000 00005
                   41        05 20001 00 4001 00112     MAXP
        FIRST      42        04 50461 00 0001 00110     MINP
                   43        06 01010 00 4001 00002     STEST
                   44        04 06550 00 0000 00006
                   45        06 20001 00 4001 00105     MINP
        STEST      46        01 21700 00 0001 00105     MAXP
                   47        01 10700 00 0001 00103     MINP
                   50        01 02510 10 0001 00101     ERROR
                   51        01 01000 00 4001 00070     SINGLR
                   52        43 50440 41 1001 00103     +0000A
                   53        04 50451 52 0400 00010     USTAR
                   54        01 10700 04 0000 00005
                   55        05 10601 22 0001 00077     DET
                   56        44 40023 06 4040 00000
                   57        04 10601 63 0400 00010     USTAR
        LOOP1      60        44 40002 41 4200 00000
                   61        41 03010 64 0000 00007
                   62        00 50451 00 0400 00010     USTAR
                   63        01 40002 00 4040 00000
        LOOP2      64        01 40001 00 4400 00007
                   65        05 10600 00 0400 00010     USTAR
                   66        01 40001 00 4020 00001
                   67        01 10401 62 0400 00010     USTAR
                   70        42 05150 00 4001 77772     LOOP2
                   71        44 05150 00 4001 77765     LOOP1
                   72        07 40004 43 4400 00006
                   73        01 21700 41 0000 00010     USTAR
                   74        43 02050 00 4200 00000
                   75        01 01000 67 4001 00005     TEST
```

```
            76     01 21700 42 0202 00000
            77     01 50401 73 0012 00000
           100     43 20001 52 4004 00000
           101     02 20001 73 4202 00000
           102     43 20001 67 4004 00000
TEST       103     47 05150 00 4001 77722     INVLP
           104     01 40003 72 4020 00000
HUNT       105     01 50400 00 0012 00000
           106     00 45062 00 4000 00017
           107     01 02010 63 4004 00000
           110     43 05150 00 4001 77773     HUNT
           111     42 02010 00 4010 00001
           112     01 01000 23 4001 00006     SWAP
           113     41 01000 47 4001 00021     FIX
LAST       114     01 40002 73 4004 77776
           115     42 05150 00 4001 77766     HUNT
OUT        116     01 01000 00 4400 00137     UNSAVE
           117     01 43005 66 4500 77776
           120     47 50470 40 0001 00034     DET
SWAP       121     41 40001 47 4020 00000
EXLOOP     122     01 50400 00 0400 00010     USTAR
           123     42 40002 43 4010 00000
           124     02 50401 00 0400 00010     USTAR
           125     42 40002 43 4010 00000
           126     02 20001 61 4400 00010     USTAR
           127     41 05150 00 4001 77771     EXLOOP
           130     01 21700 00 0210 00000
           131     01 50400 00 0204 00000
           132     01 45062 02 4000 00017
           133     01 45055 00 4000 00017
           134     01 20001 00 4210 00000
FIX        135     01 50400 00 0204 00000
           136     01 45062 00 4000 00017
           137     45 45015 00 4000 00017
           140     02 20001 00 4204 00000
           141     47 01000 41 4001 77751     LAST
SINGLR     142     00 40001 42 4000 00010     USTAR
           143     01 40000 00 4400 00135     STEX
PFSAVE     144     20 40007 41 4000 00000
           145     01 01000 21 4401 77631     ERPR
           146     00 20001 00 4001 00006     DET
           147     01 01000 00 4001 77745     OUT
EPSLN      150     03 01720 44 0000 00000
TWO47      151     06 20000 00 0000 00000
ERROR      152     00 00000 00 0000 00000
MINP       153     00 00000 00 0000 00000
MAXP       154     00 00000 00 0000 00000
DET        155     00 00000 00 0000 00000
*********
+0000A     156     01 00100 00 0000 00000


314     MCOPY    0 77776   1
315     ERPR     0 77777   1      2000000000000    0
316     SAVE     0   136   0      3000000000000    0
317     PFSAVE   0   144   1    174000000000000    0
320     USTAR    0    10   0    152000000000000    0
321     MINP     0   153   1    172000000000000    0
322     MAXP     0   154   1    165000000000000    0
323     TWO47    0   151   1    167000000000000    0
324     EPSLN    0   150   1    161000000000000    0
325     ERROR    0   152   1    157000000000000    0
326     ROWSTO   0    17   1    163000000000000    0
                                 25000000000000    0
```

```
INV              ORG                                        1
                 REM          BACK-TRANSLATION              2
L77776           REF          *MCOPY                        3
L77777           REF          *.ERRP                        4
L1        Z      BAU+2        77775,B6+1                    5
          -Z     01310        @*136,U→R                     6
          PF     RPA          L144,R→7                      7
          Z      BAU          T7,R→B3                      10
                 IF(ZER)SKP   @10,I→B1                     11
          T7     TSR          *L77776,U→B2                 12
          Z      STO          L153                         13
                 STO          L154                         14
                 CLA          B1,U→B1                      15
                 CRL          17,R→B4                      16
          -B4    CPL          @7                           17
                 FMP          L151                         20
                 VDF          L150                         21
                 STO          L152                         22
L17              LDR          B1+1,B3+1                    23
                 LLS          17,U→B5                      24
          P3     LRS          17                           25
          P      STO          B1+1,B1+1                    26
          P3     IF(POS)SKP   @34                          27
                 TRA          L17                          30
          P4     LDR          L156,U→PF                    31
          P      STO          L155                         32
L27       7      STX          7,U→T4                       33
                 SB2          PF,I→B1                      34
L31       IT6I   IF(POS)SKP   1*101                        35
          P1     LT6          *10,U→B3                     36
          P3     AB1          77776,U→T7                   37
          P1     IF(PNZ)TRA   L31                          40
          IT6I   01310        @L142                        41
                 LT5→         L154,U→T4                    42
          T5     IF(ZER)TRA   L42                          43
          T4     IF(PNZ)SKP   T5                           44
          T5     STO          L154                         45
L42       T4     LT6→         L153                         46
          T6     IF(ZER)TRA   L46                          47
          T4     IF(NNZ)SKP   T6                           50
          T6     STO          L153                         51
L46              CLA          L154                         52
                 FDV          L153                         53
                 IF(NEG)SKP   L152,R→Z                     54
                 TRA          L142                         55
          P3     LT4          -L156,U→B1                   56
          T4     LT5→         *10,R→B2                     57
                 FDV          T5,U→T4                      60
          T5     FMP→         L155,B2+1                    61
          P4     40023        @35,U→T6                     62
          T4     FMP→         *10,B3=1                     63
L60       P4     SB2          PF,U→B1                      64
          P1     IF(ZER)JMP   T7,B4-1                      65
          7      LT5→         *10                          66
                 SB2          B5                           67
L64              SB1          *7                           70
          T5     FMP          *10                          71
                 SB1          B4+1                         72
                                                           73
```

```
                   FAD+        *10,B2=1                74
          F2       IF(PNZ)TRA  L64                     75
          F4       IF(PNZ)TRA  L60                     76
          T7       SB4         *6,U+B3                 77
                   CLA         10,U+B1                 100
          F3       IF(NZE)SKP  aPF                     101
                   TRA         L103,PF-1               102
                   CLA         PF+B1,U+B2              103
                   LDR+        B1+B3,I+B3              104
          F3       STO         B3,R+B2                 105
          F        STO         PF+B1,I+B3              106
          F3       STO         B3,PF-1                 107
L103      PF       IF(PNZ)TRA  L37                     110
                   SB3         B4,I+B2                 111
L105               LDR         B1+B3                   112
          7        LLS         17                      113
                   IF(ZER)SKP  a32,B3=1                114
          F3       IF(PNZ)TRA  L105                    115
          F2       IF(ZER)SKP  a33+1                   116
                   TRA         L121,B3+1               117
          F1       TRA         L135,U+PF               120
L114               SB2         B3-1,I+B3               121
          F2       IF(PNZ)TRA  L105                    122
L116               TRA         *137                    123
L117               STX         *36-1,B6=1              124
          FF       LT7         L135,U+CC               125
L121      F1       SB1         B4,U+PF                 126
L122               LDR         *10                     127
          F2       SB2         B3,U+B3                 130
          F        LDR+        *10                     131
          F2       SB2         B3,U+B3                 132
          F        STO         *10,B1=1                133
          F1       IF(PNZ)TRA  L122                    134
                   CLA         PF+B3                   135
                   LDR         PF+B2                   136
                   LLS         17,U+P                  137
                   CRR         17                      140
                   STO         PF+B3                   141
L135               LDR         PF+B2                   142
                   LLS         17                      143
          F5       LRS         17                      144
          F        STO         PF+B2                   145
          FF       TRA         L114,U+B1               146
L143      7        SB1         10,U+B2                 147
                   TSR         *135                    150
L144      1        SPF         Z,J+B1                  151
                   TRA         *L77777,B1+1            152
L146      7        STO         L155                    153
                   TRA         L116                    154
L150               OCT         03017204400000000       155
L151               OCT         06200000000000000       156
L152               OCT         00000000000000000       157
L153               OCT         00000000000000000       160
L154               OCT         00000000000000000       161
L155               OCT         00000000000000000       162
L156               OCT         01001000000000000       163
                   END                                 164
                                                       165
                                                       166
```

GENIE

GENIE

Genie . . . .

Genie Program Format . . . .

Names . . . .

Numbers . . . .

Variables . . . .

Declarations . . . .

Functions . . . .

Constants . . . .

Remarks . . . .

Command Sequence . . . .

Arithmetic Expressions . . . .

Arithmetic Commands . . . .

Conditional Arithmetic Commands . . . .

Transfer Control Commands . . . .

Loop Control Commands . . . .

Storage Control Commands . . . .

Execute Control Commands . . . .

Input-Output Commands . . . .
(Including Sense lights)

Data Commands . . . .

Fast Registers . . . .

Assembly Language . . . .

Punctuation . . . .

Compilation Procedure . . . .

Running Genie Programs . . . .

Coding Examples . . . .

Genie Coding Conventions . . . .

The formula language for the Rice Computer is called <u>the Genie language</u>. Programs written in the Genie language are called <u>Genie programs</u>. Translation of Genie programs into machine language is accomplished by <u>the Genie compiler</u>.

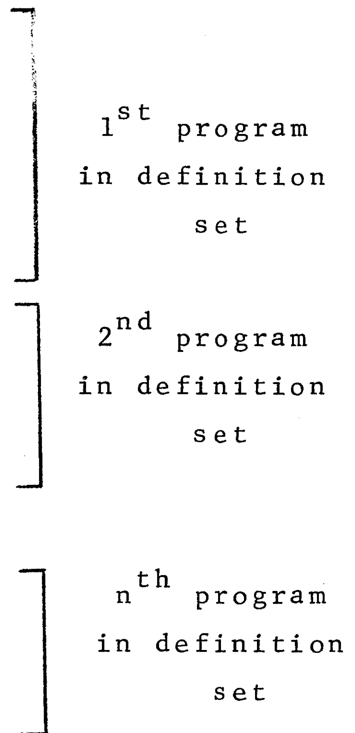
The language and the compiler are both often referred to as just <u>Genie</u>. What is meant is usually clear from the context.

Genie programs may contain instructions written in the AP2 assembly language. Hence, the AP2 assembly language is a subset of the Genie language, and the AP2 assembly program is a subset of the Genie compiler.

PROGRAM FORMAT

The unit of definition to the Genie compiler is the <u>definition set</u>, which has the form

```
          DEFINE
          declarations of external variables and
            parameters for the entire definition set
          constant codeword address specifications for external
            variables
          function definitions
PROG1(PARAM1).=SEQ
          declarations of internal variables
          remarks
          constant specifications
          command sequence for the calculation
END
PROG2(PARAM2).=SEQ
                    .
                    .
                    .
END
                    .
                    .
                    .
PROGn
                    .
                    .
                    .
END
          DEFINE
LEAVE
| cr      | 1st tab stop
```



A definition, then, is a collection of programs (in the most usual case just one) which depends on a common set of external quantities and which are completely independent with respect to their private internal symbols. The definition set has meaning only at compilation; the independent programs may be dynamically interconnected, among themselves or with programs compiled at another time, in any meaningful way at the time they are executed.

Typing of the definition set is begun by the sequence 'cr tab uc DEFINE'. This first 'DEFINE' insures that the compiler does not retain any symbols mentioned by another user of the system. Each line of a program should be begun with a case punch (uc or lc) and is ended by a carriage return (cr). If a statement is so long that it needs to be broken in typing, the sequence 'cr tab tab tab' provides continuation of the statement onto the next line. 'PROGi' designates a program name. 'PARAMi' designates the parameters of the program, a non-empty list of names separated by commas. The operation '.=' followed by the symbol 'SEQ' signals initiation of code generation for the program. Recursive code will be generated (so that a program may use itself) if 'RSEQ' is used instead of 'SEQ'. 'END', typed at the left hand margin and followed immediately by a 'cr', terminates the program, initiates final compiler output of the program, and causes the symbol table limit to be backed up so that the compiler retains only its vocabulary symbols and the external variables of the definition set. The second 'DEFINE' terminates the definition set and causes the symbol table limit to be backed up so that the compiler retains only its vocabulary symbols; all external variables backed over are printed out. 'LEAVE', typed at the left hand margin and followed immediately by 'cr cr', causes exit from the system.

Private names, those invented by a user of the Genie compiler, are formed by the following rules:

1) a single lower case Roman letter;

or 2) an upper case Roman letter, followed by upper case Roman letters, followed by lower case Roman letters, followed by numerals (no embedded spaces).

By rule 1) the following are examples of names:

a   i   p   x

By rule 2) the following are examples of names:

A   CAT   Fn   DDxy   I2   PQ29   Dog3

Concatenation of names implies multiplication of the variables specified. The following are not names:

ab   A B38   Pt4p   M5ef   w10

They are interpreted respectively as:

a×b   A×B38   Pt4×p   M5×e×f   w×10

Any number of characters may be used in a name, but only <u>five</u> are retained by the compiler. If lower case Roman letters are embedded in a name, the first is tallied as two characters.

The names

m   Man

are stored as

.M   M.AN

Names in the vocabulary of the compiler may not be used by the coder as private names. These include

names of library items -- COL, SIN, LINCT, etc.

names of various machine registers -- B1, CC, T4, etc.

names with special meaning in the Genie language -- as DATA, TRUE, LEAVE, etc.

In alphabetical order, vocabulary names are:

ACOSH	CCSH	CSOLN	FUNCT	MINSE	REPEA
ACCEPT	CDET	CSNH	FXEXP	MITIM	RESUL
ARRAY	CDIV	CSQR	GAMMA	MMPY	ROW
ASIN	CEXP	CSUB	GO	MOD	RTRAN
ASINH	CFEXP	CTAN	IM	MODUL	SCALA
ATAKE	CHISQ	CTNH	INFER	MPATC	SCRIB
ATAN	CINV	CTRAN	INPUT	MPOLA	SET
ATANH	CLENG	CVSPA	INTEG	MPOWE	SIN
B1	CLOG	CXEXP	INV	MRE	SINH
B2	CMADD	DATA	ITIME	MSPAC	SL
B3	CMCON	DEFIN	ITRAN	MSUB	SMDIV
B4	CMCPY	DET	LEAVE	MTAKE	SMMPY
B5	CMMPY	DIAG	LENGT	NEO	SOLN
BCD	CMPY	DISPL	LET	NUMBE	SQR
BOOLE	CMSPA	DPUNCH	LGAMM	ODD	STNDV
CADD	CMSUB	END	LINCT	ORTHO	T4
CARTN	CMTAK	EOV	LOG	OUTPU	T5
CACSH	COL	ERASE	LOG10	PAGCT	T6
CALL	COMPL	EVEN	MADD	PAGE	T7
CASN	CONJ	EXECU	MATRI	PLOT	TAN
CASNH	CONTR	EXP	MAX	POLAR	TANH
CATAKE	CONVL	FALSE	MCART	PRESC	TITLE
CATN	COS	FFT	MCMPL	PRINT	TRAN
CATNH	COSH	FFTC	MCONJ	PUNCH	TRUE
CC	COT	FIX	MCOPY	QCONF	TTAKE
CCEXP	CRCOR	FLEX	MEAN	RANDM	VECTO
CCOL	CROW	FLOAT	MFLT	RE	VREV
CCONT	CSIN	FOR	MIM	READ	VSPAC
CCOS	CSMDV	FORMA	MIN	REAL	Z
CCOT	CSMMP	FTRAN	MINDE	REM	

The following names may be used as private symbols in Genie language but have special meaning in the assembly language:

B6 I PF R S U X

Four character strings which are not names have special meaning to the compiler:

and if or not

A string of decimal numerals

$$DDD < 2^{14}$$

is an integer. A string of decimal numerals containing either a decimal point '.' or a power point '*' is a floating point number. The form of a floating point number is illustrated by

A.B*C

which is interpreted as

$$A.B \times 10^{C}$$

There may be as many as 14 numerals in A and B combined. C is an integer between -70 and 70; if C is not preceded by a minus sign, it is taken to be positive. Minus signs may precede decimal numbers, integer or floating point, with the usual arithmetic meaning.

A string of 18 or fewer octal numerals immediately preceded by a unary '+'

+ØØØ

is a right-adjusted octal configuration. [A '+' between two numbers is binary and does not cause the number which follows it to be octal.]

The following numbers will be understood as shown:

3	decimal, integer
-3.0	decimal, floating point
3.	decimal, floating point
3*8	decimal, floating point
3.0*-8	decimal, floating point
-0.3	decimal, floating point
.3	decimal, floating point
+30	octal

Spaces may be embedded in numbers; they are ignored. Therefore, fields within a number may be separated by spaces for ease of reading. For example, if _ represents a 'space' punch,

3_641_209.4_*_-8

is exactly equivalent to

3641209.4*-8

and

+00200_0130_0004_00257

is exactly equivalent to

+002000130000400257

In any program, each variable falls into one of three categories: internal, external, or parameters.

Internal variables must be scalars (integer, real floating point, complex, or Boolean), and these are assigned storage within the program. The names of internal variables are not retained outside the compilation of a single program; hence, the same name may be used in more than one program with a different meaning in each of the programs. Labels on statements are also internal variables.

External variables may be either scalar (integer, real floating point, complex, or Boolean) or non-scalar (program, vector, matrix, or array), and all non-scalars must be external. All external variables of a program must appear in the definition set containing that program before any '.='. External variables of any one program are the common property of all programs in which they are declared external that are in the machine at running time. The names must have unique meaning throughout the system. During program execution, each external variable has its name on the symbol table (ST, *113) and its scalar value or non-scalar codeword in the corresponding value table (VT, *122) entry.

Parameters may be either scalar or non-scalar. If they are non-scalar they must be so declared within the definition set containing the program before any '.='. Parameters are neither internal nor external with respect to the program in which they appear, but while running the arguments will fall into one of these categories with respect to dynamically higher level programs. Parameters of a program are only representative of the arguments which will be specified to the program by the dynamically higher level program which uses it while running. Within a system of programs the dynamically highest level program receives control from the operating system and cannot have arguments provided by the system; hence, the dynamically top level program should have one purely dummy parameter, a name that is never referred to in the program. The names of parameters are not retained outside the compilation

of a definition set; the same name may be used as a parameter for more than one program in a definition set, but for no other purpose in the definition set.

Declarations are used to describe variables that names represent. The simple form of declaration is illustrated by:

```
          VECTOR A
          VECTOR A , B , C
          VECTORS A , B , C
|cr       |1st tab
```

A more general form is illustrated by:

```
          INTEGER VECTOR A , B , C
          INTEGERS VECTORS A , B , C
|cr       |1st tab
```

One or more declaration words (either singular or plural) are followed by one or more variable names separated by commas.

A variable used in the Genie language is completely described by its:

	type	integer, real, complex, or Boolean
	shape	Scalar, Vector, Matrix, or Array
and	mode	function or not

A scalar is described by a type declaration:

INTEGER or INTEGERS	for integer
REAL or SCALAR or SCALARS	for real floating point
COMPLEX	for complex (Cartesian form)
BOOLEAN	for Boolean

A non-scalar is described by a shape declaration:

VECTOR or VECTORS	for vector } whose elements are scalars
MATRIX or MATRICES	for matrix } whose elements are scalars
ARRAY or ARRAYS	for non-scalars whose elements are non-scalars

and a type declaration which applies to its elements.

A function is described the mode declaration:

FUNCTION or FUNCTIONS for a private program name

and type and shape declarations which appropriately describe its implicit result, if it has one. Note: Library programs are known to Genie, and need no declarations.

Not all variables need be described by declarations. When

a variable appears on the right side of an equation in the Genie language, its type, shape, and mode will be inferred if they have not been declared:

type	real floating point
shape	scalar
mode	non-function

The INFER declaration may be used to cause other type and shape inferences:

$$\text{INFER} \quad \begin{Bmatrix} \text{INTEGER} \\ \text{REAL} \\ \text{SCALAR} \\ \text{COMPLEX} \\ \text{BOOLEAN} \end{Bmatrix} \quad \begin{Bmatrix} \text{VECTOR} \\ \text{MATRIX} \\ \text{ARRAY} \end{Bmatrix}$$

where either a type or a shape is given, or both in either order. The range of effect of an INFER is to an INFER which respecifies what it specifies, but not outside a definition set.

The name of every external variable must appear in at least one declaration before any '.='. All declarations pertaining to parameters must appear before any '.=', but they need not appear in any declaration if inference will give a proper description. Declarations pertaining to internal variables must appear within the program to which they belong, and only the type declarations are applicable since all internal variables are scalars.

Not more than one declaration in each group may be applied to a single variable, and not more than one declaration in each group may appear in a single declaration statement.

Thus,

```
          BOOLEAN MATRIX FUNCTION F
| cr      | 1st tab
```

is a legal statement, but

```
          INTEGER BOOLEAN B
| cr      | 1st tab
```

is not.

A function is a program which may be referred to in the Genie language, either for implicit execution as 'F' in the command

y=a+F(P)+b

or for explicit execution as 'G' in the command

EXECUTE G(Q)

Implicit execution is meaningful only if the function is single valued; in this case its output is not specified in the parameter list. In all other instances explicit execution is required.

The last executed command of a function to be used implicitly must define the output as follows:

RESULT=scalar or non-scalar arithmetic expression

| cr | 1st tab

In the definition of a function, its parameters are given as an ordered list of those quantities which are supplied as arguments by the program which causes it to be executed. An argument for a parameter which designates a quantity to be calculated by the function must be specified as a simple variable name; other arguments may be given by any arithmetic expression. For example, if F(A,B,C) is defined such that parameters A and B are used in the calculation of parameter C by the function F, a proper use of F would be $F(3m^2+n,V_a,P)$. But $F(SIZE, SPAN, q^2)$ is incorrect since the third argument may not be an expression. Care must be taken that parameters in the definition of a Genie program and arguments in the use of it by other Genie programs are always listed in the same order and agree in number and type.

A function may be sufficiently simple to be defined in one statement. This is done before any '.=' and is illustrated by the definition of f in the statement

$f(x,y)=3ax+a^2y$, a=2+x

| cr

The function f may then be executed implicitly within the command

sequence of a program,

$$h = k^2 f(m,n)$$

where the closed subroutine f will be applied to the arguments m and n. During compilation, output for f will be produced independent of that for the other programs in the definition set.

Every Genie program is a function. It may be used as such by any other Genie program. A Genie program begun with 'RSEQ' is a recursive function, one which may use itself. For example, the function FACTL may be executed from within the command sequence for FACTL:

```
                        :
FACTL(k).=RSEQ
                        :
          m = FACTL(n-1)
                        :
END
```

A recursive function may be executed either implicitly or explicitly, as appropriate to its definition. Genie programs begun with 'SEQ' and functions defined in one statement do not cause recursive code to be generated; they may not use themselves.

All functions except those in the library must be declared in function declarations. If a function is to be executed implicitly and its result is not to be inferred, then its name must appear in declarations to describe the result as well as in a FUNCTION declaration. Thus, the function with its arguments is an operand which must be assigned the type and shape of its output if it is to appear within an arithmetic expression.

A function name is not followed by arguments in a declaration. To specify execution, a function must be followed by arguments, as SIN X2 or CALC(q) or MAP(a+b,VAR). A function name, without arguments, may be supplied as an argument to a function which will do the execution. Thus, the program P may be defined as P(...,F,...), where the parameter F is a function, and call for execution of F(...);

then P may be executed with argument g as P(...,g,...) and the result will be execution of g(...) while running.

Note: One inconvenience is associated with this notation. If F1 is a function of a single parameter F2 which is a function, the expression

...F1(F2)...

will be misinterpreted by the compiler. One extra pair of parentheses is required, as

...(F1(F2))...

if a single parameter is a function.

Internal variables which are constants may be numerically specified by a LET statement within the program. The statement must be given before the name of the constant is used in the commands of the calculation. The form of this statement is illustrated by:

```
          LET PI=3.14159
| cr      | 1st tab
```

This is a message to the compiler which causes the number 3.14159 to be used in the program each time the internal variable name 'PI' is used. A LET statement causes no code to be generated.

The above shows specification of a real floating point value. The variable PI takes on real floating point type.

An integer value may be specified, as

```
LET K=3
```

The variable K takes on integer type.

A complex value may be specified, as

```
LET CVAL=-3.2+/5.19
```

or

```
LET POLE= 1+/0
```

The variables CVAL and POLE take on complex type.

A Boolean value (TRUE or FALSE) may be specified, as

```
LET t=TRUE
```

or

```
LET No=FALSE
```

The variables t and No take on Boolean type.

An octal configuration (right-adjusted) may be specified, as

```
LET MASK=+777777077
```

The + inflection concatenated immediately to the left of a number denotes octal interpretation of the number. The variable MASK should not be used in the Genie language.

A fixed address or codeword address may be specified, as

```
LET #TIME=+200
```

must be used for every numbered scalar, program, vector, or matrix. A Genie program may assign its own name a numerical equivalent, and the tape produced by the compiler will load with codeword at the address specified.

The values of non-scalars may not be specified in a LET statement.

The LET statement may also be used to specify the equivalence of two names. For example

LET ALPHA = BETA

causes 'BETA' to be substituted for 'ALPHA' throughout the program. Similarly

LET COUNT = B5

causes the index register B5 to be used for 'COUNT'.

More than one constant may be specified in a LET statement, if they are separated by commas, as

LET A=3, z=5.41*-6, #PROG=+127, TIME1=TIME2

There are three other commands which identify names with values. They are explained later: BCD, NUMBERS, and FORMAT in the section on data commands. These commands are non-executable and must be transferred around, and must therefore be used with care.

Printed comments in compilation output listings may be obtained by using the REM statement within the program, as illustrated by

```
          REM COMPUTE_FIRST_VALUE
```

or

```
          REM
COMPUTE_FIRST_VALUE
| cr      | 1st tab
```

where _ indicates a space typed within the remark. 'REM' is followed immediately by a single space or 'cr' which is not part of the remark, and then all following characters are taken as remark text. The statement may be continued to succeeding lines at the 3rd tab position by using the 'cr tab tab tab' sequence. The form of REM in which the text begins at the left margin causes remarks to stand out more vividly on program listings.

The REM statement does not introduce any data into the final program; its only effect is to cause the remark to be printed in the compilation output listing.

All statements of a program from the '.=' to and including the 'END', except 'LET's, remarks, and declarations, cause code to be generated. Such statements are called <u>commands</u>. The occurrence of a label on a command causes a <u>command sequence</u> to be initiated. The ordered set of all command sequences of the program is called the <u>command sequence for the calculation</u>. Each command falls into one of four categories; arithmetic, control, input-output, or data. These will be discussed in separate sections.

Any command may be labelled. The label is typed at the left-hand margin, as 'CALC' in the command

```
CALC      A=B^2+B+3.2,  B=W+5.1
| cr      | 1st tab
```

The righthand side of an equation in the Genie language must be an arithmetic expression. An arithmetic expression is just an operand. A scalar constant, a variable, an inflected variable, or a function name followed by a parenthesized list of arguments is an operand. [A single argument given as a simple variable name need not be enclosed in parentheses.] A pair of operands joined by an operation (where the triplet left operand, operation, right operand is defined in Genie) is an operand.

Any operand may be enclosed in parentheses to dictate order of computation within an expression in the conventional manner. Order is also implied by relative rank of operations. In order of decreasing rank, i.e., the most binding first, the arithmetic operations are:

unary inflections: -, |...|, and 'not'
subscription
exponentiation
× and /
+ and binary -
relations: $=, \neq, <, \nless, \leq, \nleq$

The triplets of operands joined by an arithmetic operation which are permitted in an arithmetic expression on the righthand side of an equation are given in the following paragraphs.

1) +, -, ×, / between integer, real floating point, or complex scalar operands.

If the operands are both integer or both floating point or both complex, the result is of the same type. If one operand is complex and the other is not, the non-complex operand is made complex before the operation is carried out, and the result is complex. If one operand is floating point and the other is integer, the integer is floated before the operation is carried out, and the result is floating point.

2) +/ between integer or real floating point scalar or non-scalar operands.

This is the explicit representation of a complex quantity in Cartesian form, as x+/y (written x+iy in mathematical notation). The result is complex with real and imaginary parts real floating point. The shape (scalar, vector, or matrix) of the parts determine the shape of the result; both parts must be of the same shape, and non-scalars must have the same dimensions. If the operands joined by +/ are expressions, they must be enclosed in parentheses. If the operand x+/y is combined arithmetically with other terms, it must be enclosed in parentheses.

-/ between integer or real floating point scalar operands.

The complex scalar x-/y is simply x+/(-y).

3) +(or), -(symmetric difference), ×(and), /(symmetric sum) between Boolean scalar operands.

Combination of Boolean operands yields a Boolean result, by the following rules:

\+ FALSE if both operands FALSE, otherwise TRUE

\- TRUE if operands differ, FALSE if operands the same

× TRUE if both operands TRUE, otherwise FALSE

/ TRUE if operands the same, FALSE if operands differ

The octal representations for the Boolean values are

TRUE 0077777777777777777

FALSE 0077777777777777776

4) +, -, × between non-scalar operands containing integer, real floating point or complex elements.

Standard conventions apply as to restrictions on dimensional compatibility, and the operands must be in standard form.* Addition or subtraction of two vectors or two matrices yields a vector or a matrix respectively. Multiplication of matrices yields a matrix. Multiplication of vectors yields the scalar product which is

a scalar. Multiplication of a vector and matrix yields a vector. If the operands are of the same type, the result is of that type. If the operands are of different types and one is complex, the result is complex. If one operand is integer and the other floating point, the result is floating point.

5) × between integer, real floating point, or complex scalar and integer, real floating point, or complex non-scalar.

The scalar may be on the left or the right of the non-scalar, which must be in standard form.* The result has the same form as the non-scalar operand, vector or matrix. If the operands are both integer or both floating point or both complex, the result is of the same type. An integer operand is floated before combination with a floating point operand, and the result is floating point. An integer operand is floated and then made complex before combination with a complex operand, and the result is complex. A floating point operand is made complex before combination with a complex operand, and the result is complex.

6) Division of an integer, real floating point, or complex non-scalar by an integer, real floating point, or complex scalar.

The non-scalar must be in standard form.* The result has the same form as the non-scalar operand, vector or matrix. If the operands are both integer or both floating point or both complex, the result is of the same type. An integer operand is floated before combination with a floating point operand, and the result is floating point. An integer operand is floated and then made complex before combination with a complex operand, and the result is complex. A floating point operand is made complex before combination

with a complex operand, and the result is complex.

7) Implied multiplication between operands which appear immediately next to one another, not separated by an operation. The same rules apply as for the explicit ×.

8) Exponentiation between integer, real floating point or complex scalar operands.

If either or both operands are complex, the result is complex. If neither operand is complex but either or both operands are floating point, the result is floating point and the base may not have a negative value. If both operands are integers, the result is an integer, zero if the base is > 1 in absolute value and the exponent has a negative value. Note that A^B is typed 'A sup B sub', using the superscript and sub-script keys on the flexowriter. The counter associated with these carriage moving keys should be set to zero before starting a program and <u>must</u> return to zero before the cr which ends each command.

9) Exponentiation of a short logical operand by an integer.

Short logical words are 15-bit configurations whose bits are numbered 1 to 15 from left to right. In particular SL (the sense light register) is in the vocabulary of the compiler and falls into this category. The result of exponentiation of such an operand by an integer, as SL^k, is Boolean, TRUE if bit k of SL is 1 (on) and FALSE if it is 0 (off). The value of the bit addressed is not affected by the operation. The user may also exponentiate a private variable which has been declared BOOLEAN.

10) Exponentiation of a square integer or real floating point matrix to an integer power.

If the matrix is integer it will be floating before exponentiation. The matrix must be in standard form.* The result is always a floating point matrix. If P

is the power and $P<0$, the inverse is computed. If $|P|>0$, multiplication occurs $|P-1|$ times. If $P=0$, the result is the unit matrix.

11) Subscripting of a vector by an integer scalar operand or of a matrix by a pair of integer scalar operands separated by commas.

The result is an element of the vector or matrix and is of the same type (integer, real floating point, complex, or Boolean) as the non-scalar of which it is an element. The expression A_B is typed 'A sub B sup' and return to zero carriage level <u>must</u> be observed as for exponentiation.

12) Any non-scalar may be subscripted with a total of five integer subscripts separated by commas. The operand is indirectly addressed after B1,...,B5 are loaded with the subscripts. An Array may be subscripted at both levels in one expression, e.g. $\ldots(A_{I,J,K})_{L,M}\ldots$, where A in an Array, is a reference to element L,M of the matrix $A_{I,J,K}$. The placement of the parentheses indicates the break point in the structure and the subscripting procedure is restarted with B1. The parentheses are not necessary for the first level, e.g. $\ldots B_{K,L}\ldots$, where B is an Array, is a reference to non-scalar $B_{K,L}$.

13) Relations $=$, $\neq$, $<$, $\nless$, $\leq$, $\nleq$ between integer or real floating point scalar operands.

Combination of integer or floating point operands with a relational operator yields a Boolean result, TRUE if the two operands stand in the specified relation to each other, FALSE otherwise. If the operands are not both integer or both floating point, the integer operand is floated before the comparison is made. If r and r' are relations, the form ArBr'C is tempting but <u>not</u> permitted; an equivalent form is (ArB) $\times$ (Br'C). A precise sequence of typed characters

is required:

$\neq$ is typed ' $=$ backspace uc | '

$\nless$ is typed ' $<$ backspace uc | '

$\nleq$ is typed ' $\leq$ backspace | '

Note that the relations $>$ and $\geq$ are not available, but $>$ is equivalent to $\nleq$ and $\geq$ is equivalent to $\nless$.

14) Unary - applied to an integer, real floating point, or complex scalar operand.

The negation of the operand takes place before it is combined with any other across a binary operation, except exponentiation and subscription.

15) Absolute value of an integer or real floating point scalar operand.

This inflection is denoted by absolute value bars '|' before and after the operand. These bars are simply parentheses that cause the quantity inside to be taken with positive sign.

16) Unary 'not' or - applied to a Boolean scalar operand.

The complementation of the Boolean operand takes place before it is combined with any other across a binary operation, except exponentiation and subscription. If the Boolean scalar has the value TRUE, then not A has the value FALSE; if A has the value FALSE, not A has the value TRUE.

*The standard form for vectors and matrices is that handled by VSPACE, MSPACE, and the Genie input-output commands. Generation and input-output of non-standard forms can only be handled by explicit use of SPIREL facilities.

The form of a simple arithmetic command is illustrated by:

A=arithmetic expression

| cr | 1st tab

The form of a compound arithmetic command is illustrated by:

A=arithmetic expression, B=arithmetic expression, ...

| cr | 1st tab

where more than one equation appears in the command.

If there are no interdependencies among the equations of a command, the equations are coded by Genie in the order given. If there are interdependencies, the first equation will be coded last and preference will be given to coding the remaining equations from right to left; for the second and any following equations, if the ith depends on the jth and $i>j$ (counting from left to right), then the jth equation will be coded before the ith. So the second and following equations may well be used to define subexpressions of the first (or primary) equation, producing code that will run more efficiently and copy that will be more readable. An example in which reordering will take place is

y=a+b, a=5c/d, b=6, c=b+4

| cr | 1st tab

The code generated will evaluate b, then c, then a, then y. On the other hand, the equations in

M=P+Q, a=3, i=j+1

are not dependent upon each other and will be coded in the order given.

The variable on the lefthand side of an equation may be a scalar, or a non-scalar, or a subscripted non-scalar (denoting a scalar element of a vector or matrix). All lefthand side variables in a command must be distinct, no scalar or non-scalar defined more than once. More than one element of the same non-scalar may be defined in one command.

The '=' joining lefthand side to righthand side of an equation causes storage of the computed righthand side into the loca-

tion or array specified on the lefthand side. Compatibility of types is checked for at time of compilation, and an error message is printed out if incompatibility of the two sides is detected. In every case the righthand side dominates and will be stored as calculated, no conversion taking place. If the righthand side is non-scalar, the storage addressed by the codeword on the lefthand side is freed before the store across the '=' takes place.

Genie has the ability to apply the commutative laws of arithmetic to reorder the terms of an expression to provide calculation using a minimum number of temporary stores. In the coding for a non-complex scalar expression, the compiler may use the T-registers of the computer for temporary storage. Push-down storage addressed by index register B6 is also used for this purpose. When profitable, the T-registers are used by the compiler for non-complex scalar variables that are referred to often in an equation. The codeword at machine address 10 (octal) is used in the code generated by the compiler as an accumulator for real vectors and matrices produced in the course of evaluating the righthand side of a non-scalar equation. This address may not be used by a coder. The accumulator for complex non-scalars is named CSTAR. Temporary storage for non-scalars is always on the B6-list.

A simple arithmetic command may be of <u>conditional</u> form, as illustrated by

$A = E_1$ if P_1, E_2 if P_2, ..., E_n if P_n, E_{n+1}

| cr | 1st tab

where each E_i is an arithmetic expression and each P_i is a predicate which is either true or false. The code that is generated will evaluate A as E_i for the least i for which P_i is true. If every P_i is false, then A is evaluated as E_{n+1}. If E_{n+1} is omitted, then A is not evaluated at all if every P_i is false.

Boolean valued expressions are predicates, as in the following examples:

K = 1.0 if B≤C, 2.0 if x<-12.9, 3.0

K = 1.0 if not (SL^n), 3.0

K = 1.0 if SL^5 + not (SL^n)

| cr | 1st tab

Boolean valued expressions joined by the operations 'and' and 'or' form predicates, as in the following example:

K = 1.0 if (B≤C or |C + D| ≠ 3.72) and SL^5 + not (SL^n),
2.0 if x<-12.9, 3.0

| cr | 1st tab | 2nd tab | 3rd tab

The most binding first, the operations are ordered as follows:

arithmetic operations
'and'
'or'

Parentheses may be used, as in the above example, to dictate computational order.

The predicate form F_1 r F_2 r' F_3 is tempting but <u>not</u> permitted. An equivalent permissible form is

F_1 r F_2 and F_2 r' F_3

or (F_1 r F_2) × (F_2 r' F_3)

Two exceptional Boolean predicates are 'EOV', asking if the exponent overflow light is on, and its negation 'NEO'; neither of these may be inflected by 'not'. Both of these tests turn the

light in the indicator register off.

A conditional arithmetic equation must stand alone as a command. It may not be grouped with other equations in a compound arithmetic command.

Code is generated so that the commands of the program are normally executed in the order written. An explicit variation in this order is indicated by a transfer command, illustrated by

CC = #LOOP or GO TO LOOP

| cr | 1st tab

Here 'CC' is the mnemonic for the control counter which is normally stepped sequentially through the orders of the code. 'LOOP' is a label on a command of the program, the command to which control will be passed by this transfer command. Note that 'END' is a label in every program and may be transferred to for exit from the program. The inflection '#' is required in this context to indicate that the address corresponding to LOOP, and not the contents of the location whose address is LOOP, is to be calculated on the righthand side. The '#' inflection is analagous to the 'a' bit in APl.

The conditional transfer command provides variation in the order of command execution depending upon the truth values of predicates. The form of this type of control command is shown by

CC = $\#A_1$ if P_1, $\#A_2$ if P_2, ..., $\#A_n$ if P_n, $\#A_{n+1}$ or

| cr | 1st tab GO TO A_1 if ...etc.

where the A_i are labels within the program and the P_i are predicates. The code generated causes CC to be evaluated as the first $\#A_i$ for which P_i is true. If no P_i, for i=1, 2, ..., n, is true, CC is evaluated as $\#A_{n+1}$. The term $\#A_{n+1}$ may be omitted from the command, in which case CC is unchanged if all P_i are false, so that no transfer is made. The predicates P_i are of the form described in the section on conditional arithmetic commands.

Loops may be realized in Genie language by a combination of arithmetic commands and transfer control commands. A concise notation for a popular loop structure is provided by the loop control commands. The commands of a loop are parenthesized by the FOR and REPEAT commands of the form

```
          FOR P=A , B , C
               commands of the loop
          REPEAT
| cr      | 1st tab
```

The elements of the FOR command are

parameter of the iteration, P
initial value, A
increment, B
final value, C

All elements must be scalars, either integer or floating point. In execution, the loop is traversed for $P = A + kB$, for all $k = 0,1,2,...$ such that

$$P \leq C \text{ if } B > 0$$
$$P \geq C \text{ if } B < 0$$

The element P must be given as a simple variable name. The elements A, B, and C may be given as constants or arithmetic expressions of integer or floating point type. Only if B and C are given as simple variable names may their values change during execution of the loop. Otherwise, B and C retain their values on entry to the loop throughout the execution of the loop. For example, in the loop

```
          FOR COUNT = FIRST, M+N, LAST
              :
          N=A+B
              :
          REPEAT
```

the increment value will remain constant, as computed on entry to the loop.

In the REPEAT command, 'REPEAT' is followed immediately by a

'cr'. A REPEAT must be written for every FOR.

If addressed from outside the loop, the iteration parameter has the value it had upon exit from the loop.

Loops may be nested to any level, but distinct iteration parameters must be used at each level within a nest. The 'REPEAT' is considered to be within the loop which it terminates; the 'FOR' is not. Transfer of control may be made from a command within a loop to another command within the loop or to a command outside the loop. Transfer from outside a loop to the FOR command is permitted, but transfer from outside a loop to a command within a loop is not permitted.

Any 'FOR' or 'REPEAT' may be labelled for purpose of transfer to it. The compiler generates the label '←FORn' on each FOR command and '←RPTn' on the corresponding REPEAT command, n = 1, 2, ..., 9, a, b, ... in each program. A coder's label will be used instead if it appears. Thus, FOR and REPEAT commands begin command sequences whether or not they are labelled by the coder.

The machine index registers B3, B4, B5 may be used as iteration parameters in loops and will cause significantly more efficient code to be generated, especially when a constant increment = ± 1 is specified. The section on fast registers discusses coder usage of machine registers.

Before a vector or matrix is referred to dynamically in a program it must be created, either initially from paper tape or dynamically while running.

In a Genie program, to create, or take space for, the vector named VNAME of length NELTS elements the following command is used:

```
          EXECUTE VSPACE(VNAME, NELTS)
| cr      | 1st tab
```

The vector VNAME contains zeroes initially. To create, or take space for, the matrix named MNAME of NROWS rows and NCOLS columns the following command is used:

```
          EXECUTE MSPACE(MNAME, NROWS, NCOLS)
| cr      | 1st tab
```

The matrix MNAME contains zeroes initially. The dimension arguments in both commands are integers.

The dimension arguments may be computed dynamically, so that sizes of vectors and matrices may vary from run to run. In fact, the dimension of an array may vary during a run by use of a creation command to 'recreate' an array which already exists; the old copy is automatically erased before the new one is formed.

To explicitly erase, or free the space occupied by, a vector or matrix named ARRAY on which the calculation no longer depends the following command is used:

```
          ERASE ARRAY
| cr      | 1st tab
```

Also a single ERASE command may be applied to more than one non-scalar, as illustrated by:

```
          ERASE VNAME, MNAME, ARRAY
| cr      | 1st tab
```

The erasure of a vector or matrix causes the storage occupied to be returned to a common pool, that from which storage is obtained for the creation of vectors and matrices. This pool is managed by STEX, the storage exchange program in SPIREL (explained in detail

in the literature on SPIREL), and it is called the STEX domain. STEX may move items within its domain to concentrate space if necessary to satisfy requests for space.

The command

EXECUTE PROG(PARAM)

| cr | 1st tab

causes control to be transferred to the program whose name is denoted by 'PROG' in this illustration. 'PROG' must have been declared as a function outside the command sequence for the calculation. 'PARAM' denotes a list of one or more parameters separated by commas. Parameters may be arithmetic expressions unless they designate quantities which are to be calculated by the function, in which case they must be simple variable names. Control is returned from PROG to the next command in the sequence. The interpretation given to the EXECUTE command by Genie is parallel to that for the arithmetic command, the information to the right of the space after the EXECUTE corresponding to that after the first '=' in an arithmetic command. Thus, a <u>simple conditional EXECUTE command</u> is allowed, such as

EXECUTE A(P) if a < b + c, B(Q)

| cr | 1st tab

And a <u>compound unconditional EXECUTE command</u> is allowed, such as

EXECUTE SUM(x,y), x = 2a/b, y = ab, b = 4

| cr | 1st tab

The input-output commands are:

DATA list	READ list	†PAGE list
PRINT list	INPUT list	ACCEPT list
PUNCH list	OUTPUT list	TITLE string
DPUNCH list	DISPLAY list	

| cr | lst tab

where 'list' denotes a collection of names separated by commas. Any name may be that of a scalar, other than fast registers, or of a standard vector or matrix or of a function. Expressions may not appear in the argument list, so vector and matrix elements in the subscript notation may not be designated.

The DATA command provides reading of manually punched signed decimal numbers from paper tape. The name of any type of variable may appear in the list, and any name may have been assigned a machine address in a LET statement. When the paper tape is read, if a decimal point appears the number will be converted to floating point within the machine; the absence of a decimal point causes conversion to integer form. Every number on the tape must be followed by a carriage return, tab, or comma. Integers greater than or equal to 2^{14} in absolute value are meaningless; floating point significance to more than 14 places is not meaningful. A floating point number may be followed by the sequence '* signed integer' which will cause it to be multiplied by 10 to the signed integer power upon conversion. The magnitude of such numbers must be greater than 10^{-70} but less than 10^{70}. The absence of a sign on a number implies positive sign. Then

punched	328cr	converts to	integer 328
	46.9cr		floating point 46.9
	.469*2cr		floating point 46.9
	-5391cr		integer -5391
	-69.*-1cr		floating point -6.9

†Blank or numbers 1 through 7 only

Integers and real floating point scalars are punched as single decimal numbers in the appropriate format; complex scalars are punched as real part followed by imaginary part, both floating point. A vector of length n is punched as the sequence of n+1 decimal numbers: integer n, first element, ..., n^{th} element. A matrix of m rows by n columns is punched as the sequence of mn+2 numbers: integer m, integer n, element (1,1), element (1,2), ..., element (1,n), element (2,1), ..., element (2,n), ..., element (m,1), ..., element (m,n). When the DATA command is executed, the proper tape is assumed to be in the reader. If sense light 14 is off, the line

```
          DATA  NAME
| cr      | 1st tab
```

will be printed out for each quantity read, where 'NAME' is as designated in the program containing the READ command. Thus, printer monitoring of DATA applied to parameters bears the dummy parameter name, not the name of the argument supplied as the parameter.

The <u>PRINT command</u> provides decimal output on the fast line printer of any named scalar or non-scalar quantities. These are labelled by the name given in the argument list. Any name may have been assigned a machine address in a LET statement. Scalars are printed four per line. Vectors are printed five elements per line, the leading element index in octal at the left of each line. Matrices are printed by row, five elements per line, the leading column index in octal at the left of each line. Complex variables are printed as real part followed by imaginary part; the name of the variable will be given with the real part, and "ditto" (printed '←←←←←') will label the imaginary part.

The <u>PUNCH command</u> and the <u>READ command</u> may be applied only to external variables and to parameters representing arguments which

Formatted printer output may be obtained by use of the command

```
          EXECUTE  SCRIBE(A1,...,AK,F)
| cr      | 1st tab
```

where A1,...,AK is the list of arguments to be printed, and F is the name of a FORMAT statement to be used. Any argument in A1,...,AK may be a simple name or an expression. The program SCRIBE is in the library, and its use is fully described in the library literature. A FORMAT statement gives text which will be printed directly by SCRIBE and dummy variables which will be replaced by argument values.

Page control and headings are provided with formatted printer output by use of the command

```
          EXECUTE  PRESCRIBE(A1,...,AK,F,N,NAME,LIMIT)
| cr      | 1st tab
```

where A1,...,AK,F are just as for SCRIBE, N is the number of blank lines after SCRIBE output, NAME is the name of a FORMAT statement containing pure text or a vector of BCD data to be used in the heading on each page, and LIMIT is the number of lines per page of output. The program PRESCRIBE is in the library, and its use is fully described in the library literature.

Additional forms of input and output may be obtained by use of SPIREL programs directly, but those provided by the input-output commands should be sufficient for a large number of problems. Also see the TITLE and PAGE commands on p.5.

The DPUNCH command may be applied only to external variables as explained earlier in the PUNCH command. DPUNCH provides standard decimally formatted punched tape to be later read by a DATA command only. Mixed integer and real data may be punched from scalars, vectors or matrices.

The TITLE command allows the printing of a string of literal symbols for labeling pages like SCRIBE only with greater ease. Two examples are given below. One would write:

```
          TITLE    PRINT ONE LINE HERE
          TITLE
PRINT ANOTHER LINE HERE ALSO
| cr     | 1st tab
```

The above would cause the following to be printed:

```
PRINT ONE LINE HERE
PRINT ANOTHER LINE HERE ALSO
↑
(first printer position)
```

The PAGE command allows the page to be moved to any position or by any amount easily. The 'list' consists of the integers 1,2,...,7 or no list, i.e., blank. The interpretation of the integers is given by the table below:

integer	→	move	to	next ____	page	
1	→	move	to	next 1/66	page	(one space)
2	→	move	to	next 1/22	page	
3	→	move	to	next 1/11	page	
4	→	move	to	next 1/6	page	
5	→	move	to	next 1/3	page	
6	→	move	to	next 1/2	page	
7	→	move	to	full	page	(page restore)

If the list is blank, the page is restored.

The ACCEPT command provides reading of data input through the console typewriter. The name of any type variable may be included in the list. Data may be entered when the blue light on the typewriter comes on; each line is processed before another may be typed. Decimal numbers are handled as in the DATA command; octal numbers must be preceded by a + sign. T... or F... is typed for the Boolean values. All values must be separated by commas and a line is terminated by a carriage return. For a vector or matrix of size n or nxm, n or nxm values must be typed. To change the size of a non-scalar, the new dimension(s) is enclosed by parentheses (n) or (n,m), and followed by the values to be stored. A matrix is typed by rows (as it is read in DATA). For example, where A is a vector 3 long, B is Boolean, and C is scalar:

ACCEPT A,B,C

typewriter input: -234.0, 8.34*4, .62023, T, +0142000000 cr

stores the first 3 values in A, -Z in B, and the octal number in C.

typewriter input: (4), 8.0, 9.0, -10.0, 11.0, FALSE, 345 cr

erases array A and creates a new one of length 4, stores the next 4 values in A, -1 in B, and decimal value 345 in C.

typewriter input: →, T, +002345 cr

leaves A as it is, stores -Z for B, and the octal value for C. The "forward arrow" is inserted whenever an item in the list is to remain unchanged. If sense light 14 is off, the line

ACCEPT NAME

will be printed out for each quantity in the list (as in the DATA command).

The SET command provides program control over sense light setting. It is illustrated by

SET not SL^5, SL^9, SL^1, not SL^{15}

|cr |1st tab

Any number of sense lights may be set. The notation 'SL^i' causes SL^i to be turned on; 'not SL^i' causes SL^i to be turned off. In 'SL^i' i must be numeric and may range from 1 to 15. The lights are set in the order mentioned.

Data commands cause generation of words in the program which are not instructions. These commands are not executable and all but FORMAT must be transferred around.

Alphabetic information for output on the printer may be defined by the BCD command, as illustrated by

```
MESS1     BCD _ _TEMPUS_FUGIT
```

or

```
MESS1     BCD
_ _TEMPUS_FUGIT
| cr      | 1st tab
```

where 'BCD' is followed immediately by a single space or a 'cr' which is not part of the data, and _ indicates a typed space. The command may continue onto succeeding lines at the 3rd tab position by use of the 'cr tab tab tab' sequence. A space is inserted by Genie between the last character of one line and the first of the next line. At the place such a BCD command appears in the command sequence for the program, the printer code for the information is inserted in the code for the program, nine characters per word. The label (if any) on the BCD command is associated with the first word of data.

A block of numeric data may be defined by the NUMBERS command, as illustrated by

```
CONST     NUMBERS 36.5, -2*8, 6, +774777
| cr      | 1st tab
```

In the program Genie generates, in this case,

floating point 36.5 at CONST

floating point -2.0×10^8 at CONST+1

integer 6 at CONST+2

octal 774777 (right-adjusted) at CONST+3

One or more real numbers (each but the last followed by a comma) are listed; complex numbers may not appear in the list. The list may be extended onto the succeeding lines by use of the 'cr tab tab tab' sequence. The numbers are inserted into the program in the order

given, one per word. The label (if any) on the NUMBERS command is associated with the first word of data.

Formats for the printer output programs SCRIBE and PRESCRIBE are defined by the FORMAT command, as illustrated by

```
LINE      FORMAT ddd   ITERATIONS,   CASE aa,   K=bb,   T=-d.ddce+d
or
LINE      FORMAT
ddd   ITERATIONS,   CASE aa,   K=bb,   T=-d.ddce+d
| cr      | 1st tab
```

where 'FORMAT' is followed immediately by a single space or a 'cr' which is not part of the data. The label on the FORMAT command is the name of the FORMAT which is an argument to the output programs. The format data is a "dummy line" of printer output; lower case letters and the characters '., +, -' with 'd' form dummy variables for which argument values are substituted when printing; the rest of the format data is text which is printed directly. SCRIBE and PRESCRIBE are programs in the library; their use and the details of format specification are explained fully in the library literature.

It is never necessary to use machine registers in the Genie language. But their use is permitted, with certain restrictions and with effect that more efficient code may be obtained.

T7 should never be used in the Genie language.

T6, T5, and T4 may be used as the names of scalar variables within a command. The compiler will not make use of any T-register mentioned by the coder, and code efficiency may be increased by explicit assignment of auxiliary variables to these fast registers. The values in T6, T5, T4 are not preserved by Genie from one command to another as they are subject to use by the compiler in any command in which they are not explicitly mentioned by the user.

The index registers B3, B4, B5 may be used as the names of scalar integers. These are disturbed by Genie-generated code only to address elements of arrays of more than two dimensions. (Non-standard subscripting is discussed in the section on arithmetic expressions.) Efficiency of code is gained if these registers are used as subscripts or as iteration parameters of loops with constant increment ±1. The index registers B1 and B2 may be used only if the user understands Genie coding conventions are explained in another section and can accurately anticipate the use of these registers by Genie generated code. The registers B6 and PF may not by used in Genie language.

In a Genie program, instructions in the AP2 assembly language may be interspersed at will with commands in the Genie language. AP2 is discussed in detail in the assembly language literature.

The following names identify fast registers in both Genie language and AP2:

```
     T4      T6      CC      B2      B4
     T5      T7      B1      B3      B5
```

The following names identify private quantities in Genie language and fast registers in AP2

```
             R       B6      U       I
             S       PF      X
```

Therefore, a private name I in Genie language may not be addressed in AP2 code.

Operations without mnemonics in the AP2 vocabulary may be coded in octal, as

```
                         +45061     #15
| cr        | 1st tab | 2nd tab | 3rd tab
```

Or an operation code mnemonic may be assigned with a LET statement, as

```
          LET #QSR = +45061
```

Then the instruction

```
                         QSR        #15
```

could be used instead.

In AP2 commands, the coder may make use of the fast registers, taking care to preserve the value of PF for reference to parameters and to use B6 for temporary push-down storage only. Entire functions may be written in the assembly language, but the user must first understand various Genie coding conventions, as discussed in a later section.

Normally for a Genie program initial and terminal program sequences and code to preserve parameter addressing are automatically generated by the compiler. For some programs coded predominantly in AP2, it may be desirable to avoid generation of or-

ders not explicitly coded. This may be accomplished by using 'ORG' in place of 'SEQ', as

```
PROG(PARAM).= ORG
| cr
```

to start the command sequence for the program. The first instruction of the program will be the first explicitly coded. The only words in the program generated automatically by the compiler are cross-references to external quantities and a one-word END program sequence:

```
END        TRA        Z
```

The programmer must code parameter set-up for the program, maintain PF and B6 by Genie coding conventions.

PUNCTUATION

Reference to rules of punctuation for use in the punching of Genie programs has been made in other sections. A few generalities and notes here may help the user to avoid some of the most common mistakes.

Every tape must begin with a 'cr' punch and a case punch for proper interpretation.

Every line should begin with a case punch so that it does not depend on the case at termination of the preceding line, and editing of tapes will be thus simplified.

Spaces may appear anywhere but within a name; they will be ignored.

Backspaces are ignored except within the sequence of punches for negated relations.

The superscript and subscript punches should be used only where meaningful; the sequences 'sup sub' and 'sub sup' are not equivalent to no punch at all and will not be accepted by the compiler.

The carriage counter should be set to zero before typing a program and must return to zero before the 'cr' which ends each statement.

A statement is continued onto second and succeeding lines by the sequence of punches 'cr tab tab tab'.

The operation '.=' must be punched as just those two characters in succession.

The negated relations require specific sequences of punches for proper interpretation:

$\neq$ is punched '= backspace uc |'

$\nless$ is punched '< backspace uc |'

$\nleq$ is punched '≤ backspace |'

The operations 'not', 'and', 'or', 'if' are punched in lower case and must contain no superfluous punches. All other "words" in the vocabulary of the compiler are punched fully in upper case letters.

Statement labels, the program name, function definitions 'END', and 'LEAVE' are typed at the margin; alternatively, program names and function definitions may be typed at the 1st tab position.

Since 'SEQ', 'END', and 'DEFINE' end statements, they must be followed immediately by a 'cr' punch.

Declaration identifiers, 'DATA', 'EXECUTE', 'FOR', 'LET', 'NUMBERS', 'PRINT', 'PUNCH', 'DPUNCH', 'READ', 'SET' may be followed by either a space or a tab punch.

'BCD', 'FORMAT', 'REM' may be followed by a space, a tab, or a carriage return punch.

For compilation to be terminated properly 'LEAVE' must be followed immediately by two 'cr' punches.

A Genie program is compiled by exercising option #6 in the PLACER system.

Compilation output on the printer consists or error messages, program listing, and symbol tables. These are discussed below. Compilation provides a punched paper tape to be loaded under SPIREL control. Compilation options are also discussed below.

Error messages. Genie error messages refer to carriage return number on the PLACER listing of the program. During compilation the carriage return number for the line being compiled is displayed in FT (the from-tape register). This can be useful if compilation problems arise with no error message. If a single command, statement, or instruction is continued onto more than one line, the carriage return number for the last line will pertain throughout.

Program listing. Four columns are printed, giving:

(a) The symbolic location (if any).

(b) The relative location of the word in the program, in octal.

(c) The instruction in octal, broken into fields, with tag.

(d) The symbolic address (if any).

Cross reference words and internal storage are listed after the instructions of the program, one per word with name, relative location, and content for each. The variables referenced relative to PF are then listed with name and PF increment.

Symbol tables. For each program a symbol table of internal names is printed. Of interest are columns which give the name and the relative location in the program (two to the right of the name). The column to the right of name contains descriptive information about the variable, by digits:

first - type 1, Real (floating point)
2, Integer
4, Boolean
5, Complex

second - shape and mode 0, Scalar
1, Scalar function
2, Vector
3, Vector function
4, Matrix
5, Matrix function
6, Array
7, Array function

third - 0, not a parameter
1, non-scalar parameter
2, scalar parameter

After the internal symbol table a list of programs used is given. If a program is in the library, its name is prefixed by 'GENIE'. If a name is used, this is given. If a number is used, this is given.

For each definition set a table of external names is printed in which only the names and descriptive information (as above) are of interest.

<u>Compilation options</u>. See PLACER-TRANSLATE.

RUNNING GENIE PROGRAMS

The usual procedure is to run Genie programs with SPIREL so that all library routines are immediately available.

The initial version of a program should contain liberal output of intermediate quantities. These may be conditioned on sense light settings or edited out once the program is running.

Initial runs should be made with SL14 off so that printer monitoring is provided for all SPIREL operations.

Debugging may be facilitated by a SPIREL dump of the positive portion of the Symbol Table-Value Table. This will show all named external items in the system being run, the values of scalars, and the codewords for non-scalars.

A SPIREL dump of a private program will show values of internal variables.

Arithmetic error tracing may help to locate mathematical problems.

All instructions generated by the compiler may be traced, but this is not a recommended procedure.

- Least Squares

This program computes the coefficients of a polynomial of specified degree which best fits the input data in the least squares sense. The basic method is described in "An Introduction to Numerical Mathematics", Stiefel, E.L., 1963, page 51. The only difference here being the introduction of weighting factors to the data and the use throughout of matrix algebra.

Lines 6 to 13:

Internal integers are declared and then stored into; the number of rows of XDATA and the length of COEFS (the number of coefficients is compared.

Lines 14 to 45:

The size of XDATA is expanded and is filled with the appropriate powers of X.

Lines 46 to 55:

The normal matrix is computed taking the weights into account.

Lines 56 to 67:

The coefficients, theoretical polynomial values, residues, sum of the squares, and the covariance matrix are computed.

Lines	Comments
4	Some of the parameters, the non-scalars, are declared.
6	Notice lower case alphabetic print output for characters beyond 'f'.
14-40	This AP2 code constructs control words for SPIREL to act upon; notice the labelled instruction at line 35.
41-45	Double or nested looping.
47	A matrix transpose is done here.
56-57	Non-scalar multiplications.

Lines	Comments
60	Solution of a system of equations.
63	Line is labelled but not referred to.
67	Use of matrix exponentiation to compute inverse.

```
                                                                      1
          DEFINE                                                      2
          MATRIX XDATA, SIGMA                                         3
          VECTOR YDATA, COFFS, YCALC, RESID, WGHTS                    4
PFIT(XDATA, YDATA, WGHTS, COFFS, YCALC, RESID, SQSUM, SIGMA).=SEQ
          INTEGER .N, .M, .H, .J, .I                                  6
          .N = LENGTH(YDATA)                                          7
          .M = LENGTH(COEFS)                                          10
          .H = ROW (XDATA)                                            11
          .J = .M - .H                                                12
          CC = + WATE .IF .J = 0                                      13
          7         BAJ             XDATA, U+B1                       14
                    LRS             27                                15
                    CLA             ++1150                            16
                    LRS             12                                17
                    CLA             .J, U+B2                          20
                    LRS             15, R+T7                          21
                    TSR             *+126                             22
                    SPF             *END+1                            23
                    CLA             B1, U+B4                          24
                    CRL             15, R+B3                          25
          -B2       ADD             +34+B3+1, U+B3                    26
          B3        BNA             B4+1, U+R                         27
                    CRL             15                                30
                    LRL             12                                31
                    CLA             ++0120                            32
                    LRS             12                                33
          R         CRR             15, U+T6                          34
LOOP      B3        RPA             T6, B2=1                          35
          T6        TSR             *+126, U+T7                       36
                    SPF             *END+1                            37
          B2        IF(NZE)TRA      +LOOP, B3+1                       40
          FOR     .J = 1, 1, .N                                       41
          FOR     .I = .H+1, 1, .M                                    42
          XDATA     = XDATA       xXDATA                              43
               .I, .J        .I-1, .J     .H, .J
          REPEAT                                                      44
          REPEAT                                                      45
WATE      EXECUTE VSPACE(RESID, .N)                                   46
          SIGMA = TRAN(XDATA)                                         47
          FOR .I = 1, 1, .N                                           50
          RESID   = WGHTS   x YDATA                                   51
               .I        .I        .I
          FOR  .J = 1, 1, .M                                          52
          SIGMA      = SIGMA      x WGHTS                             53
               .I, .J       .I, .J       .I
          REPEAT                                                      54
          REPEAT                                                      55
WORK      COEFS = XDATA x RESID                                       56
          SIGMA = XDATA x SIGMA                                       57
          COEFS = SOLN(SIGMA, COEFS)                                  60
```

```
               YCALC = TRAN(XDATA) × COEFS                                  61
               RESID = YDATA - YCALC                                        62
SUMM           SQSUM = 0.0                                                  63
               FOR    .I = 1, 1, .N                                         64
                                                 2
               SQSUM = SQSUM + WGHTS   + RESID                              65
                                    .I        .I
               REPEAT                                                       66
                            -1
               SIGMA = SIGMA   × (SQSUM/(.N-.M))                            67
END                                                                         70
               DEFINE                                                       71
LEAVE                                                                       72
                                                                            73
```

```
PFIT     START NEW PROGRAM                              4/06/66 14.18

*BGIN    PROGRAM SEQUENCE
LOOP     PROGRAM SEQUENCE
*FOR1    PROGRAM SEQUENCE
*FOR2    PROGRAM SEQUENCE
*RPT2    PROGRAM SEQUENCE
*RPT1    PROGRAM SEQUENCE
WATE     PROGRAM SEQUENCE
*FOR3    PROGRAM SEQUENCE
*FOR4    PROGRAM SEQUENCE
*RPT4    PROGRAM SEQUENCE
*RPT3    PROGRAM SEQUENCE
WORK     PROGRAM SEQUENCE
SUMM     PROGRAM SEQUENCE
*FOR5    PROGRAM SEQUENCE
*RPT5    PROGRAM SEQUENCE
END      PROGRAM SEQUENCE
```

```
PFIT  ,=
              *BGIN      1    10 01000 02 4400 00136

              LOOP      51    43 21601 62 0000 00006

              *FOR1     55    20 20001 00 4001 00236    ,J

              *FOR2     61    20 10000 00 0001 00231    ,H

              *RPT2    103    20 10401 00 0001 00211    ,I

              *RPT1    105    20 10401 00 0001 00206    ,J

              WATE     107    01 21702 26 0200 00005    RESID

              *FOR3    124    20 20001 00 4001 00170    ,I

              *FOR4    137    20 20001 00 4001 00154    ,J

              *RPT4    155    20 10401 00 0001 00136    ,J

              *RPT3    157    20 10401 00 0001 00135    ,I

              WORK     161    01 21700 41 0200 00000    XDATA

              SUMM     245    00 20001 00 4600 00006    SQSUM

              *FOR5    246    20 20001 00 4001 00046    ,I

              *RPT5    262    20 10401 00 0001 00032    ,I

              END      306    01 01000 00 4400 00137
                       307    01 40006 00 4000 00000
                       310    07 01000 00 4200 00000
REFFERENCE WORDS...
              SMMPY 77770     6254545770400000000
              MPOWE 77771     5457566644400000000
              MSUB  77772     5462644125400000000
              SOLN  77773     6254535525400000000
              MMPY  77774     5454577025400000000
              TRAN  77775     6361405525400000000
              VSPAC 77776     6562574042400000000
              LENGT 77777     5344554663400000000
INTERNAL STORAGE..
```

```
        .N          311                   0
        .M          312                   0
        .H          313                   0
        .J          314                   0
        .I          315                   0
        -TW47       316   6200000000000000

PARAMETERS AT PF +
        XDATA             0
        YDATA             1
        WGHTS             2
        COEFS             3
        YCALC             4
        RESID             5
        SQSUM             6
        SIGMA             7



   SUBROUTINES REFFRENCED
                                      137
  GENIF...        SMMPY
  GENIF...        MPOWE
  GENIF...        MSUB
  GENIF...        SOLN
  GENIF...        MMPY
                                      135
  GENIF...        TRAN
  GENIF...        VSPAC
                                      126
  GENIF...        LENGT
                                      136
```

```
PFIT                ORG                                     1
                    REM       BACK-TRANSLATION              2
L77770              REF       *SMMPY                        3
L77771              REF       *MPOWE                        4
L77772              REF       *MSUB                         5
L77773              REF       *SOLN                         6
L77774              REF       *MMPY                         7
L77775              REF       *TRAN                         10
L77776              REF       *VSPAL                        11
L77777              REF       *LENGT                        12
                                                            13
L1         -Z       TRA       *136,U+R                      14
L2                  SPF       B5-22                         15
           PF       RWT       L307                          16
                    CLA       PF+1,U+T7                     17
                    TSR       *L77777                       20
                    SPF       *L307                         21
                    STO       L311                          22
                    CLA       PF+3,U+T7                     23
                    TSR       *L77777                       24
                    SPF       *L307                         25
                    STO       L312                          26
                    CLA       PF,U+T7                       27
                    TSR       *L77777                       30
                    SPF       *L307                         31
                    STO       L313                          32
                    21740     L312                          33
                    SUB       L313                          34
                    STO       L314                          35
                    21740     L314                          36
                    IF(ZER)SKP a3                           37
                    TRA       L30                           40
                    CLA       aL107                         41
                    NOP       Z,U+CC                        42
L30        Z        BAU       PF,U+B1                       43
                    LRS       33                            44
                    CLA       a1150                         45
                    LRS       14                            46
                    CLA       L314,U+B2                     47
                    LRS       17,R+T7                       50
                    TSR       *126                          51
                    SPF       *L307                         52
                    CLA       B1,U+B4                       53
                    CRL       17,R+B3                       54
           -B2      ADD       a33+B4+1,U+B3                 55
           B3       00204     B4+1,U+R                      56
                    CRL       17                            57
                    LRL       14                            60
                    CLA       a120                          61
                    LRS       14                            62
           P        CRR       17,U+T6                       63
L51        B3       RPA       T6,B2-1                       64
           T6       TSR       *126,U+T7                     65
                    SPF       *L307                         66
           B2       IF(NZE)TRA L51,B2+1                     67
           I        STO       L314                          70
L56                 CLA       L311                          71
                    IF(POS)SKP L314                         72
                    TRA       L107                          73
```

```
              I     ADD           L313               74
                    STO           L315               75
L63                 CLA           L312               76
                    IF(POS)SKP    L315               77
                    TRA           L105              100
                    CLA           L314,U→T6         101
              I     BUS+2         L315,B6+1         102
              T6    NOP           Z,U→B2            103
                    21740         B6-1,U→B1         104
                    21740         *PF,B6=1          105
                    NOP           Z,U→T4            106
              T6    NOP           Z,U→B2            107
                    21740         L313,U→B1         110
                    21740         *PF               111
                    10620         T4,U→R            112
              T6    NOP           Z,U→B2            113
                    CLA           L315,U→B1         114
              R     STO           *PF               115
              I     FAD+          L315              116
                    TRA           L63               117
L105          I     FAD+          L314              120
                    TRA           L56               121
L107                CLA+2         PF+5,B6+1         122
              Z     BAU+2         @L311,B6+1        123
                    TSR           *L77776           124
                    SPF           *L307             125
                    CLA           PF,U→T7           126
                    TSR           *L77775           127
                    SPF           *L307             130
                    CLA           PF+7,U→B1         131
              Z     TSR           *135,U→B2         132
                    SPF           *L307             133
              Z     LDR+          10,R→B2           134
              R     STO           B1                135
              B1    RWT           B2                136
              I     STO           L315              137
L125                CLA           L311              140
                    IF(POS)SKP    L315              141
                    TRA           L161              142
                    21740         L315,U→B1         143
                    21740         *PF+2,U→T4        144
                    21740         L315,U→B1         145
                    21740         *PF+1             146
                    10620         T4,U→R            147
                    CLA           L315,U→B1         150
              R     STO           *PF+5             151
              I     STO           L314              152
L143                CLA           L312              153
                    IF(POS)SKP    L314              154
                    TRA           L157              155
                    CLA           L315,U→T6         156
                    CLA           L314,U→B2         157
                    21740         T6,U→B1           160
                    21740         *PF+7,U→T4        161
                    21740         T6,U→B1           162
                    21740         *PF+2             163
                    10620         T4,U→R            164
                    CLA           L314,U→B2         165
              T6    NOP           Z,U→B1            166
```

```
        F       STO      *PF+7              167
        I       FAD-     L314               170
                TRA      L140               171
L157    I       FAD-     L315               172
                TRA      L125               173
L161            CLA      PF,U-B1            174
                CLA      PF+5,U-B2          175
                TSR      *L77774            176
                SPF      *L307              177
                CLA      PF+3,U-B1          200
        Z       TSR      *135,U-B2          201
                SPF      *L307              202
        Z       LDR-     10,R-B2            203
        F       STO      B1                 204
        F1      RWT      B2                 205
                CLA      PF,U-B1            206
                CLA      PF+7,U-B2          207
                TSR      *L77774            210
                SPF      *L307              211
                CLA      PF+7,U-B1          212
        Z       TSR      *135,U-B2          213
                SPF      *L307              214
        Z       LDR-     10,R-B2            215
        F       STO      B1                 216
        F1      RWT      B2                 217
                CLA+2    PF+7,B6+1          220
                CLA+2    PF+3,B6+1          221
                TSR      *L77773            222
                SPF      *L307              223
                CLA      PF+3,U-B1          224
        Z       TSR      *135,U-B2          225
                SPF      *L307              226
        Z       LDR-     10,R-B2            227
        F       STO      B1                 230
        F1      RWT      B2                 231
                CLA      PF,U-T7            232
                TSR      *L77775            233
                SPF      *L307              234
                CLA      PF+3,U-B2          235
        Z       TSR      *L77774,U-B1       236
                SPF      *L307              237
                CLA      PF+4,U-B1          240
        Z       TSR      *135,U-B2          241
                SPF      *L307              242
        Z       LDR-     10,R-B2            243
        F       STO      B1                 244
        F1      RWT      B2                 245
                CLA      PF+1,U-B1          246
                CLA      PF+4,U-B2          247
                TSR      *L77772            250
                SPF      *L307              251
                CLA      PF+5,U-B1          252
        Z       TSR      *135,U-B2          253
                SPF      *L307              254
        Z       LDR-     10,R-B2            255
        F       STO      B1                 256
        F1      RWT      B2                 257
        Z       STO      *PF+6              260
        I       STO      L315               261
```

```
L247               CLA           L311                     262
                   IF(POS)SKP    L315                     263
                   TRA           L264                     264
                   CLA           L315,U+T6                265
                   21740         T6,U+B1                  266
                   21740         *PF+5                    267
                   FMP           U,J+T4                   270
                   21740         T6,U+B1                  271
                   21740         *PF+2                    272
                   FAD           T4                       273
                   FAD+          *PF+6                    274
          I        FAD+          L315                     275
                   TRA           L247                     276
L264               21740         L311                     277
                   SUB           L312                     300
                   53100         -U                       301
                   FMP           L316                     302
                   VDF           *PF+6,U+T4               303
                   LDR           -31                      304
                   CLA           PF+7                     305
                   TSR           *L77771                  306
                   SPF           *L307                    307
                   SB1           10                       310
          T4       TSR           *L77770                  311
                   SPF           *L307                    312
                   CLA           PF+7,U+B1                313
          Z        TSR           *135,U+B2                314
                   SPF           *L307                    315
          Z        LDR+          10,R+B2                  316
          P        STO           B1                       317
          B1       RWT           B2                       320
                   TRA           *137                     321
L307               SB6           Z                        322
          T7       TRA           PF                       323
L311               OCT           00000000000000000        324
L312               OCT           00000000000000000        325
L313               OCT           00000000000000000        326
L314               OCT           00000000000000000        327
L315               OCT           00000000000000000        330
L316               OCT           06200000000000000        331
                   END                                    332
                                                          333
                                                          334
```

• Numerical Integration

This example is adapted from Schwarz (An Introduction to ALGOL 60. Comm ACM 5:82-95 (1962)). It concerns the numerical integration of a differential equation of second order with given initial values. Schwarz chose the method of Adams' extrapolation, which consists of the following formulae:

$$y(x+h)=y(x)+hy'(x)+h^2[\tfrac{1}{2}y''(x)+\tfrac{1}{6}\nabla y''(x)+\tfrac{1}{8}\nabla^2 y''(x)+...]$$

$$y'(x+h)=y'(x)+h[y''(x)+\tfrac{1}{2}\nabla y''(x)+\tfrac{5}{12}\nabla^2 y''(x)+...]$$

where the $\nabla^k y''(x)$ are the backward differences of y'' at the point x and for the interval h. In contrast to other proposals, he starts the integration by an iterative process (lines 62 to 74) which uses the same formulae as the forward integration (lines 76 to 123).

The example consists of three separate programs: EXAMPLE3, a control program to handle input and output and execute the integration program; F, the function being integrated; and ADAMS, the numerical integration routine. EXAMPLE3 activates STEX and initiates output with a page restore and heading print, then goes into a loop in which it reads four input data from paper tape, performs the integration, prints the input and results, and returns to read more data. ADAMS receives X0,Y0,Z0, and XE as input (with the dummy names XX,YY,ZZ, and EE). M, H, and the final results X,Y, and ZED are external to both EXAMPLE3 and ADAMS.

The integration is based on the following procedure: The leading row of backward differences (which are unknown at the beginning) is first filled out with zeroes (line 52). With this leading row we integrate M steps ahead with the formulae of Adams (line 64), since R in the loop named ADMINT means the number of steps to be integrated. After this we may build up a new difference table from the Mth row backwards by keeping the Mth difference con-

stant (lines 67-73). In this way we obtain a new leading row of backward differences, with which we again integrate M steps forward. This is repeated until the Mth difference of two successive runs are nearly equal (lines 65-66 and 74; note that WE is the Mth difference of the preceding run). As soon as BETA is FALSE, we start integrating ahead a sufficient number of steps to reach XE (lines 76 to 123).

Lines	Comments
13-16	An AP2 sequence is used to initialize output and activate STEX.
61-63	Note use of the power point in arithmetic expressions.
17,23,24, 27-34	Input and results are printed with SCRIBE. The arguments in the EXECUTE command correspond in number and order to the dummy fields in the FORMATs.
11-12, 42-46	The REM may be followed by either a tab (lines 42-46) or a carriage return (lines 11-12). The same is true of FORMAT and BCD.
52-60	Extra spaces are ignored.
37	This line illustrates both the definition of a function in a single line and the use of an auxilliary equation to evaluate a common subexpression.
51	Execution of VSPACE leaves zeroes in the vector for which space is taken. This initializes W for the first pass through the loop.
61,65,66, 61,63	If a name occurs for the first time on the lefthand side of an equation, its type is inferred from the righthand side. Thus, DECIDE and BETA are inferred to be Boolean in lines 61 and 63; R, J, and V are inferred as integers in lines 61,65, and 66.
63	BETA is evaluated as TRUE if $10^{-7} < \lvert W_M - WE \rvert$; otherwise BETA is evaluated as FALSE.

Lines	Comments
72,76,105 122	These are all conditional equations. Lines 76 and 105 illustrate arithmetic conditionals; lines 72 and 122 illustrate Boolean conditionals.
22&41 102&37	The values to be used at each execution of a function are passed to the function as an ordered argument list, with the arguments corresponding in number and type to the parameters in the definition of the program.
123	The vectors for which space was taken at the beginning of ADAMS are freed at the end.

```
                                                                                 1
          DEFINE                                                                 2
          SCALARS XO,YO,ZO,XE,H,X,Y,ZED                                          3
          INTEGER M                                                              4
          VECTORS B,C,W                                                          5
          FUNCTIONS F,ADAMS                                                      6
                                                                                 7
EXAMPLE3(Z).=SEQ                                                                 10
          REM                                                                    11
THIS IS THE DRIVER PROGRAM.  IT CONTROLS INPUT, INTEGRATION, AND OUTPUT
                PAG              *Z                                              13
                SLN              +00002                                          14
                LTZ              +00000 3120 0000 00135                          15
                TSR              *+126                                           16
          EXECUTE SCRIBE(HEADER)                                                 17
          M=6, H=0.01                                                            20
LOOP      DATA XO,YO,ZO,XE                                                       21
          EXECUTE ADAMS(XO,YO,ZO,XE)                                             22
          EXECUTE SCRIBE(M,H,XO,YO,ZO,IN)                                        23
          EXECUTE SCRIBE(X,Y,ZED,OUT)                                            24
                SPA              *7                                              25
          CC=*LOOP                                                               26
HEADER    FORMAT                                                                 27
M       H             XO                YO                 ZO                    30
IN        FORMAT                                                                 31
d     d.ddddd   -ddddd.ddddd      -ddddd.ddddd      -ddddd.ddddd
OUT       FORMAT                                                                 33
                -ddddd.ddddd      -ddddd.ddddd      -ddddd.ddddd
END                                                                              35
                                                                                 36
F(XX,YY,ZZ)=ZZ(SIN(TMP)+COS(TMP)+YY^2+2), TMP=XX+YY+ZZ                           37
                                                                                 40
ADAMS(XX,YY,ZZ,EE).=SEQ                                                          41
          REM     XX,YY,ZZ ARE THE INITIAL VALUES FOR X,Y,Y PRIME
          REM     M IS THE ORDER OF THE METHOD (≤6)                              43
          REM     EE IS THE END OF THE INTEGRATION                               44
          REM     H IS THE INTEGRATION STEP                                      45
          REM     W0 IS THE SECOND DERIVATIVE, WK THE KTH BACK DIFF.
          EXECUTE VSPACE(B,7)                                                    47
          EXECUTE VSPACE(C,7)                                                    50
          EXECUTE VSPACE(W,M+1)                                                  51
          B1=1.,          C1=0.5                                                 52
          B2=0.5,         C2=1./6.                                               53
          B3=5./12.,         C3=1./8.                                            54
          B4=3./8.,          C4=19./180.                                         55
          B5=251./720.,          C5=3./32.                                       56
```

```
          B6=95./288.,           C6=863./10080.                  57
          B7=19087./60480.,          C7=275./3456.               60
          WE=1*10, R=M, DECIDE=TRUE                              61
LOOP      CC=+ADMINT                                             62
RLINT     BETA=1*-7<|WM-WE|                                      63
          WE=WM                                                  64
          FOR J=M,-1,1                                           65
          FOR V=2,1,M                                            66
          WV=WV-WV+1                                             67
          REPEAT                                                 70
          REPEAT                                                 71
          CC=+LOOP .IF BETA                                      72
          R=FIX((XE-XX)/H), DECIDE=FALSE                         73
ADMINT    X=XX, Y=YY, ZED=ZZ                                     74
          FOR J=1,1,R+1                                          75
          CC=+L14 .IF J=1                                        76
          FOR V=M,-1,1                                           77
          WV+1=WV                                               100
          REPEAT                                                101
L14       W1=F(X,Y,ZED)                                         102
          REM     F IS THE FUNCTION DEFINING THE                103
          REM     DIFFERENTIAL EQUATION                         104
          CC=+NSHIFT .IF J=C                                    105
          FOR V=2,1,M+1                                         106
          WV+1=WV                                               107
          REPEAT                                                110
NSHIFT    P=Z, Q=Z                                              111
          FOR V=1,1,M+1                                         112
          P=P+BV WV                                             113
          Q=Q+CV WV                                             114
          REPEAT                                                115
          X=X+H                                                 116
          Y=Y+H(ZED+Q H)                                        117
          ZED=ZED+P H                                           120
          REPEAT                                                121
          CC=+RLINT .IF DECIDE                                  122
          ERASE B,C,W                                           123
END                                                             124
          DEFINE                                                125
LEAVE                                                           126
                                                                127
```

```
EXAMP    START NEW PROGRAM                                  4/19/66 13.33

*BGIN    PROGRAM SEQUENCE
LOOP     PROGRAM SEQUENCE
HEADE    PROGRAM SEQUENCE
IN       PROGRAM SEQUENCE
OUT      PROGRAM SEQUENCE
END      PROGRAM SEQUENCE


EXAMP .=
               *BGIN       1      47 21641 00 0001 00101     END

THIS IS THE DRIVER PROGRAM.  IT CONTROLS INPUT, INTEGRATION, AN
D OUTPUT
               LOOP       15      01 40001 00 4000 00004

               HEADE      51      00 00000 00 0000 00007

               IN         61      00 00500 00 0000 00010

               OUT        72      00 00300 00 0000 00010

               END       103      01 01000 00 4000 00000
REFERENCE WORDS...
               ZED     77764      714443252500000000
               Y       77765      702525252500000000
               X       77766      672525252500000000
               ADAMS   77767      404340546240000000
               XE      77770      674425252500000000
               ZO      77771      710025252500000000
               YO      77772      700025252500000000
               XO      77773      670025252500000000
               *INOU   77774      755055566440000000
               H       77775      472525252500000000
               M       77776      542525252500000000
               SCRIB   77777      624261504140000000
INTERNAL STORAGE..
               *NUMB     104           3120000000135
               *NUMB     105      770024365605075341


          SUBROUTINES REFERENCED
                      ADAMS
         GENIE...     *INOU
         GENIE...     SCRIB
```

```
F          START NEW PROGRAM                                   4/19/66 13.33

+BGIN      PROGRAM SEQUENCE
END        PROGRAM SEQUENCE




F       .=
                +BGIN        1      10 01000 02 4400 00136

                END         25      01 01000 00 4400 00137
                            26      01 40006 00 4000 00000
                            27      07 01000 00 4200 00000
REFERENCE WORDS...
                COS      77776      42566225254000000
                SIN      77777      42505525254000000
INTERNAL STORAGE..
                TMP         30                      0
                +NUMB       31       10020000000000000

     PARAMETERS AT PF +
                XX               0
                YY               1
                ZZ               2



          SUBROUTINES REFERENCED
                                        137
        GENIE...       COS
        GENIE...       SIN
                                        136
```

```
ADAMS    START NEW PROGRAM                                    4/19/66 13.33

+BGIN    PROGRAM SEQUENCE
LOOP     PROGRAM SEQUENCE
RLINT    PROGRAM SEQUENCE
+FOR1    PROGRAM SEQUENCE
+FOR2    PROGRAM SEQUENCE
+RPT2    PROGRAM SEQUENCE
+RPT1    PROGRAM SEQUENCE
ADMIN    PROGRAM SEQUENCE
+FOR3    PROGRAM SEQUENCE
+FOR4    PROGRAM SEQUENCE
+RPT4    PROGRAM SEQUENCE
L14      PROGRAM SEQUENCE
+FOR5    PROGRAM SEQUENCE
+RPT5    PROGRAM SEQUENCE
NSHIF    PROGRAM SEQUENCE
+FOR6    PROGRAM SEQUENCE
+RPT6    PROGRAM SEQUENCE
+RPT3    PROGRAM SEQUENCE
END      PROGRAM SEQUENCE

ADAMS .=
                +BGIN      1      10 01000 02 4400 00136

XX,YY,ZZ ARE THE INITIAL VALUES FOR X,Y,Y PRIME
M IS THE ORDER OF THE METHOD (≤4)
EE IS THE END OF THE INTEGRATION
H IS THE INTEGRATION STEP
W+0+ IS THE SECOND DERIVATIVE, W+K+ THE KTH BACK DIFF.
                LOOP      121     01 21700 40 4001 00053     ADMIN

                RLINT     122     01 21740 41 0401 77650     M

                +FOR1     134     01 21700 00 0401 77636     M

                +FOR2     141     01 21700 00 4000 00002

                +RPT2     155     20 10401 00 0001 00230     V

                +RPT1     157     30 10401 00 0001 00225     J

                ADMIN     175     01 21700 00 0600 00000     XX

                +FOR3     202     20 20001 00 4001 00201     J

                +FOR4     215     01 21700 00 0401 77555     M

                +RPT4     230     30 10401 00 0001 00155     V

                L14       232     00 20102 26 4401 77534     X

F IS THE FUNCTION DEFINING THE
DIFFERENTIAL EQUATION
                +FOR5     244     01 21700 00 4000 00002

                +RPT5     263     20 10401 00 0001 00122     V

                NSHIF     265     00 20001 00 4001 00122     P

                +FOR6     267     20 20001 00 4001 00116     V

                +RPT6     313     20 10401 00 0001 00072     V

                +RPT3     330     20 10401 00 0001 00054     J
```

```
               END         347      01 01000 00 4400 00137
                           350      01 40006 00 4000 00000
                           351      07 01000 00 4200 00000
REFERENCE WORDS...
               F         77764      452525252540000000
               ZED       77765      714443252500000000
               Y         77766      702525252500000000
               X         77767      672525252500000000
               FIX       77770      455067252540000000
               H         77771      472525252500000000
               XE        77772      674425252500000000
               M         77773      542525252500000000
               W         77774      662525252540000000
               C         77775      422525252540000000
               VSPAC     77776      656257404240000000
               B         77777      412525252540000000
INTERNAL STORAGE..
               ←P1         352                       0
               ←ONEF       353       10010000000000000
               ←NUMB       354      772000000000000000
               ←NUMB       355       10060000000000000
               ←NUMB       356       10050000000000000
               ←NUMB       357       10140000000000000
               ←NUMB       360       10100000000000000
               ←NUMB       361       10030000000000000
               ←NUMB       362       10230000000000000
               ←NUMB       363       12640000000000000
               ←NUMB       364       13730000000000000
               ←NUMB       365       20026400000000000
               ←NUMB       366       10400000000000000
               ←NUMB       367       11270000000000000
               ←NUMB       370       20011000000000000
               ←NUMB       371       20032760000000000
               ←NUMB       372       20473000000000000
               ←NUMB       373       21124360000000000
               ←NUMB       374       23542000000000000
               ←NUMB       375       20010460000000000
               ←NUMB       376       20154000000000000
               ←NUMB       377       50022500574400000
               WE          400                       0
               R           401                       0
               DECID       402                       0
               ←NUMB       403      750015327745152745
               BETA        404                       0
               J           405                       0
               V           406                       0
               ←P2         407                       0
               P           410                       0
               Q           411                       0

   PARAMETERS AT FF +
               XX                0
               YY                1
               ZZ                2
               EE                3
```

SUBROUTINES REFERENCED

```
                                        137
                                        135
          F
GENIF...  FIX
GENIF...  VSPAC
                                        135
```

- Complex Matrix Inverse

This program inverts a square matrix whose elements are complex numbers. The method used is essentially inplace Gaussian reduction as described in "An Introduction to Numberical Mathematics", Stiefel, E.L., 1963, page 3. Each successive pivot element has the largest modulus of all the remaining choices. This insures the least possible error in the resulting inverse. If the modulus of any pivot element is too small, the matrix is numerically singular, an error message is printed.

The n X n complex matrix is stored as 2 n X n real matrices with the primary codewords in two successive memory locations. Throughout the program, subscripting and arithmetic are performed on the complex variables with the same Genie code that has heretofore been used for real variables.

Lines 11 to 13:

Working storage defined.

Line 14:

Complex matrix B is copied into A. Thus, B will be preserved after its inverse is computed.

Lines 20 to 27:

The largest remaining pivot element is found and stored in GMOD and indices stored in GG and HH.

Lines 30 to 64:

If the chosen pivot is large enough, the exchange algorithm is applied to A.

Lines 66 to 103:

Since pivot elements were not in general along the diagonal, the rows and columns of the inverse are rearranged depending on the contents of ROWW and COLL.

Line 104:

The inverse is stored in RESULT, where all results of implicit

functions are stored. Working storage is freed.

Lines	Comments
3-4	Double declarations of integer vectors and complex matrices.
10	Use of ROW function implicitly in expression.
11-12	ROWW and COLL are declared real and, thus, are created by VSPACE, the real vector space program called in the program.
13	NEW is declared complex. Thus, even though MSPACE is called by the user, CMSPACE (complex matrix space) will be the program executed.
22	Subscripting of a complex variable; use of MOD function implicitly in an expression.
24	Two equations on one line, as many as fifteen permitted.
30	Specification of a constant using power point, '*'. '←' is printed for '#'.
45	Use of '+/' to create complex variable out of two real variables.
51	Labelled REPEAT command. This is not the same as labelling the corresponding FOR statement.
60	Compound conditional transfer.
106	Unconditional transfer to labelled location.
107	Use of SCRIBE to print error message.
115-117	Terminating LEAVE statement is followed by two carriage returns.

```
                                                                        1
              DEFINE                                                    2
              COMPLEX MATRIX B,A,NEW                                    3
              INTEGER VECTOR ROWW,COLL                                  4
INVERT(B),=SEQ                                                          5
              INTEGER GG,HH,L,C,D,E                                     6
              COMPLEX G                                                 7
              L=ROW(B)                                                  10
              EXECUTE VSPACE(ROWW,L)                                    11
              EXECUTE VSPACE(COLL,L)                                    12
              EXECUTE MSPACE(NEW,L,L)                                   13
              A=B                                                       14
              FOR C=1,1,L                                               15
              GMOD=0.                                                   16
              GG=C,HH=C                                                 17
              FOR D=C,1,L                                               20
              FOR E=C,1,L                                               21
              KMOD=MOD(A    )                                           22
                        D,E
              CC=+MORE  ,IF KMOD≤GMOD                                   23
              GG=D,HH=E                                                 24
              GMOD=KMOD                                                 25
MORE          REPEAT                                                    26
              REPEAT                                                    27
              CC=+BYE  ,IF GMOD≤1.0*-12                                 30
              COLL =GG                                                  31
                  C
              ROWW =HH                                                  32
                  C
              FOR D=1,1,L                                               33
              G=A                                                       34
                 C,D
              A   =A                                                    35
               C,D  GG,D
              A    =G                                                   36
               GG,D
              REPEAT                                                    37
              FOR D=1,1,L                                               40
              G=A                                                       41
                 D,C
              A   =A                                                    42
               D,C  D,HH
              A    =G                                                   43
               D,HH
              REPEAT                                                    44
              NEW   =(1.+/0.)/A                                         45
                 C,C           C,C
              FOR D=1,1,L                                               46
              CC=+SOME  ,IF D=C                                         47
              NEW   =A   /A                                             50
                 D,C  D,C  C,C
SOME          REPEAT                                                    51
              FOR D=1,1,L                                               52
              CC=+TOME  ,IF D=C                                         53
              NEW   =-(A   /A   )                                       54
                 C,D     C,D  C,C
```

```
TOME       REPEAT                                                   55
           FOR D=1,1,L                                              56
           FOR E=1,1,L                                              57
           CC=*VOME .IF D=C .O.R E=C                                60
           NEW    =A    +(NEW   )(A   )                             61
              D,E   D,E      C,F    D,C
VOME       REPEAT                                                   62
           REPEAT                                                   63
           A=NEW                                                    64
           REPEAT                                                   65
           FOR C=L,-1,1                                             66
           HH=COLL                                                  67
                  C
           GG=ROWW                                                  70
                  C
           FOR D=1,1,L                                              71
           G=A                                                      72
              D,C
           A   =A                                                   73
            D,C  D,HH
           A    =G                                                  74
            D,HH
           REPEAT                                                   75
           FOR D=1,1,L                                              76
           G=A                                                      77
              C,D
           A   =A                                                  100
            C,D  GG,D
           A    =G                                                 101
            GG,D
           REPEAT                                                  102
           REPEAT                                                  103
           RESULT=A                                                104
           ERASE COLL,ROWW,NEW,A                                   105
           CC=*END                                                 106
BYE        EXECUTE SCRIBE(MESS)                                    107
           CC=*END                                                 110
MESS       FORMAT                                                  111
NO INVERSE DUE TO SINGULARITY                                      112
END                                                                113
           DEFINE                                                  114
LEAVE                                                              115
                                                                   116
```

```
INVER   START NEW PROGRAM                                   9/23/66 13.02

←BGIN   PROGRAM SEQUENCE
←FOR1   PROGRAM SEQUENCE
←FOR2   PROGRAM SEQUENCE
←FOR3   PROGRAM SEQUENCE
MORE    PROGRAM SEQUENCE
←RPT2   PROGRAM SEQUENCE
←FOR4   PROGRAM SEQUENCE
←RPT4   PROGRAM SEQUENCE
←FOR5   PROGRAM SEQUENCE
←RPT5   PROGRAM SEQUENCE
←FOR6   PROGRAM SEQUENCE
SOME    PROGRAM SEQUENCE
←FOR7   PROGRAM SEQUENCE
TOME    PROGRAM SEQUENCE
←FOR8   PROGRAM SEQUENCE
←FOR9   PROGRAM SEQUENCE
VOME    PROGRAM SEQUENCE
←RPT8   PROGRAM SEQUENCE
←RPT1   PROGRAM SEQUENCE
←FORa   PROGRAM SEQUENCE
←FORb   PROGRAM SEQUENCE
←RPTb   PROGRAM SEQUENCE
←FORc   PROGRAM SEQUENCE
←RPTc   PROGRAM SEQUENCE
←RPTa   PROGRAM SEQUENCE
BYE     PROGRAM SEQUENCE
MESS    PROGRAM SEQUENCE
END     PROGRAM SEQUENCE


INVER .=
                ←BGIN     1    10 01000 02 4400 00136

                ←FOR1    47    20 20001 00 4001 00511    C

                ←FOR2    64    01 21700 00 0001 00500    C

                ←FOR3    65    01 21700 00 0001 00473    C

                MORE    115    20 10401 00 0001 00450    E

                ←RPT2   117    20 10401 00 0001 00445    D

                ←FOR4   134    20 20001 00 4001 00430    D

                ←RPT4   162    20 10401 00 0001 00402    D

                ←FOR5   164    20 20001 00 4001 00400    D

                ←RPT5   212    20 10401 00 0001 00352    D

                ←FOR6   233    20 20001 00 4001 00331    D

                SOME    265    20 10401 00 0001 00277    D

                ←FOR7   267    20 20001 00 4001 00275    D

                TOME    321    20 10401 00 0001 00243    D

                ←FOR8   323    20 20001 00 4001 00241    D

                ←FOR9   327    20 20001 00 4001 00236    E
```

```
             VOME      375     20 10401 00 0001 00170     E
             ←RPT8     377     20 10401 00 0001 00165     D
             ←RPT1     420     20 10401 00 0001 00140     C
             ←FORa     422     01 21700 00 0001 00133     L
             ←FORb     435     20 20001 00 4001 00127     D
             ←RPTb     463     20 10401 00 0001 00101     D
             ←FORc     465     20 20001 00 4001 00077     D
             ←RPTc     513     20 10401 00 0001 00051     D
             ←RPTa     515     30 10401 00 0001 00043     C
             BYE       541     01 21702 26 4001 00004     MESS
             MESS      546     00 00000 00 0000 00004
             END       553     01 01000 00 4400 00137
                       554     01 40006 00 4000 00000
                       555     07 01000 00 4200 00000
   REFERENCE WORDS...
             SCRIB  77755      524261504140000000
             CADD   77756      424043432540000000
             CMPY   77757      425457702540000000
             CDIV   77760      424250652540000000
             CMPLX  77761      425457536700000000
             ←←←←←  77762      757575757500000000
             MOD    77763      545643252540000000
             CSTAR  77764      426263406140000000
             ←←←←←  77765      757575757540000000
             A      77766      402525252540000000
             ←←←←←  77767      757575757540000000
             CMCPY  77770      425442577040000000
             CMsPA  77771      425462574040000000
             NEW    77772      554466252540000000
             ←←←←←  77773      757575757540000000
             COLL   77774      425653532540000000
             VSPAC  77775      656257404240000000
             ROWw   77776      615666662540000000
             CROW   77777      426156662540000000
   INTERNAL STORAGE..
             L         556                      0
             ←P1       557                      0
             ←P2       560                      0
             C         561                      0
             GMOD      562                      0
             GG        563                      0
             HH        564                      0
             D         565                      0
             E         566                      0
             ←Q1       567                      0
             ←←←←←     570                      0
             KMOD      571                      0
             ←NUMB     572     7300106274630045557
             G         573                      0
             ←←←←←     574                      0
             ←ONEF     575      1001000000000000
        PARAMETERS AT PF +
             B                 0
             ←←←←←             1
```

SUBROUTINES REFERENCED

```
                                     137
GENIE...        SCRIB
GENIE...        CADD
GENIE...        CMPY
GENIE...        CDIV
GENIE...        MOD
                                     135
GENIE...        CMCPY
GENIE...        CMSPA
GENIE...        VSPAC
GENIE...        CROW
                                     136
```

CODING CONVENTIONS

This section discusses details of compiler generated code. It is intended for those who are particularly interested and for those who wish to code in a lower level language while maintaining compatibility with compiled programs. This material is not essential to the understanding of the Genie language and should not be read before attempting to write some programs for the compiler and gaining some familiarity with the Rice Computer, the assembly language, and the SPIREL system.

• Programs initialization and termination

The 'SEQ' or 'RSEQ' causes the compiler to generate a sequence of orders which initializes the program being compiled. The first of these orders is labelled '←BGIN', and the orders are collectively called the "←BGIN code sequence". For each 'SEQ' or 'RSEQ' there is an 'END', and an "END code sequence" corresponds to each ←BGIN code sequence. The forms of these code sequences depend on whether 'SEQ' or 'RSEQ' is used, the number of parameters (p) listed for the program and, in some cases, the types of the parameters. Each complex parameter is counted as two parameters, the real part followed by the imaginary part.

An 'SEQ' causes generation of a non-recursive program; an 'RSEQ' causes generation of a recursive program. These two types of code are distinguished functionally by the location of internal variables for the program. Constants are always stored within the program. Private storage is inside a non-recursive program and on the B6-list, addressed relative to PF, for a recursive program. Genie-generated recursive code will not alter itself while running, and a recursive program may use itself -- provided AP2 code in the program also obeys conventions necessary for recursion. The use of a program by itself is clear in a case where program A uses program A; if program A uses B which uses C which uses A, then again program A is using itself.

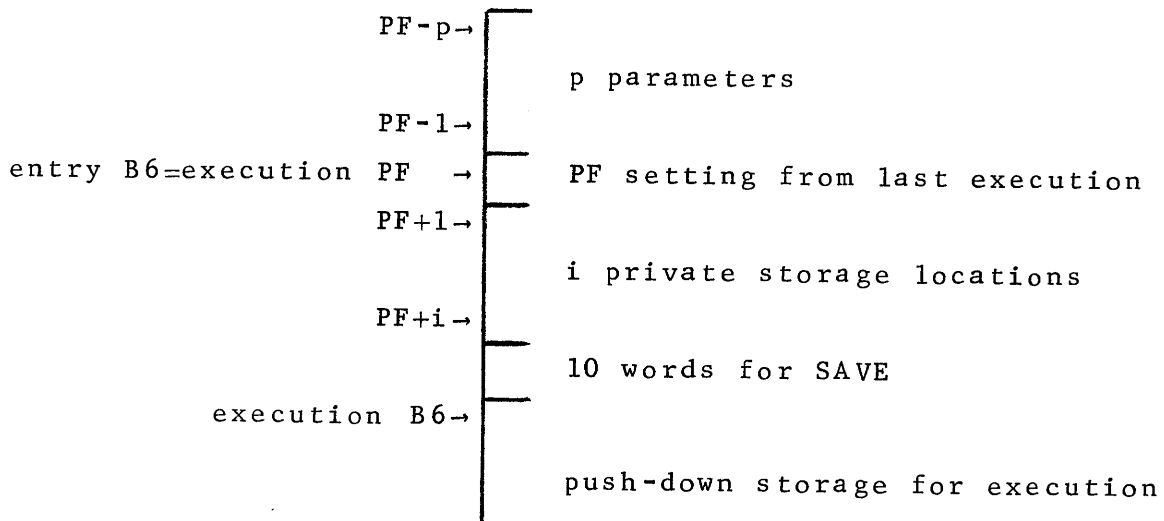
For a non-recursive program -- one begun with 'SEQ' or a one statement function ... A single fast parameter in the definition of a program is a special case which causes only PF to be saved and assumes no parameter addressing in Genie language within the program. Otherwise, fast register names should not be used as parameters in a program definition, and the following discussion applies. A single parameter enters a program in T7, the value of a scalar or * codeword address for a non-scalar. Immediately a scalar with name P in T7 is stored at internal location 'P'; a non-scalar parameter is stored on the B6-list. All fast registers

are saved; if there are parameters on the B6-list (p>1 or p=1 and a non-scalar parameter) PF is set to point to the first parameter. In this case (PF) is stored in the address portion of 'END+1' and must be maintained with this value throughout the program for the purpose of addressing parameters. The END code sequence restores the fast registers, sets B6 to free the storage occupied by any parameters on the B6-list, fetches (T7) for implicit execution, and exits to the PF setting on entry. The specific code sequences are as follows:

```
p=1         ←BGIN    PF    RWT      END
fast                        :
            END             TRA      Z

p=1         ←BGIN    -Z    TRA      *+136, U→R
scalar               T7    STO      P
                            :
            END             TRA      *+137
                     T7    TRA      PF

p=1         ←BGIN    T7    STO      B6, B6+1
non-scalar           -Z    TRA      *+136, U→R
                           SPF      B6-11
                     PF    RWT      END+1
                            :
            END             TRA      *+137
                           SB6      Z
                     T7    TRA      PF

p>1         ←BGIN    -Z    TRA      *+136, U→R
                           SPF      B6-p-10
                     PF    RWT      END+1
                            :
            END             TRA      *+137
                           SB6      Z
                     T7    TRA      PF
```

For a recursive program -- one begun with 'RSEQ' ... A single parameter enters a program in T7, the value of a scalar or * codeword address for a non-scalar. Multiple parameters enter on the B6-list at B6-p,...,B6-1, address for a scalar and * codeword address for a non-scalar. A single non-scalar parameter is stored on the B6-list. In all cases the PF setting for the last execution of the program is picked up from 'END+1' and stored just

beyond the parameters on the B6-list. This B6 value is stored in 'END+1' for the PF setting of the current execution. B6 is advanced over i private storage locations for the program. A full save is done. Then PF is set for execution -- with p parameters at PF-p,...,PF-1 and i private storage locations at PF+1,...,PF+i. A single scalar parameter named P is stored at private storage location 'P'. In the case of a single fast or a single scalar parameter, the program is considered to have no parameters. B6-list utilization by a recursive program is illustrated by:



The END code sequence restores all fast registers, restores the PF setting for the last execution at END+1, backs B6 up by p+i+1 to free all B6-list list used in execution, fetches (T7) for implicit execution, and exits to the PF setting on entry. The specific code sequences are as follows:

```
single          ←BGIN               CLA,WTG+2      END+1
scalar                     B6       RWT            END+1,B6+1
 (p=0)                     B6       ADD            a-*END+3,U→B6
                           -Z       TRA            *+136,U→R
                                    SPF            *END+1
                           T7       STO            P

single          ←BGIN      T7       STO            B6,B6+1
non-scalar                          CLA,WTG+2      END+1
 (p=1)                     B6       RWT            END+1,B6+1
                           B6       ADD            a-*END+3,U→B6
                           -Z       TRA            *+136,U→R
                                    SPF            *END+1

single fast     ←BGIN               CLA,WTG+2      END+1
 (p=0)                     B6       RWT            END+1,B6+1
and multiple               B6       ADD            a-*END+3,U→B6
 (p>1)                     -Z       TRA            *+136,U→R
                                    SPF            *END+1

all cases       END                 TRA            *+137
                                    CLA,WTG        Z
                                    STO,WTG        END+1,B6-1
                           PF       AB6            [-i],U→R
                           T7       AB6            [-p],R→CC
```

- <u>Result for implicit execution</u>

A program which is single valued may be executed implicitly; that is, it may be mentioned within the formula on the righthand side of an equation in Genie language. A non-complex scalar result must be in U upon exit from the program, a complex scalar result in the complex accumulator named CMPLX, a non-complex non-scalar result in the non-scalar accumulator whose codeword is by definition at location +10 during execution. The name 'RESULT' is interpreted by the compiler as T7 for a non-complex scalar, as CMPLX for a complex scalar, as codeword address +10 for a non-complex non-scalar and as CSTAR for a complex non-scalar. 'RESULT' may appear only on the lefthand side of an equation and must be defined in the last command executed before 'END' on all dynamic paths to 'END'. The 'END' code sequence fetches (T7) to U as it exits so that a non-complex scalar result is indeed in U upon return to the program causing the implicit execution.

- Addressing of variables

With respect to any given program every variable is in one of three categories: internal, external, parameter. All internal variables are scalar. For a non-recursive program the values of all internal variables are stored within the program. For a recursive program internal variables are of two types: constants are stored within the program; others are stored in private storage on the B6-list, the ith private storage word being addressed at (PF)+i after program initialization. External variables may be scalar or non-scalar, the address or * codeword address respectively being stored in a cross reference word within the program, the value or codeword respectively being stored in the Value Table (*+122) during execution. In the general case, reference words for parameters are stored on the B6-list. For a non-recursive program the pth parameter is addressed at (PF)+p-1 after program initialization. For a recursive program the pth parameter is addressed at (PF)-p after program initialization. Parameters of a program during execution are indeed internal or external with respect to some dynamically higher level program, but this does not affect addressing in the program where they are parameters. The following charts summarize addressing conventions for variables.

For a non-recursive program --

variable	representation	data address	codeword address	value	element
internal scalar	value in program at IS	#IS	--------	(IS)	-------
external scalar	address in program at ES	(ES)	--------	*ES	-------
external non-scalar	* codeword address in program at ENS	-------	address in (ENS)	-----	*ENS
scalar parameter	address at PF+p-1	(PF+p-1)	--------	*PF+p-1	-------
non-scalar parameter	* codeword address at PF+p-1	-------	address in (PF+p-1)	-------	*PF+p-1

For a recursive program --

variable	representation	data address	codeword address	value	element
internal constant	value in program at IC	#IC	--------	(IC)	-------
internal storage	value on B6-list at PF+i	#PF+i	--------	(PF+i)	-------
external scalar	address in program at ES	(ES)	--------	*ES	-------
external non-scalar	* codeword address in program at ENS	--------	address in (ENS)	-----	*ENS
scalar parameter	address on B6-list at PF-p	(PF-p)	--------	*PF-p	-------
non-scalar parameter	* codeword address on B6-list at PF-p	------	address in (PF-p)	-----	*PF-p

- B6-list, working storage

The SPIREL system uses the block with codeword address +112 as a working storage area. The conventions associated with this storage are that B6 points to the next available location on the list [hence, the term "B6-list"] and that the storage is used in a linear "last-in-first-out" or "push-down" fashion. Genie generated code uses the B6-list for temporary storage of intermediate quantities within the calculation of an arithmetic formula, always storing at (B6), incrementing (B6) after the store, retrieving from (B6)-1, and decrementing (B6) after retrieval. The B6-list is also used for storage of parameters before entering a program; the program then decrements (B6) over the parameters before return since the storage occupied by parameters is no longer in use. For a recursive program, a private storage area is established on the B6-list and freed prior to exit. The SAVE (*+136) and UNSAVE (*+137) programs and other SPIREL routines use the B6-list for temporary dynamic push-down storage.

Using the B6-list for temporary storage, the following sequence shows storage of A, B, C and later retrieval of C, B, A with proper maintenance of (B6) as a pointer to the B6-list:

```
    :
CLA+2    A, B6+1
    :
CLA+2    B, B6+1
    :
CLA+2    C, B6+1
        calculation perhaps involving
        use of B6-list with balance of
    :   stores and retrivals, so that
        final (B6) = initial (B6)
CLA      B6-1, B6-1
STO      C
    :
CLA      B6-1, B6-1
STO      B
    :
CLA      B6-1, B6-1
STO      A
    :
```

- <u>Parameter set-up for program execution</u>

Execution of a program with a single non-complex scalar parameter SP is preceded by code which accomplishes (SP)→T7. In the case of a single non-scalar parameter NSP, the code accomplishes *NSP→T7. For more than one parameter, representations are stored sequentially on the B6-list; if the kth parameter is a scalar SP, then SP→B6, B6+1; if the kth parameter is a non-scalar NSP, then *NSP→B6, B6+1. A complex parameter is treated as two parameters, the real part followed by the imaginary part. If one of a group of parameters is given by a number or an expression, then the quantity must be given a name before the proper parameter representation can be stored on the B6-list. For such purpose the names '←P1', '←P2', etc. for non-complex quantities are generated by the compiler. The quantity is stored at ←Pn for a scalar or *←Pn for a non-scalar is stored on the B6-list. A non-scalar at ←Pn is freed upon return from the program for which it was stored; then all ←Pn used are available for re-use. Complex quantities are stored as pairs named '←Q1', '←Q2', etc., then each part is treated like a non-complex parameter.

The execution of program **PROG** is accomplished by TSR *PROG where PROG is a cross-reference word for PROG within the program doing the execution; the codeword for PROG is in the Value Table (*+122). Thus, PROG is an external variable with respect to the program which executes it.

• Representation of Complex Variables

A complex variable is always on the first level of addressing represented by a pair of words in consecutive memory locations, the real part followed by the imaginary part. The name of a complex variable is attached to the first word of the pair, the real part; the second word of the pair has the name "ditto", printed '←←←←←'. The Cartesian form is used, and both parts are real floating point.

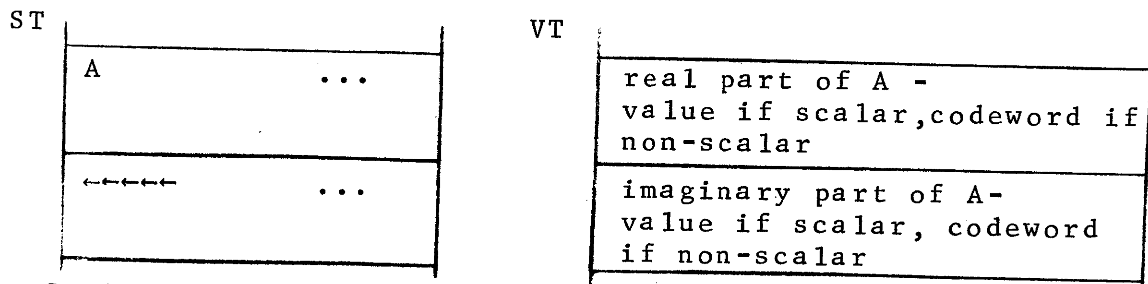
Genie generates internal storage for the complex scalar A as

A	real part of A
←←←←←	imaginary part of A

Genie generates cross-reference words for the external complex variable A as

A	name 'A' in hexads/* if non-scalar/ VT address for A
←←←←←	name "ditto" in hexads/* if non-scalar/ VT address for A's ditto

Then while running the corresponding ST-VT configuration is



Genie constructs two argument words on the B6-list for each complex argument A. The first addresses the real part of A; the second addresses the imaginary part of A.

- Subscription

In the Genie language any variable may be subscripted by from one to five indices separated by commas. The indices are assumed by the compiler to be integers: explicit numbers, simple names, or arithmetic expressions of any complexity. The indices are loaded successively into B1, B2, ..., B5 by the following procedure which allows subscripts to themselves be subscripted:

1) scan n indices from left to right, computing those which are not numbers or simple names, and storing those computed (except the last) on the B6-list;
2) scan from right to left storing (U), quantity from B6-list, named quantity, or explicit number into Bi for i=n, n-1, ..., 1.

In the sense of SPIREL, a subscripted variable is called an "array". In particular, a one-dimensional array of data is called a "vector" and is indexed by B1, and a two-dimensional array of data is called a "matrix" and is indexed by B1 and B2 in that order. But in fact an array may be of as many as five dimensions and may contain either data or programs, and its elements may be addressed in the Genie language. The indices may take on negative values if the storage configuration is correspondingly established.

• Operations on standard forms of non-scalars

In order to perform an operation between a scalar and a vector or matrix, to combine two vectors or matrices, or to store a vector or matrix the non-scalar itself must be addressed in the code. Although completely general forms of non-scalars may be created and manipulated in the SPIREL context and may have their elements addressed in the Genie language, operations on full vectors and matrices are defined only for arrays of standard form in order that execution time is not spent in handling the most general case. The standard form of non-scalars is entirely sufficient in a vast majority of applications. The definition is as follows:

standard form of one dimensional array, vector

1) loaded with STEX active
2) indexed by B1
3) initial index = 1

standard form of two dimensional array, matrix

1) loaded with STEX active
2) indexed by B1 for row specification and B2 for column specification
3) initial row index = 1, initial column index = 1

A standard complex non-scalar is a pair of standard non-scalars, as described. Codewords must be adjacent, real then imaginary; a name adheres to the real part, and the imaginary part is named "ditto" (←←←←←).

Arithmetic operations involving standard non-scalars parallels scalar arithmetic quite closely. By convention, codeword +10 is used as the non-complex non-scalar accumulator, commonly called 'U*'; the complex non-scalar accumulator is named CSTAR. The programs used for performing operations on non-scalars recognize a null codeword address for a non-scalar operand to mean that the operand is the accumulator. The creation of a new U* or CSTAR causes the storage previously addressed by that "name" to be freed. If a non-scalar in U* or CSTAR needs to be temporarily saved, this is done on the B6-list; that is, a word or pair of words on the

B6-list are taken as codewords for the storage addressed and the accumulator codewords are cleared. Note that this storage also involves adjustment of the STEX back-references to address the new codewords.

The code sequence generated by the compiler for non-complex non-scalar storage A → B is as follows:

```
        CLA       A,U→B2       ┐
  Z     TSR       *MCOPY,U→B1  │ copy A→U* only if A≠U*
≠       SPF       *END+1       ┘
        CLA       B,U→B1       ┐
  Z     TSR       *+135,U→B2   │ free storage addressed
≠       SPF       *END+1       ┘ as B only if B≠U* and
                                 not on B6-list
  Z     LDR→      +10,R→B2     ] clear U* codeword          ┐
  R     STO       B1           ] store new codeword         │
                                 for B                      │ if B≠U*
  B1    RPA,WTG   B2           ] update back-reference      ┘
```

The code sequence generated by the compiler for complex non-scalar storage A → B is as follows:

```
        CLA       A,U→B2       ┐
  Z     TSR       *CMCPY,U→B1  │ copy A→CSTAR
≠       SPF       *END+1       ┘ only if A≠CSTAR
        CLA,DBL   B,R→B1       ┐
  Z     TSR       *+135,U→B2   │ free storage addressed
        NOP       Z,B1-1       │ as B only if B≠CSTAR
  Z     TSR       *+135,U→B2   ┘ and not on B6-list
        CLA       CSTAR,U→PF   ┐                              ┐
        CLA,DBL   PF,U→B2      │ store new codewords          │
        STO,DBL   B1           ┘ for B                        │
  B1    RPA       B2,R→B2      ┐                              │ if
        NOP       Z,B1+1       │ update back-references       │ B≠CSTAR
  B1    RPA       B2,R→Z       ┘                              │
  Z     STO,DBL   PF           ] clear CSTAR                  │
≠       SPF       *ENDP1         codewords                    ┘
```

≠(PF) reset only if program is recursive or is using (PF) for reference to parameters.

- <u>Assignment of type and shape to variables</u>

In the Genie language each variable has a shape: scalar, vector, or matrix. The shape of a variable may be explicitly specified as non-scalar by a declaration: VECTOR for vector, MATRIX for matrix. Each scalar, vector, matrix, and function (result) has a type: integer, real floating point, complex, or Boolean. The type of a variable may be explicitly specified in a declaration: INTEGER for integer, REAL or SCALAR for real floating point, COMPLEX for complex, and BOOLEAN for Boolean. The standard shape/type is scalar/floating point unless otherwise specified in an INFER declaration. If the first appearance of a variable name is not in a declaration, its type is implicitly specified by the following rules:

1) If a variable name first appears on the right side of an equation, the variable is assigned the standard shape/type.
2) If a variable name first appears on the lefthand side of an equation, the variable is assigned the shape/type of the expression on the right-hand side.

In a compilation a variable will not have its type changed once it is assigned. An equation which has lefthand and righthand sides of different types will cause the compiler to comment on the equating of unlike types; code will be generated to perform a store appropriate to the quantity on the righthand side, but the type of the quantity on the lefthand side will be unaffected.

- <u>Arithmetic combination of variables of different types</u>

In arithmetic expressions Boolean and integer variables may be combined only in exponentiation, Boolean scalar variable to an integer scalar power. Boolean and floating point variables may not be combined.

Integer and real floating point scalars and non-scalars may be combined in any mathematically meaningful way. In all cases except exponentiation of a floating point scalar by a numberically specified integer ≤ 7, the integer must be floated before the combination takes place. In all cases the result of the combination is floating point. If a numerically defined integer scalar is floated, the floating point equivalent is generated at compilation time and is referenced in the generated code for the combination. Other ise, the floating of an integer scalar A is Accomplished by the following generated code:

```
-LDU      -A
FMP       ←TW47
```

where '←TW47' refers to the constant 2^{47} which will be stored within the program. The floating of an integer vector or matrix is accomplished by use of the Genie SPIREL program MFLT.

Integers and real floating point scalars and non-scalars may be combined with complex scalars and non-scalars in any mathematically meaningful way. In all cases except exponentiation of a complex scalar by an integer or floating point scalar the non-complex quantity is made complex before the combination takes place. A floating point quantity is made complex with real part equal the floating point quantity and zero imaginary part; an integer quantity is floated then made complex as a floating point quantity.

• <u>Boolean variables and operations</u>

A Boolean variable may take on the value 'TRUE' or 'FALSE', these being represented in the computer by full length quantities

TRUE = +0077777777777777777

FALSE = +0077777777777777776

The binary operations between Boolean variables to yield a Boolean value cause code to be generated as follows:

or, A+B, true if either A or B is true

```
CLA    A
ORU    B
```

and, A×B, true if both A and B are true

```
CLA    A
ORU    B
```

symmetric difference, A-B, true if A and B have different values

```
CLA    A
SYD    B
ORU    #77776
```

symmetric sum, A/B, true if A and B have the same value

```
CLA    -A
SYD    B
```

The only meaningful unary operation on a Boolean variable is complementation, not A, true if A is false

```
-I    ORU    -A
```

The machine register sense lights (SL) is a collection of 15 bits, any one of which may be individually meaningful and may be in an on or off (1 or 0) state at any time. The variable SL is Boolean and exponentiation to an integer power is defined

A^B, true if bit B of A is on (1) where the bits of A are numbered from 1 to 15, from left to right

```
CLA    A         ┐
LUR    15-B      │ if B is a number
ORU    #+77776   ┘
CLA    B         ┐
BUS    #15,U→R   │ if B is
CLA    A         │ a name
                 │ or
LUR    *R        │ an expression
ORU    #+77776   ┘
```

Although the Boolean exponential notation is particularly meaningful for the lights, it may be applied to any Boolean variable. Thus, a Boolean variable A which does not itself have a value of TRUE or FALSE may be a collection of 15 bits (the rightmost in a machine word) A^1, A^2, ..., A^{15} each with a value of TRUE or FALSE.

• <u>Loop coding</u>

In the Genie language a loop is begun by the command

FOR iteration parameter = initial, increment, final and ended by the command

REPEAT

If there are not labels on these commands, the kth loop will have the labels '←FORk' and '←RPTk' associated with it. The generalized code generated for loop control is as follows:

```
←FORk         compute initial
              initial → iteration parameter
              compute increment                                   ┐
              store increment                                     A
                                                                  ┘
              compute final                                       ┐
              store final                                         A
                                                                  ┘
[←FORk+m]           LT7           final                           B
              Z     IF(POS)SKP    increment                       ┐
              T7    IF(POS)SKP    iteration parameter, CC+1       C
              T7    IF(NEG)SKP    iteration parameter             ┘
                    TRA           ←RPTk+n
                     :
                orders of loop
                     :
←RPTk               CLA           increment                       ┐
                    FAD→          iteration parameter             D
                    TRA           ←FORk+m                         ┘
[←RPTk+n]            :
```

Seldom is the full generalized code necessary, and the following notes pertain to condensations which are provided in various specific cases.

(A) The increment and the final value are computed and stored only if they are given by expressions, that is, not simple variable names or explicit numbers.

(B) The final value will be stored in the address field of the order if it is given by an explicit integer.

(C) If the increment is given by an explicit integer, it will not be tested for being positive or negative and only the appropriate comparison of iteration parameter to final value will be generated.

(D) If the iteration parameter is a long fast register F, the ←RPTk code sequence will be

```
←RPTk     F     FAD      increment, U→F
                TRA      ←FORk+m
```

If the iteration parameter is an index register Bi and the increment is an explicit integer +1 or -1, the ←RPTk code sequence will be

```
←RPTk           TRA      ←FORk+m,Bi±1
```

- Use of fast registers in Genie generated code

Fast registers may be used in the Genie language and in assembly language coding to be used in a Genie context if there is no conflict with usage generated by the compiler:

T7 is always subject to use for special purpose temporary storage.

T7 is used for storage of a single parameter when a function is executed implicitly or explicitly.

T4, T5, T6 are subject to use in any arithmetic command for scalar temporary storage and for storage of scalars mentioned two or more times in one equation if these fast register names are not mentioned explicitly in the command.

B1 is used when loading parameters onto the B6-list if a name ←Pn is used.

B1, B2, B3, B4, B5 are used for subscripts in addressing elements of arrays. The first k are used to address an element of an array of k dimensions.

B1 and B2 are used in complex scalar arithmetic.

B1, B2, and PF may be used in operations on vectors and matrices.

B1 is used in input-output commands to specify to the program ←INOUT the operation to be performed.

B1 is used in raising an integer or a real floating point scalar to an integer power ≤ 7.

B6 always addresses the push-down B6-list which is used for temporary storage of scalars and non-scalars, for multiple parameter storage, and for private storage of a recursive program.

PF is used within a non-recursive program to address its parameters if there are more than one or if there is only one but that is a non-scalar. The appropriate value fo (PF) is, in such cases, stored in the address portion of END+1 so that resetting is easily accomplished by

```
          SPF          *END+1
```

PF is used within every recursive program to address parameters and private storage locations. The appropriate value of (PF) is stored in the address portion of END+1 so that resetting is easily accomplished by

```
          SPF          *END+1
```

- Rearrangement of arithmetic formulae for efficient evaluation

The compiler has the ability to rearrange the terms in addition (or subtraction) and multiplication (or division) strings. Constant terms are shifted to the left in the formula. Terms which are themselves expressions, rather than simple variable names or numbers, are shifted to the left to save temporary stores that would be required were such complex terms to appear to the right in a string. The ordering of the complex terms is determined by the number of temporary stores required to evaluate each; the complex term requiring the most temporary stores will be shifted farthest to the left.

If the order of evaluation within a formula is of importance, this rearrangement may be avoided by defining each complex term in a separate equation, thereby giving each a name. Then the original formula will involve only simple variable names, and rearrangement will not take place.

SPIREL

SPIREL

Concepts . . . .

Codewords . . . .

System Organization . . . .
- Memory Utilization
- B6-list
- System Components

Control Words . . . .
- Program *126, XCWD
- Control Word Format
- Address Specification
- Basic SPIREL Operations
- More SPIREL Operations
- Recursive Application of SPIREL
- Symbolic Addressing
- Summary of Control Words

Console Communication . . . .

Use of SPIREL . . . .
- Normal Running
- Input Paper Tape Format
- Tracing
- Arithmetic Error Monitor
- Block Bounds Check
- Diagnostic Dump
- High Speed Memory Dump
- Error Messages in SPIREL
- Symbol Table-Value Table Print Format
- SPIREL System on Magnetic Tape

Storage Control . . . .
- Linear Consumption by TAKE
- Activation of STEX and its Domain
- Memory Configuration Generated by STEX
- Use of STEX
- Deactivation of STEX

System Components . . . .
- Vectors and Print Matrix
- Programs
- Component Linkages

System Duplicator . . . .
- Purpose of the Duplicator
- Use of the Duplicator
- SPIREL Generation

Backing Storage System . . . .

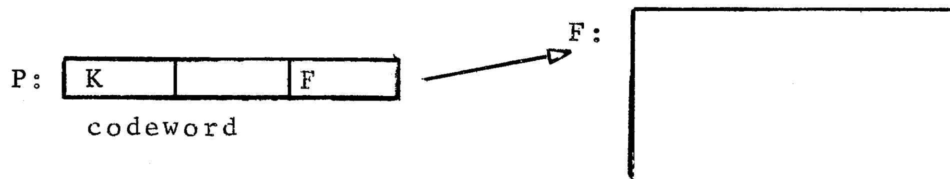
A computer serves a user by decoding instructions and performing the operations specified. In an analagous manner, the SPIREL system serves a user by decoding control words and performing the operations specified. In fact, once in the machine SPIREL may be thought of as an extension of the Rice Computer.

As instructions are used to dictate computer operations on single words in the memory, so are control words used to dictate SPIREL operations on blocks. Physically, a block is a set of contiguous words in memory. Associated with each block is a one-word "label" called a codeword. A block is a logical unit from the point of view of the user; it may contain a program, a vector of data, or codewords which in turn label other blocks. In general, an array is a logical structure which consists of a codeword which labels a block of codewords which label blocks, and so on until on the lowest level are blocks which do not contain labels. The depth or dimension of an array is just the number of codeword levels in the array. A program is a single block with one codeword, a one-dimensional array. A data vector is a single block with one codeword, a one-dimensional array. A matrix of data is a vector of data vectors, a two-dimensional array. Collections of programs and data vectors may be logically grouped to form program and data arrays of any depth deemed organizationally useful to the SPIREL user.

An array is uniquely associated with its single highest codeword, the primary codeword. In most applications all addressing of information in arrays (contents in lowest level blocks) is done through the primary codeword. Thus, access to information in an array depends on only one address, the codeword address for the array. The physical location of blocks is irrelevant to the user, so allocation of storage for blocks is performed by SPIREL, and addressing through levels of codewords constructed by SPIREL is accomplished by the indirect addressing of the hardware. A fixed

region of the memory, locations 200 through 277 (octal), is by convention reserved for primary codewords, and allocation in this area is the responsibility of the individual user. The unique correspondence of a primary codeword address to its array provides an informative "name" for the array. A program with codeword address 225 may be called program *225, and a matrix with primary codeword at 271 may be called matrix *271; the '*' symbolizes indirect addressing in the assembly language, and here serves to emphasize this operation in connection with codewords.

A program *P of length K may occupy a block beginning at machine address F. Then program *P is represented in the machine as



In programming, control is passed to this program by the code

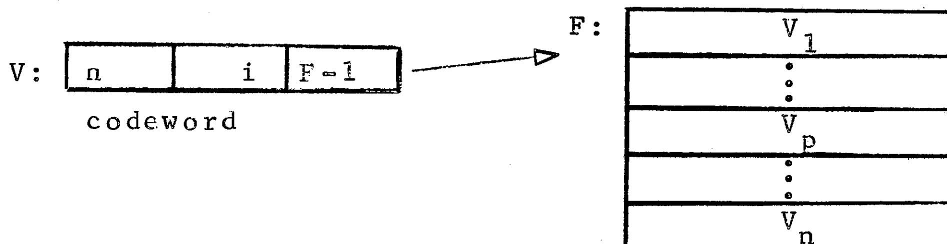
TSR *P

which becomes

TSR F

when the hardware indirect addressing is carried out. The address formed is that of the first word of the program.

A vector *V of n data elements V_1, V_2, ..., V_n may occupy a block beginning at machine address F. Then vector *V is represented in the machine as



In programming, the data element V_p is addressed by the code

p → index register i

then

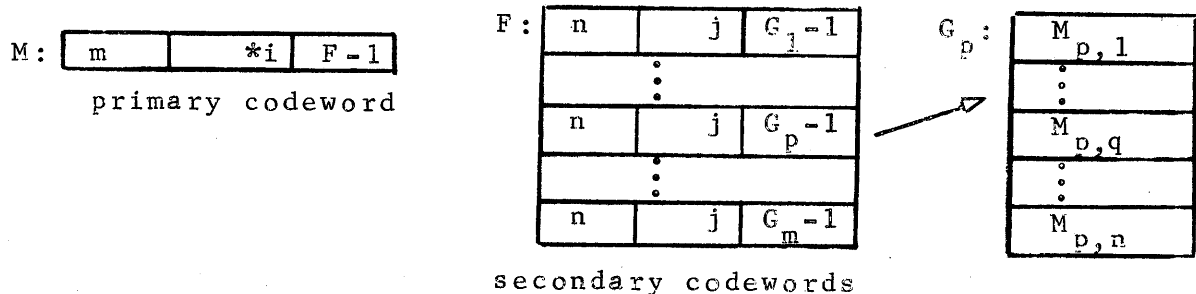
operation *V

which becomes

operation $p+F-1$

when the hardware indirect addressing is carried out. The address formed is that of V_p, as desired. The standard vector form uses index register B1 for element addressing. Non-standard forms permit variability of indexing and even allow the first word of the block containing the vector elements to be addressed as V_K where K is any integer.

A matrix *M of m rows by n columns of data $M_{1,1},\ldots,M_{m,n}$ is stored one row per block and is represented in the machine as



In programming the data element $M_{p,q}$ is addressed by the code

p → index register i

q → index register j

then

operation *M

which becomes

operation $*p+F-1$

and then

operation $q+G_p-1$

when two levels of hardware indirect addressing are carried out. The address formed is that of $M_{p,q}$, as desired. This addressing in no way depends on the size of the matrix *M. The standard matrix form is for a rectangular matrix, using B1 for row specification and B2 for column specification. Non-standard forms permit variability of indexing and non-rectangular structures and even allow the first element to be addressed as $M_{K,L}$ for K and L any integers.

SPIREL operations on arrays are dictated by control words

which specify the primary codeword address for the array. Such operations are:

- to take space for a program, vector, or standard matrix
- to print the lowest level blocks in an array
- to punch an array
- to execute a program
- to free the space occupied by an array
- and many others

Of particular importance is the fact that arrays may be dynamically created and erased or changed in size and structure so that only immediately pertinent arrays occupy space at any time during the run of a user's system.

SPIREL control words are 18-place octal configurations, i.e., one machine word in length. With the SPIREL system in the machine, control words may be transmitted to the system in two ways:

- internally under program control by the user --

  control word → T7

  and TSR *126

- externally to the SPIREL communication routine from paper tape --

  control word preceded by 'carriage return' punch on paper tape in the reader

  or from the typewriter --

  control word $\xrightarrow{type}$ U-register.

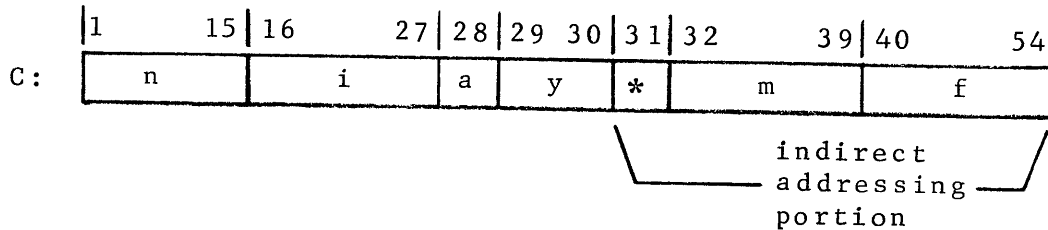
In all these cases the control word is in fact transmitted to XCWD (Execute Control Word), program *126 in the SPIREL system. This program is the nucleus of SPIREL; it interprets each control word and may use other programs in the system to carry out specified operations. SPIREL is, then, a collection of programs, and any of these may be utilized directly by the user of the system.

Details about codewords, system organization, control word decoding and formats, storage control, and the SPIREL programs are given in the succeeding sections.

With every block of memory is uniquely associated a codeword which has two primary functions:

- description of the block, including current length and location, current type of block content, and printing format for the block
- indirect addressing portion appropriate for programmed addressing through the codeword into the block.

The format for a codeword at address C, which labels a block beginning at address F=f+i or F=f+i-1 is:



where

n=length of the block.

i=initial index of the block in 1's complement format, where zero i is denoted by i=7777; standard SPIREL provision is for i=1.

For a block with B-mods in its codeword (a vector), the first word is addressed as element C_i.

For a block with no B-mods in its codeword (a program), the first word of the block is addressed as word 1 of C; if i<1, the words preceding word 1 are cross reference words.

a=1 if block contains codewords; empty (0) otherwise.

y=printing format to be used for output of block if none given in print control word

0: octal, 4 words per line (standard SPIREL provision)
1: hexad, 108 characters per line
2: octal, 1 word per line in program layout
3: decimal, 5 words per line

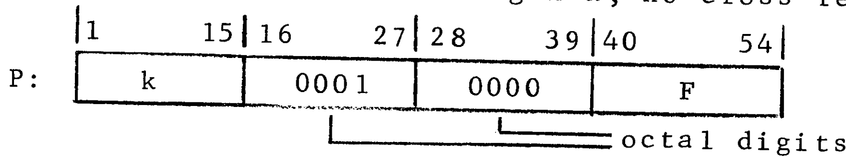
*,m=indirect addressing and B-modification bits, effective in addressing indirectly through the codeword.

f=F-i if block has B-mods in codeword
F-i+1 if block has no B-mods in codeword

F=the address of the first word of the block

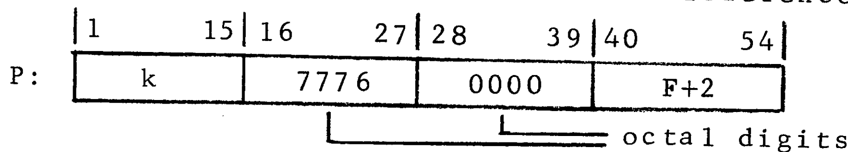
A codeword is completely formed by SPIREL at the time the corresponding block is created. This creation is the result of a control word to take space (structure formed and lowest level blocks filled with zeroes) or one to read a block (structure formed and lowest level blocks filled with words read from paper tape). The most frequently used codeword forms are illustrated below.

- For a program *P of length k, no cross references:



where the first word in the block and the first word of code in the program coincide at address F.

For a program *P with two cross references prior to code:

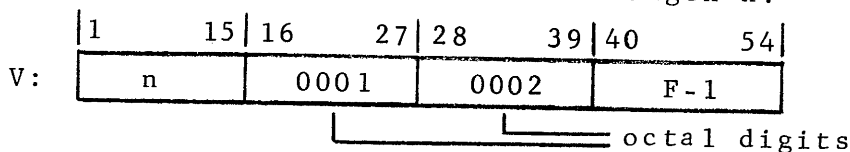


where the first word of the block is located at address F, and the first word of code in the program is located at address F+2.

In both cases the block occupied by the program is k words in length, and control is passed to the program by the code

```
TSR     *P
```

- For a standard vector *V of length n:

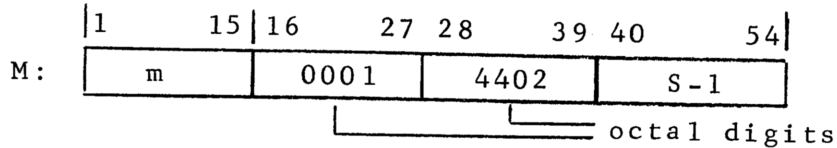


where the vector element V_1 is located at address F, the initial index=1, and a B1 modifier is used. The element V_p is addressed by the code

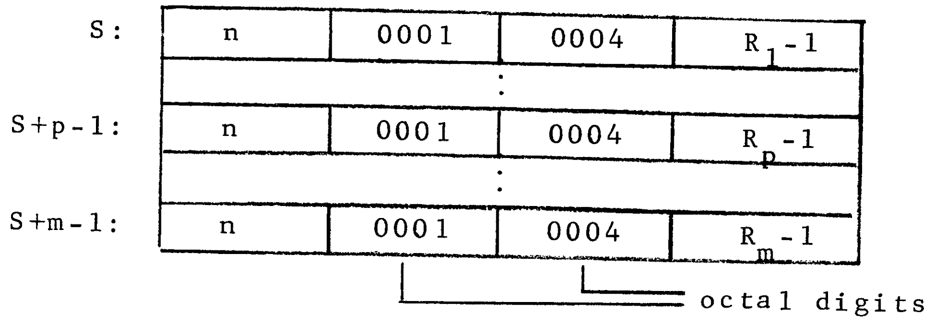
```
SB1     p
CLA     *V
```

- For a standard matrix *M of m rows by n columns:

primary codeword



secondary codewords



where each row is stored in a separate block and the matrix element $M_{p,1}$ is located at address R_p for each p, the initial row and column indices=1, a B1 modifier is used for row specification, a B2 modifier is used for column specification, the primary codeword contains an a-bit and a *-bit. The element $M_{p,q}$ is addressed by the code

```
SB1     p
SB2     q
CLA     *M
```

An array whose primary codeword address is utilized in code is a numbered array. There are also named arrays, whose names are stored in a system vector called the Symbol Table with primary codewords stored in the parallel Value Table. A named array is addressed in code indirectly through a cross reference word which contains the name of the array. All cross reference words for a program are located within the program before the code. The code is then one level removed from the primary codeword for a named array, and the linkage from cross reference word to the codeword in the Value Table is maintained by SPIREL.

The user specifies the primary codeword address or the name for each array, but the location of blocks in the memory is left

to the "discretion" of SPIREL. Any given block may be located variously from run to run, and may be moved even during a run if the STEX storage control mechanism in SPIREL is active and such manipulation is necessary for another requested allocation.

- Memory Utilization

The SPIREL system itself is a collection of programs, tables, and individual constants. System conventions provide memory utilization as follows:

octal addresses	use
00000-00007	machine full-length fast registers Z,U,R,S, T4,T5.T6.T7
00010	used as codeword address by library routines
00011-00020	machine trap locations
00021-00022	not used
00023-00024	link to SPIREL console communication routine
00025-00026	not used
00027-00036	entry to SPIREL program *120, diagnostic dump
00037	console entry to magnetic tape system (explained in MAGNETIC TAPE SYSTEM section)
00040-00077	used by SPIREL program *120, diagnostic dump, and by magnetic tape system programs
00100-00177	system codewords and individual constants
00200-00277	region for primary codewords of numbered arrays or numbered constants of the system user
00300-[E-400]*	storage of blocks labelled by system or user codewords, each block containing a program, data, or codewords which in turn label other blocks
[E-377]-[E-100]*	main magnetic tape system program
[E-77]-E*	magnetic tape system communication program

*E represents the end of memory:

E=17777 for 8K
37777 for 16K
57777 for 24K
77777 for 32K

- B6-list

The working push-down storage area is addressed by index register B6 and is commonly called the B6-list. SPIREL programs use the B6-list; programs of the system user may similarly use the B6-list; and index register B6 may be used for other purposes only if the working storage setting is maintained for those programs which depend on it.

Conventional use of the B6-list depends on one fact: that B6 contains the address of the first word of a block of storage not in use. Therefore, if one word of temporary storage is required, it is taken at B6 and B6 is incremented by one; if the last word stored on the B6-list is retrieved from the address B6-1 and is in fact no longer resident on the list, B6 is decremented by 1, and the storage location may be reused.

The B6-list exists in memory as program block *112. The initial system setting of B6 is to the first word address of this block (given in codeword). The standard length of the block is 200 (octal) locations.

A frequent application of the B6-list is for temporary storage of fast registers to be used by a subroutine, but to appear undisturbed to the program using the subroutine. A program wishing use but preserve T4, T5, and T6 might use the B6-list as follows:

— upon entry

```
T4    STO    B6,B6+1
T5    STO    B6,B6+1
T6    STO    B6,B6+1
```

— computation with private use of T4, T5, T6 and any desired use of the B6-list

— prior to exit

```
      LT6    B6-1,B6-1
      LT5    B6-1,B6-1
      LT4    B6-1,B6-1
```

— exit with B6 setting same as that upon entry.

- System Components

The programs, tables, and individual constants which comprise the SPIREL system are listed below. The programs are fully explained in later sections, and the diagram of SPIREL component linkage shows how the various components are functionally interconnected.

INDIVIDUAL CONSTANTS

Address	Name	Function
100	STORAG	describes available storage
101	FIRSTEX	first word address of storage exchange domain
102	LASTEX	last word address +1 of storage domain
107	NUMBER	relative Symbol Table address of SPIREL operand
114	PRCT	current active length of ADDR
115	ACWD	used by PUNCH
117	STPNT	gives index of last active entry in ST and VT
121	FWA	first word address of last program tagged (used by TRACE and TAGSET)
124	NAME	symbolic name (if any) of block currently being operated on by SPIREL

VECTORS, PRINT MATRIX, B6-LIST

Codeword Address	Name	Use	Length_8
112	LISTB6	Push-Down Storage Area	200
113	ST	Symbol Table	400
116	PM	Print Matrix	200
122	VT	Value Table	400
125	ADDR	Base Address Vector	6
174	TEXT	Console Input Text	14

PROGRAMS

Codeword Address	Name	Use
13	TRACE	Trace
14	ARITH	Arithmetic Error Monitor
20	CHECK	Check Block Bounds
110	HDPR	Print Control Word
111	MATRX	Process Matrix
120	DIADMP	Diagnostic Dump
126	XCWD	Execute from Control Word
127	SETPM	Set Up Print Matrix
130	SMNAM	Find Symbolic Name
131	DATIME	Print Date and Time
132	CLOCK	Decode Clock
133	PCNTRL	Punch Control Word
135	STEX	Storage Exchange
136	SAVE	Save Fast Registers
137	UNSAVE	Unsave Fast Registers
140	DELETE	Insert or Delete Space
141	CHINDX	Change Initial Index
142	TAGSET	Tagset
143	CONVRT	Convert from Decimal
144	PRINT	Print
145	PUNCH	Punch
146	XCWSQ	Execute Control Word Sequence
147	PFTR	PF Trace
150	MAP	Map STEX Domain
151	PRSYM	Print Symbol and Value Tables
152	PWRTN	Conversion of Powers of Ten
153	MRDDC	Multiple Read Decimal
155	BINDC	Binary to Decimal Conversion
156	RDCHK	Read with Checksum
157	PUNCHK	Punch with Checksum
170	PLOT	Plot Character on Scope
171	SAMPLE	Sample Typewriter for Console Input
172	ERPR	Print Error Messages
173	CONSOL	Interpret Console Input
175	CNTXT	Determine Context
176	TLU	Table Look-Up

Program *126, XCWD

The nucleus of the SPIREL system is the program *126, XCWD (Execute from Control Word). This routine interprets control words received in T7 and carries out the specified operations. The work of *126 may in most cases be described in the following steps:

1) address determination -- consists of determining the address of the first word to be operated upon and the number of words to be operated upon
2) operation determination -- consists of determining the operation to be performed
3) operation execution -- consists of performing the operation or using another SPIREL program to carry out the operation

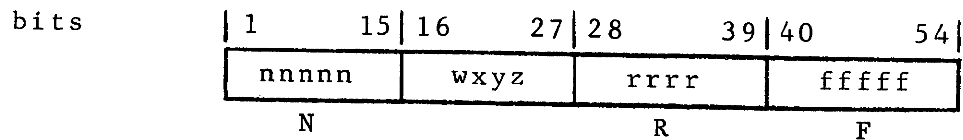
XCWD accepts a control word in T7 and disturbs no other fast registers. Two sense lights affect the behavior of *126:

SL14 off causes XCWD to print one line for each control word it executes. The information printed includes the control word, the operation, the name and relative symbol table address of the block operated on, the location and number of words operated on, the number of free words of storage remaining and a notation if the storage exchange system is active. This SPIREL monitoring provides useful load records and may help in debugging. A system which has been checked out would probably run with SL14 on to suppress monitoring.

SL15 on causes "reading" and "correcting" operations of SPIREL to bypass storage of what is read and instead compare it with what is currently in the locations where storage would otherwise take place.

- Control Word Format

A control word is divided into seven fields: N,w,x,y,z, R, and F. These are arranged as follows:



bits	1 — 15	16 — 27	28 — 39	40 — 54
	nnnnn	wxyz	rrrr	fffff
	N		R	F

where each lower case letter represents one octal digit. In general, N,x,R, and F concern address specification; w,y, and z concern operation specification. x indicates the shape of the operand F

x = 0 → absolute address
x = 1 → relative (single level) address
x = 2 → relative with Bl modifier (single level)
x = 4 → relative (all levels) address
x = 5 → relative on symbol table
x = 3, 6, and 7 are given under More SPIREL Operations, p.29,19,20.

- <u>Address Specification</u>

General rules for address specification may be stated; exceptions exist and are noted in the list of control words later in this section.

The control word fields N,x,R, and F provide information which determines

$\mathfrak{F}$, the address of the first word to be operated on
and η, the number of words to be operated on.

To specify SPIREL operation on a set of locations whose absolute machine locations are known:

x=0
F= $\mathfrak{F}$
N= η
R irrelevant

For example, it might be useful to have SPIREL print the user's codeword region, 100 (octal) words starting at location 200 (octal).

To specify SPIREL operation on all of or a portion of a <u>block</u> labelled by a numbered codeword:

x=1
F=codeword address

If $\mathfrak{F}$ is to be the <u>first word</u> of the block or blocks operated on:

R=0

R≠0 specifies the relative program word or vector element, counting from the initial index of the block in either case, where initial index=1 for programs containing no cross references prior to code and for standard vectors. Note that the 0th word of a program or 0th element of a vector is specified by R=7777.
If η is to be the current length of the block or blocks operated on:

N=0

otherwise:

N= η

To specify SPIREL operation on the entire contents of each of the lowest level blocks of an array which is labelled by a codeword:

x=4

N=0

R=0

(More general application of x=4 is explained in the section on Recursive Application of SPIREL.)

To specify SPIREL operation on a named scalar or block, the name is given after the control word on paper tape or in R for internal control. Then the control word to address a named scalar or block contains:

x=0 for a scalar, 1 otherwise

F=0

and the control word to address all of the lowest level blocks of a named array contains:

x=4

F=0

To specify SPIREL operation, other than read, on a named scalar or all of the lowest level blocks of a named array, the relative Symbol Table (ST) address of the name may be used in typed external communication from the console. The typed control word then contains:

x=5

F=relative ST address of name

Addressing of named quantities is discussed in detail in the section on Symbolic Addressing. In the sections which follow, with the exception of the read operation, a control word which utilizes

x=0

F=location

or

x=1

F=codeword address

may also take the form

x=5

F=relative ST address of name

if the operation can be meaningfully applied to a named scalar or all of the lowest level blocks of a named array.

- Basic SPIREL Operations

The most basic SPIREL operations and the corresponding control word forms are described in this section. This set is sufficient for initial understanding and use of the system.

Read

Read operations are specified by control words with w=0 and x=0,1,2,4, or 5. The y field specifies the read mode:

y=0, octal from paper tape -- 18 octal digits per word, each preceded by a carriage return

=1, hexad from paper tape -- 9 hexads per word, each preceded by a carriage return

=2, zeroes generated by SPIREL

=3, decimal from paper tape, each word followed by a carriage return and in the form discussed in the section on paper tape input formats under Use of SPIREL

=4, hexad with tags and checksum from paper tape in the form punched by SPIREL

For y=0,1 or 2 words are stored with the tag given by z. As words are read (except SPIREL-generated zeroes, specified by y=2), they are checked for proper storage in memory. If the check fails, the word as stored and the location of the word are typed in octal on the console typewriter. The location given is an absolute machine address if x=0 in the control word; it is the relative location in the block if x≠0 in the control word. If SL15 is on, the words read (except SPIREL-generated zeroes, specified by y=2) are not stored but are compared to the contents of the memory location where storage would normally take place. If the comparison fails the word actually stored in memory and its location are typed on the console typewriter as above.

Read control word forms are as follows:

nnnnn 00yz 0000 fffff Read N words in mode y and store with tag z (if y=0,1, or 2) beginning at location F.

nnnnn 01yz rrrr fffff If codeword at F addresses an array and STEX storage control is active, free that array. Create a block of length N and form its codeword at F. Place octal configuration given by R in the corresponding bit positions of codeword at F. (The last three triads of R specify the B-modification and indirect-addressing bits to be used. The first triad describes the contents of the block being read; its interpretation is described under Codewords.) Set initial index=1. Read N words in mode y and store with tag z (if y=0,1, or 2) into the block labelled by the codeword at F.

nnnnn 02yz 0000 fffff Exactly equivalent to the control word nnnnn 01yz 0002 fffff, used to read a standard vector of data.

nnnnn 04yz rrrr fffff If codeword at F addresses an array and STEX storage control is active, free that array. Create the structure of a standard matrix with N rows and R columns with primary codeword at F. Read N×R words, successive complete rows, in mode y with tag z (if y=0,1, or 2) into the matrix with primary codeword at F.

• Basic SPIREL Operations (continued)

The most basic SPIREL operations and the corresponding control word forms are described in this section. This set is sufficient for initial understanding and use of the system.

Correct

The correct operation is specified by a control word with w=1 and x=0,1,2,4, or 5. Correction implies that some part of an existing unit is to be replaced with new information without completely recreating the unit. Therefore, a correct is just a read into an existing block over the previous contents. The fields y and z specify mode and tag as explained for read operations, and again SL15 on causes comparison instead of storage of what is read.

The correct control word form is as follows:

nnnnn 11yz rrrr fffff Read N words into the block labelled by the codeword at F, where N and R are specified according to the standard rules for address specification. Read in mode y with tag z (if y=0,1, or 2).

• Basic SPIREL Operations (continued)

The most basic SPIREL operations and the corresponding control word forms are described in this section. This set is sufficient for initial understanding and use of the system.

Tagset

Tagset operations are specified by control words with w=2 and x=0,1,2,4, or 5. The purpose is to set tags on words in the memory, and the tag to be set is given by z=0,1,2, or 3 (tag 0 meaning no tag). If y=3, all words in the address range specified are tagged. Otherwise, all words in the address range specified are considered instructions and tagging is selective, with a word being tagged only if its "class" triad = y. (The class of an instruction is given in the third triad from the left.)

The tagset operation is most often used to set tag 3 on instructions in programs to be executed. Then if execution is carried out in the trapping mode, the trace program in SPIREL monitors on the printer the execution of the tagged instruction. This trace output is explained in detail in the section on Use of SPIREL.

The tagset control word forms are as follows:

nnnnn 20yz 0000 fffff Set tag z on all words of class y from location F to location F+N-1, inclusive.

nnnnn 21yz rrrr fffff Set tag z on all words of class y in the block labelled by the codeword at F, where N and R are specified according to the standard rules for address specification.

• Basic SPIREL Operations (continued)

The most basic SPIREL operations and the corresponding control word forms are described in this section. This set is sufficient for initial understanding and use of the system.

Execute

The exeucte operation is specified by wxyz=3100 or 3500 and is designed so that SPIREL will, in essence, transfer control to the address specified as the entry to a closed subroutine. This operation is usually employed as an external directive to SPIREL. Primary control is then with SPIREL; successive programs may be executed, with other SPIREL operations interspersed as desired.

The form of the execute control word is as follows:

00000 3100 rrrr fffff Transfer control to word R of the program which comprises the block labelled by the codeword at F. As a special case, if R=0, transfer control to word 1, or the first word of executable code in the program as determined by the initial index. At entry to the specified program, all fast registers except PF and T7 (and T4 if a named program is specified) are set to the values they had at the time the control word was given to SPIREL. The program executed should be written as a closed subroutine, i.e., it should exit to the address contained in PF upon entry.

The first execute will be used to start a system running after loading as necessary into a fresh SPIREL. This first execute has special effect if STEX has not been activated. All memory in use is consolidated so that items previously loaded may be moved to fill gaps of free storage. All free storage is then available for further allocations by TAKE. In effect, STEX is deactivated prior to the execution called for. See section on storage control for more details.

• <u>Basic SPIREL Operations</u> (continued)

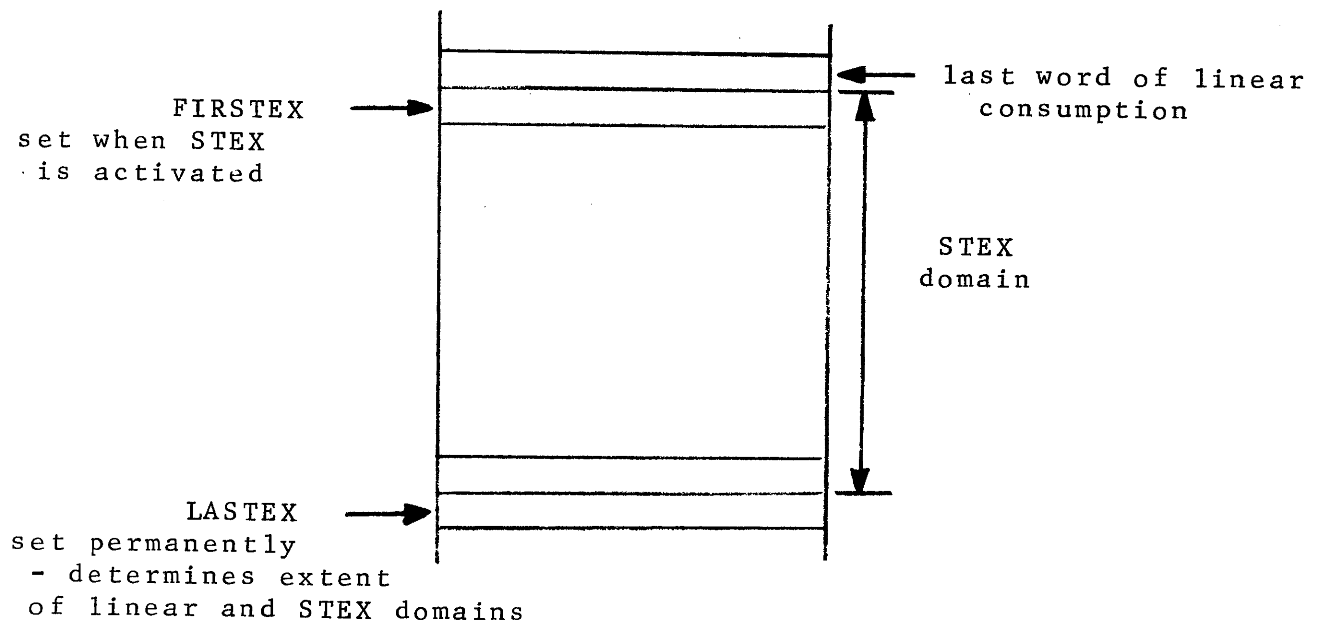
The most basic SPIREL operations and the corresponding control word forms are described in this section. This set is sufficient for initial understanding and use of the system.

<u>Activate STEX</u>

The simple storage control algorithm in SPIREL operates on a principle of linear consumption of space in memory. If STEX, the storage exchange program, is active at the time blocks are created, later redefinition of these blocks will result in the space previously occupied being returned to the system for re-use. Thus, at any given time only space which is currently labelled by a codeword is in use. Activation of STEX causes the STEX domain to be defined

- from the word beyond the extent of linear consumption, this address being stored at FIRSTEX, location 101
- through the word before the address stored at LASTEX, location 102.

This definition is illustrated by:



Any block created after STEX is activated is said to be loaded under STEX control and will be loaded in the STEX domain. Any block may be recreated under STEX control, but old space will be available for re-use only if it was orginally taken in the STEX domain.

The STEX storage control system is described in detail in another section.

The control word which causes STEX to be activated is

00000 3120 0000 00135

It belongs to the execute class of control words. If STEX is activated, subsequent activations are meaningless but harmless.

The first activation of STEX, which may be used after some loading into a fresh SPIREL, has special effect if it precedes the use of an execute control word. All memory in use is consolidated so that items previously loaded may be moved to fill gaps of free storage. The last word of memory in use is then taken as the last word of linear consumption and STEX is activated as described above. In effect, STEX is deactivated prior to the activation called for. See section on storage control for more details.

• Basic SPIREL Operations (continued)

The most basic SPIREL operations and the corresponding control word forms are described in this section. This set is sufficient for initial understanding and use of the system.

Print

Print operations are specified by control words with w=4 and x=0,1,2,4, or 5. The format of output is given by y as follows:

y	format	words/line
0	octal	4
1	hexad	12 (108 chars.)
2	octal, program format	1
3	decimal	5
4	octal, for 8½x11" pages	3
5	octal, with tag	1
6	decimal, with tag	1
7	decimal, for 8½x11" pages	3

For all options except y=1 (hexad), the value of z controls printing at the left of each line the location of the first word on the line:

z = 0, to print location in octal
1, not used
2, to not print location
3, to print location in decimal

The number printed is the relative location in a block for x=1 or 4 (relative) or the absolute machine address for x=0 (absolute). The format for each decimal number printed is:

floating point

-d.ddddddddddde±dd

- "-": if number is negative
- "d.ddddddddddd": 12 decimal digits
- "±dd": exponent, 2 decimal digits

fixed point integer

-ddddddd (1-15 decimal digits; "-" if number is negative)

The SPIREL monitoring provided if SL14 is off provides a printed identifier with each block printed.

The print control words are as follows:

<u>nnnnn 40yz 0000 fffff</u> Print in format y N words beginning at location F, with location format given by z.

<u>nnnnn 41yz rrrr fffff</u> Print in format y from the block labelled by codeword at F, where N and R are specified according to the standard rules for address specification and the location format is given by z.

<u>00000 44yz 0000 fffff</u> Print in format y all of the lowest level blocks in the array labelled by codeword at F, with location format given by z.

• <u>Basic SPIREL Operations</u> (continued)

The most basic SPIREL operations and the corresponding control word forms are described in this section. This set is sufficient for initial understanding and use of the system.

<u>Punch</u>

Punch operations are specified by control words with w=5 and x=0,1,2,4, or 5. The punch format is given by y and corresponds to the mode in which the information punched will later be read:

y=0, octal
=1, hexad
=2, no information (zeroes to be generated internally at read time to correspond to information specified in punch control word)
=3, decimal format
=4, hexad with tags and checksum
=5, high density hexads with tags, parity and checksum

If z is even, every item punched is preceded and followed by control words sufficient to cause the information punched to be read at a later time into logical position identical to that at time of punching; this is the usual procedure. If z is odd, no control words accompany the information punched.

If z is even and a complete array is punched, the control words punched cause recreation of an identical array when the punched tape is later read. If z is even and only part of an array is punched, the control words punched assume that an identical structure exists when the punched tape is later read, and the punched information is corrected into the structure.

Any tape which is punched with the SPIREL punch operations may then be read under SPIREL control with SL15 on to effect a validation of the punched tape by comparison with the information that was to be punched.

The punch control word forms are as follows:

nnnnn 50y0 0000 fffff Punch in mode y N words beginning at location F.

nnnnn 51y0 rrrr fffff Punch in mode y from the block labelled by the codeword at F, where N and R are specified according to the standard rules for address specification.

00000 54y0 0000 fffff Punch in mode y all of the array labelled by the codeword at F.

- More SPIREL Operations

The SPIREL operations described in this section are not necessary for initial understanding and use of the system.

Address Base Manipulation

SPIREL operations may be applied to blocks and arrays when the pertinent codeword address is known. The primary codeword address of any array is fixed and known to the user, and is in the range 200-277 (octal). When the F fields of control words with x=1 are interpreted as machine addresses, the SPIREL address base is said to be set to zero. For the purpose of applying SPIREL operations to sub-arrays, the SPIREL address base may be set so that the F field of control words is interpreted relative to the 0th element of a block of codewords. Each base change in a sequence of base changes will progress one level into an array, and such a sequence may go to depth 4.

Consider an array of three dimensions with primary codeword at address A and B-modification on each level. Assuming that A exists in the machine, the following series of SPIREL operations illustrates the use of address base manipulations:

operation	effect
----	address base initially set to zero
set address base to A 00000 0600 0000 aaaaa	address base at A_0
print I in octal 00000 4400 0000 iiiii	print two-dimensional array A_I in octal format
set address base to J 00000 0600 0000 jjjjj	address base at $A_{J,0}$
print M in octal, one word per line, with tag 00000 4150 0000 mmmmm	print block $A_{J,M}$ in the format one word per line, octal with tag

operation	effect
set address base back one level 00000 0700 0000 00001	address base at A_0
print K in decimal 00000 4430 0000 kkkkk	print two-dimensional array A_K in decimal format
set address base to N 00000 0600 0000 nnnnn	address base at $A_{N,0}$
print from R,P words at Q in octal ppppp 4100 qqqq rrrrr	print words $A_{N,R,Q},\ldots,A_{N,R,Q+P}$ in octal format
set address base to zero 00000 0700 0000 00000	return to initial address base setting

Note that the 0th element of a block is denoted by 77777, and negative element addresses are specified in one's complement form.

The address base manipulation control word forms are as follows:

00000 06y0 rrrr fffff If y=0 and address base is set to zero, set address base down to F if R is null, to F_R if R is not null. If y=0 and address base is set down to A, set address base down one level to A_F if R is null, two levels to A_{F_R} if R is not null. If y=1 and address base is set down to A, increment base to A+F on same level, then set down to $(A+F)_R$ if R is not null.

00000 0700 0000 fffff Set address base back F levels. If F=0, set address base back to zero.

SPIREL execution of these control words causes no monitoring on the printer, but monitoring of SPIREL operations performed with the address base set to other than zero reflects the successive levels of address base settings in effect.

• More SPIREL Operations (continued)

The SPIREL operations described in this section are not necessary for initial understanding and use of the system.

Insert and Delete Words in Blocks

The insert operation provides a facility for lengthening and shortening blocks labelled by codewords. The insert control word form is

nnnnn 1150 rrrr fffff

The Rth word of the block labelled by codeword at F (counting from the initial index) is addressed, and N words are inserted at that point in the block. N is interpreted in one's complement arithmetic. If N>0, N words containing zeros are inserted beginning at the Rth word, and the former R^{th} word becomes the $R+N^{th}$ word; the length of the block is increased by N. If N<0, N words are deleted beginning at the Rth word, and former $R+N^{th}$ word becomes the Rth word; the length of the block is decreased by N. If R is empty, N words are added to the end of the block. If N is empty, the Rth word and all following are deleted.

The insert operation requires that space for the new form of the block be available while the old form still exists. If STEX is active, the space occupied by the old form is freed when the new form is complete.

The delete operation generates the new form of the block on top of the old form and copies only the segment after that deleted. If STEX is active, the space deleted is freed. If codewords are deleted and freed, the space labelled by the codewords is also freed.

In addition to x=1, x=2,4, or 5 are also allowed.

- More SPIREL Operations (continued)

The SPIREL operations described in this section are not necessary for initial understanding and use of the system.

Change Initial Index

The initial index of the block labelled by the codeword at F is set to N by the control word

nnnnn 1160 0000 fffff

N is specified in one's complement form, and an initial index of zero is designated by N=77777.

If the codeword at F contains B-mods, the first word of the block is subsequently addressed as vector element F_N.

If the codeword at F contains no B-mods, the words preceding word 1 in program F are understood to be symbolic cross references and are immediately loaded. Programs which are written to refer to named quantities, in particular Genie generated programs, require these cross references to scalars and codewords in the Value Table (VT, SPIREL system vector *122) in positions parallel to the positions of the names of the quantities in the Symbol Table (ST, SPIREL system vector *113). The cross references must be loaded, linked to the current VT, each time the program is loaded and prior to its execution.

In addition to x=1, x=2,4, or 5 are also allowed.

• More SPIREL Operations (continued)

The SPIREL Operations described in this section are not necessary for initial understanding and use of the system.

Inactivate Storage

The inactivate operation may be applied to any array by using the control word

00000 1170 0000 fffff

If the array labelled by the codeword at F is in the STEX domain, all storage for the array is freed and F is cleared.

In addition to x=1, x=2,4, or 5 are also allowed.

• More SPIREL Operations (continued)

The SPIREL operations described in this section are not necessary for initial understanding and use of the system.

Monitor

If the STEX storage control system is active, blocks in the STEX domain are subject to being physically moved when it is necessary to concentrate free space. At the console, the user may wish to obtain information about the location of a particular block or the number of words of free storage. The control word

00000 3110 rrrr fffff

causes SPIREL monitoring to occur if SL14 is off. R is interpreted as in standard address specification as the word in the block labelled by the codeword at F for which monitoring is desired. No SPIREL operation is performed on the block.

• More SPIREL Operations (continued)

The SPIREL operations described in this section are not necessary for initial understanding and use of the system.

Reorganize

If STEX, the storage exchange program, is active, all free space in memory may be collected into one contiguous block by the control word

00000 3130 0000 00135

This operation is called reorganization, and the word REORGANIZATION is provided on the printer if SL 14 is off. If STEX is not active, this control word has no effect.

• More SPIREL Operations (continued)

The SPIREL operations described in this section are not necessary for initial understanding and use of the system.

Execute Control Word Sequence

A block may contain SPIREL control words. The execute control word sequence operation, when applied to the block, will cause SPIREL to interpret the words addressed as control words and carry out the specified operations in order. If in the sequence a control word specifies a named block by F=0, then the next word in the sequence must contain the name of the block (five printer hexads left-adjusted in the format discussed in the section on symbolic addressing. Therefore, a control word sequence N words in length which contains $\mathfrak{m}$ names then contains N-$\mathfrak{m}$ control words.

The control word forms which instruct SPIREL to execute a control word sequence are as follows:

nnnnn 3040 0000 fffff   Execute control word sequence N words in length stored in memory from location F to location F+N-1.

nnnnn 3140 rrrr fffff   Execute control word sequence in the block labelled by codeword at F, where N and R are specified according to the standard rules for addressing.

• More SPIREL Operations (continued)

The SPIREL operations described in this section are not necessary for initial understanding and use of the system.

Map STEX Domain

The structure, i.e. codewords, for arrays in the STEX domain is printed by use of the control word

00000 3150 0000 00135

If a codeword outside the STEX domain addresses a block inside the STEX domain, it is taken as the primary codeword for an array. The structure of the array is then printed. If z is odd, all inactive and active block headers are printed. e.g.,

00000 3151 0000 00135

• More SPIREL Operations (continued)

The SPIREL operations described in this section are not necessary for initial understanding and use of the system.

Deactivate STEX

The control word which causes STEX to be deactivated is

00000 3160 0000 00135

It belongs to the execute class of control words.

The operation involves reorganization of the STEX domain and recourse to linear storage consumption by TAKE in the inactive area of storage. More details are given in the section on Storage Control.

If STEX is not active, deactivation is meaningless but harmless. Activation of STEX after deactivation causes creation of a new empty STEX domain.

- More SPIREL Operations (continued)

The SPIREL operations described in this section are not necessary for initial understanding and use of the system.

Obtain Date and Time

The date and time of day are available in the computer. SPIREL will format this information (14 positions in length) for printing when the control word

00000 4300 rrrr 00131

is executed. The next line actually printed will contain the date and time beginning at print position R. SPIREL will format and print the date and time beginning at print position R when the control word

00000 4310 rrrr 00131

is executed. The print positions are numbered 1-108 (decimal) from left to right across the page. In both cases, if R is empty, the print position is set by SPIREL to 48 so that the date and time will appear at the right side of a page $8\frac{1}{2}$ inches wide.

• <u>More SPIREL Operations</u> (continued)

The SPIREL operations described in this section are not necessary for initial understanding and use of the system.

<u>Print or Punch Symbol and Value Tables</u>

System elements may have names which correspond to single word storage addresses and codeword addresses for arrays. When loaded under SPIREL control, such elements have their names in the SPIREL system vector *113, the Symbol Table (ST). The parallel SPIREL system vector *122, the Value Table (VT), contains the corresponding single word or primary codeword. The index of the last active ST-VT entry is maintained within the SPIREL system as a constant at 117, STPNT.

These tables may be printed with basic SPIREL control words, but a special printing format appropriate to the contents of the tables is provided when the following control word is used:

<u>nnnnn 45y0 rrrr 00000</u>

N words of ST and VT are printed, beginning at ST_R and VT_R. If N and R are empty, the range for printing is implied by the value of y:

y=0 or 3	all entries in the currently active ST-VT
y=4 or 7	all entries with positive indices in the currently active ST-VT

In any case, y=3 or 7 causes only the entries in context (discussed below) in the range specified or implied to be printed out.

Some or all of the quantities with names in the Symbol Table may be punched in checksum format for symbolic loading. To consider for punching N items beginning with the Rth in the ST-VT the following control word is used:

<u>nnnnn 55y0 rrrr 00000</u>

If N and R are empty, the range of entries considered for punching is implied by the value of y:

y=0,1,2, or 3	all entries in the currently active ST-VT
y=4,5,6, or 7	all entries with positive indices in the currently active ST-VT

In any case, the value of y specifies which items to punch:

y=0 or 4	all programs, then all data* in the range specified or implied
y=1 or 5	programs only in the range specified or implied
y=2 or 6	data* only in the range specified or implied
y=3 or 7	all programs, then all data* in context in the range specified or implied

*data consists of scalars in VT and arrays with codewords in VT; data on tape is preceded by a control word to activate STEX.

As a special case, any ST-VT can be printed with the following control word:

nnnnn 46y0 rrrr fffff

with fffff being the codeword address of the desired ST. The codeword for the corresponding VT must be in the memory location immediately following the ST codeword. N, R and Y are used as described for the system ST-VT.

Prior to the printing or punching called for by these control words, context is automatically determined. This means that the ST entries for all items with negative ST indices which are necessary for support of the programs currently loaded are "marked" with a tag 0; all others will have a tag 1. All items with positive ST-VT indices are automatically taken to be in context, whatever their tags. This assumes that all "library" items are loaded with negative ST-VT indices and that all ST entries with negative indices are given a tag 1 prior to loading of any "private" programs numerically or with positive ST-VT indices; this is the case with SPIREL and the standard library. The same situation may be generated by any user with SPIREL, his own library, and his own private routines.

• <u>Recursive Application of SPIREL</u>

In general, SPIREL control words with x=1 cause the specified operation to be applied to the block labelled by codeword at F. If meaningful, x=1 may be replaced by x=4 and the operation will be applied through the <u>array</u> labelled by codeword at F. This recursive application is accomplished by the use of SPIREL by SPIREL. In other words, when the SPIREL program *126 (XCWD) encounters a control word C with x=4 (except in the case of read, w=0), and F labels a block of codewords, the address base is set down to F, SPIREL is applied to the first N blocks on the next level with N control words C' in which N'=R and R'=0, and the address is set back up one level. If a control word with x=4 is applied to a block which does not contain codewords, the behavior of *126 is as if x=1, and the recursion is terminated. Thus the depth of the recursion is determined by the structure or depth of the array addressed.

As an example, consider the control word to print in decimal

00003 4430 0002 00200

where the array *200 is a standard data matrix. Since the block labelled by the codeword at 200 contains codewords, these control words are generated and delivered for SPIREL execution:

00000 0600 0000 00200
to set address base to 200

00002 4430 0000 00001
to print the first two words of row 1

00002 4430 0000 00002
to print the first two words of row 2

00002 4430 0000 00003
to print the first two words of row 3

00000 0700 0000 00001
to set address base back one level, to zero.

If row 2 in the array *200 had contained codewords for 4 blocks of

data, the control word

00002 4430 0000 00002

in the above sequence would have caused further generation of the control words:

00000 0600 0000 00002

to set address base to 200_2

00000 4430 0000 00001

to print all of block $200_{2,1}$

00000 4430 0000 00002

to print all of block $200_{2,2}$

00000 0700 0000 00001

to set address base back one level, to 200

• Symbolic Addressing

SPIREL provides facilities for addressing scalars, programs, vectors, and matrices by name. A control word with a null F field will cause program *126 (XCWD) to read what follows on paper tape as a 5-hexad name preceded by a cr punch. The name is added to the Symbol Table (ST,*113) if it is not already present. Then the F field is assigned the address in the Value Table (VT,*122) which parallels the name in ST. Under program control a control word with null F may be given in T7, a 5-hexad name left-adjusted in R, and entry made to the second order of *126 with the order

TSR *126, CC+1

Names are represented by 5 printer hexads and are formed by the following rules:

- upper case letters A,B,...,Z are represented by the hexads 40,41,...,71
- lower case letters a,b,...,z are represented by the hexads 40,41,...,71
- the first in a sequence of lower case letters is preceded by a '26' hexad (backspace punch on flexowriter)
- the numerals 0,1,...,9 are represented by the hexads 00,01,...,11
- a name containing less than 5 hexads is filled to 5 hexads by '25' hexads (tab punch on flexowriter) on the right

Examples of 5-hexad names are

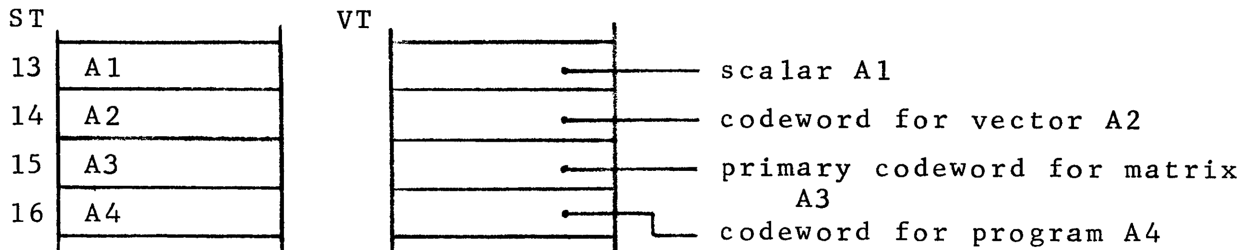
54 40 55 25 25	for	MAN
54 26 40 55 25	for	Man
54 40 55 01 25	for	MAN1
26 54 25 25 25	for	m

The printed load record for a run gives the name of any quantity loaded or referenced symbolically. To the right of each name appears a number F which is the relative Symbol Table address of that name. A SPIREL control word of the form

nnnnn w5yz rrrr fffff

is equivalent to symbolic reference to the Fth Symbol Table entry for all operations except READ. This form is easily typed for console communication to SPIREL.

Consider the ST-VT configuration



The control word with symbol

00001 0030 0000 00000 cr 40 01 25 25 25

will cause the scalar A1 in decimal form to be read into A1's VT entry.

The control word with symbol

00000 4130 0000 00000 cr 40 02 25 25 25

or the control word

00000 4530 0000 00014

will cause the vector A2 with codeword in A2's VT entry to be printed in decimal form.

The control word with symbol

00000 5440 0000 00000 cr 40 03 25 25 25

or the control word

00000 5540 0000 00015

will cause the matrix A3 with primary codeword in A3's VT entry to be punched with symbol. The tape punched will load at a later time, creating a matrix with primary codeword in A3's VT entry, even if this entry is not in exactly the same relative VT location.

The control word with symbol

00004 0420 0003 00000 cr 40 03 25 25 25

will cause the space currently addressed by the codeword in A3's VT entry to be freed. Then a 4 by 3 matrix of zeroes to be created and addressed by the codeword in A3's VT entry.

The control word with symbol

00000 4100 0000 00000 cr 40 04 25 25 25

or the control word

00000 4500 0000 00016

will cause the program A4 with codeword in A4's VT entry to be printed out in octal.

The control word with symbol

00001 4030 0000 00000 cr 40 01 25 25 25

or the control word

00001 4530 0000 00013

will cause the scalar A1, stored in A1's VT entry, to be printed out in decimal.

The name of a double operand (such as a complex scalar or non-scalar) is attached to the first component of the pair. The second component is named "ditto" which is printed '←←←←←' and is represented by the hexad string

75 75 75 75 75 for ←←←←← ("ditto")

For any double operand, its name and "ditto" appear consecutively on the Symbol Table. If K is a double operand, given the name K, SPIREL will operate on the first component; then given the name "ditto", SPIREL will operate on the second component. To operate on the second component of K independent of the first, SPIREL must be given the name K before the name "ditto" with a control word. This may be accomplished by use of the monitor control word which designates no operation to SPIREL:

00000 3110 0000 00000 cr 52 25 25 25 25
nnnnn wxyz rrrr 00000 cr 75 75 75 75 75

If K has ST relative address 31, then "ditto" for K is at 32. The first component is addressed by the control word

nnnnn w5yz rrrr 00031

and the second component by

nnnnn w5yz rrrr 00032

Summary of Control Words

y \ w	0 read	1 correct	2 tagset	3 execute	4 print	5 punch
0	octal	octal	class 0	execute program	octal 4 wds/line	octal
1	hexad	hexad	class 1	on-line control word print only	hexad, 108 chars/line	hexad
2	zeroes	zeroes	class 2	activate STEX	octal-program format 1 wd/line	zeroes
3	decimal	decimal	all classes	reorganize	decimal, 5 wds/line	decimal
4	hexad + tag + checksum	hexad + tag + checksum	class 4	execute control word sequence	octal 3 wds/line $8\frac{1}{2}$x11"	hexad +tag + checksum
5		insert/ delete space	class 5	map codewords in STEX domain	octal with tag, 1 wd/line	high density hexad + tag + parity + checksum
6		change initial index	class 6	deactivate STEX	decimal with tag, 1 wd/line	
7		free storage	class 7	initialize STEX	decimal 3 wds/line $8\frac{1}{2}$x11"	

Input through the Console Typewriter

When a SPIREL system comes off magnetic tape, control is in the console communication loop, at PAUSE -- so named because the message *PAUSE* appears on the display scope. The blue light on the console typewriter will be on.

At any time, PAUSE may be obtained by going to location 23 or 24.

At PAUSE the console typewriter is used to input commands to the SPIREL SYSTEM. As text is input, it is displayed on the scope. The 'bs' (backspace) key causes the last character entered to be erased. Typing a question mark '?' causes the line to be erased. A carriage return or semi-colon ';' causes the accumulated command to be interpreted and printed.

To simply have text transmitted to the printer, '*' should be used as the first character.

The system interprets and obeys a variety of commands which may be input at the console:

a) Control commands to halt or have SPIREL read control words from paper tape;

b) SPIREL commands to have control words formed and passed to XCWD;

c) REGISTER commands to cause machine registers to be loaded;

d) MAGNETIC TAPE commands to search or read the system tape or pass control to manual mag-tape system;

e) ARITHMETIC commands to perform execution of arithmetic statements. See LIBRARY -- ←IFE.

It is important to understand that while in the console communication loop, at PAUSE or with unterminated text displayed, the only means of communication is through the console typewriter; field switches may not be used. Direct manual control may be exercised at 'HALT', obtained by issuing the 'halt' CONTROL command. At HALT the machine stops; all registers are set as directed by the user.

- Control Commands

Control commands are designated by single characters or keys. In all cases, the command is displayed on the scope and printed.

The control commands are:

halt -- (uc) H(cr)
: The machine stops with 'HALT' displayed in U. Lights and registers (except U,R,S,T7,P2,B6) are set as at the last halt, or as changed by intervening executions and register commands. B6 is set to the top of the SPIREL B6-list (*112).[†] Lights and registers may be reset manually at the halt. If a control word is typed into U, pushing CONTINUE causes the typed control word to be passed on to XCWD. If 'HALT' is left in U and there is paper tape in the reader, pushing CONTINUE causes one control word from tape to be passed on to XCWD and FETCHing causes control words on the tape to be processed until the end of tape or a null control word is encountered. Pushing CONTINUE with 'HALT' in U and no paper tape in the reader causes return to the PAUSE.

fetch -- (uc) F(cr) or "index"
: Control words are read from paper tape and passed to XCWD. Control returns to the console communications loop when the end of tape or a null control word is encountered. The fetch command is equivalent to halt and FETCH with the field switch.

cont -- (uc) C(cr)
: A single control word is read from paper tape and passed to XCWD. Control then returns to the console communications loop. The continue command is equivalent to halt and CONTINUE with the field switch.

[†]This occurs only if Console (*173) was entered at the 1st instruction or from location 23. If entry is at the 2nd instruction or from location 24, B6 is left as it was at entry.

• SPIREL Commands

SPIREL commands provide a convenient expression of control word input to XCWD. Each command is input as text followed by 't'[1] which causes the command to be interpreted and the corresponding control word to be passed to XCWD. The form of a SPIREL command text is a two-letter key followed by qualification information in which

n denotes name or octal codeword address
i,j,k denote octal numbers, complemented by minus sign
s denotes type or selected portion[2]
o denotes octal absolute address
f denotes format specifier
d denotes string of data words separated by commas[2]

The SPIREL commands are:

key	function	commands	
AS	activate STEX	AS	activate STEX
BD	base down	BD n s	set address base down to n
		BD n↓i,...j s	set address base down to $n_{i,...,j}$
BU	base up	BU	set address base up one level
		BU i	set address base up i levels
BZ	base to zero	BZ	set address base to zero
CB	create a block	CB i AT n s	create a block (no B-mods) of zeroes i long with codeword at n
CH	check bounds	CH n s	set tag to check bounds on n
CM	create a matrix	CM i BY j AT n s	create a standard matrix of zeroes i rows by j columns with codeword at n
CO	correct	CO n↓i,...,j s = d	correct data words into array n, starting at word $n_{i,...,j}$
		CO o s = d	correct data words into locations o, o+1,...

[1] carriage return or ';'
[2] Form explained in later section

key	function	commands	
CV	create a vector	CV i AT n s	create a standard vector of zeroes i long with codeword at n
CW	control word	CW i j k n s	send control word with N field=i, WXYZ field=j R field=k, F field=n or null with name n to XCWD
DS	deactivate STEX	DS	deactivate STEX
ER	erase	ER n s	erase array with code-word at n
EX	execute	EX n	execute program with code-word at n
		EX n↓i	execute n starting at word i
ID	insert/delete	ID i AT n↓j s	insert (i>0) or delete (i<0) |i| words at n_j
		ID i AT n s	insert (i 0) i words at end of n
		ID n↓i s	delete word n_i and all following
IN	initial index	IN i AT n s	set initial index of n to i
MO	monitor	MO n s	monitor all of n
		MO n↓i,...,j s	monitor word $n_{i,...,j}$
		MO i AT n↓j,...,k s	monitor i words in array n, starting at word $n_{j,...,k}$
		MO o A	monitor one word at location o
		MO i AT o	monitor i words starting at location o
MP	map	MP	map STEX domain
PR	print	PR n f s	print all of n
		PR n↓i,...,j f s	print word $n_{i,...,j}$
		PR i AT n↓j,...,k f s	print i words in array n, starting at $n_{j,...,k}$
		PR o A f s	print one word at location location o
		PR i AT o f	print i words starting at location o

key	function	commands	

where f blank causes printing in decimal
f = O causes printing in octal
H causes printing in hexad
P causes printing in program format

key	function	commands	
PU	punch	PU n f s	punch all of n
		PU n↓i,...,j f s	punch word $n_{i,\ldots,j}$
		PU i AT n↓j,...,k f s	punch i words in array n, starting at word $n_{j,\ldots,k}$
		PU o A f s	punch one word at location o
		PU i AT o f	punch i words starting at location o

where f blank causes punching in hexads with checksum
f = O causes punching in octal
H causes punching in hexads
Z causes punching of space-taking control words only

key	function	commands	
RE	reorganize	RE	reorganize STEX domain
ST	print ST	ST	print all of ST-VT
		ST U	print user's (positive) portion of ST-VT
		ST i AT j	print i words of ST-VT starting at relative location j
T0	set tag 0	T* n s	set tag on all of n
T1	set tag 1	T* n↓i,...,j s	set tag on word $n_{i,\ldots,j}$
T2	set tag 2	T* i AT n↓j,...,k s	set tag on i words in array n, starting at word $n_{j,\ldots,k}$
T3	set tag 3		
TR	set tag to trace		
		T* o A s	set tag on location o
		T* i AT o	set tag on i words starting at location o

where * = 0,1,2,3, or R

- Register Commands

Register commands cause machine registers to be loaded. Each command is input as text followed by 't' which causes the command to be interpreted and the specified register set. The form of a register command is

R = d

where R is the register name and d denotes a single data word (form explained in later section).

The registers which may be loaded and R for them are:

index registers   CC(effects transfer),B1,B2,B3,B4,B5

- Special Options

(1) Any SPIREL command of the form:

XX i AT n (↓u....w)

where XX is any valid command, may be written:

XX n (↓u...v) FROM j TO k

where j and k are program order numbers, etc. Thus the need to know the actual number of words (in octal) being operated on is eliminated. Note that j must be strictly less than k. For example:

PR 5 AT V↓2   would become

PR V FROM 2 TO 7

a more natural form.

(2) If a GENIE program is compiled with the SL[12] option, any internal variable name or statement label may be used as a subscript in a SPIREL command. However if a GENIE program was begun with RSEQ only statement labels may be used.

For example: if A is an internal constant in program F, one might say

PR F↓A

which will print the value of A. Or to trace the second FOR loop of program F:

TR F FROM ←FOR2 TO ←RPT2

thus eliminating the necessity to know either order number or how many words the loop contains.

- Magnetic Tape Commands

Magnetic tape commands provide communication with the magnetic tape system with manual control or for reading of the system tape and tape searching. (See separate section for details on MT System). Each command is input as text followed by 't' which causes the command to be interpreted and obeyed.

The magnetic tape commands are:

command	function
PL	read nearest PLACER
MT	go to MT 'arrow' halt
MT i	read block i (octal number) from MT system tape
SP	read nearest SPIREL
TS i	search (overlapped with computation) to block i (octal numbers) on MT system tape

- Data Input Formats

Data may be input from the console into registers (by the register commands) and into absolute locations and arrays (by the SPIREL command to correct).† Each data word specifies the content of one computer word and may be input in decimal, octal, or alphabetic form.

Decimal integers, as

1968, -29

may contain up to 18 characters and none may be '.' or '*' or space.

Decimal floating point numbers, as

.1, -2.95678, -0.5*6, 91.7*-9, 5*3

may contain up to 18 characters, no spaces, and either '.' or '*' or both must appear.

Octal numbers, such as

+252525, +01.01000.00.4001.77765, +10, +3120.0000.00135

may contain up to 18 digits after the '+' (which designates octal) and not including any '.' which may be used as a spacer. Leading zeroes are assumed if fewer than 18 digits are given.

Alphabetic words, such as

A BC DEF,* 123X YZ*

may contain up to 9 characters after the initial '*' (which designates alphabetic) and not including the final '*'. The final '*' may be omitted if the word ends the command. Trailing spaces are assumed if fewer than 9 characters are given.

†The number data words is found by actual count and any other number is ignored. For complex arrays or locations the data string is split in half, (the first half real and the second half imaginary), and the number of complex pairs is half the number of single words.

- Type Format (s)

The s type format specifier may be blank, R, I, or C. Blank means to determine the type, real or complex (double), and perform the operation accordingly. (Complex can only be implied if a name is on the symbol table.) R (real) and I (imaginary) mean perform the operation on that half of the complex named location or array. C (complex) means imply complex type on the name which causes both real and imaginary parts to be operated on. When a CO (correct) is used and complex may be implied, refer to page 8, Data Input Formats, for details.

- Mode Format (f)

A — absolute. If referring directly to a core address and not a name on the symbol table the A specifier must be used if the memory location is not to be taken as the location of a codeword, from which an array would be printed.

†H — hexad (BCD or literal)

†O — octal

†D — decimal

P — program (only for print)

- Errors

(1) REFERENCE MADE TO SYMBOLIC NAME NOT ON SYMBOL TABLE

†Meaningful only with print and punch command.

- <u>Normal Running</u>

  The procedure for running with SPIREL is usually as follows:

  1) load SPIREL from magnetic tape
  2) load private programs and any data which may be outside the STEX domain from paper tape
  3) if data is to be loaded before execution, activate STEX with the control word

     00000 3120 0000 00135

     to pack all items previously loaded and relegate all free storage to dynamic allocation by STEX -- then load data
  4) position "run tape" which contains control word to start execution, any data to be read under program control, and perhaps control words for further SPIREL operations
  5) CONTINUE to start system running; if STEX has not been activated, first execution will do so -- pack all items previously loaded and relegate all free storage to dynamic allocation by STEX

- Input Paper Tape Formats

The SPIREL read control words designate the read mode to be employed in reading the pertinent data from paper tape. For each read mode there is an appropriate punch format for the data on paper tape.

The octal format (y=0) prescribes that each word consist of exactly 18 octal digits and that each word be preceded by a "spill character", usually a 'carriage return' punch. Octal tapes may be punched manually on the flexowriter. Also, a punch control word with y=0 produces octal format on paper tape, but this is inefficient use of paper tape since only three channels are utilized.

The hexad format (y=1) prescribes that each word consist of exactly 9 hexads and that each be preceded by a "spill character", usually a 'carriage return' or 'tab' punch. The hexad format utilizes all six data channels on paper tape and is equivalent to the octal format at twice the density. Hexad tapes are usually not punched manually on the flexowriter but are easily produced through SPIREL with a punch control word in which y=1.

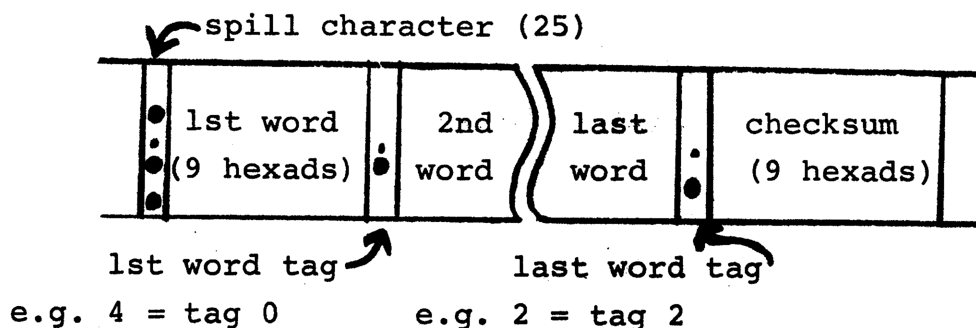
The hexad with tag and checksum format (y=4) is produced only by a SPIREL punch control word with y=4. For example, output tapes from the assembly program and the compiler are in this form. The advantages of this format over the plain hexad format are implied by the name:

- Tags on words are represented on paper tape and reproduced when the tape is read.
- A sum over all words is formed as the tape is punched and represented on the tape. This sum is recomputed when the tape is read and the computed sum is compared to that punched on the tape. This provides a check on both punching and reading.

The format for one word consists of 9 hexads and one tag triad per word, where tag representations are

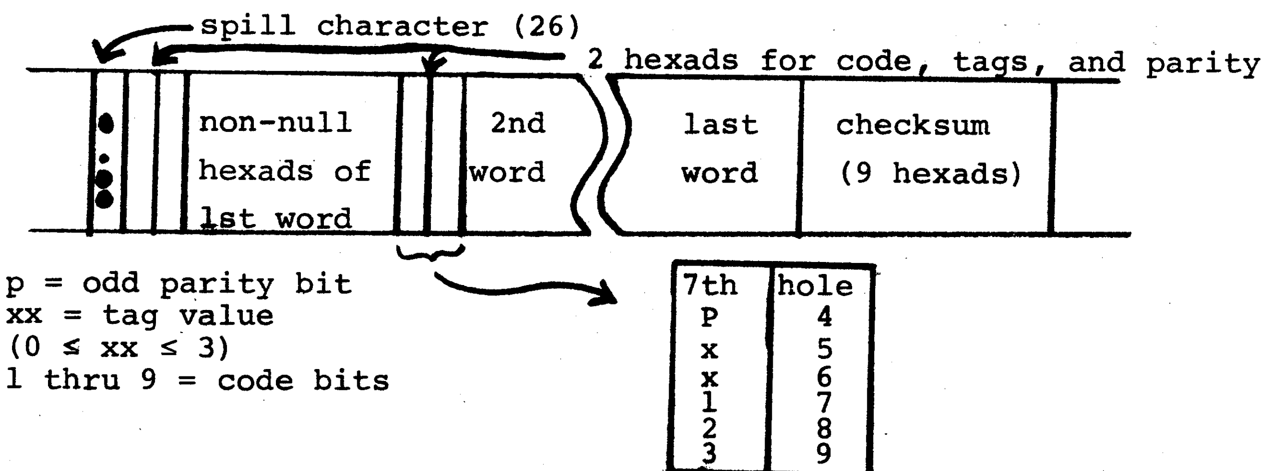
1 for tag 1
2 for tag 2
3 for tag 3
4 for no tag

A spill character (25) is always punched before the data. The physical format of the punched tape is shown below.



The <u>high density hexad with tag, parity, and checksum</u> format (y=5) is produced only by a SPIREL punch control word with y=5. The high density is obatined by not punching null hexads and is preceeded by two hexads of code, tags, and parity. Errors can be more readily detected since the total odd parity on the word, its tag, and the code is punched with each word. A spill character (26) is always punched before the data which is followed by the usual checksum. Tag 0 is punched as 0 and is added as a 0 to the checksum.

The code in the first two hexads of every word show the position of the non-null hexads in the original word in order to read the data back in. The 0's in the nine bit code show the position of the null hexads. Thus the number of hexads punched is the number of 1's in the code. The tapes punched in this format are virtually impossible to read by hand. The physical format of the high density punched tape is shown below.



The words with tag are not separated by any "spill characters" but are immediately adjacent to each other. A set of words punched with a single punch control word is preceded by a single "spill character' and followed by the checksum, one hexad word which is the fixed point sum of right half-word plus left half-word plus tag representation for all words in the set.

The decimal format (y=3) for a single word depends on the internal representation desired for the number. Tapes in decimal format may not be punched by SPIREL, but this is the format most frequently utilized for manually prepared data input tapes. A set of decimal words to be read due to the execution of a single read control word with y=3 is begun with a 'lower case' punch. Spaces and case punches are then ignored, and a punch other than one of

0 1 2 3 4 5 6 7 8 9 . + - e f t *

terminates a number which is being read. The character punched after each number to terminate it is most frequently a 'carriage return' punch. In the particular formats which follow, the letter d stands for a decimal digit, one of

0 1 2 3 4 5 6 7 8 9

The punches 'e' and '*' may be used interchangeably. If a decimal point is punched in representing a number, it may begin or end the number. The particular decimal formats are as follows:

integer ±D of no more than 14 decimal digits, with tag G=1,2,3,4. G=1,2,3 causes the number to be stored with tag 1,2,3. G=4 causes the number to be stored without changing the tag in memory.

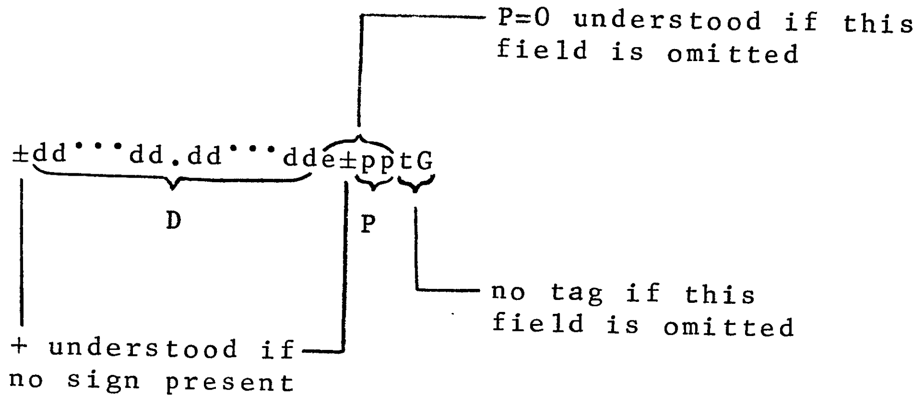
±dd···dddtG

D (under dd···ddd) — tG: no tag if this field omitted

± : + understood if no sign present

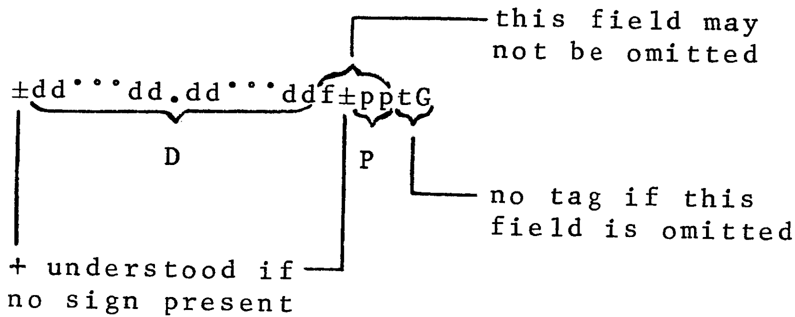
The decimal point is assumed to be to the right of the least significant digit punched and at the right end of the machine word. Integer arithmetic is to be employed.

<u>floating point</u> $\pm D\times10^{\pm P}$, in absolute value between 10^{-78} and 10^{74}, with tag G=1,2,3,4



Decimal digits in D beyond the 14th are ignored. Floating point arithmetic is to be employed. The floating point form is recognized only if D contains a decimal point or if the field e±pp (or *±pp) appears.

<u>fixed point fraction</u> $\pm D\times10^{\pm P}$ in absolute value $\geq 2^{-47}$ and<1, with tag G=1,2,3,4



Decimal digits in D beyond the 14th are ignored. The representation in the machine assumes that the decimal point is at the left end of the mantissa. Fixed point arithmetic is to be employed.

- <u>Tracing</u>

Tracing is a means of observing the execution of instructions. This facility is provided in the SPIREL system. The instructions to be traced must bear a tag 3. Mode light 3 and trapping light 6 must be turned on prior to a run in which tracing is to occur; this is the normal ML, TL configuration when SPIREL is initially loaded. The trace is provided by the SPIREL system program *13 (TRACE).

Trace options exist to obtain arithmetic register content (U,R,S) in either octal or decimal and B-register content (B1,...,B6,PF) in octal. For each tagged instruction the first five fields of trace output are as follows:

(CC): address of the instruction relative to the first word of the last program tagged with a tagset control word

(CC'): address of the next instruction to be executed relative to the first word of the last program tagged with a tagset control word; printed only for non-sequential transfer

(M'): the final address formed as a result of decoding field 4 of the instruction

(ATR): contents of the arithmetic tag register (1,2, or 3) after field 4, <u>not</u> after execution of the instruction; blank if the tag is zero

(TF'): T-flags, if they are on; printed as two octal digits, the rightmost bit for T7, next for T6, etc.

(I): the instruction executed, formatted into fields

The arithmetic register trace then provides in octal or decimal:

(S): the contents of S as a result of execution of field 4 of the instruction, before the operation in field 2 is carried out

(U'): contents of U after execution of the instruction

(R'): contents of R after execution of the instruction

And the B-register trace provides (B1'), (B2'), (B3'), (B4'), (B5'),

(B6'), (PF') --- the contents of the seven B-registers after execution of the instruction.

Mode Lights may be used at the console for selections of trace options.

To erase the tag from each instruction traced, turn on ML15.

To produce trace output only if a branch of control occurs, turn on ML14.

To obtain B-register trace only, turn on ML13.

To obtain arithmetic trace in decimal, turn on ML10; this option is effective only if T-flags are not present on T4, T5, and T6; this option uses the SPIREL print matrix (*116) and should not be used on programs which are doing set-ups for printing.

To obtain both arithmetic and B-register trace (two lines per instruction traced) turn on ML9 with ML13 off.

The trace procedure saves all machine registers, even the T-flags, except S and P2. If the contents of S is to be preserved from one instruction to the next, neither may be traced. Orders which set P2 or assume that P2 in unchanged may not be traced --- with one exception, that an order which sets P2 to be used only as a transfer address may be traced, but subsequent orders which assume an unchanged P2 may not be traced. This exception allows transfers to the SPIREL programs *136(SAVE) and *137(UNSAVE) to be traced, but these routines themselves may not be traced.

An order to be executed in the repeat mode may be tagged for tracing only if the order which causes entry into the repeat mode is also traced and is of the form

MLN

or SBi,ERM

or ABi,ERM

and the instruction executed immediately after the repeated instruction is also traced. The trace output for the traced repeated instruction consists of one printed line in which (CC), (ATR), (I), and (S) pertain to the first execution and (CC'), (TF'), (M'), (U'), (R'), and (Bi') pertain to the last execution.

* <u>Arithmetic Error Monitor</u>

While running with SPIREL the following arithmetic error conditions may be monitored: mantissa overflow, exponent overflow, and improper division. To monitor a condition the corresponding trap light must be set on: respectively MOV, EOV, and ÷. If an error which is being monitored occurs, a message is printed. Information provided includes the error made, the location of the instruction generating the error, and the name or codeword address of the program containing that instruction.

• Block Bounds Check

While running with SPIREL, addressing into arrays may be monitored to check that references are not made outside the bounds of the array. This is accomplished by placing a tag 1 on the primary codeword for the array and running with trapping light 13 on, as when SPIREL is initially loaded. Checking is provided by the SPIREL program *20(CHECK).

If a tag 1 is encountered in indirect addressing in field 4 of an instruction, the program CHECK gets control through a hardware trap. The index value k at each level of indirect addressing is then checked to see that

$$i \le k < i+n$$

where i is the initial index and n is the length in the codeword for that level. If an error is detected, information is printed which includes the array name, index values at each level, the location of the offending instruction, and the name or codeword address or the program containing that instruction. Storing out of bounds is inhibited; otherwise, the checked instruction is executed normally.

To successfully check bounds on an array, the tag on the codeword must be preserved during execution. All SPIREL and library routines and all Genie-generated code preserve tags on codewords. Preservation in private code is the responsibility of the user.

The program CHECK uses control trapping on tag 1 after execution of the checked instruction, but this trapping facility may simultaneously be applied to non-checked instructions by the user.

• <u>Diagnostic Dump</u>

If a program stops unexpectedly, the contents of machine registers, the codeword region, the B6-list, and a map of the STEX domain may be printed by using the diagnostic dump, SPIREL system program *120, with entry prefix in machine locations 00027-00036 (octal). The procedure for obtaining this output is as follows:

--Record, mentally or otherwise, the contents of CC as displayed on the console if the instruction at which the stop occurred is of interest.

--Type 00027 (octal) into CC and FETCH to pass control to the diagnostic dump routine.

--A programmed halt occurs within the diagnostic dump routine. Type the recorded (CC) into U and CONTINUE; or simply CONTINUE if (CC) was not recorded.

--Diagnostic dump output is provided on the printer:

Register contents, (CC) and (I) given as 0 if nothing was typed into U at the halt.

Codeword region, with SPIREL address base shown in heading output.

B6-list.

STEX domain map.

--Control is returned to the loop for console communication with SPIREL. Registers are restored as follows: T-registers without flags except T7, B-registers except CC and B6 and PF, special purpose registers except P2, all lights. The SPIREL address base is set to zero, and B6 is set to the first word of the B6-list. The user may continue to use SPIREL.

• High-Speed Memory Dump

If a program fails in such a way that the SPIREL system cannot be reached, a printed record of the memory configuration at the time of the failure is occasionally of assistance in debugging. For this purpose a self-loading High-Speed Dump tape is available at the console.

To load the tape, position the Dump tape in the reader, depress RESET, then LOAD. Do not CLEAR. The program loads at 57400. The contents of machine registers are printed out, and a halt occurs with

(U): 57400 0000 0000 0010

Pushing CONTINUE causes dumping of the contents of memory, from location 10 to location 57400. To change this dump range type into U at the halt type the new upper bound in the first five traids and the new lower bound in the last five traids.

The dump output is printed with four full words per line and the address of the first word at the left of the line. Each full word is split into five fields, corresponding to the fields of the machine instruction. If a word is tagged, an a, b, or c (corresponding to tag 1, tag 2, or tag 3) is printed immediately to the right of the tagged word.

Error Messages in SPIREL

There are no error halts in the SPIREL system. Nearly all error conditions result in an error message being printed and a return through location 24 to the console communication loop. The printing results in a minimal disturbance to the system so that the source of the error can better be determined.

The message is printed by ERPR *172. Where possible ERPR determines the program and order number where the error originated and prints this information with the message. The possible error messages are listed below and are self explanatory:

1. CHECKING ERROR ON PAPER TAPE INPUT,
2. INSUFFICIENT SPACE IN MEMORY FOR READ,
3. INSUFFICIENT SPACE IN MEMORY FOR INSERT,
4. IMPROPERLY FORMATTED DECIMAL NUMBER,
5. ORDER TRACED WHICH REQUIRED P2 SAVED,
6. ATTEMPT TO SET ADDRESS BASE TO FIFTH LEVEL,
7. ATTEMPT TO SET ADDRESS BASE BACK TOO MANY LEVELS,
8. ATTEMPT TO SET ADDRESS BASE BACK WHEN SET TO ZERO,
9. ATTEMPT TO TAKE BLOCK OF ZERO LENGTH,
10. ATTEMPT TO EXECUTE NON-EXISTENT PROGRAM,
11. ATTEMPT TO SET ADDRESS BASE TO NULL CODEWORD,
12. ATTEMPT TO SET ADDRESS BASE PAST LAST CODEWORD LEVEL,
13. REFERENCE MADE TO SYMBOLIC NAME NOT ON SYMBOL TABLE,
14. SYMBOL TABLE-VALUE TABLE IS FULL,
15. IMPROPER MEMORY CONFIGURATION.

It is not suggested to use the system after any of the above error conditions occur without first rectifying the error.

- Symbol Table-Value Table Print Format

The SPIREL control word

nnnnn 45y0 rrrr 00000

provides parallel printing of corresponding Symbol Table (ST) and Value Table (VT) entries. These "tables" are SPIREL system standard vectors *113 and *122 respectively. The output is in seven fields as follows:

(1) relative address in vector, i.e., index
(2) symbol from bits 1-30 of ST entry
(3) bits 31-39 of ST entry in octal:
    000 if VT entry contains a scalar
    400 if VT contains a codeword
(4) address field of ST entry, giving the absolute address of the corresponding VT entry in octal
(5) tag on ST entry: 0 if item is in context, 1 otherwise
(6) VT entry: decimal value associated with name in ST entry if it is a single-word quantity; codeword (in octal) for array associated with name in ST, empty if the array does not currently exist.
(7) tag on VT entry

nnnnn 46y0 rrrr fffff

provides parallel printing of the ST-VT whose ST codeword is at fffff. The VT codeword must be in the memory location immediately following the ST codeword.

5/7

- SPIREL System on Magnetic Tape

Copies of the SPIREL system for 24K memory are located on the MT System magnetic tape. When one of these copies is read into the memory, control is in the console communications loop. The Mode and Trapping Lights are set to permit tracing of tagged instructions and block bounds checking. Loading of "private" blocks will begin at about location 10000 (octal) and may extend to location 57400 (octal).

● <u>Linear Consumption by TAKE</u>

The simple storage control algorithm in the SPIREL system is called TAKE (in program *135). TAKE is given all of inactive storage as its domain when STEX is deactivated. TAKE operates on the principle of linear consumption of memory. A pointer to the first inactive word of storage, address L, is maintained. A request for M words is satisfied by an allocation of M words at L, and L is incremented by M. This is an irreversible procedure in that space, once allocated, may not be reclaimed for use in later allocations. L is stored in the address field of STORAG, location 100 (octal).

The upper bound on allocatable storage is specified by the contents of location 102 (octal):

LASTEX=last allocatable address +1

In the standard SPIREL only the magnetic tape system communication routine is above allocatable storage. In producing a SPIREL (see System Duplicator section) any upper bound may be specified.

TAKE may be utilized "privately" to obtain blocks of memory in a way compatible with allocation by the SPIREL system. On entry to program *135, (B2)=number of words desired. On exit (B1)=address to first word of block allocated. If the request for a space cannot be satisfied, (B1)=0 on exit.

- <u>Activation of STEX and its Domain</u>

The SPIREL Storage Exchange algorithm STEX (in *135) is activated by the control word

00000 3120 0000 00135

If L is the address of the first word of inactive storage at the time STEX is activated, the STEX domain is [L,(LASTEX)-1], all inactive storage. The address of the first word in the STEX domain is stored in the address field of FIRSTEX, location 101 (octal). Deactivation of STEX for recourse to the TAKE system is explained in a later section. STEX provides optimal use of storage because

- blocks may be freed explicitly, making such space as becomes logically unnecessary available for reassignment to blocks logically required;
- blocks whose codewords are re-used to label new blocks are automatically freed for re-use;
- if the total free space in memory is sufficient to satisfy a request for a block but the "first" free block (explained later) is not large enough, free space is automatically concentrated and the allocation is made.

<u>Any</u> block may be freed at any time. Only if STEX is active and the block is <u>in</u> the STEX domain will the space be available for re-use in later allocations.

The first STEX activation in a fresh SPIREL has special effect if it precedes the use of an execute control word. All memory in use is first concentrated outside the STEX domain which is defined as all free memory. Items loaded prior to this first STEX activation may be moved to fill gaps of free space.

FIRSTEX indicates whether or not STEX is active:

(FIRSTEX) null if STEX is not active

(FIRSTEX) not null if STEX is active

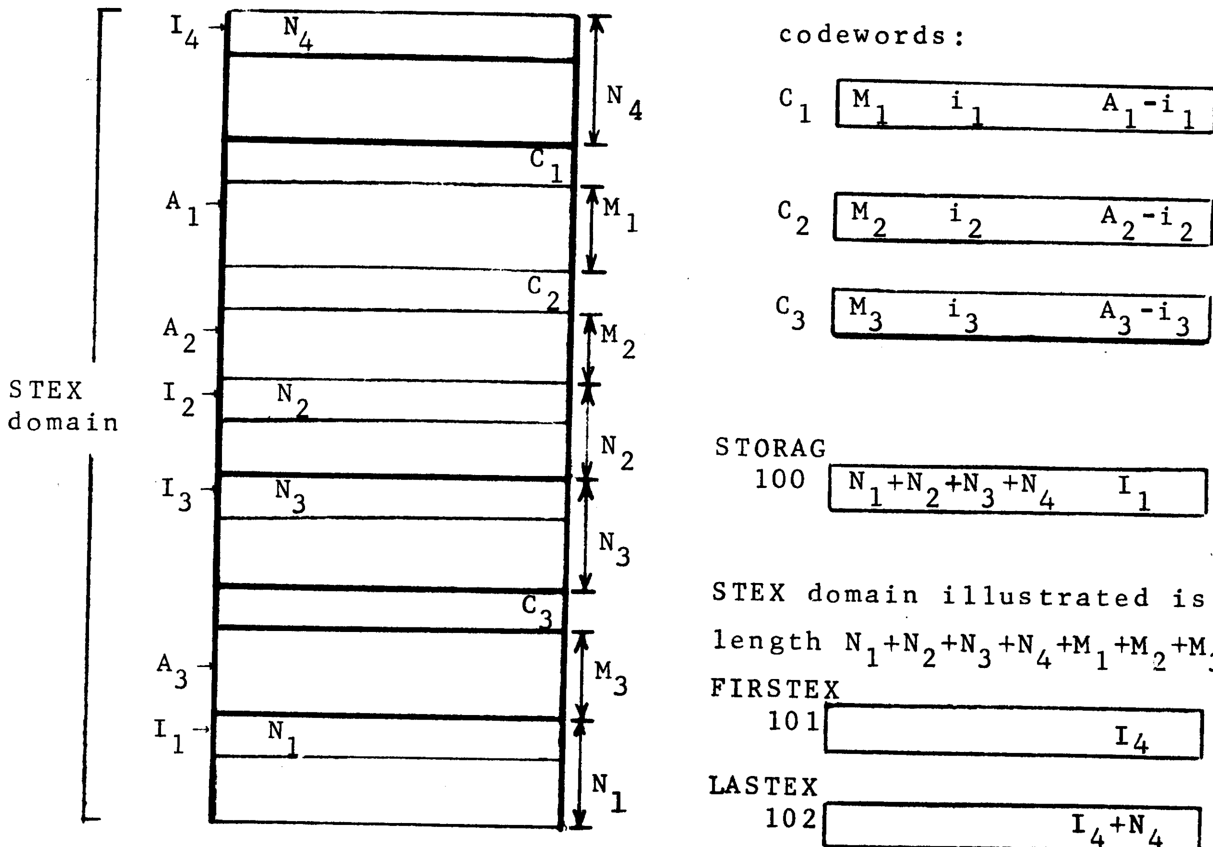
with exponent not null prior to first execute or activate STEX control word

with exponent null after user activation

- Memory Configuration Generated by STEX

The domain of STEX is divided into active and inactive areas. Each area is further segmented into blocks, not necessarily adjacent. Each active block is labelled by a codeword. The first word of an active block is not a part of the block from the user's point of view; it is called the back-reference for the block and contains in its address field the codeword address of the block. One inactive block may be found from location 100, which contains the total number of inactive locations in the memory in its first 15 bits and the first word address of this inactive block in its last 15 bits. Each inactive block contains in its first word the length of the block it heads in the first 15 bits. The first word of an inactive block is inactive in the sense that it may be activated just as any other word in the block.

An illustration of the STEX memory configuration is given below:



In general, the length of the STEX domain is given by

$$K = \sum_{i=1}^{n} N_i + \sum_{j=1}^{m} M_j + m$$

where the domain is divided into n inactive blocks and m active blocks. Note that K is a constant, determined at the time STEX is activated, and K = (LASTEX)-L, where L is the address of the first word of inactive storage at the time STEX is activated.

The codewords that address the active blocks may be located anywhere in the memory, but the blocks of every array must lie wholly inside or outside the domain of STEX.

A duplicate codeword may exist for any array. Its existence is indicated by a duplicate back-reference stored in the same word as the normal back-reference but in the 15 bits adjacent to the address field. Inactivating the duplicate will leave the array unchanged and clear the duplicate. Inactivating the original codeword will have the additional effect of clearing the duplicate.

If it is necessary to move a codeword for an active block, the back-reference for the block must be appropriately altered. For instance, when the block addressed through codeword A is activated the back-reference for the block contains the address A; if (A) is stored at B, back-reference for the block must be changed to B.

STEX offers many advantages for data storage control, but programs may also be loaded into the STEX domain with a few restrictions. Since pathfinder settings are absolute and return is not made through codewords, programs should not be moved in a memory reorganization which STEX may have to perform. This possibility is eliminated if programs are loaded before any space is taken for data that may ever be inactivated. This rule should always be followed.

- Use of STEX

Once activated, STEX may be used directly by the coder with entry parameters

(B1) = codeword address of block on which STEX is to operate,

(B2) = length of block.

STEX first tests the word addressed by (B1). If this codeword is not null, the storage addressed through this codeword to all sub-levels is inactivated by STEX and all codewords are made null. If the codeword is null, no inactivation occurs. Then (B2) is tested. If (B2) $\neq$ 0, a block of storage of length (B2) is activated with back-reference to the address (B1), the codeword at (B1) remains null.

If the old codeword at (B1) had no a-bit and if sufficient inactive storage is available in the high order locations adjacent to the old first word address then the new FWA will be the same as the old. Note that in the case of a new array of size less than or equal to the old array addressed through the appropriate codeword address, the user is assured that the freed space will be reused. In any case (B1) is set to the FWA of the block activated.

To take N locations to be addressed through codeword C, set (B1) = C, (B2) = N and enter STEX. All space formerly occupied by array C will be inactivated and all associated codewords cleared. Exit will be made with (B1) = FWA of a new block to be addressed through C and (B2) unchanged. The back-reference for the new block is supplied by STEX. If the inactive area is not sufficiently large to meet a request for space, exit is made with (B1) = 0.

To simply inactivate memory addressed through C, enter STEX with (B1) = C and (B2) = 0.

Freeing a block is accomplished by inserting an inactive header in place of the back-reference word. Activating a block is accomplished by obtaining space from the inactive block addressed by 100. If the address portion of 100 is null, then a reorganization will occur before activation. In this case all active memory is written to the low address end of the STEX domain, leaving one inactive block at the high address end. Space is then obtained from this new (and only) inactive block. Reorganization will also occur if the block addressed by 100 is not large enough to accommodate the request for storage.

If reorganization occurs and SL14 is off, the message REORGANIZATION is printed. If (B1) = 0 on entry to STEX, reorganization is performed and no space is allocated. A reorganization may also be forced by the control word

00000 3130 0000 00135

When STEX is used directly, the coder must generate his own codewords. The alternative of taking space with a "read" control word provides generation of codewords for the coder.

- <u>Deactivation of STEX</u>

The SPIREL Storage Exchange algorithm STEX may be deactivated by the control word

00000 3160 0000 00135

If STEX is not active, this control word has no effect. If STEX is active, the following procedure is carried out:

--The STEX domain is reorganized so that all inactive space is collected in a single block at the high end of the domain.

--The TAKE algorithm is reinstated.

--STORAG is set so that linear consumption will commence from the beginning of currently inactive storage, and FIRSTEX is cleared to indicate that there is no STEX domain established.

Two points should be carefully noted:

--Deactivation of STEX involves reorganization, so absolute addresses in the STEX domain may become meaningless as a result of this operation. In particular, a program in the STEX domain should not ask for deactivation of STEX.

--Items in the STEX domain at the time of deactivation will not be in the domain if STEX is subsequently reactivated. Each activation creates a new <u>empty</u> domain beyond all storage in use.

It is difficult to imagine an application of STEX deactivation while a system is running. But a very useful application is in maintenance of a system of programs. A collection of system items may be loaded with STEX active and kept on magnetic tape as a master. For running, STEX may be deactivated prior to loading further. But items may be deleted, added, and changed in the master without any wasted space.

• Vectors, Print Matrix, B6-List

Any component which is of use to the individual programmer is denoted by Δ in the margin next to its name.

Δ *112, B6-List, LISTB6

Length: 200 (octal)

Function: This block is not B-modified. The area is used for working push-down storage, called the B6-list. Index register B6 is initially set to point to the first word in the block. The B6 setting is maintained dynamically as a pointer to the next word in the block which may be used for push-down storage.

*113, Symbol Table, ST

Length: 400 (octal)

Function: This is a standard B1-modified vector. Each entry contains the name and descriptive parameter for an item in the total system being run, an item which is identified symbolically rather than by its address or codeword address. The parallel entry in the Value Table, *122, contains the item or codeword corresponding to the item name in the Symbol Table. The index of the last active entry in the Symbol Table is dynamically maintained at location 117.

Δ *116, Print Matrix, PM

Length: 200 (octal)

Function: This block is not B-modified. The address of the first word of *116 is used as the address field in all SPIREL print orders, except in octal tracing. The print matrix is always cleared immediately after SPIREL-controlled printing.

*122, Value Table, VT

Length: 400 (octal)

Function: This is a standard B1-modified vector. Each entry contains the value of or the codeword for an item in the total system being run, an item which is identified symbolically rather than by address or codeword address. The parallel entry

in the Symbol Table, *113, contains the name and descriptive parameter for the item. The index of the last active entry in the Value Table is dynamically maintained at location 117.

*125, Base Address Vector, ADDR

Length: 6

Function: This block is not B-modified. It is used by SPIREL to dynamically maintain a record of all levels through which the base address has been set down and to compute effective addresses from those specified in control words.

*174, Text Vector, TEXT

Length: 14 (octal)

Function: This is a standard B1-modified vector. It contains the current text supplied by *171, SAMPLE, which *173, CONSOLE operates on. The text is in the form of printer hexads.

- Programs

Since programs *135(STEX), *136(SAVE), and *137(UNSAVE) are necessary components of the SPIREL system, they are not included in the lists of supporting routines.

*13, Trace, TRACE

Length: See SYSTEM DUPLICATOR, P. 4.

Function: The SPIREL trace program receives control through hardware trapping due to tag 3 on instructions, both before and after execution of the instruction. Information for each line of trace output is derived, formatted, and printed by this program.

Registers Not Preserved: P2,S

Supporting Routines: ARITH(*14), SETPM(*127), PM(*116) for decimal option only

*14, Arithmetic Error Monitor, ARITH

Length: See SYSTEM DUPLICATOR, P. 4.

Function: This program receives control through hardware trapping due to exponent overflow, mantissa overflow, or improper division. The type and source of error are printed.

Registers Not Preserved: P2,S

Supporting Routines: SETPM(*127), PFTR(*147)

*20, Check Block Bounds, CHECK

Length: See SYSTEM DUPLICATOR, P. 4.

Function: This program receives control through hardware trapping due to tag 1 on codewords as they are used in indirectly addressing. The values of indices on each level of indirect addressing are checked for being legal. If an error is detected, the source of the error, the array addressed, and index values are printed.

Registers Not Preserved: P2,S

Supporting Routines: TRACE(*13), SETPM(*127), PFTR(*147)

*110, Print Control Word, HDPR

Length: See SYSTEM DUPLICATOR, P. 4.

Function: This program is used in the SPIREL system for online control word monitoring when SL14 is off.

Supporting Routines: SETPM(*127)

*111, Process Matrix, MATRX

Length: See SYSTEM DUPLICATOR, P. 4.

Function: This program is used in the recursive application of SPIREL as explained elsewhere.

Supporting Routines: all SPIREL components, but only those necessary to perform the specified operation on any particular utilization; see section on SPIREL Component Linkages.

*120, Diagnostic Dump, DIADMP

Length: See SYSTEM DUPLICATOR, P. 4.

Function: This program is used in the SPIREL system when control is passed to location 00027 (octal). The diagnostic dump formats and prints the contents of the fast registers as explained in the section on Use of SPIREL.

?IREL

Δ *126, Execute Control Word, XCWD

Length: See SYSTEM DUPLICATOR, P. 4.

Function: This program is the nucleus of the SPIREL system. It interprets control words and may use other system programs to carry out specified operations. External communication to XCWD from paper tape and from the console is provided within the system and is explained elsewhere. For internal communication, control should be passed to the second word of *126 if the SPIREL operation specified is to be performed on a named item; otherwise control is given to the first word of *126.

Input: (T7)=SPIREL control word to be executed.
(R)=5 left-adjusted printer hexads (with '25' fill if necessary) for name of item to be operated on if control is given to *126 at the second word; in this case F in the control word in T7 is empty.

Registers Not Preserved: none (and SPIREL cannot operate on fast registers)

Supporting Routines: all SPIREL components, but only those necessary to perform specified the operation on any particular utilization; see section on SPIREL Component Linkages.

Δ *127, Set Up Print Matrix, SETPM

Length: See SYSTEM DUPLICATOR, P. 4.

Function: This program will format a single word into the print matrix for printing at a specified position on a line if appropriately instructed. It will print the contents of the print matrix and clear the print matrix if appropriately instructed. Common usage of SETPM for the printing of a single line consists of an entry for each word of information to be printed and an entry to have the collection printed and the print matrix cleared. Thus, SETPM provides a facility for com-

position and output of lines on the printer.

Input: (T7)=information to be set up in print matrix, if any

(B1)=parameter which controls operation of SETPM

(B3)=print position, 1-108 (decimal) at which field set-up should begin, if relevant (print position ≤ 0 is permitted, in which case nothing set up to the left of position 1 is printed)

Operation: on the basis of (B1) on entry:

†(B1)<0, octal format of last 5 triads of (T7) at print position (B3), and increment (B3) by 5

entry (B3) → ooooo ← exit (B3)

1-5 octal digits, filled with leading blanks

†(B1)=0, octal format of (T7) in 18-position field at print position (B3), and increment (B3) by 18

entry (B3) → oooooooooooooooooo ← exit (B3)

1-18 octal digits, filled with leading blanks

†(B1)=1, hexad format of (T7) in 9-position field at print position (B3), and increment (B3) by 9

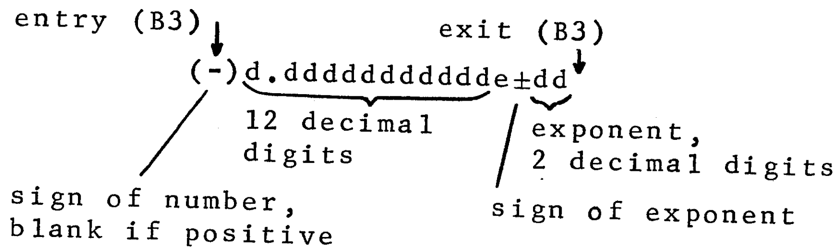
entry (B3) → hhhhhhhhh ← exit (B3)

9 hexads

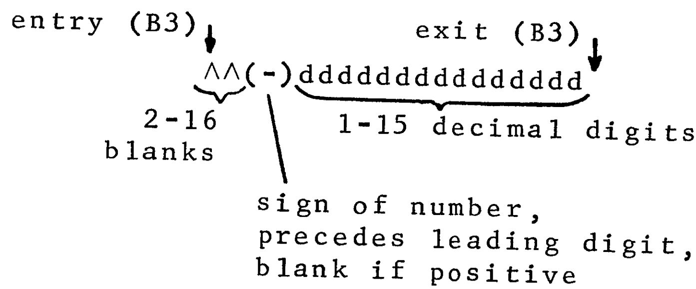
(B1)=2, print and clear PM, (B3) set to one and (T7) meaningless

†(B1)=3, general long decimal format of (T7) in 18-position field at print position (B3), and increment (B3) by 18

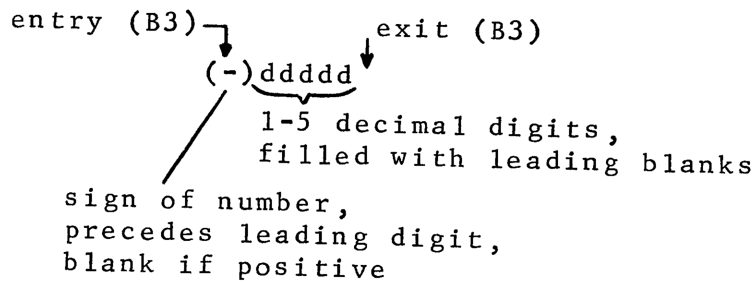
floating point --



fixed point --

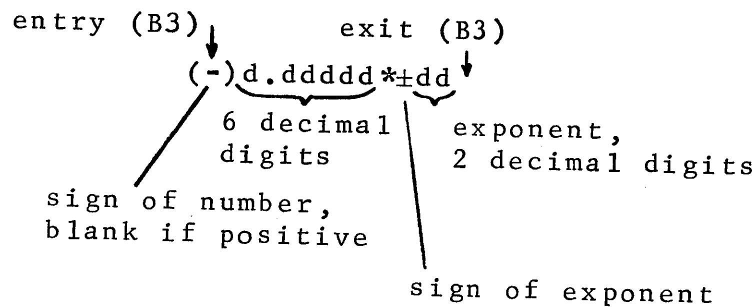


†(B1)=4, short decimal integer format of mantissa of (T7) in 6-position field at print position (B3), and increment (B3) by 6

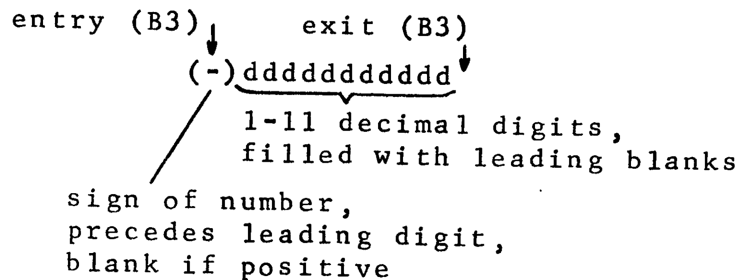


†(B1)=5, general short decimal format of (T7) in 12-position field at print position (B3), and increment (B3) by 12

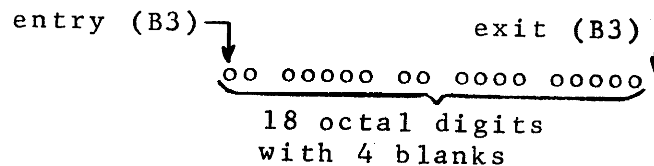
floating point --



fixed point --



†(B1)=6, octal instruction format of (T7) at print position (B3), and increment (B3) by 22



†(B1)=10 (octal), format of (T7) on basis of (B2) at print position (B3):

(B2) < 0	as for (B1) < 0, short octal but with leading zeroes printed (no leading blanks)
(B2) = 0	as for (B1) = 0, long octal but with leading zeroes printed (no leading blanks)
(B2) = 1	as for (B1) = 1, hexad but with leading zeroes deleted (filled with leading blanks)

†print and clear PM after set-up if SETPM entered at second word.

Registers Not Preserved: T7

Supporting Routines: BINDC(*155) when decimal formatting is used.

*130, Find Symbolic Name, SMNAM

Length: See SYSTEM DUPLICATOR, P. 4.

Function: This program is used in the SPIREL system for operations which are performed on items with symbolic names.

Supporting Routines: TLU(*176)

Δ *131, Print Date and Time, DATIME

Length: See SYSTEM DUPLICATOR, P. 4.

Function: This program reads the digital clock through

*132 (CLOCK), sets up the 14-character date and time in the print matrix, and prints the contents of the print matrix if requested to do so. It is used by XCWD to perform "obtain date and time" operations; it may also be used directly.

Input: T7=00000 00p0 rrrr 00000,

where p=0 causes set-up only
p=1 causes set-up, print, and clear
r specifies the print position of the date
r=0 causes set-up for 8 1/2 x 11" pages
(first character in print position 48)

Registers Not Preserved: T7

Supporting Routines: CLOCK(*132), SETPM(*127)

Δ <u>*132, Decode Clock, CLOCK</u>

Length: See SYSTEM DUPLICATOR, P. 4.

Function: This program interrogates the digital clock and calendar in the machine and translates the coded time and date into printer hexads. The time is based on a 24-hour clock. CLOCK is used by DATIME(*131).

Input: none

Output: Printer hexads in T5, T6. For example, if CLOCK were executed at 9:45 pm on May 17, 1964, the CLOCK output would be:

T5=05/17/64_

T6=21.45_ _ _ _

where _ denotes "space".

Registers Not Preserved: B2,T4,T5,T6

Supporting Routines: none

<u>*133, Punch Control Word, PCNTRL</u>

Length: See SYSTEM DUPLICATOR, P. 4.

Function: This program is used in the SPIREL system for any punch operation executed with z even.

Supporting Routines: none

Δ ***135, Storage Exchange, STEX**

Length: See SYSTEM DUPLICATOR, P. 4.

Function: This program performs storage allocation as explained in detail in the section on Storage Control.

Input: (B1)=codeword address of array or block which is to be freed or for which space is to be allocated; 0 if reorganization is desired.

(B2)=length of block to be allocated; 0 to only free space.

Output: (B1)=address of first word of block allocated; 0 if allocation requested and insufficient space available; same as entry if no allocation is requested.

(B2) same as on entry.

Registers Not Preserved: none

Supporting Routines: SETPM(*127)

Δ ***136, Save Fast Registers, SAVE**

Length: See SYSTEM DUPLICATOR, P. 4.

Function: This program uses 12(octal) words on the B6-list for storage of all fast registers and a word denoting the registers to be saved.

Input: (R), bits 46-54, to specify registers to be saved:

45 46 54

not meaningful to SAVE

The registers are stored on the B6-list in the order shown from right to left (i.e., PF saved first), and then (R) is itself stored on the B6-list. Notice that (R)=-Z on entry causes all nine registers to be saved.

Use: Control should be passed to SAVE by a TRA (not a TSR) instruction which may be traced.

Registers Not Preserved: none

Supporting Routines: none

Δ *137, Unsave Fast Registers, UNSAVE

Length: See SYSETM DUPLICATOR, P. 4.

Function: This program complements SAVE. It obtains from the B6-list the (R) stored by the complementary execution of SAVE, restores all fast registers designated to be saved, and decrements B6 by 12(octal). The T-flags are cleared by UNSAVE.

Use: Control should be passed to UNSAVE by a TRA (not a TSR) instruction. UNSAVE returns via (P2) on entry, and the instruction

TRA *137

may be traced.

Registers Not Preserved: none

Supporting Routines: none

*140, Insert or Delete Space, DELETE

Length: See SYSTEM DUPLICATOR, P. 4.

Function: This program is used in the SPIREL system for the operation of inserting or deleting words in blocks.

Supporting Routines: TAKE or STEX(*135) for insert

*141, Change Initial Index, CHINDX

Length: See SYSTEM DUPLICATOR P. 4.

Function: This program is used in the SPIREL system for the operations of changing the initial index of a block and loading cross reference words of programs.

Supporting Routines: TLU(*176) for programs with cross references

*142, Tagset, TAGSET

Length: See SYSTEM DUPLICATOR, P. 4.

Function: This program is used in the SPIREL system for the operation of tag setting.

Supporting Routines: none

*143, Convert from decimal, CNVRT

Length: See SYSTEM DUPLICATOR, P. 4.

Function: This program is used to convert from a general decimally formatted number to binary.

Input: T6, T7 contain printer hexads representing decimal number. R = number of hexads. Hexads are assumed to be left justified.

Error Message: IMPROPERLY FORMATTED DECIMAL NUMBER

Output: Binary number in T7.

Resisters Not Preserved: X, T7.

Supporting Routines: PWRTN(*152)

*144, Print, PRINT

Length: See SYSTEM DUPLICATOR, P. 4.

Function: This program is used in the SPIREL system for all print operations.

Supporting Routines: SETPM (*127)

*145, Punch, PUNCH

Length: See SYSTEM DUPLICATOR, P. 4.

Function: This program is used in the SPIREL system for all punch operations.

Supporting Routines: CLOCK(*132); PCNTRL(*133) for punching tapes with control words (i.e., with z even); PUNCHK(*157) for punching tapes in the hexad with tag and checksum format; BINDEC (*155) for punching decimal tapes.

*146, Execute Control Word Sequence, XCWSQ

Length: See SYSTEM DUPLICATOR, P. 4.

Function: This program is used in the SPIREL system for the operation of executing a control word sequence.

Supporting Routines: all SPIREL components, but only those necessary to perform the operations specified by the control words in the sequence being executed; see section on SPIREL Component Linkages.

*147, PF Trace, PFTR

Length: See SYSTEM DUPLICATOR, P. 4.

Function: This program is used to determine relative value of PF in a program with named or numbered codeword.

Supporting Routines: none

*150, Map STEX Domain, MAP

Length: See SYSTEM DUPLICATOR, P. 4.

Function: This program prints the structure of all arrays in the STEX domain, i.e., codewords.

Supporting Routines: XCWD(*126),SETPM(*127).

*151, Print Symbol and Value Tables, PRSYM

Length: See SYSTEM DUPLICATOR, P. 4.

Function: This program is used in the SPIREL system for the printing and punching of ST and VT in the special format described elsewhere.

Supporting Routines: SETPM(*127), CNTXT(*175)

Δ *152, Conversion of Powers of Ten, PWRTN

Length: See SYSTEM DUPLICATOR, P. 4.

Function: This program is used in the SPIREL system for reading of decimal numbers from paper tape. Given the floating point number P and the integer Q, this program computes the floating point number $N=P\times10^{Q}$.

Input: (T5)=signed floating point number P
(B2)=integer Q in one's complement form, less than 75 (decimal) in absolute value

Output: (T5)=$P\times10^{Q}$, (PF)=0 if $|Q|<75$
(T5)=0, (PF)=1 if $|Q|\geq75$

Registers Not Preserved: B1, T4

Supporting Routines: none

Δ *153, Multiple Read Decimal, MRDDC

Length: See SYSTEM DUPLICATOR, P. 4.

Function: This program is used in the SPIREL system for the reading of data in decimal input formats from paper tape. It may be used directly to read decimal numbers, convert them as explained elsewhere, and store them.

Input: (B1)=address at which to begin storing the numbers read.
(B2)=number of numbers to read

Output: (B1) same as on input
(B2)=0

Error Message: IMPROPERLY FORMATTED DECIMAL NUMBER occurs if an improper decimal number is read, one which is out of the range permitted for the format used.

Registers Not Preserved: none

Supporting Routines: PWRTN(*152)

Δ *155, Binary to Decimal Conversion, BINDC

Length: See SYSTEM DUPLICATOR, P. 4.

Function: This program is used in the SPIREL system for conversion of a number from its internal binary representation to a decimal representation in printer hexads. It may be used directly for the same purpose.

Input: (T4) = number to be converted, fixed point integer if exponent empty, floating point otherwise.

Output: (T4), (T5) = number in decimal printer hexad form, 18 hexads:

floating point

```
±d.ddddddddddde±dd
 12 decimal      exponent,
   digits        2 decimal digits
```

fixed point integer

```
  ...     ±dd···dd
 2-16         1-15 decimal
 blanks            digits
```

Registers Not Preserved: T6,T7,B1,B2,B3

Supporting Routines: none

*156, Read with Checksum, RDCHK

Length: 57 (octal)

Function: This program is used in the SPIREL system for reading tapes in the hexad with tag and checksum format or high density hexad with tag, parity and checksum format explained in the section on Use of SPIREL. It may be used for the same purpose by an individual.

Input: (B1) = address at which to store first word read.
(B2) = number of words to read.

Error Message: CHECKSUM ERROR ON ABSOLUTE TAPE occurs if the checksum computed while reading does not agree with that read from paper tape.

Registers Not Preserved: T4,T5,T6,T7,B2,B3,B4,B5 ((B2)=0 on exit)

Supporting Routines: none

Note: RDCHK determines the hexad format by the value of the spill character punched by PUNCHK(*157). A 25 is for normal hexads and a 26 is for high density hexads.

Δ *157, Punch with Checksum, PUNCHK

Length: See SYSTEM DUPLICATOR, P. 4.

Function: This program is used in the SPIREL system for punching tapes in the hexad with tag and checksum format at the normal entry or high density hexad, with parity, tag, and checksum format at the second instruction. The "spill character" (one hexad) that precedes a checksummed sequence is also provided by this program. It may be used for the same purpose by an individual. See the section on Use of SPIREL.

Input: (B1) = address of first word to be punched.
(B2) = number of words to be punched.

Registers Not Preserved: T4,T5,T6,T7,B2,B3,B4,B5 ((B2)=0 on exit)

Supporting Routines: none

Δ *170, Plot Character on Scope, PLOT

Length: See SYSTEM DUPLICATOR, P. 4.

Function: This routine plots one character on the display scope.

Input: (B2) = character as hexad in printer code.
(B5) = position, $0 \leq (B5) < 216_{10}$, for maximum of eight lines, 27 characters per line.

Output: (B5) incremented by 1.

Registers Not Preserved: B2,T4,T5,T6,T7

Supporting Routines: none

<u>*171, Sample Typewriter for Console Input, SAMPLE</u>

Length: See SYSTEM DUPLICATOR, P. 4.

Function: This routine takes input from the console typewriter, maintains the input text on the scope and in memory, and exits when control input is received.

Output: return to PF if 'carriage return' character received, otherwise to PF+1 with (B1)=0 for 'halt', typed 'H'
1 for 'fetch', 'F' or
2 for 'cont', 'C' 'index'

Every command is displayed on the scope and printed before exiting. When SAMPLE is ready to receive new input from the console typewriter, the message *PAUSE* appears on the scope.

Registers Not Preserved: all

Supporting Routines: SETPM(*127), PLOT(*170)

<u>*172, Error Printer, ERPR</u>

Length: See SYSTEM DUPLICATOR, P. 4.

Function: This program prints error messages for the SPIREL system programs. See USE OF SPIREL, P. 9, for list of messages and details.

Registers Not Preserved: none

Supporting Routines: SETPM(*127), PFTR(*147)

<u>*173, Interpret Console Input, CONSOL</u>

Length: See SYSTEM DUPLICATOR, P. 4.

Function: This routine interprets commands received through the console typewriter. It will pass control words to XCWD, communicate with the magnetic tape system, set registers, and allow the user to communicate with XCWD from paper tape or the typewriter.

Input: (CC)=23 gives control to CONSOL, text to be interpreted is in vector TEXT(*174).

Output: One or more control words passed on to XCWD in T7.

Registers Not Preserved: T7

Supporting Routines: XCWD(*126, CNVRT(*143), SAMPLE(*171)

*175, Context Determination, CNTXT

Length: See SYSTEM DUPLICATOR, P. 4.

Function: This program determines all named programs with negative ST indices that are necessary for the support of all numbered programs and named programs with positive ST indices. Library routines may be loaded with negative indices to -0 and ST entries bearing tag 1. Then all private routines are loaded. CNTXT will operate on the ST entries with negative indices, leaving tag 1 on those not necessary for support and clearing the tag on all those "in context".

Supporting Routines: none

Δ *176, Table Look-Up, TLU

Length: See SYSTEM DUPLICATOR, P. 4.

Function: This program is used in the SPIREL system to determine the index of the Symbol Table entry for a name. If the name "looked for" does not appear on the Symbol Table, TLU adds it and increments the current last active index at location 117 by one.

Input: (T4) = left-adjusted 5 printer hexad representation of the name to be "looked for" on Symbol Table, *113; the rest of (T4) is irrelevant to TLU. Entry at CC+1 will cause the name looked for not to be placed on the Symbol Table if it was not found.

Output: (B1) = index of ST entry for the name "looked for".
(PF) = 1 if the entry was added to the active portion of the Symbol Table; 0 otherwise.

Registers Not Preserved: B1, PF

Supporting Routines: none

• Purpose of the Duplicator

When a system of programs has been debugged and extensive production running is contemplated or when memory space becomes critical, it may be advantageous to produce a self-loading paper tape bearing necessary SPIREL components, library routines used, and the coder's private system elements. The system duplicator SPIDUP, program *10, is designed for this purpose. This program is not itself a SPIREL component. SPIDUP has codeword address 10 (octal) and this will not conflict with normal codeword locations.

If the amount of space allowed for the SPIREL system is known, one may determine if the desired SPIREL system can be accommodated. This is simply done by adding up the lengths of all programs and vectors needed plus 300 (octal), the normal loading address of SPIREL. Since the system is being continually modified, the current lengths of all its components are listed on Page 4 of this section.

• Use of the Duplicator

A dump tape is used to tell program *10 what system elements are to be punched. The dump tape consists of a series of dump words, each preceded by a 'carriage return' punch and consisting of 18 octal digits. The dump word forms are as follows:

00000 0100 0000 fffff for all of the block (program or vector) with codeword at F.

nnnnn 0000 0000 fffff for N words beginning at machine address F.

nnnnn 0120 rrrr fffff for a control word which will cause a block of N zeros to be created with codeword at F and R in the dump word at the corresponding position in the codeword. If N in the dump word is empty, N and R will be obtained from the codeword at F at time of punching.

nnnnn 1100 rrrr fffff for N words beginning at the Rth element of the block with codeword at F. A "correct" control word is punched which is meaningful at later reading only if the block has been previously created.

A null dump word (18 '0' punches preceded by a 'carriage return') terminates the list of system components to be punched. Hexads on the dump tape after the null dump word will be reproduced onto the tape punched. Except for blocks of zeros, all components are punched in the hexad with tag and checksum format.

The complete system from which components are to be punched to produce a self-loading system tape should be in the machine. STEX must not be active. Then execution of program *10 results in the following steps:

- A leader is punched. It contains the SPIREL loader and RDCHK, program *156.
- The programmed stop

  Z HTR CC

  occurs, and the dump tape should be readied in the reader.
- CONTINUEing causes the dump tape to be read and the specified system components to be punched, until the null dump word is read.

- The SPIREL system tail is punched.
- Any information on the dump tape beyond the null dump word is duplicated, hexad for hexad.

- SPIREL Generation

A complete SPIREL system is produced by use of a dump tape as follows:

dump words		components and length (octal)	
00001 0000 0000 00102	†°	LASTEX	
00000 0120 0000 00112	†	LISTB6	200
00000 0120 0000 00113		ST	400
00000 0120 0000 00116	†	PM	200
00000 0120 0000 00122		VT	400
00000 0120 0000 00174	†	TEXT	14
00000 0100 0000 00013		TRACE	335
00000 0100 0000 00014		ARITH	125
00000 0100 0000 00020		CHECK	263
00000 0100 0000 00110		HDPR	106
00000 0100 0000 00111		MATRX	213
00000 0100 0000 00120		DIADMP	130
00000 0100 0000 00125	†	ADDR	6
00000 0100 0000 00126	†	XCWD	433
00000 0100 0000 00127	†	SETPM	155
00000 0100 0000 00130		SMNAM	57
00000 0100 0000 00131		DATIME	14
00000 0100 0000 00132		CLOCK	55
00000 0100 0000 00133		PCNTRL	14
00000 0100 0000 00135	†	STEX	364
00000 0100 0000 00136	†	SAVE	7
00000 0100 0000 00137	†	UNSAVE	36
00000 0100 0000 00140		DELETE	277
00000 0100 0000 00141		CHINDX	52
00000 0100 0000 00142		TAGSET	15
00000 0100 0000 00143		CNVRT	53
00000 0100 0000 00144		PRINT	113
00000 0100 0000 00145		PUNCH	160
00000 0100 0000 00146		XCWSQ	27
00000 0100 0000 00147		PFTR	46
00000 0100 0000 00150		MAP	141
00000 0100 0000 00151		PRSYM	152
00000 0100 0000 00152		PWRTN	41
00000 0100 0000 00153		MRDDC	154
00000 0100 0000 00155		BINDC	103
00000 0100 0000 00157		PUNCHK	56
00000 0100 0000 00170	†	PLOT	141
00000 0100 0000 00171	†	SAMPLE	203
00000 0100 0000 00172		ERPR	225
00000 0100 0000 00173	†	CONSOL	1035
00000 0100 0000 00175		CNTXT	54
00000 0100 0000 00176		TLU	43
00000 0000 0000 00000		null word to end dump	
77777 1160 0000 00172		correct index of ERPR	
77776 1160 0000 00173		correct index of CONSOL	

†components which must be included in minimal SPIREL.

9/6

°The dump word for LASTEX may be omitted; the end of allocatable memory will then be set to E-377 (where E is the last word in memory) to exclude only the magnetic tape system programs. A small setting will provide private absolute storage if such is desired.

For determination of a sufficient set of SPIREL components to satisfy the requirements of a private system, reference should be made to the diagram of SPIREL component linkages. If HDPR, program *110, is to be eliminated, all control words must be executed with SL14 on unless location 110 routes control immediately to the program using *110. Set

(110): 00000 0000 0001 00000

before execution of *10 and include the dump word

00001 0000 0000 00110

in the dump tape.

To alter the initial SPIREL loading address (normally octal 300) or the message printed upon completion of paper tape reading, refer to the symbolic listing of SPIDUP in the SPIREL reference notebook.

● Purpose

Backing storage can be used as an extension of effective memory space or as a fast alternative to paper tape for input and output. For such applications the SPIREL-compatible backing storage system (SBSS) is provided. Data created and processed by SPIREL in memory may be output onto magnetic tape by SBSS; data input from magnetic tape by SBSS may be processed by SPIREL.

The SPIREL backing storage system uses codeword addresses 160-167 and 177. It may be loaded with any SPIREL. STEX must be active for use of SBSS.

- Use

The SPIREL backing storage system, SBSS, provides a data transmission link between memory and a device. For transmission to occur the logical device (numbered 0-7) must be attached to a tape unit (numbered 0-3). A unit must be detached before it is manipulated other than under SBSS control, by programs or manually.

The logical unit of information on a device is a file. While a unit is in use for output an output file is open on the device; while it is in use for input an input file is open. Physically, a file consists of a sequence of records of uniform length. For an open file there exists a buffer area in memory which contains the last record read or the accumulation for the next record to be written. An output file must be closed to assure that all data output to the file is actually out of the buffer.

Output files are written sequentially on a device and are numbered 1 to 252_{10}; files may not be overwritten. A device may be positioned at any file. For input, the file at which the device is positioned is the one read.

SBSS maintains status and control information about each device in a device status block.

There are three levels of SBSS usage:

1) On the array level the user may read and write SPIREL arrays by simply specifying the codeword address and the device number to the program ARRAY(*166). All buffering is handled automatically on input and output. Storage allocation for arrays is provided on input.

2) On the buffered level the user may read and write any memory area by specifying the first word address, the number of words, and the device number to the programs GET(*160) for input and PUT(*161) for output. All buffering is handled automatically.

3) On the _direct_ level the user may read and write buffer loads (records) by specifying the device number to the programs READ(*162) for input and WRITE(*163) for output.

On all levels, the program CONTRL(*167) is used for logical (non-transmission) operations:

to attached and detached devices;

to open and close input and output files;

to specify buffer length for a device;

to position a device at a specified file, by rewinding, by backspacing one file or record, or near the end of information on the device;

to obtain status information about a device;

to set control options for a device;

to reposition after restart

Various comments on the use of SBSS are appropriate. Exercise of the system will no doubt suggest more.

• Files need not be explicitly open by the user. They will be opened when data transmission occurs.

• Output files should be closed as soon as possible after they are completed. While an output file is open, there is constant danger of transport misbehavior causing a file to become unreadable.

• When attaching a device to a tape unit, the exact number of files on the tape must be given or all sorts of bad things can happen.

• Never have more than one device attached to the same unit. It leads to immense confusion and almost certain file destruction.

- Tape Format

Marker allocation on an SBSS written magnetic tape is as follows:

marker 0 -- unused

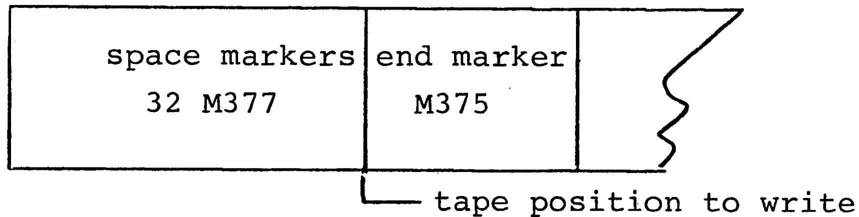
markers 1-374_8 -- file markers so that 252_{10} files are allowed

marker 375 -- end-of-information marker

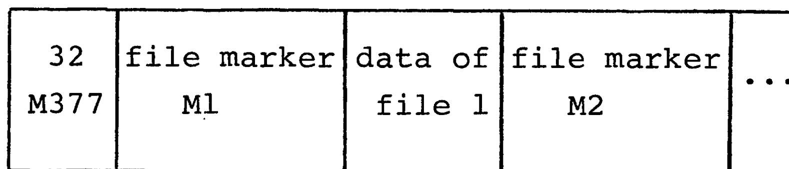
marker 376 -- record marker

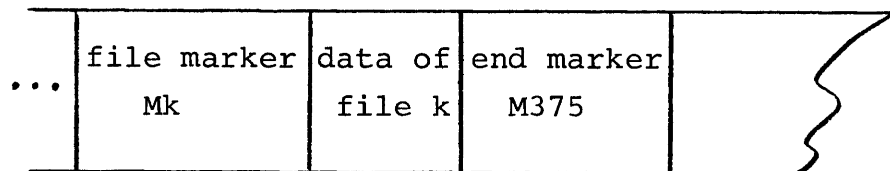
marker 377 -- space marker

A new reel of tape containing no files, but ready to be written, bears only a leader and an end-of-information marker.

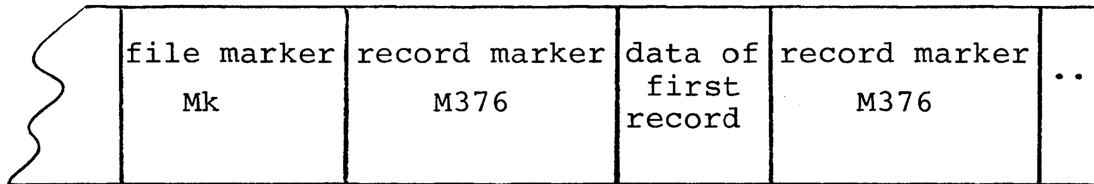


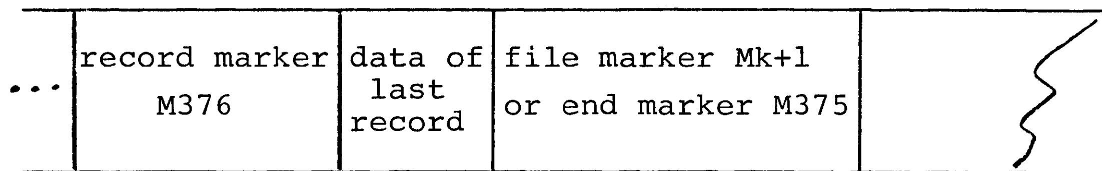
A reel containing k files bears k file markers and an end-of-information marker.



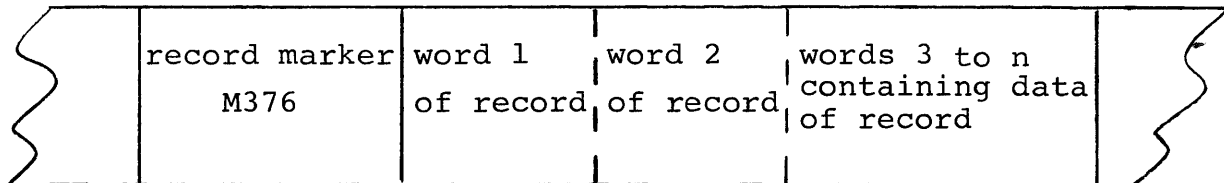


The kth file on a reel begins with a file marker and contains any number of record markers and records, all of the same length.





The ith record in the kth file on a reel begins with a record marker and contains n words of which the first two describe the record.



Word 1 contains

n(15 bits), length of the record;

k(9 bits), file number;

number of data words used (15 bits), null if this is not the last record in file k so that n-2 words are used;

i(15 bits), record sequence number.

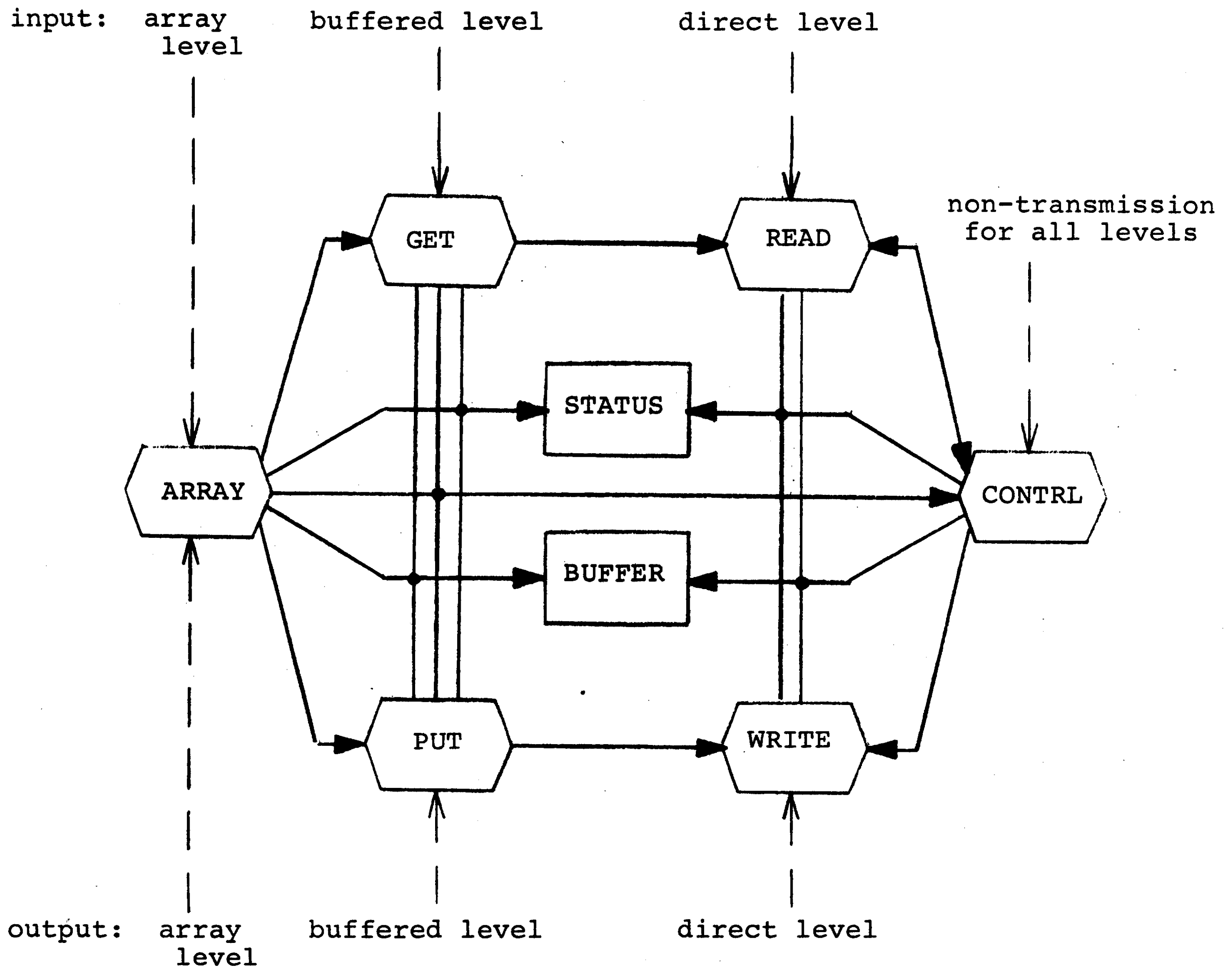
Word 2 contains a symmetric difference checksum for the record.

- Organization

The SPIREL backing storage system, SBSS, uses codewords as follows:

160	GET	read (B2) words to memory starting at address (B1) from device (B3)
161	PUT	write (B2) words from memory starting at address (B1) to device (B3)
162	READ	read one record from device (B3) into buffer for that device
163	WRITE	write buffer for device (B3) as one record on that device
164	BUFFER	buffer matrix with row i as the buffer for device i (i=0,1,...,7) and row -1 as the check buffer for all open output files needing one
165	STATUS	device status matrix with row i containing status for device i (i=0,1,...,7)
166	ARRAY	read or write array with codeword address (B1), relative to SPIREL address base setting, to or from device (B3)
167	CONTRL	for device (B3), perform non-transmission operation specified by (B1), with parameters specified in B2 and B4
177	ERRT	tape error routine

The levels of SBSS usage and interconnection of components is shown in the following diagram.



- SBSS Programs

All programs preserve registers except T7 which are not used for input or output.

*160, Input from Device to Memory, GET

Length 63 (octal)

Function: This routine transmits data from a device buffer to the user's memory locations, opening an input file if necessary and causing the buffer to be filled from the device as required.

Input: (B1)=address at which to begin storing the data read
(B2)=number of words to read
(B3)=device number

Special entry: at second order to simply skip over data; transmission suppressed and input (B1) irrelevent.

Output: (T7)=number of words transmitted (or skipped)

Return: to PF+1 normally
to PF if end-of-file encountered before specified number of words transmitted (or skipped) or input (B2)=0 and end-of-file encountered

Errors: none

Supporting Routines: READ(*162), CONTRL(*167)

*161, Output from Memory to Device, PUT

Length: 42 (octal)

Function: This routine transmits data from the user's memory locations to a device buffer, opening an output file on the device if necessary and causing the buffer to be emptied to the device as required.

Input: (B1)=address at which to begin obtaining data to write
(B2)=number of words to write
(B3)=device number

Return: to PF

Errors: none

Supporting Routines: WRITE(*163), CONTRL(*167)

*162, Input from Device to Buffer, READ

Length: 77 (octal)

Function: This routine reads the next record of the current input file on a device into the buffer for that device, opening an input file if necessary.

Input: (B3)=device number

Special entry: at second order to obtain first word of next record as output in T7 but remain positioned at same next record.

Return: to PF if no errors

Errors: 01 record sequence error detected
02 cannot read record in 8 attempts

Supporting Routines: CONTRL(*167), ERRT(*177)

*163, Output from Buffer to device, WRITE

Length: 103 (octal)

Function: This routine writes the buffer for a device into the current output file for that device and checks what is written. An output file must be open on the device.

Input: (B3)=device number

Return: to PF if no errors

Errors: 03 no open output file
04 UME in memory
05 cannot write record in 8 attempts

Supporting Routines: ERRT(*177)

*166, Transmit Array to or from Device, ARRAY

Length: 204 (octal)

Function: This routine transmits a SPIREL array, with tagw, between memory and a device, opening an input or output file as necessary.

Write

Entry: at first order

Input: (B1)=codeword address of array to be written, relative to SPIREL address base setting
(B3)=device number

Return: to PF

Errors: none

Read

Entries: at second order to read an array
at third order to skip an array

Input: (B1)=+0 to read array with same codeword address (or name) as when written
=codeword address (≠+0) at which to read array, relative to SPIREL address base setting

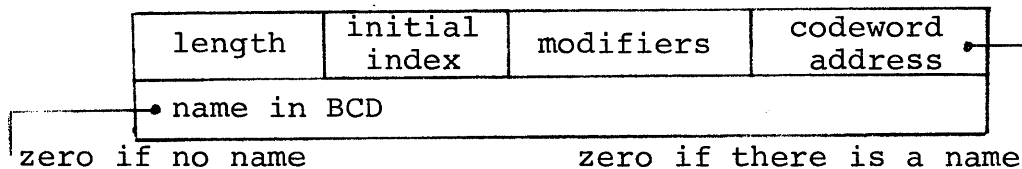
Return: to PF+1 normally
to PF if there is no data to be read from the file upon entry

Errors: 06 end-of-file encountered before read (or skip) complete

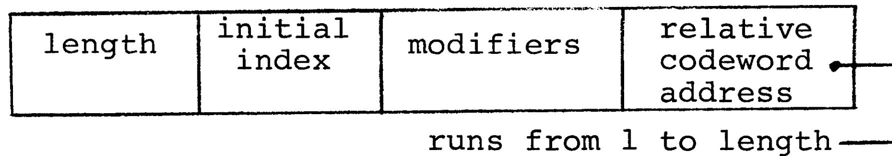
Write and Read

On Tape:

primary codeword



subsidiary codeword



Supporting Routines: XCWD(*126), GET(*160), PUT(*161), CNTRL(*167), ERRT(*177)

*167, Control for Tape System, CONTRL

Length: 430 (octal)

Function: This routine performs non-transmission functions in the SPIREL backing storage system; file control, logical operations, and positioning.

Input: (B1) to specify operation

(B3)=device number

(B2), (B4) may contain parameters

Output: may be in T7

Return: to PF if no error

Operations: on the basis of $(B1)_8$ on entry

(B1)=0, open input file

If device (B3) is not attached to a unit, no operation is performed.

Any open file is first closed

If device (B3) is positioned at the end of information, an input file cannot be opened and error 07 occurs.

If all is well, the next input file on device (B3) is opened and a buffer is created if necessary. If there is not memory space for the buffer, error 13 occurs.

(B1)=1, close input file

If device (B3) is not attached to a unit or there is no input file open on the device, no operation is performed.

Otherwise, the current input file on device (B3) is closed; the associated buffer is freed if the switch in the device status block dictates.

(B1)=2, open output file

If device (B3) is not attached to a unit, no operation is performed.

Any open file is first closed.

If the tape is full (252_{10} files), error 10 occurs.

If the switch in the device status block indicates that the device is write-protected, error 11 occurs.

If all is well, a new output file is opened on device (B3) and a buffer is created if necessary. If there is not memory space for the buffer, error 13 occurs. On exit (T7)=number of file opened.

(B1)=3, close output file

If device (B3) is not attached to a unit or there is no output file open on the device, no operation is performed.

Otherwise, the buffer for device (B3) is emptied if necessary and the current output file is closed. The buffer is freed if the switch in the device status block dictates, and the check buffer may be freed also. On exit (T7)=number of file closed.

(B1)=4, attach device

and (B4)=logical tape unit, 0-3 .

(B2)=number of files on the tape, 0 for a new output tape

If device (B3) is already attached to a unit, no operation is performed.

If a non-existent unit is specified in B4, error 12 occurs.

Otherwise, device (B3) is attached to unit (B4). The tape is rewound and a leader is written if input (B2)=0 (new tape).

(B1)=5, set buffer length

and (B2)=length, number of words

If a file is open on device (B3), no operation is performed. Buffer length may be set only when no file is open, but a buffer is not created until it is needed for transmission.

If no buffer is specified, devices 0-3 have buffer length 400_8, devices 4-7 have buffer length 1000_8.

(B1)=6, detach device

If device (B3) is not attached to a unit, no operation occurs.

Any open file is first closed.

Device (B3) is detached from its unit; the tape is rewound, and the device is left positioned at file 1.

(B1)=7, rewind

If device (B3) is not attached to a unit, no operation is performed.

Any open file is first closed.

If there are files on device (B3), the tape is rewound and left positioned at file 1.

(B1)=10, backspace file

If device (B3) is not attached to a unit, no operation is performed.

Any open file is first closed.

Then the tape is positioned at the beginning of a file.

On exit

(T7)=-1 if tape was at beginning of file 1 and could not be backspaced

=0 if tape was positioned within a file and was backspaced to the beginning of that file, or was backspaced to the beginning of the preceding file.

(B1)=11, backspace record

On exit

(T7)=-1 if device (B3) is not attached to a unit or no file is open on the device and no positioning is done
=0 if tape was positioned at the beginning of a file and could not be backspaced within the file
=1 if tape was backspaced one record in the current file

(B1)=12, position at file

and (B2)=file number

On exit

(T7)=-1 if device (B3) is not attached to a unit or a file is open on the device and no positioning is done
=0 if tape is positioned at the beginning of file (B2)
=1 if file (B2) does not exist on the device and no positioning is done

(B1)=13, position near end-of-information

On exit

(T7)=-1 if device (B3) is not attached to a unit or a file is open on the device and no positioning is done
=0 if the tape is positioned at the last file

(B1)=14, get field from device status block

and $(B2)_8$ specifies the field, as explained in the section on device status:

(B2)=1, LEFT	(B2)=4, FILE NO.	(B2)=7, FILES
=2, NEXT	=5, LAST	=10, UNIT
=3, LENGTH	=6, REC NO.	=11, SWITCH

On exit, (T7)=field value for device (B3), 0 if (B2) is not meaningful.

(B1)=15, set switches

and (B4) specifies switches to turn off

(B2) specifies switches to turn on

Switches, as explained in the section on device status, for device (B3) are turned on for 1's in (B4), then off for 1's in (B2). System switches are immune from setting by this operation.

(B1)=16, re-position after restart

If device (B3) is not attached to a unit, no operation is performed.

Otherwise, the tape is rewound and then positioned as designated in the device status block for device (B3).

Errors:
- 07 cannot open input file
- 10 cannot open output file
- 11 attempt to output on protected device
- 12 attempt to attach device to non-existent unit
- 13 insufficient space for buffer

Supporting Routines: STEX(*135), READ(*162), WRITE(*163) ERRT(*177)

- <u>Device Status and Control</u>

The device status matrix, STATUS(*165), contains one device status block per row, row i for device i, i=0,1,...,7. B3 is the row modifier, and there is no column modifier.

Each device status block is three words long and contains information about the device and its buffer:

DEVICE STATUS BLOCK

word 1	LEFT			NEXT
word 2	LENGTH	FILE NO.	LAST	REC NO.
word 3	FILES	ERROR		SWITCH

Word 1 describes buffer status:

LEFT -- number of words in the buffer yet to be obtained for input or supplied for output

NEXT -- relative location of the next word in the buffer to be obtained for input or supplied for output

Word 2 duplicates the first word of a record on tape:

LENGTH* -- number of words in the buffer, and number of words in a record (data words +2)

FILE NO. -- number of file at which currently positioned

LAST -- non-zero only for the last record of a file and then gives number of data words in the record

REC NO. -- number of the record at which currently positioned

*If not specified through use of CONTRL(*167), buffer length is 400_8 for devices 0-3, 1000_8 for devices 4-7.

Word 3 contains control information:

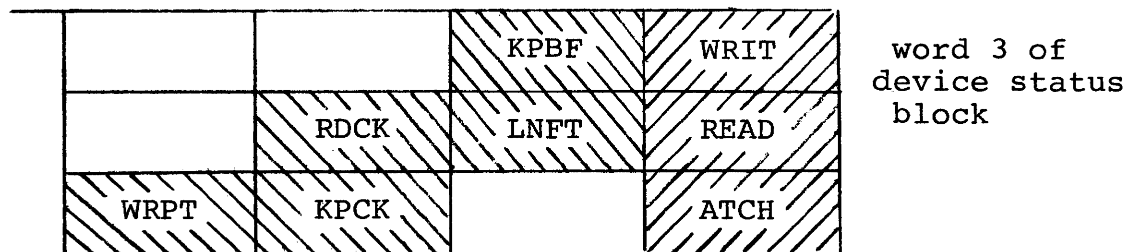
FILES -- number of files currently on the tape

UNIT -- logical tape unit number to which the device is attached, if at all

ERROR -- codeword address for user's error routine -- not used now

SWITCH -- set of binary switches to describe status and prescribe user options -- see below for details

The twelve bits of SWITCH are shown in detail:



<u>User</u> option switches -- all off initially

WRPT, write protect -- if on, do not permit output to this unit

RDCK, read check -- if off, check output by read without store
if on, check output by word-for-word comparison in check buffer

KPCK, keep check buffer -- if on, do not free check buffer when output file is closed

KPBF, keep data buffer -- if on, do not free buffer for device when file is closed

LNFT, length from tape -- if off, use LENGTH in status block as buffer length for input
if on, get buffer length from tape on input

<u>System</u> status switches -- may not be set by user

WRIT, write -- if on, output file open on device

READ, read -- if on, input file open on device

ATCH, attach -- if on, device is attached to unit

- Buffers

The buffer matrix, BUFFER(*164), contains one device buffer per row, row i for device i, i=0,1,...,7. Row -1 is used as a check buffer for all devices requiring one. BUFFER is modified by B3=-1,0,...,7 for rows and B5=1,... for columns.

A device buffer is created, if necessary, when a file is opened on a device. It is freed when a file is closed, unless the device status block specifies to keep the buffer.

The check buffer is created or lengthened, if necessary, when an output file is opened on a device for which read checking is specified in the device status block. The check buffer is freed when an output file is closed on a device for which the device status block specifies not to keep the check buffer <u>and</u> no other device is using it.

A device buffer is just a copy of a record on tape. The first two words are for control; the first is the same as word 2 of the device status block, the second the symmetric difference checksum. The remaining words contain user data.

DEVICE BUFFER

word 1	LENGTH	FILE NO.	LAST	REC NO.
word 2	CHECKSUM			
	data			

- Errors

SBSS errors occur when conditions arise due to physical tape problems or logical programming errors from which the system cannot recover. An error message is printed giving the error number, device, and information from the device status block including the unit and current tape position. Then a halt occurs with the error number and 'TAPE' in U.

Error	Program	Condition
01	162,READ	record sequence numbers out of order - probably missed a marker
02	162,READ	tape cannot be read - tried to read record 8 times
03	163,WRITE	trying to write without output file open - can occur only on direct access level
04	163,WRITE	WMTE light came on while writing, signifying UME in memory
05	163,WRITE	tape cannot be written - tried writing twice in each of 8 spots
06	166,ARRAY	end-of-file encountered while reading an array
07	167,CONTRL	cannot open input file because positioned at end of information
10	167,CONTRL	cannot open output file because maximum number (252) exist already
11	167,CONTRL	cannot open output file because device is write-protected
12	167,CONTRL	cannot attach device to non-existent unit
13	167,CONTRL	insufficient space in memory for buffer

LIBRARY

LIBRARY

Purpose. . . . .

Library Content. . . . .
Scalar Routines
Complex Scalar Routines
Matrix Routines
Complex Matrix Routines
Software Routines
Debug Routines
Constants

Programmed Use of Library Routines . . . .
Types of Routines
Use in the Genie Language
Use in Assembly Language

Running. . . . .
Compression
Diagnostic Procedures

Library Routines . . . .

System Maintenance . . . .
Punching
Editing

The SPIREL System is composed of the SPIREL operating system plus a library. The library is a collection of named programs and constants which may be utilized by a user system loaded into SPIREL.

The names of all library items are known to the Genie compiler. The user does not have to make any declarations about library items in a Genie program.

The names of library items are not known to the assembly program. The user must use pseudo-orders to establish linkage to library items from AP1 programs.

Memory space for the full library is required while user programs are being loaded. But then the library may be compressed to just those programs required for support of the user programs, the decision about what is necessary being made automatically. Dynamically, memory space is required for only the essential library items.

- Scalar Routines

These programs are mathematical functions, taking a single floating point argument and giving a floating point result. Detailed explanations are given in the Library Routines section.

All programs are for implicit execution, as discussed in Programmed Use of Library Routines.

NAME	DESCRIPTION
SQR[†]	square root argument ≥ 0; if argument < 0, gives result = 0 and prints error message
EXP[†]	exponential |argument| ≤ 170.0; if argument < -170.0, gives result = 0; if argument > 170.0, gives result for argument = 170.0 and prints error message
LOG[†]	natural logarithm argument > 0; if argument ≤ 0, gives result = argument and prints error message
LOG10	logarithm, base 10 argument > 0; if argument ≤ 0, gives result = 0 and prints error message
SIN[†]	sine
COS[†]	cosine
TAN[†]	tangent
COT[†]	cotangent
ASIN[†]	arc sine |result| < π/2, |argument| ≤ 1.0; if |argument| > 1.0, gives result = 0 and prints error message
ATAN[†]	arc tangent |result| < π/2
SINH[†]	hyperbolic sine |argument| ≤ 170.0; if |argument| > 170.0, gives result for |argument| = 170.0 and prints error message
COSH[†]	hyperbolic cosine |argument| ≤ 170.0; if |argument| > 170.0, gives result for |argument| = 170.0 and prints error message
TANH[†]	hyperbolic tangent |argument| ≤ 170.0; if |argument| > 170.0, gives |result| = 1.0
ASINH[†]	arc hyperbolic sine
ACOSH[†]	arc hyperbolic cosine argument ≥ 1.0; if argument < 1.0, gives result = 0 and prints error message

NAME	DESCRIPTION
ATANH[†]	arc hyperbolic tangent $\|argument\| < 1.0$; if $\|argument\| \geq 1.0$, gives result = 0 and prints error message
GAMMA	gamma function $-27.0 < argument < 55.0$ and argument not a negative integer; if argument out of range or a negative integer, gives result = 0 and prints error message
LGAMMA	log gamma argument ≥ 0; if argument < 0, gives result = 0 and prints error message

[†] In the Genie language, if argument is complex, corresponding complex routine will be used.

• Complex Scalar Routines

These programs are functions of a single complex argument, giving a real or complex result. Detailed explanations are given in the Library Routines section.

All programs are for implicit execution, as discussed in Programmed Use of Library Routines.

NAME	DESCRIPTION
RE†	real part of a complex number
IM†	imaginary part of a complex number
CARTN†	conversion of complex number from polar to Cartesian form
POLAR†	conversion of complex number from Cartesian to polar form
MOD	modulus of a complex number
CONJ†	conjugate of a complex number
ITIMES†	i times a complex number
CSQR	complex square root
CEXP	complex exponentiation $\lvert$real part of argument$\rvert \leq 170.0$; if real part < -170.0, gives result $= 0$; if real part > 170.0, gives result for real part $= 170.0$ and prints error message
CLOG	complex log argument $\neq 0$; if argument $= 0$, gives result $= 0$ and prints error message
CSIN	complex sine $\lvert$imaginary part of argument$\rvert \leq 170.0$; if $\lvert$imaginary part$\rvert > 170.0$, gives result for $\lvert$imaginary part$\rvert = 170.0$ and prints error message
CCOS	complex cosine $\lvert$imaginary part of argument$\rvert \leq 170.0$; if $\lvert$imaginary part$\rvert > 170.0$, gives result for $\lvert$imaginary part$\rvert = 170.0$ and prints error message
CTAN	complex tangent $\lvert$imaginary part of argument$\rvert \leq 85.0$; if $\lvert$imaginary part$\rvert > 85.0$, gives result for $\lvert$imaginary part$\rvert = 85.0$ and prints error message; if argument near singularity, gives result = tangent of real part of argument and prints error message
CCOT	complex cotangent $\lvert$imaginary part of argument$\rvert \leq 85.0$; if $\lvert$imaginary part$\rvert > 85.0$, gives result for $\lvert$imaginary part$\rvert = 85.0$ and prints error message

NAME	DESCRIPTION
CASN	complex arc sine
CATN	complex arc tangent argument $\neq \pm i$; if argument = $\pm i$, gives result = 0 and prints error message
CSNH	complex hyperbolic sine |real part of argument| $\leq$ 170.0; if |real part| > 170.0, gives result for |real part| = 170.0 and prints error message
CCSH	complex hyperbolic cosine |real part of argument| $\leq$ 170.0; if |real part| > 170.0, gives result for |real part| = 170.0 and prints error message
CTNH	complex hyperbolic tangent |real part of argument| $\leq$ 85.0; if |real part| > 85.0, gives result for |real part| = 85.0 and prints error message; if argument near singularity, gives result = tangent of real part and prints error message
CASNH	complex arc hyperbolic sine
CACSH	complex arc hyperbolic cosine
CATNH	complex arc hyperbolic tangent argument $\neq \pm 1$; if argument = ± 1, gives result = 0 and prints error message

†In the Genie language, if argument is vector or matrix, corresponding complex matrix routine will be used.

● Matrix Routines

These programs operate on standard vectors and matrices in the STEX domain. Detailed explanations are given in the Library Routines section.

A matrix routine prints an error message if a non-scalar operand does not exist and then performs no operation.

The EXECUTION column below gives the type of routine as explained in Programmed Use of Library Routines.

NAME	DESCRIPTION	EXECUTION
MCOPY	copy of vector or matrix	special
MADD	addition of two vectors or matrices if dimensions not compatible, prints error message	special
MSUB	subtraction of two vectors or matrices if dimensions not compatible, prints error message	special
MMPY	multiplication of two vecotrs, two matrices, or a vector and a matrix -- if dimensions not compatible, prints error message	special
INV†	inverse of matrix if matrix contains more rows than columns or is singular, performs no operation and prints error message	implicit
TRAN†	transpose of matrix	implicit
SMMPY	multiplication of scalar and vector or matrix	special
SMDIV	division of vector or matrix by a scalar if scalar = 0, performs no operation and prints error message	special
MFLT	floating point equivalent of integer matrix	special
MCMPL	complex equivalent of real floating point matrix	special
MINDEX	change initial index and B-mods for a vector or matrix	explicit
MINSERT	insert or delete elements of a vector or rows or columns of a matrix	explicit
MPATCH	move part of one vector or matrix into another vector or matrix	explicit

NAME	DESCRIPTION	EXECUTION
MPOWER	matrix to an integer power if matrix not square and power ≥ 0, prints error message and uses square portion; if matrix not square and power < 0, prints error message and performs no operation; if power < 0 and matrix singular, performs no operation and prints error message	special
DIAG	diagonalization of a matrix, with eigen-vectors if desired	explicit
ORTHOG	Gram-Schmidt orthonormalization of the rows of a matrix -- if rows not linearly independent, prints error message and performs no operation	implicit
SOLN†	solution of a system of linear equations if dimensions not proper or solution not defined, prints error message and performs no operation	implicit
DET†	determinant of a matrix if matrix not square, gives result = 0 and prints error message	implicit
STNDV	standard deviation of a vector	implicit
CHISQ	χ^2 for two vectors if vectors not the same length, gives result = 0 and prints error message	implicit
QCONF	χ^2 confidence level between two vectors if vectors not the same length, gives result = 1.0 and prints error message; if lengths < 2, gives result = 0 and prints error message	implicit
CRCOR	vector cross-correlation or auto-correlation if length of result > sum of input lengths - 1 for cross-correlation or > length of input for auto-correlation, prints error message and performs no operation	implicit
CMCON	construct complex vector or matrix if parts do not have same dimensions prints error message and performs no operation	special
FTRAN	Fourier transform of a real vector if arguments are not meaningful, prints error message and performs no operation	implicit

NAME	DESCRIPTION	EXECUTION
ITRAN	inverse Fourier transform of a complex vector -- if arguments are not meaningful, prints error message and performs no operation	implicit
VREV	order reversal of vector elements	explicit
CONVL	convolution of two vectors	implicit

+In the Genie language, if argument is complex, corresponding complex routine will be used.

- Complex Matrix Routines

These programs operate on standard complex vectors and matrices in the STEX domain. Detailed explanations are given in the Library Routines section.

A complex matrix routine prints an error message if a non-scalar operand does not exist and then performs no operation.

The EXECUTION column below gives the type of routine as explained in Programmed Use of Library Routines.

NAME	DESCRIPTION	EXECUTION
CMCPY	copy of a complex vector or matrix	special
CMADD	addition of two complex vectors or matrices -- if dimensions not compatible, prints error message	special
CMSUB	subtraction of two complex vectors or matrices -- if dimensions not compatible, prints error message	special
CMMPY	multiplication of two complex vectors, two complex matrices, or a complex vector and a complex matrix -- if dimensions not compatible, prints error message	special
CINV	inverse of complex matrix if matrix contains more rows than columns or is singular, performs no operation and prints error message	implicit
CTRAN	transpose of complex matrix	implicit
CDET	determinant of complex matrix if matrix not square, gives result = 0 and prints error message	implicit
CSOLN	solution of a system of linear equations with complex coefficients -- if dimensions not proper or solution not defined, prints error message and performs no operation	implicit
CSMMP	multiplication of a complex scalar and a complex vector or matrix	special
CSMDV	division of a complex vector or matrix by a complex scalar -- if scalar = 0, performs no operation and prints error message	special

NAME	DESCRIPTION	EXECTUION
MRE	real part of a complex vector or matrix	implicit
MIM	imaginary part of a complex vector or matrix	implicit
MCARTN	conversion of a complex vector or matrix from polar to Cartesian form	implicit
MPOLAR	conversion of a complex vector or matrix from Cartesian to polar form	implicit
MCONJ	conjugate of a complex vector or matrix	implicit
MITIMES	i times a complex vector or matrix	implicit

• Software Routines

These programs perform miscellaneous operations on data. Detailed explanations are given in the Library Routines section.

The EXECUTION column below gives the type of routine as explained in Programmed Use of Library Routines.

NAME	DESCRIPTION	EXECUTION
LENGTH†	length of a vector	implicit
CLENGTH	length of a complex vector	implicit
ROW†	number of rows in a matrix	implicit
CROW	number of rows in a complex matrix	implicit
COL†	number of columns in a matrix	implicit
CCOL	number of columns in a complex matrix	implicit
MAX	index of maximum element in vector	implicit
MIN	index of minimum element in vector	implicit
VSPACE†	dynamic creation of standard vector of zeroes	explicit
CVSPACE	dynamic creation of a standard complex vector of zeroes	explicit
MSPACE†	dynamic creation of standard matrix of zeroes	explicit
CMSPACE	dynamic creation of a standard complex matrix of zeroes	explicit
MTAKE†	dynamic creation of n-dimensional array of zeroes	explicit
CMTAKE	dynamic creation of n-dimensional complex array of zeroes	explicit
FXEXP	integer or floating point number to an integer power -- if base = 0 and exponent ≤ 0, gives result = 0 and prints error message	special
FLEXP	floating point number to a floating point power -- if base ≤ 0, gives result = 0 and prints error message	special
EVEN	test integer for being even	implicit
FIX	integer nearest to floating point number, if |argument| ≥ 16383.5, gives result = 0 and prints error message	implicit
FLOAT	floating point equivalent of integer	implicit

NAME	DESCRIPTION	EXECUTION
RANDM	floating point random number between 0.0 and 1.0	implicit
CADD	addition of complex scalars	special
CSUB	subtraction of complex scalars	special
CMPY	multiplication of complex scalars	special
CDIV	division of complex scalars	special
CXEXP	complex scalar to an integer power -- if base = 0 and exponent ≤ 0, gives result = 0 and prints error message	special
CFEXP	complex scalar to a real floating point power -- if base = 0, gives result = 0 and prints error message	special
CCEXP	complex scalar to a complex power if base = 0, gives result = 0 and prints error message	special
CONTROL†	application of SPIREL to non-complex item	explicit
CCONTROL	application of SPIREL to complex item	explicit
SCRIBE	formatted line printing	explicit (Genie programs only)
PRESCRIBE	formatted line printing with page control	explicit (Genie programs only)
PLOT	plot on printer of one vector versus another or of a vector versus its indices	explicit

†In the Genie language, if argument is complex, corresponding complex routine will be used.

- Debug Routines

These utility programs are never used in assembly language coding. Detailed explanations are given in the Library Routines section.

NAME	DESCRIPTION	EXECUTION
←COMP	library compression	internal library use only
←INOUT	input/output from compiled programs	used only by Genie-generated code
←ERPR	prints error message	internal library use only
EDIT	communication for compression and system maintenance	console use only
←ENTRY	records information for error print	internal library use only
INPUT	user's input routine	used only by Genie-generated code
OUTPUT	user's output routine	used only by Genie-generated code

• Constants

NAME	DESCRIPTION
LINCT	number of lines used on current page updated by SCRIBE and PRESCRIBE
PAGCT	number of page being printed updated by PRESCRIBE
←TEMP	used for storage control by EDIT only
CMPLX	complex scalar accumulator, double word operand with name "ditto" (←←←←←) on second part
←ELOC	information used in printing error message

- Types of Routines

The library routines may be classified by execution characteristics.

Functions are programs which accept arguments by the following rules:

for a single scalar argument, its value in T7 -- may not be output

for a single non-scalar argument, * codeword address in T7 -- may be input, output, or both

for N arguments, N > 1, the address of a scalar and * codeword address of a non-scalar on the B6-list at B6-N,..., B6-1 -- any may be input, output, or both (B6 decremented by N on exit)

A function which provides no output or has one or more output arguments is for explicit execution only.

A function which provides a single output which is not specified as an argument is for implicit execution. The single output is provided as follows:

real scalar in U and T7

complex scalar in the complex scalar accumulator, CMPLX

real non-scalar (vector or matrix) in the real non-scalar accumulator, *10 (octal)

complex non-scalar (vector or matrix) in the complex non-scalar accumulator, CSTAR

Note: Each complex argument is actually two arguments: the real part, then the imaginary part. Thus, for a function with one or more complex arguments, the number of arguments N ≥ 2 always. Two words are used on the B6-list for each complex argument.

Programs which do not accept arguments in T7 or on the B6-list by the above rules are not functions. They require special set-up for execution, e.g., arguments may be given in index registers.

- <u>Use in the Genie Language</u>

In the Genie language only function execution may be specified.

<u>Explicit execution</u> of the function EXPLC with arguments A,B, and C is specified by the command

    EXECUTE EXPLC(A,B,C)

Example:

    EXECUTE VSPACE(V,K)

to execute the program VSPACE for dynamic creation of the vector V of length K.

<u>Implicit execution</u> of the function IMPLC with arguments A,B, and C is specified on the righthand side of an equation, alone or in an expression:

    ... = ...IMPLC(A,B,C)...

Example:

$$P = (SIN(X^2)+SIN(Y^2))/(X-Y)$$

involving two executions of the SIN program.

An argument which is input only may be an expression. An output argument must be given as a simple name, scalar (not an element of a vector or matrix) or non-scalar as appropriate.

Scalar input arguments may be specified numerically, e.g.,-5.39. Non-scalar arguments must be specified by name, not by number, e.g., the vector with codeword at +200 (octal) may not be referred to as +200 but must be assigned the value by

    LET #V = +200

and then be referred to as V.

Genie language may cause code to be generated which calls for execution of library routines which are not functions.

Example: For scalar S and matrix M, the command

$$Q = S^3M^2$$

causes execution of FXEXP, MPOWER, and SMMPY which are library routines for special execution. The command

    PRINT S,M,Q

causes execution of the library routine ←INOU.

- <u>Use in Assembly Language</u>

In either AP1 or AP2 execution of any library routine may be specified by code to set up arguments and then TSR to the program.

In an AP1 program every named item which is referenced, including library routines and constants, must be given a cross-reference word through which it is indirectly addressed. This is accomplished with a REF pseudo-order as explained in the assembly language literature. Cross-reference words are set up automatically in Genie programs for all external named items referenced in the Genie language or AP2, including library routines and constants.

- Compression

The set of library routines may be reduced to just that set necessary for support of any user system. This compression is provided by execution, from the console, of the program EDIT. All memory space occupied by unnecessary programs is made available for user storage.

Library routines have negative Symbol Table indices (...,-2, -1,-0), and their Symbol Table entries bear a tag 1 initially.

Private named items receive positive Symbol Table indices (1,2,...).

Printing or punching ST-VT through SPIREL and execution of the library program EDIT cause context to be determined. This just means that all ST entries with negative indices have tag 1 changed to tag 0 (no tag) if the item represented is required for support of private programs loaded (on the basis of reference only, not dynamic use). Thus, only library items not in context bear tag 1 when a ST-VT print-out is obtained through SPIREL. All private named and numbered items are always in context.

To use EDIT, obtain SPIREL and load private programs and any data which may be located outside the STEX domain. Do not activate STEX or execute any program. Execute the named program EDIT with a control word to SPIREL from the console (manually or off paper tape). Context is determined, and space occupied by all items not in context is freed. All free space is consolidated into a single area. If SL14 is off, a print-out is given of ST-VT entries for items in context.

The "compressed" system may be written on tape for future use or may be used immediately for a run. Activate STEX, if desired, and load items into the STEX domain. Use an execute control word to start the system running.

A "compressed" system (one in which EDIT has been executed) may not be compressed again. This would be meaningless and is impossible since the EDIT routine "erases" itself.

- Diagnostic Procedures

The first version of a program should contain ample print outs that provide display of intermediate results. These may be edited out of the final version of the program or they may be executed conditionally on the basis of sense light settings.

A program should be tested with sense light 14 off. This causes monitoring on the printer of all SPIREL operations, all input-output operations, and all space taking operations. Such information is often a valuable debugging aid.

All the SPIREL diagnostic features available: tracing, arithmetic error monitoring, block bounds checking, diagnostic dump, memory dump. In addition, improper input to many library routines will cause an error message on the printer. Information provided includes the error made, the location of the transfer to the program detecting the error, and the name or codeword address of the calling program.

In this section the library routines are listed in alphabetical order. The details of input, output, and operation are given for each program. The programs ←ERPR and ←ENTRY and the constant ←ELOC may be used by any routine; they are not mentioned as support.

ACOSH, arc hyperbolic cosine

Function: This routine computes the arc hyperbolic cosine of a number.

Execution: implicit

ACOSH(A)

where argument A is floating point scalar input
result is floating point scalar

Errors: If A < 1.0, ACOSH gives result = 0 and prints error message.

Support: programs LOG, SQR

ASIN, arc sine

Function: This routine computes the arc sine of a number.

Execution: implicit

ASIN(A)

where argument A is floating point scalar input

result is floating point scalar, $|ASIN(A)| < \pi/2$

Errors: If $|A| > 1.0$, ASIN gives result = 0 and prints error message.

Support: program SQR

ASINH, arc hyperbolic sine

Function: This routine computes the arc hyperbolic sine of a number.

Execution: implicit

ASINH(A)

where argument A is floating point scalar input
result is floating point scalar

Errors: none

Support: programs LOG, SQR

ATAKE, array take

Function: This routine creates an m-dimensional array, the lowest level of which contains primary codewords addressing n-dimensional arrays of zeroes.

Execution: explicit

ATAKE(A,$D_1 \ldots D_m$,Z,$D_1 \ldots D_n$,N)

where argument A is the real array to be created,
argument D_i is the length in the ith dimension,
argument Z indicates the break between m and n, and
argument $N \leq 6$ is the number of dimensions (m+n)+1.

Space formerly addressed as A is freed. An array of size $D_1 x \ldots x D_m$ is created, to be indexed by registers B1...Bm. Then arrays of size $D_1 x \ldots x D_n$ (with primary codewords at level D_m) are created, to be indexed by registers B1...Bn.

Errors: If $N > 6$, or if any $D_i < 1$, or if the Z parameter is missing or improperly located, an error message is printed.

Support: MTAKE

ATAN, arc tangent

Function: This routine computes the arc tangent of a number.

Execution: implicit

ATAN(A)

where argument A is floating point scalar input

result is floating point scalar, $|ATAN(A)| < \pi/2$

Errors: none

Support: none

ATANH, arc hyperbolic tangent

Function: This routine computes the arc hyperbolic tangent of a number.

Execution: implicit

ATANH(A)

where argument A is floating point scalar input

result is floating point scalar

Errors: If $|A| \geq 1.0$, ATANH gives result = 0 and prints error message.

Support: program LOG

CACSH, complex arc hyperbolic cosine

Function: This routine computes the complex arc hyperbolic cosine of a complex number.

Execution: implicit

CACSH(A)

where argument A is complex scalar input
result is complex scalar in CMPLX

In the Genie language, same execution is specified by

ACOSH(A)

for complex argument A.

Errors: none

Support: program CASNH

CADD, complex add

Function: This routine forms the sum of two complex scalars.

Execution: special

input (B1) = address of real part of first operand

(B2) = address of real part of second operand

result in U, R and in complex scalar accumulator, CMPLX.

Complex scalars must occupy consecutive memory locations, real part followed by imaginary part.

Errors: none

Support: scalar CMPLX

CARTN, polar to Cartesian conversion

Function: This routine converts a double word scalar in polar form to a complex scalar in Cartesian form.

Execution: implicit

CARTN(A)

where argument is double word scalar input in polar form, i.e., represented by floating point scalars r and θ such that $A = re^{i\theta}$

result is complex scalar in standard Cartesian form, i.e., represented by floating point scalars x and y such that $A = re^{i\theta} = x + iy$

The polar form of a complex operand is a complex operand in the Genie language, but the arithmetic operations are not defined for this representation; input, output, and storage across an equals are meaningful for the polar form and useful.

Errors: none

Support: program SIN; scalar CMPLX

CASN, complex arc sine

Function: This routine computes the complex arc sine of a complex number.

Execution: implicit

CASN(A)

where argument A is complex scalar input
result is complex scalar in CMPLX

In the Genie language, same execution is specified by

ASIN(A)

for complex argument A.

Errors: none

Support: program CASNH; scalar CMPLX

CASNH, complex arc hyperbolic sine

Function: This routine computes the complex arc hyperbolic sine of a complex number.

Execution: implicit

CASNH(A)

where argument A is complex scalar input
result is complex scalar in CMPLX

In the Genie language, same execution is specified by

ASINH(A)

for complex argument A.

Errors: none

Support: programs CADD, CLOG, CMPY, CSQR; scalar CMPLX

CATAKE, complex array take

Function: This routine creates an m dimensional complex array; the lowest level of which contains codewords addressing n-dimensional arrays of zeroes.

Execution: explicit

$$\text{CATAKE}(A,D_1\ldots D_m,Z,D_1\ldots D_n,N)$$

where argument A is the complex array to be created,

argument D_i is the length in the ith dimension,

argument Z indicates the break between m and n, and

argument $N \leq 6$ is the number of dimensions $(m+n)+1$.

Space formerly addressed as A is freed. A complex array of size $D_1x\ldots D_m$ is created, to be indexed by registers B1...Bm. Then complex arrays of size $D_1x\ldots xD_n$ (with primary codewords at level D_m) are created, to be indexed by registers B1...Bn.

Errors: If $n > 6$, or any $D_i < 1$, or if the Z parameter is either missing or improperly located, an error message is printed.

Support: ATAKE,MTAKE

CATN, complex arc tangent

Function: This routine computes the complex arc tangent of a complex number.

Execution: implicit

CATN(A)

where argument A is complex scalar input
result is complex scalar in CMPLX

In the Genie language, same execution is specified by

ATAN(A)

for complex argument A.

Errors: If A = 0±/1.0, CATN gives result = 0 and prints error message.

Support: programs CASN, CATNH; scalar CMPLX

CATNH, complex arc hyperbolic tangent

Function: This routine computes the complex arc hyperbolic tangent of a complex number.

Execution: implicit

CATNH(A)

where argument A is complex scalar input

result is complex scalar in CMPLX

In the Genie language, same execution is specified by

ATANH(A)

for complex argument A.

Errors: If A = ±1.0, CATNH gives result = 0 and prints error message.

Support: programs CDIV, CLOG; scalar CMPLX

CCEXP, complex-complex exponentiation

Function: This routine computes the exponentiation of a complex scalar to a complex power.

Execution: special

input address portion of (U) = address of real part of complex scalar to be raised to power

address portion of (R) = address of real part of complex power

result in U,R and in complex scalar accumulator, CMPLX.

Complex scalars must occupy consecutive memory locations, real part followed by imaginary part.

Errors: If base = 0, CCEXP gives result = 0 and prints error message.

Support: programs CEXP, CLOG, CMPY; scalar CMPLX

CCOL, number of columns in a complex matrix

Function: This routine provides the number of columns in a complex matrix.

Execution: implicit

CCOL(A)

where argument A is complex matrix input

result is the integer number of columns in matrix A.

In the Genie language the same execution is specified by

COL(A)

for complex argument A.

Errors: If A does not exist, result = 0 and an error message is printed.

Support: program COL

CCONTROL, application of SPIREL to complex operand

Function: This routine composes control words and applies SPIREL to the real then the imaginary part of a named complex quantity.

Execution: Explicit

CCONTROL(N,WXYZ,R,CNAME)

where arguments N, WXYZ, and R are control word fields as for CONTROL

argument CNAME is the complex scalar or non-scalar to which the SPIREL operation is to be applied

See description of CONTROL for further details and examples. In the Genie language the same exeuction is specified by

CONTROL(N,WXYZ,R,CNAME)

for complex argument CNAME.

Errors: none

Support: program CONTROL

CCOS, complex cosine

Function: This routine computes the cosine of a complex number.

Execution: implicit

CCOS(A)

where argument A is complex scalar input
result is complex scalar in CMPLX

In the Genie language, same execution is specified by

COS(A)

for complex argument A.

Errors: If |imaginary part of A| > 170.0, CCOS gives result for |IM(A)| = 170.0 and prints error message.

Support: programs SIN, SINH; scalar CMPLX

CCOT, complex cotangent

Function: This routine computes the cotangent of a complex number.

Execution: implicit

CCOT(A)

where argument A is complex scalar input
result is complex scalar in CMPLX

In the Genie language, same execution is specified by

COT(A)

for complex argument A.

Errors: If |imaginary part of A| > 85.0, CCOT gives result for |IM(A)| = 85.0 and prints error message.

Support: programs COT, SIN, SINH; scalar CMPLX

CCSH, complex hyperbolic cosine

Function: This routine computes the complex hyperbolic cosine of a complex number.

Execution: implicit

CCSH(A)

where argument A is complex scalar input
result is complex scalar in CMPLX

In the Genie language, same execution is specified by

COSH(A)

for complex argument A.

Errors: If |real part of A| > 170.0, CCSH gives result for |RE(A)| = 170.0 and prints error message.

Support: programs SIN, SINH; scalar CMPLX

CDET, complex determinant

Function: This routine computes the determinant of a square standard complex matrix in the STEX domain.

Execution: implicit

CDET(A)

where argument A is square complex matrix input

result is complex scalar in CMPLX

SPIREL monitoring for creation of intermediate results is provided if SL14 is off. In the Genie language the same execution is specified by

DET(A)

for complex argument A.

Errors: If A does not exist, CDET prints an error message and performs no operation. If A is not square, CDET gives result = 0 and prints an error message.

Support: program CINV; non-scalar CSTAR

CDIV, complex divide

Function: This routine forms the quotient of two complex scalars.

Execution: special

input (B1) = address of real part of numerator

(B2) = address of real part of denominator

result in U,R and in complex scalar accumulator, CMPLX.

Complex scalars must occupy consecutive memory locations, real part followed by imaginary part.

Errors: none

Support: scalar CMPLX

CEXP, complex exponential

Function: This routine computes the exponential of a complex number.

Execution: implicit

CEXP(A)

where argument A is complex scalar input

result is complex scalar in CMPLX

In the Genie language, same execution is specified by

EXP(A)

for complex argument A.

Errors: If real part of A < -170.0, CEXP gives result = 0; if real part of A > 170.0, CEXP gives result for RE(A) = 170.0 and prints error message.

Support: programs EXP, SIN; scalar CMPLX

CFEXP, complex-floating point exponentiation

Function: This routine computes the exponentiation of a complex scalar to a real floating point power.

Execution: special

input address portion of (U) = address of real part of complex scalar to be raised to power

(R) = floating point scalar power

result in U,R and in complex scalar accumulator, CMPLX

Complex scalars must occupy consecutive memory locations, real part followed by imaginary part.

Errors: If base = 0, CFEXP gives result = 0 and prints error message.

Support: programs CARTN, FLEXP, POLAR; scalar CMPLX

CHISQ, χ^2

Function: This routine computes χ^2, measure of fit, for two floating point vectors of equal length.

Execution: implicit

CHISQ(A,B)

where argument A is the theoretical distribution, real floating point vector input

argument B is the observed distribution, real floating point vector input

result is real scalar, computed as

$$\sum_{i=1}^{n} \left(\frac{(B_i - A_i)^2}{A_i} \right)$$

where vectors are of length n

Errors: If A and B are not equal in length or if either does not exist, CHISQ prints an error message and gives result = 0.

Support: none

CINV, complex matrix inverse

Function: This routine forms the inverse of a standard complex matrix in the STEX domain.

Execution: implicit

CINV(A)

where argument A is standard complex matrix input

result is in complex non-scalar accumulator, CSTAR

Input matrix is not destroyed. SPIREL monitoring for creation of the result is provided if SL14 is off.

The usual application of CINV is to compute the inverse of a square matrix. CINV will work on a matrix containing more columns than rows, say n rows and m columns with m > n. In this case m - n systems of linear equations are represented

$$p^{th} \text{ system } \begin{cases} A_{1,1}X_1+\ldots+A_{1,n}X_n-A_{1,n+p} = 0 \\ \\ A_{n,1}X_1+\ldots+A_{n,n}X_n-A_{n,n+p} = 0 \end{cases}$$

for p = 1,...,m-n. Column p of the result matrix contains the solution of the pth system:

$$X_1 \text{ in } A_{1,n+p},\ldots,X_n \text{ in } A_{n,n+p}$$

The left square portion (n Columns) of the matrix result is the inverse of the left square portion of the input. In the Genie language the same execution is specified by

INV(A)

for complex argument A.

Errors: If A does not exist or if A contains more rows than columns, CINV prints an error message and performs no operation. If A is singular, no result is given, and CINV prints an error message.

Support: programs CADD, CDIV, CMCPY, CSUB, MOD; scalar CMPLX, non-scalar CSTAR

CLENGTH, length of a complex vector

Function: This routine provides the length of a complex vector.

Execution: implicit

CLENGTH(A)

where argument A is complex vector input

result is integer length of vector A.

In Genie programs the same execution is specified by

LENGTH(A)

for complex argument A.

Error: If A does not exist, result = 0 and an error message is printed.

Support: program LENGTH

CLOG, complex natural logarithm

Function: This routine computes the natural logarithm of a complex number.

Execution: implicit

CLOG(A)

where argument A is complex scalar input

result is complex scalar in CMPLX with imaginary part ≥ 0

In the Genie language, same execution is specified by

LOG(A)

for complex argument A.

Errors: If A = 0 (both real and imaginary parts = 0), CLOG gives result = 0 and prints error message.

Support: programs LOG, POLAR; scalar CMPLX

CMADD, complex matrix add

Function: This routine forms the sum of two standard complex vectors or two standard complex matrices in the STEX domain.

Execution: special

input (B1) = codeword address of real part of first operand

(B2) = codeword address of real part of second operand

result in complex non-scalar accumulator, CSTAR

If either (B1) or (B2) null on entry, the corresponding operand is taken as the complex non-scalar accumulator, CSTAR. An operand which is not CSTAR is not destroyed. The two codewords for a complex non-scalar must occupy consecutive memory locations, real part followed by imaginary part. SPIREL monitoring for creation of the result is provided if neither operand is CSTAR and SL14 is off.

Errors: If either operand does not exist, CMADD prints error message and performs no operation. If dimensions of the two operands are not the same, CMADD uses the subset of the larger which corresponds to the smaller, performs the addition, and prints error message.

Support: programs MADD, MSUB; non-scalar CSTAR

CMCON, complex matrix construction

Function: This routine forms a standard complex vector or matrix from two standard vectors or matrices in the STEX domain. The operands must be of the same dimensions and should contain floating point elements.

Execution: special

input (B1) = codeword address of real part

(B2) = codeword address of imaginary part

result in complex non-scalar accumulator, CSTAR

If either (B1) or (B2) null on entry, the corresponding operand is taken as the non-scalar accumulator, *10. An operand which is not *10 is not destroyed. SPIREL monitoring for creation of the result is provided if SL14 is off.

Errors: If either operand does not exist or if the operands are not of the same dimension, CMCON prints error message and performs no operation.

Support: program MCOPY, non-scalar CSTAR

CMCPY, complex matrix copy

Function: This routine copies a standard vector or matrix in the STEX domain.

Execution: Special

Input: (B1) = codeword address of real part of copy
(B2) = codeword address of real part of vector or matrix to be copied.

If (B1) is null on entry, (B2) is copied into CSTAR, the complex non-scalar accumulator. (B2) is never erased after the copy. If (B1) = (B2), (B2) is assumed to be a duplicate codeword (see SPIREL section on Storage Control). Then an actual copy is made of (B2) and the duplicate backreference is removed. SPIREL monitoring for creation of the copy is provided if SL14 is off.

Errors: If the complex non-scalar (B2) to be copied does not exist, CMCPY prints an error message and performs no operation.

Support: program MCOPY; non-scalar CSTAR.

CMMPY, complex matrix multiply

Function: This routine forms the product of two standard complex vectors (dot product, a scalar), a standard complex vector and a standard complex matrix (a vector), or two standard complex matrices (a matrix). Operands must be in the STEX domain.

Execution: special

input (B1) = codeword address of real part of lefthand operand
(B2) = codeword address of real part of righthand operand
scalar result in CMPLX; non-scalar in CSTAR

If either (B1) or (B2) null on entry, the corresponding operand is taken as the complex non-scalar accumulator, CSTAR. An operand which is not CSTAR is not destroyed. Note that vector × matrix treats the vector as a one-column matrix. The dot product given by vector A × vector B is defined as $\Sigma A_i \bar{B}_i$ where $\bar{B}_i$ is the conjugate of B_i. The two codewords for a complex non-scalar must occupy consecutive memory locations, real part followed by imaginary part. SPIREL monitoring for creation of the result is provided if SL14 is off.

Errors: If either operand does not exist, CMMPY prints error message and performs no operation. If the non-scalar operands do not have dimensions compatible for multiplication, CMMPY uses the subset of the operand with the larger pertinent dimension which corresponds to the operand with the smaller pertinent dimension, performs the multiplication, and prints an error message.

Support: programs MADD, MMPY, MSUB; scalar CMPLX; non-scalar CSTAR

CMPY, complex multiply

Function: This routine forms the product of two complex scalars.

Execution: special

input (B1) = address of real part of first operand

(B2) = address of real part of second operand

result in U,R and in complex scalar accumulator, CMPLX

Complex scalars must occupy consecutive memory locations, real part followed by imaginary part.

Errors: none

Support: scalar CMPLX

CMSPACE, complex matrix space

Function: This routine creates a standard complex matrix of zeroes in the STEX domain.

Execution: explicit

CMSPACE(A,B,C)

where argument A is complex matrix to be created
argument B is integer number of rows in A
argument C is integer number of columns in A

Storage addressed formerly as A is freed; then, if both B > 0 and C > 0, a complex matrix (two matrices with adjacent codewords) with B rows and C columns is created. SPIREL monitoring for freeing or creating is provided if SL14 is off. In the Genie language the same execution is specified by

MSPACE(A,B,C)

for complex argument A.

Errors: none

Support: program MSPACE

CMSUB, complex matrix subtract

Function: This routine forms the difference of two standard complex vectors or two standard complex matrices in the STEX domain.

Execution: special

input (B1) = codeword address of real part of operand to be subtracted from

(B2) = codeword address of real part of operand to be subtracted

result in complex non-scalar accumulator, CSTAR

If either (B1) or (B2) null on entry, the corresponding operand is taken as the complex non-scalar accumulator, CSTAR. An operand which is not CSTAR is not destroyed. The two codewords for a complex non-scalar must occupy consecutive memory locations, real part followed by imaginary part. SPIREL monitoring for creation of the result is provided if neither operand is CSTAR and SL14 is off.

Errors: If either operand does not exist, CMSUB prints error message and performs no operation. If dimensions of the two operands are not the same, CMSUB uses the subset of the larger which corresponds to the smaller, performs the subtraction, and prints error message.

Support: program CMADD

CMTAKE, complex matrix take

Function: This routine creates an n-dimensional complex array of zeroes.

Execution: explicit

$$CMTAKE(A,D_1,\ldots,D_N,N)$$

where argument A is complex array to be created
argument D_i is length in the ith dimension
argument $N \le 5$ is the number of dimensions

Space addressed formerly as A is freed; then an array of size $D_1 x \ldots x D_N$ is created, to be indexed by registers B1,...,BN. SPIREL monitoring for creating A is provided if SL14 is off. In the Genie language the same execution is specified by

$$MTAKE(A,D_1,\ldots,D_N,N)$$

for complex argument A.

Errors: If any $D_i < 1$, $N < 1$, or $N > 5$, CMTAKE gives no result and prints an error message.

Support: program MTAKE

COL, number of columns in a matrix

Function: This routine provides the number of columns in a matrix.

Execution: implicit

COL(A)

where argument A is a matrix input

result is the integer number of columns in matrix A.

Errors: If A does not exist, result = 0 and an error message is printed.

Support: none

<u>CONJ, conjugate of complex scalar</u>

Function: This routine provides the conjugate of a complex scalar.

Execution: implicit

CONJ(A)

where argument A is complex scalar in standard Cartesian form

result is complex scalar in CMPLX

Errors: none

Support: scalar CMPLX

CONTROL, application of SPIREL

Function: This routine composes a control word and applies SPIREL to a named quantity.

Execution: explicit

CONTROL(N,WXYZ,R,NAME)

where argument N is an integer, the "N" field (first five traids) of the control word to be executed

argument WXYZ is an integer, the "wxyz" field (next four triads) of the control word to be executed, usually given as an octal configuration

argument R is an integer, the "R" field (next four triads) of the control word to be executed

argument NAME is the scalar or non-scalar to which the SPIREL operation is to be applied

If NAME is a scalar, the N field in the control word must be 1, the x triad in the control word must = 0 so that WXYZ would be given as +w0yz in Genie. If NAME is a non-scalar (vector, matrix, or program), the x triad in the control word should = 4 so that WXYZ would be given as +w4yz in Genie.

Examples:

EXECUTE CONTROL(n,+4400,k,BLOCK)

in the Genie language would cause SPIREL to print n words of the vector or program BLOCK in octal, starting at the kth.

EXECUTE CONTROL(0,+5440,0,DATA)

in the Genie language would cause SPIREL to punch all data of the array DATA in hexad with tag and checksum format.

EXECUTE CONTROL(1,+4030,0,RATE)

in the Genie language would cause SPIREL to print the scalar RATE in decimal.

Errors: none

Support: none

CONVL, convolution

Function: This routine computes the convolution of two real floating point vectors.

Execution: implicit

CONVL(A,B)

where argument A is the shorter (filter) input vector
argument B is the longer (data) input vector
result is real floating point vector in the non-scalar accumulator, *10, of length = (length of B)-(length of A)+1

In computing dot products, CONVL does not extend the filter beyond the ends of the data; but "slow start-up" may be obtained by having sufficient zeroes at the ends of the data.

Errors: If A or B does not exist, CONVL prints an error message and performs no operation.

Support: program VREV

COS, cosine

Function: This routine computes the cosine of a number.

Execution: implicit

COS(A)

where argument A is floating point scalar input

result if floating point scalar

Also, (R) = SIN(A) on exit.

Errors: If $|A| \geq 2^{47}$, COS gives result = 0 and prints an error message.

Support: program SIN

COSH, hyperbolic cosine

Function: This routine computes the hyperbolic cosine of a number.

Execution: implicit

COSH(A)

where argument A is floating point scalar input
result is floating point scalar
Also, (R) = SINH(A) on exit.

Errors: If $|A| > 170.0$, COSH gives result for $|A| = 170.0$ and prints error message.

Support: program SINH

COT, cotangent

Function: This routine computes the cotangent of a number.

Execution: implicit

COT(A)

where argument A is floating point scalar input

result is floating point scalar

Error: If $|A|$ = multiple of $3\pi/2$, COT gives result = 0 and prints an error message.

Support: program TAN

CRCOR, cross-correlation or auto-correlation

Function: This routine performs the cross-correlation of two vectors of equal length or auto-correlation on a single vector.

Execution: implicit

CRCOR(A ,B ,C)

where argument A is real floating point vector input
argument B is real floating point vector input, B≢A for cross-correlation, B≡A for auto-correlation
argument C is integer specifying length of result, ≤ 2(input length)-1 for cross-correlation, ≤ input length for auto-correlation, C≡Z for maximum length result
result is standard floating point vector in the non-scalar accumulator, *10

Errors: If C > 2(input length)-1 for cross-correlation or C > input length for auto-correlation, CRCOR prints an error message and gives maximum length result. If A and B are not equal in length or either does not exist, CRCOR prints an error message and performs no operation.

Support: none

CROW, number of rows in a complex matrix

Function: This routine provides the number of rows in a complex matrix.

Execution: implicit

CROW(A)

where argument A is complex matrix input
result is integer number of rows in matrix A.

In the Genie language the same execution is specified by

ROW(A)

for complex argument A.

Error: If A does not exist, result = 0 and an error message is printed.

Support: program CLENGTH

CSIN, complex sine

Function: This routine computes the sine of a complex number.

Execution: implicit

CSIN(A)

where argument A is complex scalar input
result is complex scalar in CMPLX

In the Genie language, same execution is specified by

SIN(A)

for complex argument A.

Errors: If |imaginary part of A| > 170.0, CSIN gives result for |IM(A)| = 170.0 and prints error message.

Support: programs COS, SINH; scalar CMPLX

CSNH, complex hyperbolic sine

Function: This routine computes the complex hyperbolic sine of a complex number.

Execution: implicit

CSNH(A)

where argument A is complex scalar input
result is complex scalar in CMPLX

In the Genie language, same execution is specified by

SINH(A)

for complex argument A.

Errors: If |real part of A| > 170.0, CSNH gives result for |RE(A)| = 170.0 and prints error message.

Support: programs SIN, SINH; scalar CMPLX

CSMDV, complex scalar-matrix divide

Function: This routine divides a standard complex vector or matrix in the STEX domain by a complex scalar.

Execution: special

input (B1) = codeword address of real part of non-scalar operand

(U) = address of real part of scalar operand

result in complex non-scalar accumulator, CSTAR

If (B1) null on entry, the non-scalar operand is taken as CSTAR. A non-scalar which is not CSTAR is not destroyed. The two words for a complex scalar and the two codewords for a complex non-scalar must occupy consecutive memory locations, real part followed by imaginary part. SPIREL monitoring for creation of the result is provided if SL14 is off.

Errors: If the non-scalar operand does not exist or if the scalar = 0, CSMDV prints error message and performs no operation.

Support: programs CMCPY, CDIV; scalar CMPLX; non-scalar CSTAR

CSMMP, complex scalar-matrix multiply

Function: This routine forms the product of a complex scalar and a standard complex vector or matrix in the STEX domain.

Execution: special

input (B1) = codeword address of real part of non-scalar operand
(U) = address of real part of scalar operand

result in complex non-scalar accumulator, CSTAR

If (B1) null on entry, the non-scalar operand is taken as CSTAR. A non-scalar which is not CSTAR is not destroyed. The two words for a complex non-scalar must occupy consecutive memory locations, real part followed by imaginary part. SPIREL monitoring for creation of the result is provided if SL14 is off.

Errors: If non-scalar operand does not exist, CSMMP prints error message and performs no operation.

Support: programs CMCPY, CMPY; non-scalar CSTAR

CSOLN, complex linear equations solution

Function: This routine provides the solution to a system of linear equations represented by a square standard complex matrix of coefficients and a standard complex vector of constants. The operands must be in the STEX domain.

Execution: implicit

CSOLN(A,B)

where argument A is square complex matrix of coefficients
argument B is complex vector of constants
representing a system of n equations of the form

$A_{i,1}X_1+\ldots+A_{i,n}X_n = B_i \quad , \quad i = 1,2,\ldots,n$

result is complex vector in the complex non-scalar accumulator, CSTAR, with the value of X_i as the ith element

SPIREL monitoring for creation of the result is provided if SL14 is off. In the Genie language the same execution is specified by

SOLN(A)

for complex argument A. In assembly language coding the argument B may be a matrix of m columns representing m systems of equations of the form

$A_{i,1}X_{1,j}+\ldots+A_{i,n}X_{n,j} = B_{i,j} \quad , \quad i = 1,2,\ldots,n \text{ and } j = 1,2,\ldots,m$

Then the result in CSTAR is a matrix with the solution to the jth system as the jth column

$X_{1,j}, X_{2,j}, \ldots, X_{n,j}$

Errors: If A or B does not exist or if dimensions are not proper of if a solution is not defined, CSOLN prints an error message and performs no operation.

Support: program CINV; non-scalar CSTAR

CSQR, complex square root

Function: This routine computes the square root a complex number.

Execution: implicit

CSQR(A)

where argument A is complex scalar input

result is complex scalar in CMPLX with imaginary part ≥ 0

In the Genie language, same execution is specified by

SQR(A)

for complex argument A.

Errors: none

Support: programs CARTN, POLAR, SQR; scalar CMPLX

CSUB, complex subtract

Function: This routine forms the difference of two complex scalars.

Execution: special

input (B1) = address of real part of the operand to be subtracted from

(B2) = address of real part of the operand to be subtracted

result in U,R and in complex scalar accumulator, CMPLX

Complex scalars must occupy consecutive memory locations, real part followed by imaginary part.

Errors: none

Support: scalar CMPLX

CTAN, complex tangent

Function: This routine computes the tangent of a complex number.

Execution: implicit

CTAN(A)

where argument A is complex scalar input
result is complex scalar in CMPLX

In the Genie language, same exeuction is specified by

TAN(A)

for complex argument A.

Errors: If |imaginary part of A| > 85.0, CTAN gives result for |IM(A)| = 85.0 and prints error message. If A is near singularity of complex tangent, CTAN gives result = tangent of real part of A and prints error message.

Support: programs SIN, SINH, TAN; scalar CMPLX

CTNH, complex hyperbolic tangent

Function: This routine computes the complex hyperbolic tangent of a complex number.

Execution: implicit

CTNH(A)

where argument A is complex scalar input
result is complex scalar in CMPLX

In the Genie language, same execution is specified by

TANH(A)

for complex argument A.

Errors: If |real part of A| > 85.0, CTNH gives result for |RE(A)| = 85.0 and prints error message. If A is near singularity, CTNH gives result = tangent of real part of A and prints error message.

Support: programs CASN, CTAN; scalar CMPLX

CTRAN, complex matrix transpose

Function: This routine forms the transpose of a standard complex matrix in the STEX domain.

Execution: implicit

CTRAN(A)

where argument A is standard complex matrix input

result is in complex non-scalar accumulator, CSTAR

Note that CTRAN forms the matrix B, transpose of A, such that $B_{i,j}=A_{j,i}$. SPIREL monitoring for creation of the result is provided if SL14 is off. In the Genie language the same execution is specified by

TRAN(A)

for complex argument A.

Errors: If A does not exist, CTRAN prints an error message and performs no operation.

Support: programs CMCPY, TRAN; non-scalar CSTAR

CVSPACE, complex vector space

Function: This routine creates a standard complex vector of zeroes in the STEX domain.

Execution: explicit

CVSPACE(A,B)

where argument A is complex vector to be created
argument B is integer length of A

Storage addressed formerly as A is freed; then, if B > 0, a complex vector (two vectors with adjacent codewords) of length B is created. SPIREL monitoring for freeing or creation of A is provided if SL14 is off. In the Genie language the same execution is specified by

VSPACE (A,B)

for complex argument A.

Errors: none

Support: program VSPACE

CXEXP, complex-fixed point exponentiation

Function: This routine computes the exponentiation of a complex scalar to an integer power.

Execution: special

input address portion of (U) = address of real part of complex scalar to be raised to power

(R) = integer power

result in U,R and in complex scalar accumulator, CMPLX, zero if base = 0 and input (R) > 0

Complex scalars must occupy consecutive memory locations, real part followed by imaginary part.

Errors: If base = 0 and input (R) ≤ 0, CXEXP gives result = 0 and prints error message.

Support: programs CARTN, FXEXP, POLAR; scalar CMPLX

DET, determinant

Function: This routine computes the determinant of a square standard matrix of floating point type.

Execution: implicit

DET(A)

where argument A is square floating point matrix input
result is floating point scalar

A is destroyed only if it is the non-scalar accumulator, *10. In any case *10 is freed. SPIREL monitoring is provided for creation of an intermediate non-scalar if SL14 is off.

Errors: If A does not exist, DET prints an error message and performs no operation. If A is not square, DET gives result = 0 and prints an error message.

Support: program INV

<u>DIAG, matrix diagonalization</u>

Function: This routine diagonalizes a symmetric matrix of floating point type in the STEX domain. The initial indices of the matrix must be one. It also provides eigenvectors if desired.

Execution: explicit

DIAG(A ,B ,C)

where argument A is the matrix to be diagonalized and will contain the result

argument B is the matrix to contain eigenvectors as rows, null if no eigenvectors desired

argument C floating point scalar to be used as upper bound on off-diagonal elements of diagonalized matrix, null for upper bound as 2^{-48} times smallest diagonal element in result

The input matrix may be stored in upper triangular form, initial row indices 1,2,...,n for an n $\times$ n matrix

Errors: none

Support: program SQR

EDIT, library edit

Function: This routine performs library maintenance operations and is executed for library compression.

Execution: from the console only -- from word 1 for compression (see RUNNING section), from word 2 for punching (see MAINTENANCE section), from word 3 for initialization only (see MAINTENANCE section).

Errors: none

Support: program ←COMP

EVEN, test integer even

Function: This routine tests an integer for being even.

Execution: implicit

EVEN(A)

where argument A is integer input

result is Boolean value TRUE (represented by integer -0) if A is even, Boolean value FALSE (represented by integer -1) if A is odd

Errors: none

Support: none

EXP, exponential

Function: This routine computes the exponential of a number.

Execution: implicit

EXP(A)

where argument A is floating point scalar input

result is floating point scalar

Errors: If A < -170.0, EXP gives result = 0. If A > 170.0, EXP gives result for A = 170.0 and prints error message.

Support: none

FFT, fast Fourier transform

Function: This program does a discrete Fourier transform or inverse, as directed by the parameters; it is also used by FFTC.

$$A'_j = \frac{1}{B} \sum_{k=0}^{N-1} A_k e^{-i2\pi jk/N} \quad \text{(exponent sign is + for an inverse)}$$

Execution: explicit

FFT(A,B,C)

where argument A is the input/output vector (complex)

B is the real scale constant

C is a Boolean variable: true if the sign of the exponent is negative, false otherwise.

result is stored in A since the computation is done in place. FFT is most efficient for highly composite N, that is, if $N = n_1 . n_2 . n_3 . \ldots . n_m$. If N is prime the running time is on the order of N^2, otherwise it is on the order of

$$N\left(\sum_{i=1}^{m} n_i\right).$$

Errors: if A is non-existent, or the values require too much scratch storage, an error message is printed.

Support: program COS.

FFTC, fast Fourier transform control program

Function: This program causes a discrete Fourier transform or inverse to be done on an original input vector, as directed by the combination of input parameters. See FFT and RTRAN.

Execution: explicit

FFTC(A,B,C,D,E)

where argument A is the input vector
B is the output vector
C is the real scale constant
D is a Boolean variable: true for a transform, false for an inverse
E is a Boolean variable: true if the complex vector is conjugate symmetric (i.e., only the right half of the vector is supplied); false, if not.

The Fourier transform requires a complex input; therefore if input A is real, FFTC makes it complex under the following conditions:

1. if a) input A has odd length N, and output B is specified as symmetric,
   or b) output B is specified as non-symmetric (parameter E false),

   FFTC makes A complex by creating an imaginary part: a vector of zeroes N long. The output is complex vector B, each part of which is N long.

2. If input A has even length N and B is specified as symmetric (E true), FFTC saves time by creating a complex vector from A: the real part is the odd elements of A and the imaginary part is the even elements. This complex vector is N/2+1 long (the +1 being a zero added by the program to provide necessary working space for RTRAN). FFT is then entered at the third instruction, causing the transform to

be done on N/2 elements. RTRAN is done next, on N/2+1 elements and the result is complex vector B (each part of which is N/2+1 long).

If the input A is already complex, FFTC does the following:

1. if A is N long and not symmetric, the transform or inverse is done directly and the output is a complex vector N long (the input and output vectors may both be A if the input doesn't need to be saved).
2. If A is symmetric, FFTC considers its length N to represent one half of the vector plus 1 (the mid-point), and the real output B will be 2(N-1) long. (This case is the reverse of 2 above, i.e., it does RTRAN first, followed by FFT entered at the third instruction. By definition then, this case is an inverse and parameter D must be false).

The scale constant for a transform is usually 1.0; for an inverse 1.0/length N. If the input and output vectors are different lengths, use the length of the one which is real.

Errors: The following combinations of parameters produce error messages:

1. A real, B real, C,D,E
2. A complex, B complex, C,D,E true
3. A complex, B real, C,D,E false

Support: programs FFT, RTRAN, MCOPY, CMCPY; non-scalars ←CSTAR, USTAR, ←DUMY.

FIX, convert to integer

Function: This routine computes the integer closest to a floating point scalar.

Execution: implicit

FIX(A)

where argument A is floating point scalar input

result is integer closest to A, rounded up in absolute value

Errors: If $|A| \geq 16383.5$, FIX gives result = 0 and prints error message.

Support: none

<u>FLEXP, floating point exponentiation</u>

Function: This routine computes exponentiation of a floating point number to a floating point power.

Execution: special

input (U) = floating point scalar to be raised to power

(R) = floating point scalar power

result in U and T7 which is $(U)^{(R)}$

Errors: If input $(U) \leq 0$, FLEXP gives result = 0 and prints error message.

Support: programs EXP, LOG

FLOAT, convert to floating point

Function: This routine provides the floating point equivalent of an integer.

Execution: implicit

FLOAT(A)

where argument A is scalar input, integer or floating point

result is floating point equivalent of A, just A if A is floating point

Errors: none

Support: none

FTRAN, Fourier transform

Function: This routine computes the complex frequency (Hz) spectrum of a real function of time. The discrete time series must be stored in a floating point vector in the STEX domain, and the values must be equally spaced in time.

Execution: implicit

FTRAN(A,B,C,D,E,F)

where argument A is the real time series vector input,
argument B is the upper time limit, floating point scalar
argument C is the lower time limit, floating point scalar
argument D is the upper frequency limit, floating point scalar
argument E is the lower frequency limit, floating point scalar
argument F is the frequency increment, floating point scalar
result is complex vector in CSTAR, the complex non-scalar accumulator

The Fourier integral is approximated by the trapezoidal quadrature formula. SPIREL monitoring for creation of the result is provided if SL14 is off.

Errors: If A does not exist or $B-C \le 0$ or $D-E < 0$, FTRAN prints an error message and performs no operation.

Support: programs CVSPACE, SIN; non-scalar CSTAR

FXEXP, fixed point exponentiation

Function: This routine computes exponentiation of a number to an integer power.

Execution: special

input (U) = floating point or integer scalar to be raised to power

(R) = integer power

result in U and T7 which is $(U)^{(R)}$ and of same type as input (U), zero if input (U) is integer and $|(U)| > 1$ and input $(R) < 0$, zero if input $(U) = 0$ and input $(R) > 0$

Errors: If input $(U) = 0$ and input $(R) \leq 0$, FXEXP gives result = 0 and prints error message.

Support: none

GAMMA, gamma function

Function: This routine computes the gamma function of a real floating point number.

Execution: implicit

GAMMA(A)

where argument A is real floating scalar input

result is real floating point scalar computed by Stirling's logarithmic approximation

Errors: If A<-27, A>55.0, or A is a negative integer, GAMMA gives result=0 and prints an error message.

Support: programs EXP, LOG

IM, imaginary part

Function: This routine provides the imaginary part of a complex scalar.

Execution: implicit

IM(A)

where argument A is complex scalar input

result is real floating point scalar

If coded in the Genie language, the library routine is not used; but the routine may be used in assembly language coding. IM may be used on any double word scalar argument to provide the second part as a single word scalar.

Errors: none

Support: none

INPUT, special input

Function: This routine is supplied by the user for special input to programs written in the Genie language.

Execution: in Genie language only, by the command

INPUT list

where the program INPUT is entered once for each named variable in the list. A complex variable is treated as two items, the real part with the name of the variable and the imaginary part with the name "ditto". The program INPUT must be coded in the assembly language, AP1. Information is given in T7 on entry to INPUT as follows:

bits	1-30	name in BCD as given in list
	31-33	not used
	34-36	octal 0 for scalar
		2 for vector
		4 for matrix
	39-41	not used
	40-54	address of scalar, codeword address for non-scalar

INV, matrix inverse

Function: This routine forms the inverse of a standard matrix in the STEX domain. The matrix must be of floating point type.

Execution: implicit

INV(A)

where argument A is floating point standard matrix input

result is floating point standard matrix in the non-scalar accumulator, *10

Input matrix which is not *10 is not destroyed. SPIREL monitoring for creation of the result is provided if SL14 is off. The determinant of A is given in T7 on exit, floating point scalar.

The usual application of INV is to compute the inverse of a square matrix. INV will work on a matrix containing more columns than rows, say n rows and m columns with m > n. In this case m - n systems of linear equations are represented

$$p^{th} \text{ system } \begin{cases} A_{1,1}X_1+\dots+A_{1,n}X_n-A_{1,n+p} = 0 \\ A_{n,1}X_1+\dots+A_{n,n}X_n-A_{n,n+p} = 0 \end{cases}$$

for p = 1,...,m-n. Column p of the result matrix contains the solution of the pth system:

$$X_1 \text{ in } A_{1,n+p},\dots,X_n \text{ in } A_{n,n+p}$$

The left square portion (n columns) of the matrix result is the inverse of the left square portion of the input, and the determinant computed is that of the left square portion of the input.

Errors: If A does not exist or if A contains more rows than columns, INV prints an error message and performs no operation. If A is singular, no result is given, and INV prints an error message.

Support: program MCOPY

ITIMES, i times complex scalar

Function: This routine computes i times a complex scalar.

Execution: implicit

ITIMES(A)

where argument A is complex scalar, x+iy

result is complex scalar in CMPLX, -y+ix

Errors: none

Support: scalar CMPLX

ITRAN, Inverse Fourier Transform

Function: This routine computes the real time spectrum of a complex function of frequency (HZ), where $F(HZ)=\overline{F(-HZ)}$. The positive portion of the frequency domain must be stored in a complex, floating point vector in the STEX domain.

Execution: implicit

ITRAN(A,B,C,D,E,F)

where argument A is the complex frequency vector input,
argument B is the upper frequency limit, floating point scalar,
argument C is the lower frequency limit, floating point scalar,
argument D is the upper time limit, floating point scalar,
argument E is the lower time limit, floating point scalar,
argument F is the time increment, floating point scalar,
result is real vector in the real non-scalar accumulator, *10.

The Fourier integral is approximated by the trapezoidal quadrature formula. SPIREL monitoring for creation of the result is provided if SL14 is off.

Errors: If A does not exist or $B-C\leq 0$ or $D-E<0$, ITRAN prints an error message and performs no operation.

Support: programs SIN, VSPACE.

LENGTH, length of vector

Function: This routine provides the length of a vector.

Execution: implicit

LENGTH(A)

where argument A is vector input

result is integer length of vector A.

Errors: If A does not exist, result = 0 and an error message is printed.

Support: none

LGAMMA, log gamma

Function: This routine computes the logarithm of the gamma function of a non-negative real floating point number.

Execution: implicit

LGAMMA(A)

where argument A is real floating point scalar input

result is real floating point scalar

Error: If A<0, LGAMMA gives result=0 and prints an error message.

Support: programs GAMMA, LOG

LOG, natural logarithm

Function: This routine computes the natural logarithm of a number.

Execution: implicit

LOG(A)

where argument A is floating point scalar input

result is floating point scalar

Errors: If $A \leq 0$, LOG gives result = argument and prints error message.

Support: none

LOG10, logarithm, base 10

Function: This routine computes the common logarithm of a number.

Execution: implicit

LOG10(A)

where argument A is floating point scalar input

result is floating point scalar

Errors: If $A \le 0$, LOG10 gives result = 0 and prints error message.

Support: program LOG

MADD, matrix add

Function: This routine forms the sum of two standard vectors or two standard matrices in the STEX domain. The operands must agree in type, floating point or integer, and the result will be of the same type.

Execution: special

input (B1) = codeword address for first operand

(B2) = codeword address for second operand

result in non-scalar accumulator, *10

If either (B1) or (B2) null on entry, the corresponding operand is taken as the non-scalar accumulator, *10. An operand which is not *10 is not destroyed. SPIREL monitoring for creation of the result is provided if neither operand is *10 and SL14 is off.

Errors: If either operand does not exist, MADD prints error message and performs no operation. If dimensions of the two operands are not the same, MADD uses the subset of the larger which corresponds to the smaller, performs the addition, and prints error message.

Support: none

MAX, vector maximum

Function: This routine computes the index of the element with the largest numeric value in a vector of floating point numbers.

Execution: implicit

MAX(A)

where argument A is floating point vector input

result is integer

Errors: If A does not exist, MAX prints an error message and gives result=0.

Support: none

MCARTN, matrix polar to Cartesian conversion

Function: This routine converts a double word vector or matrix in the STEX domain in polar form to a complex vector or matrix in Cartesian form.

Execution: implicit

MCARTN(A)

where argument is double word vector or matrix input in polar form, i.e., each element represented by floating point scalars r and θ such that the element = $re^{i\theta}$

result is complex vector or matrix in CSTAR in standard Cartesian form, i.e., each element represented by floating point scalars x and y such that the element = $re^{i\theta} = x+iy$

SPIREL monitoring for creation of the result is given if SL14 is off. The polar form of a complex operand is a complex operand in the Genie language, but the arithmetic operations are not defined for this representation; input, output, and storage across an equals are meaningful for the polar form and useful. In the Genie language same execution is specified by

CARTN(A)

for non-scalar argument.

Errors: If A does not exist, MCARTN prints an error message and performs no operation A.

Support: programs CARTN, CMCPY, POLAR; scalar CMPLX; non-scalar CSTAR

MCMPL, matrix complex

Function: This routine provides the complex equivalent of a real floating point vector or matrix in the STEX domain.

Execution: special

input (B1)=codeword address for operand

result in complex non-scalar accumulator, CSTAR

If (B1) null on entry, the operand is taken as the real non-scalar accumulator, *10.

Errors: If operand does not exist, MCMPL prints error message and performs no operation.

Support: program MCOPY; non-scalar CSTAR

MCONJ, matrix conjugate

Function: This routine provides the conjugate of a complex vector or matrix.

Execution: implicit

MCONJ(A)

where argument A is complex vector or matrix input

result is complex vector or matrix in CSTAR, each element being the conjugate of the corresponding element of A

SPIREL monitoring for creation of the result is given if SL14 is off. In the Genie language same execution is specified by

CONJ(A)

for non-scalar argument A.

Errors: If A does not exist, MCONJ prints an error message and performs no operation.

Support: program CMCPY; non-scalar CSTAR

MCOPY, matrix copy

Function: This routine copies a standard vector or matrix in the STEX domain.

Execution: Special

Input: (B1) = codeword address for copy,
(B2) = codeword address of vector or matrix to be copied.

If (B1) is null on entry, (B2) is copied into *10, the non-scalar accumulator. (B2) is never erased after the copy. If (B1) = (B2), (B2) is assumed to be a duplicate codeword (see SPIREL section on Storage Control). Then an actual copy is made of (B2) and the duplicate backreference is removed. SPIREL monitoring for creation of the copy is provided if SL14 is off.

Errors: If the non-scalar (B2) to be copied does not exist, MCOPY prints an error message and performs no operation.

Support: programs MSPACE and VSPACE.

MEAN, Average Values

Function: This routine computes the mean of a standard vector.

Execution: implicit

MEAN(A)

where argument A is a floating point vector

result is a floating point scalar

Errors: If A does not exist, MEAN gives result = 0 and prints an error message.

Support: none

MFLT, matrix float

Function: This routine provides the floating point equivalent of an integer vector or matrix in the STEX domain.

Execution: special

input (B1)=codeword address for operand

result in non-scalar accumulator, *10

If (B1) null on entry, the operand is taken as the non-scalar accumulator, *10.

Errors: If operand does not exist, MFLT prints error message and performs no operation.

Support: program MCOPY

<u>MIM, matrix imaginary part</u>

Function: This routine provides the imaginary part of a complex vector or matrix.

Execution: implicit

MIM(A)

where argument A is complex non-scalar input

result is in non-scalar accumulator, *10

MIM may be used on any double word non-scalar argument to provide the second part as a single word non-scalar. SPIREL monitoring for creation of the result is provided if SL14 is off. In the Genie language the same exeuction is specified by

IM(A)

for non-scalar argument A.

Errors: If A does not exist, MIM prints error message and performs no operation.

Support: program MCOPY

MIN, vector minimum

Function: This routine computes the index of the element with the smallest numeric value in a vector of floating point numbers.

Execution: implicit

MIN(A)

where argument A is floating point vector input

result is integer

Errors: If A does not exist, MIN prints an error message and gives result=0.

Support: program MAX

MINDEX, matrix index

Function: This routine changes the initial indices and B-mods for a vector or matrix in the STEX domain.

Execution: explicit

MINDEX(i,b,V) for vector

where argument i is integer, initial index (for i = zero, use -Z)

argument b is integer (=1,2,...,7) for B-mod or zero to not change B-mod

argument V is vector operand

If both i and b are zero, the vector V is changed to standard form (initial index = 1 and B1-mod).

MINDEX(i_r,b_r,i_c,b_c,M) for matrix

where arguments i_r and i_c are integers, row and column initial indices respectively

arguments b_r and b_c are integers (=1,2,...,7) for row and column B-mods respectively or zero to not change B-mod

argument M is matrix operand

If arguments i_r,b_r,i_c, and b_c are zero, the matrix M is changed to standard form (initial indices = 1 and B1-mod for rows, B2-mod for columns).

Errors: If operand does not exist, MINDEX prints error message and performs no operation.

Support: none

MINSERT, matrix insert

Function: This routine inserts or deletes elements of a vector in the STEX domain or rows or columns of a matrix in the STEX domain.

Execution: explicit

MINSERT(n,r,V) for vector

where argument n is integer, number of elements to insert as zeroes if > 0, number of elements to delete if < 0
argument r is integer, index of first element inserted or deleted
argument V is vector operand

MINSERT(n,r,k,M) for matrix

where argument n is integer, number of rows or columns to insert as zeroes if > 0, number of rows or columns to delete if < 0
argument r is integer, index of first row or column inserted or deleted
argument k is integer, k = 1 to operate on rows, k = 2 to operate on columns

If argument n is zero, MINSERT deletes element, row, or column r and all following. If argument r is null, MINSERT inserts n elements, rows, or columns after the last.

Errors: none

Support: none

MITIMES, i times complex matrix

Function: This routine computes i times a complex vector or matrix.

Execution: implicit

MITIMES(A)

where argument A is complex vector or matrix input

result is complex vector or matrix in CSTAR, each element being i times the corresponding element of A

SPIREL monitoring for creation of the result is given if SL14 is off. In the Genie languate same execution is specified by

ITIMES(A)

for non-scalar argument A.

Errors: If A does not exist, MITIMES prints an error message and performs no operation

Support: program MCONJ; non-scalar SCTAR

MMPY, matrix multiply

Function: This routine forms the product of two standard vectors (dot product, a scalar), a standard vector and a standard matrix (a vector), or two standard matrices (a matrix). Operands must be in the STEX domain; they must agree in type, floating point or integer, and the result will be of the same type.

Execution: special

input (B1) = codeword address for lefthand operand
(B2) = codeword address for righthand operand

scalar result in U and T7; non-scalar in accumulator, *10

If either (B1) or (B2) null on entry, the corresponding operand is taken as *10. An operand which is not *10 is not destroyed. Note that vector X matrix treats the vector as a one-row matrix, and matrix X vector treats the vector as a one-column matrix. SPIREL monitoring for creation of the result is provided if SL14 is off.

Errors: If either operand does not exist, MMPY prints error message and performs no operation. If the non-scalar operands do not have dimensions compatible for multiplication, MMPY uses the subset of the operand with the larger pertinent dimension which corresponds appropriately to the operand with the smaller pertinent dimension, performs the multiplication, and prints an error message.

Support: none

MOD, modulus of complex scalar

Function: This routine computes the modulus of a complex scalar.

Execution: implicit

MOD(A)

where argument A is complex scalar input

result is real floating point scalar

Errors: none

Support: program SQR

MODUL, Compute Remainder

Function: This routine computes A modulo B

Execution: implicit

MODUL(A,B)

where A and B are integers.

result is an integer

Errors: none

Support: none

MPATCH, matrix patch

Function: This routine moves part of one vector or matrix in the STEX domain into another vector or matrix in the STEX domain.

Execution: explicit

MPATCH([what],[from],[to])

where [from] arguments are i, FV to move from vector FV starting at element i (integer)

[from] arguments are i, j, FM to move from matrix FM starting at element i, j (integers)

[to] arguments are k, TV to move to vector TV starting at element k (integer)

[to] arguments are k, ℓ, TM to move to matrix TM starting at element k, ℓ (integers)

[what] arguments are given by the chart:

from \ to	TV_k	$TM_{k,\ell}$
FV_i	m elements [what] as m	m elements into row k [what] as 1,m n elements into col ℓ [what] as n,1
$FM_{i,j}$	m elements from row i [what] as 1,m n elements from col j [what] as n,1	n rows × m cols [what] as n,m

Errors: none

Support: none

MPOLAR, matrix Cartesian to polar conversion

Function: This routine converts a complex vector or matrix in the STEX domain in Cartesian form to a double word vector or matrix in polar form.

Execution: implicit

MPOLAR(A)

where argument A is complex vector or matrix input in standard Cartesian form, i.e., each element represented by floating point scalars x and y such that the element = x + iy

result is double word vector or matrix in CSTAR in polar form, i.e., each element represented by floating point scalars r and θ such that the element = $x+iy = re^{i\theta}$; $0 \le \theta < 2\pi$; if $x = y = 0$, then $r = \theta = 0$

SPIREL monitoring for creation of the result is given if SL14 is off. The polar form of a complex operand is a complex operand in the Genie language, but the arithmetic operations are not defined for this representation; input, output, and storage across an equals are meaningful for the polar form and useful. In the Genie language the same execution is specified by

POLAR(A)

for non-scalar argument A.

Errors: If A does not exist, MPOLAR prints an error message and performs no operation.

Support: program MCARTN

MPOWER, matrix power

Function: This routine raises a square standard matrix in the STEX domain to an integer power, generating a unit matrix for zero power, inverting for a negative power, and multiplying for powers > 1 in absolute value. The matrix must be of floating point type.

Execution: special

input (U) = codeword address of matrix
(R) = integer power

result in non-scalar accumulator, *10

If (U) null on entry, the input matrix is taken as *10. A matrix which is not *10 is not destroyed. SPIREL monitoring for creation of the result is provided if SL14 is off and the input is not *10 with power one.

Errors: If matrix does not exist, MPOWER prints error message and performs no operation. If power ≥ 0 and matrix is not square, MPOWER uses square portion, performs operation, and prints error message. If power < 0 and matrix is not square, MPOWER prints error message and performs no operation. If power < 0 and matrix is singular, no result is given, and MPOWER prints error message.

Support: programs INV, MCOPY, MMPY

MRE, matrix real part

Function: This routine provides the real part of a complex vector or matrix.

Execution: implicit

MRE(A)

where argument A is complex non-scalar input

result is in non-scalar accumulator, *10

MRE may be used on any double word non-scalar argument to provide the first part as a single word non-scalar. SPIREL monitoring for creation of the result is provided if SL14 is off. In the Genie language the same execution is specified by

RE(A)

for non-scalar argument A.

Errors: If A does not exist, MRE prints error message and performs no operation.

Support: program MCOPY

MSPACE, matrix space

Function: This routine creates a standard matrix of zeroes in the STEX domain.

Execution: explicit

MSPACE(A,B,C)

where argument A is matrix to be created
argument B is integer number of rows in A
argument C is integer number of columns in A

Storage addressed formerly as A is freed; then, if both B > 0 and C > 0, a matrix with B rows and C columns is created. SPIREL monitoring for freeing or creating A is provided if SL14 is off. If SL14 is on, MSPACE takes "fast" space by bypassing XCWD(*126).

Errors: none

Support: none

MSUB, matrix subtract

Function: This routine forms the difference of two standard vectors or two standard matrices in the STEX domain. The operands must agree in type, floating point or scalar, and the result will be of the same type.

Execution: special

input (B1) = codeword address for the operand to be subtracted from

(B2) = codeword address for the operand to be subtracted

result in non-scalar accumulator, *10

If either (B1) or (B2) null on entry, the corresponding operand is taken as the non-scalar accumulator, *10. An operand which is not *10 is not destroyed. SPIREL monitoring for creation of the result is provided if neither operand is *10 and SL14 is off.

Errors: If either operand does not exist, MSUB prints error message and performs no operation. If dimensions of the two operands are not the same, MSUB uses the subset of the larger which corresponds to the smaller, performs the subtraction, and prints error message.

Support: program MADD

MTAKE, matrix take

Function: This routine creates an n-dimensional array of zeroes.

Execution: explicit

$MTAKE(A,D_1,\ldots,D_N,N)$

where argument A is array to be created

argument D_i is length in the ith dimension

argument $N \leq 5$ is the number of dimensions

Space addressed formerly as A is freed; then an array of size $D_1 x \ldots x D_N$ is created, to be indexed by registers B1,...,BN. SPIREL monitoring for creating A is provided if SL14 is off. If SL14 is on, MTAKE takes "fast" space by bypassing XCWD(*126).

Errors: If any $D_i < 1$, $n < 1$, or $n > 5$, MTAKE gives no result and prints an error message.

Support: none

ODD, test integer odd

Function: This routine tests an integer for being odd.

Execution: implicit

ODD(A)

where argument A is the integer input

result is Boolean value TRUE (represented by integer-0) if A is odd; Boolean value FALSE (represented by -1) if A is even.

Errors: none

Support: none

ORTHOG, matrix orthonormalization

Function: This routine orthonormalizes (by the Gram-Schmidt method) the rows of a standard matrix in the STEX domain. The matrix must be of floating point type.

Execution: implicit

ORTHOG(A)

where argument A is floating point standard matrix input

result is floating point standard matrix in the non-scalar accumulator, *10

Input matrix which is not *10 is not destroyed. SPIREL monitoring for creation of result is provided if input is not *10 and SL14 is off.

Errors: If A does not exist or if the rows are not linearly independent, ORTHOG prints an error message and performs no operation.

Support: programs MCOPY, SQR

OUTPUT, special output

Function: This routine is supplied by the user for special output from programs written in the Genie language.

Execution: in the Genie language only, by the command

OUTPUT list

where the program OUTPUT is entered once for each named variable in the list. A complex variable is treated as two items, the real part with the name of the variable and the imaginary part with the name "ditto". The program OUTPUT must be coded in the assembly language, AP1. Information is given in T7 on entry to OUTPUT as follows:

bits	
1-30	name in BCD as given in list
31-33	not used
34-36	octal 0 for scalar
	2 for vector
	4 for matrix
39-41	not used
40-54	address for scalar, codeword address for non-scalar

PLOT, plot on the printer

Function: This routine plots on the printer one floating point vector versus another or a floating point vector versus its indices.

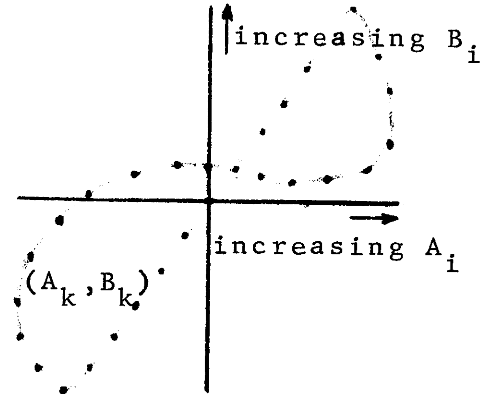
Execution: explicit

PLOT(A,B)

where argument A is a vector of x-values to be plotted across the page from min on the left to max on the right

argument B is a vector of y-values to be plotted down the page from max at the top to min at the bottom

result is one-page plot of points (A_k, B_k), k in the index range of both A and B

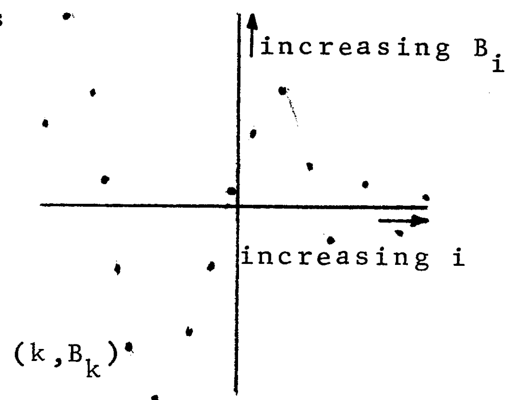


PLOT(Z,B)

where argument Z (actually zero on the B6-list) specifies use of indices for x-values to be plotted across the page from min on the left to max on the right

argument B is a vector of y-values to be plotted down the page from max at the top to min at the bottom

result is one-page plot of points (k, B_k), k in the index range of B



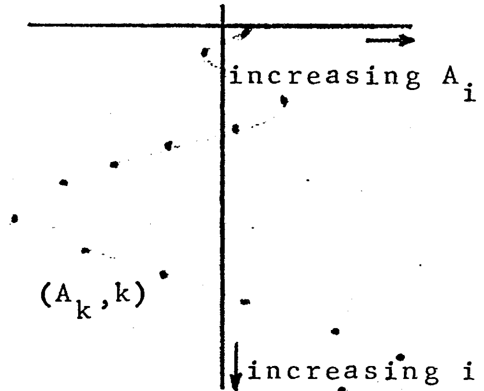
PLOT (continued)

PLOT(A ,Z)

where argument A is a vector of x-values to be plotted across the page from min on the left to max on the right

argument Z (actually zero on the B6-list) specifies use of indices for y-values to be plotted down the page from min at the top to max at the bottom

result is plot on one or more pages of points (A_k, k), k in the index range of A, one point per index value



No vector operands are destroyed.

Errors: If both arguments are Z, an error message is printed.

Support: none

POLAR, Cartesian to polar conversion

Function: This routine converts a complex scalar in Cartesian form to a double word operand in polar form.

Execution: implicit

POLAR(A)

where argument A is complex scalar input in standard Cartesian form, i.e., represented by floating point scalars x and y such that $A = x + iy$

result is double word operand in polar form, i.e., represented by floating point scalars r and θ such that $A = x + iy = re^{i\theta}$; $0 \le \theta < 2\pi$; if $x = y = 0$, then $r = \theta = 0$

The polar form of a complex operand is a complex operand in the Genie language, but the arithmetic operations are not defined for this representation; input, output and storage across an equals are meaningful for the polar form and useful.

Errors: none

Support: programs ASIN, MOD; scalar CMPLX

PRESCRIBE, line print with format and page control

Function: This routine is used in Genie programs only instead of SCRIBE to produce printed SCRIBE output with page control, headings, and page numbers.

Execution: explicit*

PRESCRIBE(A1,...,AK,F,N,TITLE,LIMIT)

where argument F is the name of the format to be used
arguments A1,...,AK are variables whose values are to be substituted successively for dummy variables in the format
(exactly as for SCRIBE)

argument N is the number of spaces after output of SCRIBE (A1,...,AK,F)

argument TITLE is either the name of a format containing only text or the name of a vector containing hexad data, to be used for title on pages (may be more than 108 characters to exceed one line)

argument LIMIT is the number of lines to be printed per page of output

and * one additional argument is supplied automatically by the Genie compiler -- minus the number of arguments A1,...,AK,N,TITLE,LIMIT stored directly on the B6-list as a negative integer after the arguments A1,...,AK,N,TITLE,LIMIT

If LINCT exceeds LIMIT on entry, PRESCRIBE prints a heading containing the specified title at the top of the next page. Then SCRIBE(A1,...,AK,F) is executed. Finally, N spaces are provided. LINCT is updated to reflect all printing and spacing by PRESCRIBE.

The heading is provided by PRESCRIBE after incrementing PAGCT by 1. It consists of a 1/2 inch margin (3 blank lines) at the top of the page, then the lines:

...data and time... ...page no...

...title supplied by user...

...blank line...

PAGCT is used as the page number, and LINCT is set to 5 plus the

PRESCRIBE (continued)

number of lines in the title after a heading print.

In a fresh Genie SPIREL, LINCTR = PAGCTR = Q. Either may be used within private programs. Both should be updated if printing is done other than through SCRIBE or PRESCRIBE. Setting LINCTR = 0 forces a new page on the next entry to PRESCRIBE. LIMIT = 60 provides a 1/2 inch margin at the bottom of the page to match the margin at the top.

Errors: same as for SCRIBE

Support: program SCRIBE; constants LINCT, PAGCT

QCONF, χ^2 confidence

Function: This routine computes the χ^2 confidence level between two floating point vectors of equal length.

Execution: implicit

QCONF(A,B)

where argument A is the theoretical distribution, real floating point vector input

argument B is the observed distribution, real floating point vector input

result is real scalar, computed as

$$Q(\chi^2 | \nu) = \int_{\chi^2}^{\infty} e^{-\frac{t}{2}} t^{\frac{\nu}{2}-1} dt$$

with degrees of freedom ν=vector length -1

Errors: If A and B are not equal in length or if either does not exist, QCONF prints an error message and gives result=1.0. If vector length <2, QCONF prints an error message and gives result=0.

Support: programs CHISQ, EXP, SQR

RANDM, random number generator

Function: This routine computes the first or the next in a sequence of pseudo-random floating point numbers evenly distributed between 0.0 and 1.0.

Execution: implicit

RANDM(A)

where argument $A \neq 0$ causes generation of the first random number, i.e., restarts the generation procedure

argument $A \equiv Z$ causes generation of the next random number (starts the generation procedure on first execution)

result is floating point scalar

Errors: none

Support: none

RE, real part

Function: This routine provides the real part of a complex scalar.

Execution: implicit

RE(A)

where argument A is complex scalar input

result is real floating point scalar

If coded in the Genie language, the library routine is not used; but the routine may be used in assembly language coding. RE may be used on any double word scalar argument to provide the first part as a single word scalar.

Errors: none

Support: none

ROW, number of rows in a matrix

Function: This routine provides the number of rows in a matrix.

Execution: implicit

ROW(A)

where argument A is matrix input

result is integer number of rows in matrix A.

Errors: If A does not exist, result = 0 and an error message is printed.

Support: program LENGTH

RTRAN, real fast Fourier transformation

Function: this program is used in conjunction with FFT by FFTC in those cases where 1) the complex input is conjugate symmetric and the output real; or 2) the input is real and even in length, and the output is complex and conjugate symmetric.

Execution: explicit (see FFTC)

RTRAN(A,B,C)

where argument A is the input/output vector (complex)

B is a Boolean variable: true if the sign of the exponent is negative, otherwise false.

C is a Boolean variable: true for a transform, otherwise false.

Errors: If A doesn't exist, an error message is printed.

Support: programs COS, SIN.

SCRIBE, line print with format

Function: This routine substitutes variables for dummy fields in a line skeleton called a format and prints the result. It may be used in Genie programs only.

Execution: explicit*

SCRIBE(A1,...,AK,F)

where argument F is the name of the format to be used

argument A1,...,AK are variables whose values are to be substituted successively for dummy variables in the format

and * one additional argument is supplied automatically by the Genie compiler -- minus the number of arguments A1,...,AK,F stored directly on the B6-list as negative integer after the arguments A1,...,AK,F

result is that printing occurs and the constant LINCT is incremented by 1 for each line.

A format is a line skeleton written as a FORMAT statement in the Genie language, as a BCD pseudo-order in AP1. A format contains text and dummy variables. Special characters are used to form dummy variables: lower case letters 'a,b,c,d,e,f', the characters '+,-,.' with 'd' and 'e', and the digits '0' thru '9' with 'f'. A dummy variable is any consecutive sequence of special characters in the format. All other characters in the format are characters of text. The use of special characters to form dummy variables is explained below.

SCRIBE operates by transferring text directly to the printed output and substituting argument values for dummy variables. The number of arguments need not equal the number of dummy variables in the specified format. If the number of arguments is less than the number of dummy variables, processing will cease when a dummy variable is encountered for which there is no argument. If the number of arguments is greater than the number of dummy variables,

SCRIBE (continued)
the format will be used as many times as necessary to substitute all arguments, and each re-use of the format will cause a new line of printing to be initiated. The processing of a non-scalar argument is handled by replacement of successive dummy variables in the specified format with successive array elements -- all words of a program, all vector elements, all matrix elements by row, and generally all data words of any array.

One or more lines may be printed on a single entry to SCRIBE. Line termination occurs due to:

- special position notation 'f0f'
  -- causes printing and initialization of the next line at print position 1 (as on entry), but scan of the format continues.
- end of format
  -- if no arguments remain to be processed, causes printing and exit
  -- if more arguments remain to be processed, causes printing, initialization of the next line at print position 1, and reinitialization of the format scan.
- dummy variable in format and no arguments remain to be processed
  -- causes printing and exit

A dummy variable consists of a string of special characters with no embedded blanks. The representation determines the type of conversion to be applied to an argument and the appearance of the output. The types of dummy variables are as follows:

A hexad dummy is formed by a string of 'a's with possibly embedded 'c's. The occurrence of any 'a' specifies hexad conversion of the argument. Each 'a' specifies the position of a hexad character, and each 'c' specifies a space within the field. Hexads are taken from the left end of the word: a hexad dummy with three 'a's will cause the three leftmost hexads of the argu-

ment specified to be printed. A machine word contains nine hexads; if there are more than nine 'a's in single hexad dummy, then more than one word of input must be used. If the argument is a scalar, successive words in memory will be used. If the argument is a non-scalar, successive array elements will be used.

An octal dummy is formed by a string of 'b's with possibly embedded 'c's. The occurrence of any 'b' specifies octal conversion of the argument. Each 'b' specifies the position of an octal digit, and each 'c' specifies a space within the field. A machine word contains eighteen octal digits, so no more than eighteen 'b's in a dummy variable are meaningful. Octal digits are taken from the right end of the word: a dummy variable with four 'b's will cause the four rightmost octal digits of the argument specified to be printed.

A decimal dummy is formed by a string of 'd's with possibly embedded 'c's, and perhaps the special characters '+,-,.,e'. The occurrence of any 'd' specifies decimal conversion of the argument. Each character in the dummy specifies a position in the decimal output. The general form of a decimal dummy is:

±d···d.d···de±d···d

The decimal point '.' will appear in the output as in the dummy. It must appear to get the fractional part of a floating point argument, and then the fractional part is rounded in the last digit. If no decimal point appears, the last digit in the integer is rounded.

The character 'e' appears in the printed output and indicates that the integer following it is the power of ten for the number in front of it. The 'e' causes output of a full field of decimal digits before the decimal point and an appropriate exponent.

Each 'd' specifies a decimal digit position in the output, before or after the decimal point or in the exponent.

A character '+' or '-' specifies a sign position in

the output, either on the number or on its exponent. '-' specifies to print a sign (minus) only if the number which follows is negative. '+' specifies to always print a sign (plus or minus). A sign is always printed immediately to the left of the most significant digit of the number to which it applies.

A decimal dummy without 'e' may be overflowed by an argument value. The number will then be truncated on the left, X being printed as the leftmost character. Thus with a dummy -dd.ddd the value 5763.4587 will be printed as X63.459.

A position dummy does not use an argument; it is formed by a pair of 'f's which bracket an unsigned integer which is the print position to move to in forming the line of output. The 'f's and the bracketed number do not appear on the printed output. The print positions are numbered from 1 to 108 from left to right. Any number of pairs of 'f's may appear anywhere except within variables on a dummy line. As a special case, 'f0f' causes printing and initialization of the next line at print position 1.

Examples: [Note _ denotes space]

dummy variable	argument value	output
aaa	hexad ABCDEFGHI	ABC
aacaa	hexad ABCDEFGHI	AB_CD
aaaaaaaaa	hexad THE_END__	THE_END__
bb	octal ...461	61
bb	octal 0...01	_1
bbcb	octal ...461	46_1
d.d	decimal 3.59	3.6
dd.d	decimal 3.51	_3.5
d	decimal 3.5	4
dd	decimal 3	_3
dd.d	decimal 3	_3.0
-d.d	decimal 3.52	_3.5

SCRIBE (continued)

dummy variable	argument value	output
-dd.d	decimal -0.52	_-0.5
+d.d	decimal 3.52	+3.5
+d.d	decimal -0.52	-0.5
d.d	decimal -3.52	3.5
d.d	decimal 35.67	5.7 or X.7
ddcddd	decimal 1024	_1_024
-dd.ddde+dd	decimal 45784.734	_45.785e_+3
+dd.ddde-dd	decimal 45.784834	+45.785e__0
+dd.dddce-dd	decimal 45.784734	+45.785_e__0

In Genie, the format given by the statement

```
SKEL        FORMAT
FIRST RESULTS    aaaacaaa    A=bbbcbbb    B=-d.dddd    C=-dd.ddce+d
```

might be used in the explicit execution command

```
EXECUTE        SCRIBE(ALPHA,AONE,BTWO,A+B,SKEL)
```

to cause the printed output

```
FIRST RESULTS    TRUE END    A=147 003    B= 4.5969    C=-47.594 e-1
```

Errors: X in decimal field as explained under decimal dummy.

Support: constant LINCT

SIN, sine

Function: This routine computes the sine of a number.

Execution: implicit

SIN(A)

where argument A is floating point scalar input

result is floating point scalar

Also, (R) = COS(A) on exit.

Errors: If $|A| \geq 2^{47}$, SIN gives result = 0 and prints an error message.

Support: none

SINH, hyperbolic sine

Function: This routine computes the hyperbolic sine of a number.

Execution: implicit

SINH(A)

where argument A is floating point scalar input

result is floating point scalar

Also, (R) = COSH(A) on exit.

Errors: If $|A| > 170.0$, SINH gives result for $|A| = 170.0$ and prints error message.

Support: program EXP

SMDIV, scalar-matrix divide

Function: This routine divides a standard vector or matrix in the STEX domain by a scalar. The operands must agree in type, floating point or integer, and the result will be of the same type.

Execution: special

input (B1) = codeword address for non-scalar operand

(U) = scalar operand

result in non-scalar accumulator, *10

If (B1) null on entry, the non-scalar operand is taken as *10. A non-scalar operand which is not *10 is not destroyed. SPIREL monitoring for creation of the result is provided if the non-scalar operand is not *10 and SL14 is off.

Errors: If the non-scalar operand does not exist or if the scalar = 0, SMDIV prints an error message and performs no operation.

Support: program SMMPY

SMMPY, scalar-matrix multiply

Function: This routine forms the product of a scalar and a standard vector or matrix in the STEX domain. The operands must agree in type, floating point or integer, and the result will be of the same type.

Execution: special

input (B1) = codeword address for non-scalar operand

(U) = scalar operand

result in non-scalar accumulator, *10

If (B1) null on entry, the non-scalar operand is taken as *10. A non-scalar operand which is not *10 is not destroyed. SPIREL monitoring for creation of the result is provided if the non-scalar operand is not *10 and SL14 is off.

Errors: If non-scalar operand does not exist, SMMPY prints an error message and performs no operation.

Support: program MCOPY

SOLN, linear equations solution

Function: This routine provides the solution to a system of linear equations represented by a square standard matrix of coefficients and a standard vector of constants, both of floating point type and in the STEX domain.

Execution: implicit

SOLN(A,B)

where argument A is a square floating matrix of coefficients
argument B is a floating point vector of constants
representing a system of n equations of the form

$$A_{i,1}X_1 + \dots + A_{i,n}X_n = B_i \quad , i=1,2,\dots,n$$

result is floating point vector in the non-scalar accumulator, *10, with the value of X_i as the ith element.

An operand which is not *10 is not destroyed. SPIREL monitoring for creation of the result is provided if SL14 is off.

Errors: If A or B does not exist or if dimensions are not proper or if a solution is not defined, SOLN prints an error message and performs no operation.

Support: program INV

SQR, square root

Function: This routine computes the square root of a number.

Execution: implicit

SQR(A)

where argument A is floating point scalar input

result is floating point scalar

Errors: If $A < 0$, SQR gives result = 0 and prints error message.

Support: none

STNDV, standard deviation

Function: This routine computes the standard deviation of any vector of floating point numbers.

Execution: implicit

STNDV(A)

where argument A is floating point vector input

result is floating point scalar

Errors: If A does not exist, STNDV prints an error message and performs no operation.

Support: program SQR

TAN, tangent

Function: This routine computes the tangent of a number.

Execution: implicit

TAN(A)

where argument A is floating point scalar input

result is floating point scalar

Errors: If $|A| \geq 2^{47}$, or if $|A|$ is a multiple of $\pi/2$, TAN gives result = 0 and prints an error message.

Support: none

TANH, hyperbolic tangent

Function: This routine computes the hyperbolic tangent of a number.

Execution: implicit

TANH(A)

where argument A is floating point scalar input

result is floating point scalar

If $|A| > 170.0$, TANH gives $|result| = 1.0$.

Errors: none

Support: program EXP

TRAN, matrix transpose

Function: This routine forms the transpose of a standard matrix in the STEX domain.

Execution: implicit

TRAN(A)

where argument A is standard matrix input

result is standard matrix in the non-scalar accumulator,*10

Input matrix which is not *10 is not destroyed. SPIREL monitoring for creation of the result is provided if SL14 is off.

Errors: If A does not exist, TRAN prints an error message and performs no operation.

Support: none

TTAKE, Triangular Matrix Take

Function: This routine creates an upper triangular matrix of zeroes in the STEX domain.

Execution: explicit

TTAKE(A,N)

where argument A is the matrix to be created
argument N is the size of the matrix (N×N)

Storage formerly addressed as A is freed; then, if N > 0, a triangular matrix is created. If N = 0, any storage for A is freed and A is cleared.

Errors: none

Support: none

VREV, vector reversal

Function: This routine reverses the order of the elements of a vector.

Execution: explicit

VREV(A)

where argument A is vector input
result replaces vector A

Errors: If A does not exist, VREV prints an error message and gives no result.

Support: none

VSPACE, vector space

Function: This routine creates a standard vector of zeroes in the STEX domain.

Execution: explicit

VSPACE(A ,B)

where argument A is vector to be created

argument B is integer length of A

Storage addressed formerly as A is freed; then, if B > 0, vector A of length B is created. SPIREL monitoring for freeing or creation of A is provided if SL14 is off. If SL14 is on, VSPACE takes "fast" space by bypassing XCWD(*126).

Errors: none

Support: None

←CDUP, duplicate U, R to CSTAR

Function: This routine takes a complex codeword in U, R and duplicates it into CSTAR, the complex non-scalar accumulator. It is used when complex arrays are subscripted to one level in GENIE.

Support: CSTAR, complex non-scalar

←COMP, compression

Function: This routine performs compression of the library. It is never executed by a user, either manually or under program control.

Execution: internal library use only -- receives control from EDIT

Errors: none

Support: none

←CSAV, save CSTAR on B6 list

Function: This routine saves or duplicates CSTAR, the complex non-scalar accumulator, on to the B6 list. It is used for complicated complex non-scalar expressions in GENIE.

Support: CSMT, complex non-scalar

←CSWP, swap (B1), (B2), or (B1) + 1 to CSTAR

Function: This routine swaps the codeword in CSTAR to (B1), (B2), and changes the backreference. (B1), (B2) is erased first if normal entry is used but not so if entry is at order 2. In the latter case, the input is assumed to be (B1) and (B1) + 1. It is used in complex non-scalar stores in GENIE.

Support: CSTAR, complex non-scalar

←ENTRY, entry to library

Function: This routine records information about entry to a library routine, the name of the routine and the PF setting.

Execution: internal library use only

Errors: none

Support: constant ←ELOC

←ERPR, error print

Function: This routine prints error message containing text from the calling program and information about the PF setting at time of the error.

Execution: internal library use only, from all programs supplying error messages

Errors: none

Support: constant ←ELOC

←EXEC, Arithmetic Evaluator

Function: This routine carries out arithmetic operations called for by arithmetic statements input to CONSOL

Execution: internal system use only, called by ←IFE

Errors: none

Support: ←TYPE, vector ←CT, FXEXP, FLEXP, ←INOU

←INOUT, input/output

Function: This routine does input and output from compiled programs to carry out DPUNCH, READ, PRINT, PUNCH, DATA, INPUT, OUTPUT, DISPLAY, and ACCEPT commands in the Genie language.

Execution: from Genie-generated code only -- by TRA (not TSR) which may be traced

input (B1) specifies operation: 0,1,2,3,4,5,6,7,10 for DPUNCH READ, PRINT, PUNCH, DATA, INPUT, OUTPUT, DISPLAY, ACCEPT respectively

list of arguments one per word following TRA, terminated by null word; each word containing name in BCD and addressing information

return to location following null word

Monitoring of the input/output operation is provided if SL14 is off.

Errors: none

Support: programs INPUT, OUTPUT

←FETC, Interpreter Fetch

Function: Used by ←IFE to fetch characters from *TEXT (174).

Execution: Internal system use only.

Errors: none

Support: none

←IFE, Interpretive Formula Evaluator

Function: This routine performs statement scanning of arithmetic expressions input to CONSOL.

Execution: Internal system use only.

Errors: If illegal syntactic expression found, prints error message and returns to CONSOL.

Support: ←LASC, ←LOOK, ←TYPE, ←FETC
vector ←CT.

- Description of Use:

  Names: The name of any library subroutine may not be used for a private name. Also any 2 character sequence which is the mnemonic for a SPIREL command may not be used as a private name. All names are external system quantities, as they are on the Symbol Table (*113).

  Type of Results: Floating point always takes precedence over fixed point. The type of a variable becomes fixed when it first appears on the L.H.S. of an equation. Storing of an integer R.H.S. to a floating point L.H.S. will cause the integer to be floated before the store. However, a floating point R.H.S. will always be stored that way regardless of the type of the L.H.S.

  Arithmetic operators: The standard set of operators are available:

  binary: +, -, /, x(lower case x)
  Multiplication may be implied as in GENIE, when unambiguous. However if A1 and B are names, A1B will not be taken as A1 x B; but A1 B will be, where the ' ' represents a space.

  unary: -, | |(abs. value bars), and +(indicates what follows is to be interpreted as an octal number).

Superscripts and subscripts are allowed following the standard GENIE conventions.

NOTE: Genie interprets $-A^B$ as $(-A)^B$. ←IFE interprets it the way it looks: $-(A^B)$

Functions: Any library or user function may be executed implicitly in an ←IFE statement. Arguments may be scalar or non-scalar, but in no case may they be complex. ←IFE cannot operate in any case on complex quantities.

For convenience of notation, functions may be raised to a power immediately after the function name and preceeding the '(args)'. For example

$$A = SIN(X)^2 + COS(X)^2$$

may be written as

$$A = SIN^2(X) + COS^2(X).$$

However, this operation is meaningful only if the function has a single, non-complex, scalar result.

NOTE: For any function which has a single scalar parameter, that parameter will be floated before execution takes place. Therefore the function FLOAT may not be used in the ←IFE language.

Explicit execution: A function may be explicitly executed with args in the following manner. A dummy variable is used on the L.H.S. of a statement that would otherwise call for implicit

execution. For example:

A = TTAKE(B,5)

will create a triangular matrix at B.
If a function executed in this manner has a non-scalar result, and does not work in place, the resulting array will be in USTAR (*10).

<u>Summary and Further Comments</u>:

1) Type of variables: real or integer scalar only, except that real non-scalars may appear in function arguments. (A single element of a non-scalar is a scalar).
2) Rank of operations:
+, -, x, /, -(unary), | |, function call, ↓, ↑.
3) Number format:
integer: 5, 376
real: .5, 5.1, 5.3*-3
octal: +533
4) Statement length: May not exceed four (4) lines on the display scope.
5) Special Display Option: If the L.H.S. is a non-subscripted variable, an equal sign ('=') if placed at the end of the statement will cause the value stored to be displayed on the scope. An HTR -- will occur. Press continue to return to CONSOL COMMUNICATION LOOP.
6) More than one statement may be included in a line, provided they are separated by a comma ','. No interdependencies are accounted for. The statements

will be evaluated in the order in which they appear. For example:

A = B+C =, F = SIN(A) =, G = G =

will display A(=B+C) and then compute and display F, and display G.

7) Other I-0:
Printing must be done with the standard SPIREL print command. There is no ←IFE equivalent to the GENIE-DATA statement.

8) ←IFE statements and SPIREL commands may not appear on the same line.

9) ←IFE and all associated programs are edited out of the library with EX EDIT. To keep them in such a system, a dummy AP1 program must be included with a single REF to ←IFE.

←LASC, Statement Scanner

Function: Performs conversion to reverse polish of an arithmetic expression of the form accepted by ←IFE.

Execution: internal system use only.

Errors: none

Support: ←EXEC, vector ←CT

←LOOK, check for special command

Function: Used by ←IFE to look for special command sequences.

Execution: internal system use only.

Errors: none

Support: none

<u>←RDUP, duplicate U to *10</u>

Function: This routine takes a codeword in U and duplicates it into *10, the non-scalar accumulator. It is used when arrays are subscripted to one level in GENIE.

Support: none

←RSAV, save *10 on B6 list

Function: This routine saves or duplicates *10, the non-scalar accumulator, on to the B6 list. It is used for complicated non-scalar expressions in GENIE.

Support: none

←RSWP, swap (B1) to *10

Function: This routine swaps the codeword in 10 to (B1) and changes the backreference. (B1) is erased first if normal entry is used but not so if entry is at order 2. It is used in non-scalar stores in GENIE.

Support: none

←TYPE, determine shape of ST entry

Function: Used by ←IFE and ←EXEC to determine shape of symbol table entry.

Execution: internal system use only

Error: none

Support: none

- Punching

The program EDIT is used for punching all library items. The punch procedure is:

(1) Load SPIREL from paper tape or magnetic tape.

(2) Load all necessary updates to the library routines -- corrections, new versions of programs and new programs.

(3) Erase (ER at console) any programs to be deleted from the package.

(4) Do not activate STEX or execute any program other than EDIT.

(5) Execute from word 2 of the program EDIT with a control word to SPIREL, manually or off paper tape.

(6) Initialization occurs -- ST and VT indices set so that all entries have tag 1 and negative indices and last entry in use is at -0. ST and VT are alphabetized.

(7) Punching of the package starts. Interrupt by turning on SL15 when enough tape is punched. CONTINUE to resume punching. EDIT exits when punching is finished.

(8) To check new punched tapes, load with SL15 on.

- Editing

The program EDIT is used for updating the SPIREL library in memory or on magnetic tape.

The edit procedure is:

(1) Load SPIREL from paper tape or magnetic tape.

(2) Load all necessary updates to library routines, named and numbered -- corrections, new versions of programs, and new programs.

(3) Do not activate STEX or execute any program other than EDIT.

(4) Execute from word 3 of the program EDIT with a control word to SPIREL, manually or off paper tape.

(5) Initialization occurs -- ST and VT indices set so that all entries have tag 1 and negative indices and last entry in use is at -0.

(6) Control returns to the console communications loop, and the library is updated in memory.

(7) Write on magnetic tape, if desired.

MAGNETIC TAPE

MAGNETIC TAPE SYSTEM

Introduction . . . .

Usage . . . .

Manual READ

Manual WRITE

Programmed READ

Programmed WRITE

System Organization . . . .

Public Systems Tape . . . .

Dynamic Dumping . . . .

Tape Format . . . .

Tape Preparation . . . .

MARK

COPY

The magnetic tape system provides areas on magnetic tape for storage of 24K memory loads of SPIREL-compatible systems. Reading and writing may be done both manually (from the console) and under program control, as for dynamic dumping of production runs.

The tape is pre-marked into blocks numbered from 1 to 40 (octal), each containing a full 24K memory load. Reading and writing operations are carried out by the magnetic tape system programs which are located at the high end of memory. There are four means of communication with the magnetic tape system:

- in the SPIREL system (and the PLACER systems) from the console entry at location 37 for manual read or manual write -- set (CC) = 37 and fetch
- with the console bootstrap tape, BOOT, for manual read or manual write
- by transferring to an octal location (57700) for programmed reading
- by transferring to an absolute location (57701) for programmed writing.

- Manual READ

From the console a memory load may be obtained from magnetic tape by the following procedure:

1) Give the "MT i" command from the console or FETCH from location 37 (octal) if the appropriate tape routines are in memory, or LOAD the bootstrap tape for the tape unit to be used.

   The magnetic tape system programs are read from tape, and the machine halts with an arrow in U. The current position (block number) is displayed in IL.

2) If IL addresses the block desired, simply CONTINUE; otherwise, set SL to the desired block number and CONTINUE.

   The block will be read from tape and checksummed. All lights and registers (except T7) will be restored to their values at the time the block was written. The control word, if any, provided when the block was written will be executed. Control will be returned to the console communication loop if the block was written with a manual WRITE or to the location immediately following the transfer to 57701 for a programmed WRITE.

3) If an invalid block number is set into SL, SL will be cleared and the machine will halt again with the arrow in U. Go back to step 2).

4) If the designated block cannot be read, the machine will halt with NO in U. Go back to step 1) to try again.

- Manual WRITE

From the console a memory load may be written onto magnetic tape. Obtain the memory load by reading from magnetic tape or LOADing the "CLEAR" tape, then reading from paper tape as desired. All lights (except IL 10, 11, 12, 13 -- the tape search lights) and registers (except T7) will be stored on tape with the memory load and should be set as desired. The manual WRITE procedure is as follows:

1) Give the "MT" command from the console or FETCH from location 37 (octal) if the appropriate tape routines are in memory, or LOAD the bootstrap tape for the unit to be used.

   The magnetic tape system programs are read from tape, and the machine halts with an arrow in U. The current position (block number) is displayed in IL.

2) Turn off "NOT WRITE" light on the transport to be used.
3) Turn on SL[1].
4) If you wish to have a control word executed when the block is read from tape, type it into U.
5) If IL addresses the block desired, simply CONTINUE: otherwise, set the desired block number into the right end of SL and CONTINUE.

   The memory will be written at the designated block and control will be returned to the console communication loop. The control word to be executed when the block is read is not executed at this time.

6) If an invalid block number is set into SL, SL will be cleared and the machine will halt again with the arrow in U. Go back to step 3).
7) If the block cannot be written without error, the machine halts with NO in U. The memory load is not destroyed. Go back to step 3) to try again.
8) Turn on "NOT WRITE" light on the transport used.

- Programmed READ

From a program, control may be passed to the magnetic tape system to read a memory load or to go into the manual READ procedure.

1) To read block K from magnetic tape, set (T7)=K and TRA to the programmed READ location, 57700.

   Operation continues as for manual READ, step 2) -- and halt with NO in U will show block which could not be read as current position in IL.

2) To pass control to the manual READ procedure, set (T7)≤0 and TRA to the programmed READ location, 57700.

   Operation continues at step 2) in manual READ, with an arrow in U and current position (block number) in IL.

- <u>Programmed WRITE</u>

From a program, the magnetic tape system may be used to dump a memory load, control then being returned to program after the write. The procedure is as follows:

1) Code

```
            set (T7)
        TRA     57701
    [return after read]
    [return after write]
```

2) The value (T7) = K, K > 0 will cause block K to be written
(T7) ≤ 0 will cause halt with arrow in U and current position (block number) in IL; then set block number for write in SL and CONTINUE

3) Control after write returns to second order beyond TRA to 57701 with all registers restored, except (T7) which indicates action taken:
(T7) = 0 if write was successful
(T7) =-1 if invalid block number given
(T7) =-2 if successful write could not be performed

4) Control upon subsequent READ of block goes to first order beyond TRA to 57701 -- no control word being executed and all registers but T7 as before WRITE.

The magnetic tape system consists of the programs BOOT, CALL, MAIN, MARK, and COPY.

BOOT is used to initialize the system from the console. With the LOAD switch it goes into memory at location 57400, searches tape backward to the nearest copy of CALL, reads CALL and transfers to it.

CALL normally remains in memory at locations 57700 to 57777. The entry points for programmed operations are in CALL, as is the entry from BOOT. CALL saves all registers except T7, CC, P2, and S, then reads MAIN from magnetic tape and transfers control there. CALL tries five times to read the nearest MAIN without error; if this fails, it searches backward for the next MAIN to try again.

MAIN normally remains in memory at locations 57400 to 57677. It controls the logic of positioning, reading, writing, handling tape errors, and unsaving. As in CALL, five attempts are made to do each read or write correctly. MAIN does not have to remain in memory. The standard end of allocatable memory for SPIREL is set at 57377. If the 300 words occupied by MAIN are required for dynamic allocation in a SPIREL system, the end of memory may be set to 57700, allowing MAIN to be overwritten. In this case, a SPIREL REORGANIZATION of the STEX domain should be requested and location 100 checked before writing to ensure that no information will be over-stored when MAIN is brought in for the WRITE operation.

The memory arrangement used by the magnetic tape system is as follows:

10-57377	dumped on and read from magnetic tape
40-77	used by MT system to store fast and special registers
57400-57677	MAIN program
57700-57777	CALL program

A public systems tape for transport 3 is maintained by the Computer Project. It contains **copies** of the programming systems: SPIREL and PLACER. Each user is assigned a block for storage of a private system. For the protection of all users, programmed writing is inhibited on the public systems tape.

System tape maintenance involves three system tape reels. Tape A is in current use and is copied once a week. Tape B is 0 - 1 week old, and tape C is 1 - 2 weeks old.

Once a week tape A becomes tape B, tape B becomes tape C, and tape C is used for the new tape A. As much as possible of tape B is copied onto tape A. Blocks which cannot be read from tape B are written as zeros on tape A.

If tape A becomes unusable, a new tape A will be copied from tape B. If tape A becomes unusable and tape B cannot be copied, tape C becomes tape B, and tape B may be copied as tape C to have three reels on hand.

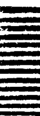
It may be necessary for a user to fill or update his block on a new system tape A. Users have access to tapes B and C for reading, NOT FOR WRITING, but this will not solve all problems. A user should always be able to regenerate his block from paper tape.

At the time a new tape A is written re-allocation of blocks may be made on the basis of computer usage; watch for this so that you are always writing in your own block.

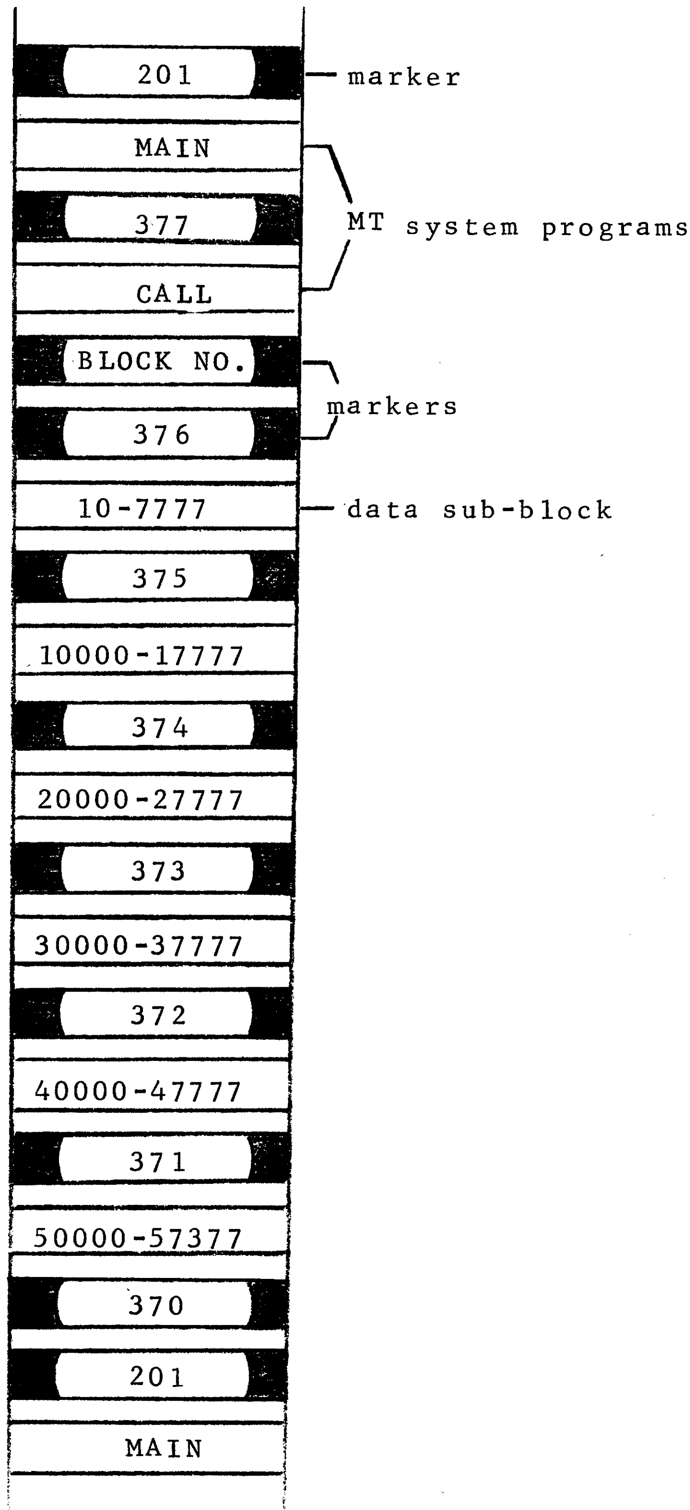
The magnetic tape system programs are easily used in a dump-restart procedure for a production package. Dumping may be cycled through a series of blocks, so that the past several dumps are always available.

At each dump the date, time, and number of the dump along with any results of interest may be printed. P2 and T7 may be saved if these are necessary in the running program. After each dump control returns to the second order beyond the TRA to 57701, and the system may continue to run.

Restart is accomplished by a manual READ, and control returns to the first order beyond the TRA to 57701. Initialization of any sort may be done, and then the system may continue to run.



The present system provides 40 blocks written in sub-blocks of about 10,000 words separated by markers as shown below. The number of blocks on the tape, the size of the segments, the total amount of memory written, and the logical transport on which the system runs may be changed by simple edits of the symbolic programs.



- MARK

The MARK program loads at location 57300, marks a system tape in the format described, fills all blocks with the same content, does not check for bad areas on the tape or for unsuccessful writing. The system used as content for the blocks may not have memory in use above location 57277.

To prepare a tape with MARK:

1) either a) clear memory with the "CLEAR" tape and load SPIREL from paper tape
   or b) load SPIREL from magnetic tape
   or c) load some other system, which you wish to have written in all blocks on the tape.

2) Self-load CALL for the transport to be used; machine will stop with

   (I): Z HTR Z

3) Self-load MAIN for the transport to be used; machine will stop with

   (I): Z HTR Z

4) Self-load MARK for the transport to be used; machine will stop with

   (I): HTR 57300

5) Turn off "NOT WRITE" light and CONTINUE. Tape will be marked, and machine will halt as for manual READ. Normally 40 (octal) blocks are provided. The number of the block being written is kept in B1, and the program may be stopped short of 40 blocks.

Paper tape copies of CALL, MAIN, and MARK for transports 2 and 3 are available in the programming office.

- COPY

The COPY program loads at location 7000 and marks a tape in the format described. It writes each block on the new tape as directed by a control paper tape -- as a copy of a block on the old tape or blank. As each block on the new tape is written, it is checked for being written correctly and for being readable. "Bad spots" on the tape are detected and avoided.

To prepare a tape with COPY:

1) Mount the old tape as logical tape 3.
2) Mount the new tape as logical tape 2 with the "NOT WRITE" light OFF.
3) Self-load CALL for the logical unit on which the new tape will be used.
4) Self-load MAIN for the logical unit on which the new tape will be used.
5) Self-load COPY, and a HTR to the first instruction of COPY (at location 7000) will occur.
6) Position control paper tape (format described below) in the reader.
7) CONTINUE to location 7000 to rewind both tapes and start the copy procedure. Bypass the order at location 7000 to avoid rewinding the old tape which must be positioned before the first block to be copied. Bypass the order at location 7001 to avoid rewinding the new tape. In any case, the copy procedure is begun at location 7002 by writing leader of markers 201 on the new tape.
8) The copy procedure reads from the control paper tape for each block of the new tape. The number of the block being written is maintained in PF.
9) If there is no paper tape in the reader, COPY will hang on a a read order and the control information for the block number shown in PF may be typed into U as it would have been read from the control paper tape.

10) All reads from the old tape are checked for parity and checksum; they are repeated until both are correct. Five read failures on the same sub-block (there are six in each block) will cause COPY to halt; three options are then available:

    (a) Pushing CONTINUE will cause writing of a sub-block of zeros instead of the copy from the old tape. The block will be readable, and the checksum correct.

    (b) Typing a number into B6 and fetching from F0 will cause that number of further read attempts to be made. If they all fail, COPY will return to the same halt.

    (c) Typing a number into U and fetching from F3 will cause the block with that number on the old tape to be written instead of the one found unreadable.

11) All writes on the new tape are checked for parity and word-to-word correspondence to what should have been written. After five unsuccessful writes, COPY fills the unwritable section with markers 200 and tries again.

12) The copy procedure is terminated by simply letting COPY hang on a read paper tape order.

The COPY control tape contains a directive for each block to be written on the new tape, these being given for blocks 1,2,... in order. Each directive consists of exactly three punches:

cr NN

where NN is the two-digit octal block number of the block to be copied from the old tape. If a block on the new tape is not to contain a copy of a block from the old tape but is to be written as zeros, punch NN as 00.