N O T E S

on the

G E N I E C O M P I L E R

for the

R I C E U N I V E R S I T Y C O M P U T E R

January, 1964

TABLE OF CONTENTS

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

```
          DEFINE
          declarations of external variables and non-scalar
             parameters for the entire definition set
          constant codeword address specifications for external
             variables
          function specifications
PROG1(PARAM1).=SEQ                                           1st program
          declarations of internal variables                 in definition
          remarks                                            set
          constant specifications
          command sequence for the calculation
END
PROG2(PARAM2).=SEQ                                           2nd program
                    .                                        in definition
                    .                                        set
                    .
END
                    .
                    .
                    .
PROGn                                                        nth program
                    .                                        in definition
                    .                                        set
                    .
END
          DEFINE
LEAVE
|cr stop  |1st tab stop
```

A definition, then, is a collection of programs (in the most usual case just one) which depend 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 operator '.=' followed by the symbol 'SEQ' signals initiation of code generation for the program. '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.

NAMES

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 spaces intervening).

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   12   PQ29   Dog3

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

ab   A B38   Pt4p   M5ef   w10

and will be interpreted respectively as:

a×b   A×B38   Pt4×p   M5×e×f   w×10

In scanning from left to right to collect the characters which comprise a name, the appearance of a character which cannot be concatenated by rule 2) or of a space will terminate the collection. Any number of characters may be used in a name, but only <u>five</u> will be retained by the compiler. If lower case Roman letters are imbedded in a name, the first is tallied as two characters. The names

m   Man

will be printed and stored internally as

.M   M.AN

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

and	COL	FALSE	LOG	READ	T5
ATAN	CONTR	FIX	MATRI	REM	T6
B1	COS	FOR	MSPAC	REPEA	T7
B2	COT	FUNCT	not	RESUL	TAN
B3	DATA	I	NEO	ROW	TRAN
B4	DEFIN	if	NUMBE	S	TRUE
B5	END	IL	or	SCALA	U
B6	EOV	INTEG	PF	SIN	VECTO
BCD	EVEN	INV	PRINT	SL	VSPAC
BOOLE	EXECU	LENGT	PUNCH	SQR	WAIT
CC	EXP	LET	R	T4	X
					Z

NUMBERS

A string of decimal numerals

$$DDD < 2^{14}$$

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

A.B*C

which is interpreted to mean

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

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 <u>octal configuration</u>. [A '+' between two numbers is binary and will 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
+3	octal

VARIABLES

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

Internal variables must be scalars (integers or floating point numbers), and these are assigned storage within the program. Internal variables do not retain their names after compilation; 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 (floating point scalar, integer, or Boolean), or non-scalar (program, vector, or matrix), and all non-scalars must be external. At the time the program is run, an 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. External variables of any one program are the common property of all programs in the machine at running time, and the names must have unique meaning throughout the system. All external variables of a program must appear in the definition set containing that program before any 'SEQ'.

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 'SEQ'. Parameters are neither internal nor external with respect to the program in which they appear, but while running will fall into one of these categories with respect to dynamically higher level programs. Parameters of a program are only representative of those variables 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 its own system variables specified as parameters; 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 used only in compilation, and are not retained while running a program.

DECLARATIONS

The form permissible for declarations are illustrated by:

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

Either a singular or a plural declaration identifier is permitted; it is followed by one or more variable names, separated by commas.

Before any 'SEQ' all external variables and those parameters which are not floating point scalars must have their types specified. Declarations for use in this area are:

INTEGER	for integer scalar, vector of integer elements, matrix of integer elements, or function with integer result
SCALAR	for floating point scalar
BOOLEAN	for Boolean scalar, vector of Boolean elements, matrix of Boolean elements, or function with Boolean result
VECTOR	for data vector, elements assumed to be floating point scalars unless also declared 'INTEGER' or 'BOOLEAN'
MATRIX	for data matrix, elements assumed to be floating point scalars unless also declared 'INTEGER' or 'BOOLEAN'
FUNCTION	for program whose name is not in the vocabulary of the compiler, result assumed to be floating point scalar unless declared to be non-scalar ('VECTOR' or 'MATRIX') and/or non-floating point ('INTEGER' or 'BOOLEAN')

Not more than one declaration in each group may be applied to a single variable.

Internal variables are scalars: integers, floating point numbers, or Boolean variables. If the first appearance of an internal scalar is on the left hand side of an equation, it assumes the type of the expression on the right hand side. If its first appearance is on the right hand side of an equation, an internal scalar is assumed to be floating point unless it has been explicitly declared as an integer or a Boolean variable. The only declarations meaningful for internal variables are:

INTEGER	for integer scalar
BOOLEAN	for Boolean scalar

FUNCTIONS

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)

Implicite 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 result as follows:

RESULT=scalar or non-scalar arithmetic expression

|cr |1st tab

The parameters of a function are given as an ordered list of those quantities which are supplied to the function routine by the program which causes it to be executed. When a function is used within a program a parameter which designates a quantity to be calculated by the function must be specified as a simple variable name; other parameters may be given by any arithmetic expression. For example, if F(A,B,C) is defined such that A and B are used in the calculation of 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 parameter may not be an expression. In the definition of a Genie program and in the use of it in other Genie programs care must be taken that parameters are always listed in the same order and that the number of parameters and their types are the same at each occurrence. In a Genie program a function name must appear with parameters following, as SIN X2 or CALC(q) or MAP(g,VAR), except in declarations. As a consequence, function names may not be used as parameters of other functions.

If a function is to be executed implicitly and its output is not a floating point scalar, then its name must appear in declarations to define the output as well as in a function declaration. Thus, the function with its parameters is an operand which must be assigned the type of its output if it is to appear within an arithmetic expression.

Every Genie program is a function. It may be used as such by any other Genie program but it may not use itself. The appendix discusses details that will be of interest to the user who wishes to code functions in a lower level language.

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

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

|cr |1st tab

The function f may then be used implicitly within the command sequence of a program in the definition set, as in the command

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

where the closed subroutine f will be applied to the parameters m and n. During compilation, output for f will be produced independent of that for the programs in the definition set. The function is external to the programs in the definition set and may be used implicitly by any program at running time since its name will appear on the symbol table.

There is a collection of function names known to Genie. These names need not be declared as functions.

NAME	CODEWORD ADDRESS	DESCRIPTION	
*** for implicit execution only ***			
SIN(A)	200		'A' floating point scalar input; result floating poing scalar
COS(A)	201		
SQR(A)	202	$A \geq 0$	
EXP(A)	203	$A < 170.0$	
LOG(A)	204	$A > 0$	
ATAN(A)	205	$\|result\| < \pi/2$	
TAN(A)	206	$\|A\| < \pi/2$	
COT(A)	207		
LENGTH(A)	210	'A' vector; result integer length of A	
ROW(A)	210	'A' matrix; result integer number of rows in A	
COL(A)	211	'A' matrix; result integer number of cols in A	
FIX(A)	217	'A' floating point; result integer nearest to A	
INV(A)	224	'A' matrix; result matrix which is inverse of A, if A non-singular	
TRAN(A)	225	'A' matrix; result matrix which is transpose of A	
EVEN(A)	227	'A' integer scalar; result Boolean true or false	
*** for explicit execution only ***			
†VSPACE(A,B)	213	'A' vector, 'B' integer; takes space for A of length B	
†MSPACE(A,B,C)	214	'A' matrix, 'B' integer; 'C' integer takes space for A, B rows by C cols	
†CONTROL(n,+WXYZ,r,f)	230	'n' integer, 'WXYZ' octal, 'r' octal or integer, 'f' name; control word is composed and *126 in SPIREL is executed, as explained in write-up of SPIREL	

†SPIREL monitoring on the printer is provided if sense light 14 is off.

CONSTANTS

Constants of a program may be numerically specified by a 'LET' statement appearing (except as noted below) within a 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 floating point number 3.14159 to be used in the program each time the internal variable name 'PI' appears. A 'LET' statement causes no code to be generated.

An internal integer value may be specified if the variable has first been appropriately declared, as

```
          LET K=3
```

An octal configuration (right justified) may be specified, but the variable should not be declared as an integer, as

```
          LET MASK=+777777077
```

where the + inflection concatenated immediately to the left of a number denotes octal conversion of the number.

A Boolean value (TRUE or FALSE) may be specified if the variable has first been appropriately declared, as

```
          LET t=TRUE
```

or

```
          LET No=FALSE
```

A fixed codeword address may be specified, as

```
          LET #CDWD=+265
```

so that the codeword for the function, vector, or matrix named CDWD will be addressed at machine address 265 instead of in the value table. This is the only LET which may appear in a definition set outside a program. 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 valuse of non-scalars may not be specified in a 'LET' statement.

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

```
LET A=3, z=5.41, #PROG=+247
```

There are two other commands which identify names with values. They are explained later: BCD in the section on alphabetic printing, and NUMBERS in the section on assembly language. Both of these commands are non-executable and must be transferred around, and must therefore be used with care.

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'.

REMARKS

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

```
            REM          _ _COMPUTE _FIRST_VALUE
|cr         |1st tab     |2nd tab
```

where _ indicates a typed space. The statement may be continued to succeeding lines at the 3rd tab position by using the 'cr tab tab tab' sequence.

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 final output listing.

All statements of a program from the 'SEQ' to and including the 'END', except 'LET's, remarks, and declarations, cause code to be generated. Such statements are called commands. The occurrence of a label on a command causes a command sequence to be initiated. The ordered set of all command sequences of the program is called the command sequence for the calculation. Each command falls into one of three categories; arithmetic, control, or input-output. 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 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, . . .

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 i^{th} depends on the j^{th} and i>j (counting from left to right), then the j^{th} equation will be coded before the i^{th}. 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.

An operand in Genie is a single variable, a function name followed by a parenthesized list of arguments, or an expression enclosed in parentheses which dictate order of computation 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 arithemtic operations are:

unary inflections: -, |...|, and 'not'

subscription

exponentiation

X and /

+ and binary -

Arithmetic operations that are permitted within an arithmetic expression on the right hand side of an equation are:

1) +, -, X, / between integer or floating point scalar operands.
If the operands are both integer or both floating point, the result will be of the same type. If the operands are of different types, the integer will be floated before the operation is carried out, and the result will be floating point.

2) +(or), -(symmetric difference), X(and), /(symmetric sum) between two Boolean scalar operands.
Combination of Boolean operands yields a Boolean result, by the following rules:

+	TRUE	FALSE
TRUE	TRUE	TRUE
FALSE	TRUE	FALSE

-	TRUE	FALSE
TRUE	FALSE	TRUE
FALSE	TRUE	FALSE

X	TRUE	FALSE
TRUE	TRUE	FALSE
FALSE	FALSE	FALSE

/	TRUE	FALSE
TRUE	TRUE	FALSE
FALSE	FALSE	TRUE

The octal representations for the Boolean values are

TRUE 0077777777777777777
FALSE 0077777777777777776

3) +, -, X between two non-scalar operands containing integer or floating point 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 two matrices yields a matrix. Multiplication of a vector and a matrix yields a vector. And multiplication of two vectors yields the scalar product which is a scalar. If the operands are both integer or both floating point, the result will be of the same type. If the operands are of different types, the integer operand will be floated

before the operation is carried out, and the result will be floating point.

4) × between integer or floating point scalar and integer or floating point 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, the result will be of the same type. If the operands are of different types, the integer operand will be floated before the multiplication is carried out, and the result will be floating point.

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

6) Exponentiation between two integer or floating point scalars.

If either or both of the operands is floating point, the result will be floating point. If both of the operands are integers, the result is an integer, zero if the exponent has a negative value. Note that A^B is typed 'A sup B sub', using the superscript and subscript 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.

7) 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) and IL (the indicator light register) are in the vocabulary of the compiler and fall 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 on and FALSE if it is 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.

8) 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 or floating point) 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 must be observed as for exponentiation.

9) Unconventional subscripting by integer scalar operands.

Under normal conditions, only standard vectors and matrices will have their elements addressed with the subscript notation.* But any operand may be subscripted by as many as five integer operands separated by commas. The operand which is subscripted will be indirectly addressed after the integer subscripts are loaded into B1, ..., B5 from left to right. Data arrays and arrays of programs can be handled with SPIREL if such elaborate addressing is desired.

10) Unary - applied to an integer or floating point scalar operand.

The negation of the operand takes place before it is combined with any other across a binary operation. This rule is unambiguous but leads to a possibly unexpected interpretation in the case of $-A^B$. Code is generated to form $(-A)^B$. Inflection of the expression A^B should be written $-(A^B)$.

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

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

12) Unary 'not' 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. The complementation rule is

not A=FALSE if A=TRUE
=TRUE if A=FALSE

The variable on the left hand 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 left hand side variables in a command <u>must</u> be distinct, no scalar or non-scalar defined more than once and not more than one element of one non-scalar defined in any one command.

The '=' joining left hand side to right hand side of an equation causes storage of the computed right hand side into the location or array specified on the left hand 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 right hand side dominates and will be stored as calculated, no conversion taking place. A non-subscripted non-scalar on the left hand side must have base indices one. If the right hand side is non-scalar, the storage addressed by the codeword on the left hand side is freed through STEX, the storage control routine in SPIREL, before the store across the '=' takes place.

Genie has the ability to apply the commutative laws of arithemtic to reorder the terms of an expression to provide calculation using a minimum number of temporary stores. In the coding for a scalar expression, the compiler may use the fast 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 scalar variables that are referred to often in an equation. The codeword at machine address 240 is used in the code by the compiler as an accumulator for vectors and matrices produced in the course of evaluating the right hand side of a non-scalar equation. This address may not be used by a coder. Temporary storage for non-scalars is always on the B6-list. See the appendix for more details.

*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. Standard forms of non-scalars are discussed further in the appendix.

CONDITIONAL
ARITHMETIC COMMANDS

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 the E_i are arithmetic expressions and the P_i are <u>predicates</u>, expressions which are 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 no P_i is true, for i = 1, 2, ..., n, then A is evaluated as E_{n+1}. E_{n+1} may be omitted from the command, in which case A is not evaluated if all predicates are false. A <u>Boolean predicate</u> is simply a Boolean expression. An <u>arithmetic predicate</u> is of the form L r R, where L and R are arithmetic expressions and r is a <u>relation</u>, one of =, ≠, <, ≮, ≤, ≰. A <u>compound predicate</u> is formed by joining simple predicates with the operations 'and' and 'or', as in

$A = 1.0$ if ($B \le C$ or $|C+D| \ne 3.72$) and SL^5 + not(SL^n)
$D < m+p$, 2.0 if $x < 0.0$, 3.0

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

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

arithmetic operations
relations
'and'
'or'

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

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

F_1 r F_2 and F_2 r' F_3

Genie requires a precise sequence of typed characters for the negated relations:

≠ is typed ' = backspace uc | '
≮ is typed ' < backspace uc | '
≰ is typed ' ≤ backspace | '

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 trnasfer command, illustrated by

CC = #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 right hand side. The '#' inflection is analagous to the 'a' bit in APl.

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

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

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.

LOOP
CONTROL 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 parameter of the iteration is P. The initial value of P is given by A, which may be a constant, a single variable, or an arithmetic expression. The positive or negative increment by which P is stepped at the end of each iteration is given by B, which may be a constant, a variable, or an arithmetic expression. The final value of P is given by C, and the loop will be traversed until P exceeds C in numerical value. The elements of the FOR command must be scalars, either integers or floating point numbers. A 'REPEAT', followed immediately by a carriage return, must be written for every 'FOR'.

Loops may be nested to any level, but distinct iteration parameters must be used at each level within a nest. 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. The 'REPEAT' is considered to be within the loop which it terminates; the 'FOR' is not. Any 'FOR' or 'REPEAT' may be labelled for purpose of transfer to it. If addressed from outside the loop, the iteration parameter will have the value it had upon exit from the loop.

The code generated by the compiler when a FOR command is encountered:

1) sets the iteration parameter to the initial value
2) transfers control to the command beyond the corresponding

REPEAT if the current value of the increment is positive/negative and the current value of the iteration parameter is greater than/less than the the final value, or else to the first of the commands of the loop.

The code generated when a REPEAT command is encountered:

1) sets the iteration parameter to its current value plus the increment (which may be negative), as specified in the corresponding FOR command

2) transfers to step 2) of the FOR sequence described above.

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 when a constant increment = ± 1 is specified. The section on fast registers discusses coder usage of machine registers.

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 simple conditional EXECUTE command is allowed, such as

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

And a compound unconditional EXECUTE command is allowed, such as

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

The input-output commands are:

DATA list
PRINT list
PUNCH list
READ list

|cr |1st tab

where 'list' denotes a collection of names (which may have been assigned machine addresses in 'LET' statements), not expressions, of scalars or of non-scalars with base indices equal to one. Functions (or program) names may not appear in the argument list of an input-output command. Neither may vector or matrix elements in the subscript notation be designated in such an argument list.

The DATA command provides reading of manually punched signed decimal numbers from paper tape. The list given in the command may contain any type of variable. 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^{15} 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 'e 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	328 cr	converts to	integer 328
	46.9cr		floating point 46.9
	.469e2cr		floating point 46.9
	-5391cr		integer -5391
	-69.e-1cr		floating point -6.9

Scalars must be punched as single numbers in the format described. A vector of length n is punched as the sequence of n+1 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

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 PRINT command provides output on the fast line printer of any named scalar or non-scalar quantities. These are labelled by the name given in the routine in which the PRINT command appears. Scalars are printed one per line. Vectors are printed five elements per line. Matrices are printed by row, five elements per line.

The PUNCH command and the READ command may be applied only to variables which are named on the symbol table at the time the command is executed. All external variables of the program in which the 'PUNCH' appears and those parameters which at the time of execution are indeed external in some dynamically higher level program fall into this category. Care must be taken to apply these commands properly to parameters as there are no checks built into the compiler or input-output program to insure presence of a particular name on the symbol table. 'PUNCH' provides, for each variable listed, a single control word, followed by the name as it appears on the symbol table, followed by the data in hexad with checksum. For a scalar the SPIREL control word has wxyz=0040; for a vector the control word has wxyz=0240; for a matrix the control word has wxyz=0440. These output paper tapes may be loaded through SPIREL symbolically or they may be read with a READ command. In fact, only tapes of the form produced by a PUNCH command may be read by a READ command.

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.

T7 may be used only for output of a scalar from a single valued function that will be executed implicitly. The command executed immediately before 'END' in such a program may be of the form

```
                T7 = calculated output
|cr             |1st tab
```

T6, T5, and T4 may be used within a command as the names of scalar variables computed in other than the first equation of the command. Genie 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. Only T6, T5, T4 are available for this purpose, and they should be called upon in this order since Genie will use only Ti for i less than the smallest Tj mentioned by the coder. The command

$$M=T6/T5,\ T6=a+b,\ T5=(c^2+c-4.1)/d$$

is an example of coder use of fast registers. The values in T6, T5, T4 are not preserved by Genie from one command to another as they are subject to use in Genie-generated code 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 commands.) Efficiency of code is gained if these registers are used as subscripts or as iteration parameters of loops with explicit increment ±1. The index registers B1 and B2 may be used only if the user understands Genie coding conventions as explained in the appendix and can accurately anticipate the use of these registers by Genie generated code. The registers B6 and PF may not be used in Genie language but may be used in the assembly language if compatibility with Genie generated code is maintained.

The assembly language recognized by Genie is called AP2. Instructions in the AP2 language may interspersed at will with commands in the Genie language within the command sequence for a Genie program. AP2 is discussed in detail in a separate write-up.

Frequent use will probably be made of AP2 language for setting of sense lights since no notation for this operation exists within the Genie language. To turn on sense light 3:

```
                    SLN        +10000
|cr        |1st tab  |2nd tab  |3rd tab
```

When the assembly language is employed, it may be desirable to dictate placement of numbers within a program at a particular point. The Genie command illustrated by

```
CONST     NUMBERS 36.5, -2.8, 6, +774777
|cr       |1st tab
```

provides this facility. In the program Genie generates, in this case,

floating point 36.5 at CONST
floating point -2.8 at CONST+1
integer 6 at CONST+2
octal 774777 (right justified) at CONST+3

The command may or may not be labelled. One or more numbers (each but the last followed by a comma) are listed, and the list may be extended onto succeeding lines by use of the 'cr tab tab tab' sequence. The words generated are not executable, so transfer around NUMBERS commands must be explicitly coded.

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 the appendix.

ALPHABETIC PRINTING

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

```
MESS1      BCD        _ _TEMPUS_FUGIT
|cr        |1st tab   |2nd tab
```

where _ indicates a space when typing. 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. Of course, what is generated is not executable, so transfer around BCD commands must be explicitly coded.

Once alphabetic information has been specified, it may be set into the print matrix at any position on the line, one word (i.e., nine characters) at a time, and then printed with program *127 in SPIREL. An AP2 code sequence for printing MESS1 starting at print position 12 is

```
           PF         RPA        RSPF
           Z          SB3        12,U→B1
                      CLA        MESS1,U→T7
                      TSR        *+127,B1+1
                      CLA        MESS1+1,U→T7
                      TSR        *+127
                      TSR        *+127,B1+1
RSPF                  SPF        Z
|cr        |1st tab   |2nd tab   |3rd tab
```

Detailed discussion of program *127 may be found in the write-up on SPIREL. For printing MESS1 at the left hand margin, the Genie language command

```
           EXECUTE CONTROL(2,+4010,0,MESS1)
|cr        |1st tab
```

with SL14 on will provide the desired output. The parameters in this command indicate that two words starting at the location named MESS1 are to be printed in hexad form. Printing is produced 108 characters per line, as many lines as necessary. In the example 14 characters require two words of storage, hence the value 2 for the first parameter to CONTROL. The function CONTROL is explained in the FUNCTIONS section.

SIZE RESTRICTIONS

The sizes of command sequences and programs generated by the Genie compiler are limited by the size of the memory. With 8K of memory no command sequence may cause generation of more than 300 (octal) instructions, and the entire program may not exceed 1000 (octal) instructions in length. The compiler does not check for overflow, but it should be apparent at time of compilation if the limits are exceeded. No absolute correspondence can be established between the length of a Genie program in symbolic form and the length of the absolute program it causes the compiler to generate. Roughly, though, a page of Genie language segmented into four command sequences should not exceed the size restrictions imposed on the code generated. A remedy for size restrictions on programs is found in the ability to break a single program into several within the same difinition set.

While compiling, the number of private symbols which may be stored is 70 (decimal). While running a system, the standard SPIREL allows for 64 external names on the symbol table.

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.

Only statement labels, the program name, 'END', and 'LEAVE' are typed at the margin.

'REM' and 'BCD' are followed by a 'tab' punch.

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

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

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.

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

Spaces may appear anywhere but within a name or number; 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 | '

$\not<$ is punched ' < backspace uc | '

$\not\leq$ 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.

Function definitions and program name may appear either at the left margin or at the 1st tab position.

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

The Genie PLACER system provides operations on symbolic and absolute Genie tapes. It is located on the MT System magnetic tape at block 101.01. When this PLACER is read into memory program *240 is executed, and the stop

(I): 00 HTR CC

occurs. The set of options to be exercised should 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	compile (edited) symbolic tape
SL^7	back-translate absolute tape

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 options in PLACER. Pushing CONTINUE causes the specified operations to be carried out in order as described below:

<u>SL^7, BACK-TRANSLATE.</u> The stop

(I): 07 HTR CC

occurs if the absolute tape to be translated is not in the reader. Options as explained in the separate section on the back-translator may be set into the sense lights. Pushing CONTINUE causes the translator to read the tape and create in the machine a symbolic tape image.

<u>SL^1, READ.</u> The symbolic tape to be read must contain <u>only one</u> definition set, this begun with one carriage return and terminated by two carriage return punches. 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.

<u>SL^2, EDIT.</u> The stop

(I): 02 HTR CC

occurs. The edit tape is placed in the reader. Pushing CONTINUE causes this tape, which must contain <u>only</u> 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 correction is specified by three parameters: the initial carriage return number (i), the final carriage return number (f), and the number of lines in octal in the symbolic correction (n). A line in a symbolic tape is terminated by a carriage return, these being numbered from 1 on listings. The n lines of a correction will replace the portion of the program read from and not including carriage return i through carriage return f. Note that n=0 effects a deletion. The last line of a symbolic tape must not be replaced. On a single edit tape f of one correction may not equal i of another correction. The format for punching the correction parameters is:

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

SL^3, PUNCH. The tape image in the machine is punched out on paper tape.

SL^4, LIST. The tape image in the machine is listed on the fast line printer with carriage return numbers. A lower case Roman letter is printed as '. upper case letter '. Superscripts and subscripts are printed above and below the main line. Unfortunately, '$\neq$' prints as '|', the '=' being lost because the two characters are too close to each other on the print wheel.

SL^5, CHECK. The stop

(I): 05 HTR CC

occurrs if the tape to be checked is not in the reader. 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.

SL^6, COMPILE. The stop

(I): 06 HTR CC

occurs. The symbolic Genie tape is placed in the reader. Pushing CONTINUE causes the tape to be read. This reading is very irregular as the text is being processed by Genie as it is read.

When an 'END' statement is read, output of the program is provided on the printer and the absolute tape is punched. The final 'DEFINE' statement causes printing of the external variables of the program just compiled. Then the 'LEAVE' statement causes exit from the compiler to PLACER control program *240.

It is sometimes desirable to obtain symbolic AP1 listings or tapes for programs which exist in absolute form only. These programs may have been compiled or absolute-coded so that no listings exist, or listings which once existed may have been lost.

Symbolic listings for documentation and tapes which may be edited are generated by the AP1 Back-Translator loaded as part of Genie PLACER.

The back-translation is in the form of a symbolic tape image in the same form as is generated when an ordinary symbolic tape is read under PLACER control. All operation mnemonics in the extended AP1 vocabulary are recognized, and symbolic addressing is set up when instructions reference locations within the program. For most programs, instructions are distinguished from data words, and the data words are translated to OCT pseudo-orders.

The types of tapes which may be back-translated are:

1. SPIREL-loading relative programs in any punch format
2. SPIREL-loading absolute programs in any punch format
3. SELF-loading programs in octal or hexad.

The first word on the tape determines the type of the tape; it is not necessary to make any other indication. If single control words, such as base-changing control words, are on the tape, they are passed over; sections of tape with symbolic cross-references are also ignored.

Usage

If SL^7 is turned on at the normal halt (I): 00 HTR CC in Genie PLACER, a program will be read from paper tape and a symbolic tape image constructed in memory. This symbolic tape image is equivalent to one generated by the READ option (SL^1), and may then be listed, punched, or edited.

If SL^7 is the only sense light turned on, or if there is no paper tape in the reader, a halt occurs with (I): 07 HTR CC. At this time, certain sense light options on the back-translation may be selected (see below). After the symbolic tape image is created, control retruns to PLACER at the normal halt ((I): 00 HTR CC).

If more than one sense light is turned on at the 00 HTR CC and if there is tape in the reader, the 07 HTR CC will be by-passed. In either case, a symbolic tape image will be generated first, and then the other specified options (print, punch, etc.) will be performed on the new image.

Options

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

1. A flow analysis of the program to determine which words may be executed as instructions and which are internal data words or constants
2. the 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	Data word not explicitly referenced in the program
tag 1	Data word explicitly referenced in the program
tag 2	Instruction not explicitly referenced in the program
tag 3	Instruction 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 types of programming which cannot be analyzed properly are those which involve

1. entry points at other than the first instruction of a program,
2. use of transfer vectors or computed transfers within a program (e.g., TRA CC+B3),

3. use of the X register, as in JMP in the operation field or CC+X in the auxiliary.

Four sense light options are provided to make it possible to specify as executable instructions those words which would not otherwise be identified as such. These sense lights must be set at the 07 HTR CC as described above.

SL^{12}. Do not perform control flow analysis, but translate on the basis of the tags on the program as read.

SL^{13}. Accept a list of extra entry points or other words which must be identified as instructions. If this option is selected, a 13 HTR CC will occur immediately after the program tape is read. At this time the back-translator will accept added entry points from paper tape punched in the special format

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

where [cr] is a carriage return and AAAAA, BBBBB, CCCCC, ... are five-digit (octal) relative locations in the program. The process is terminated when the end of tape is detected. Note that it is only necessary to specify the first word of a block of instructions (a block is ended by an unconditional transfer instruction, either explicit or implicit).

SL^{14}. Punch the program with tags after the flow analysis.

SL^{15}. Do not perform translation to symbolic tape image.

SYMBOLIC ADDRESSING IN SPIREL

In the Genie language quantities are normally identified by name, not by the machine address where the corresponding value or codeword is located. The SPIREL system 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 justified in T4, and entry made to the second order of *126 with the AP2 order

TSR *+126, CC+1

Again, the f field is assigned the appropriate VT address.

The <u>name</u> must be given as exactly 5 <u>printer</u> hexads, as

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

These configurations are not always conveniently punched on the flexowriter since case punches may <u>not</u> appear, the '26' hexad is given by a backspace punch, and the '25' fill hexad is given by the tab punch.

Given the ST-VT configuration

ST	VT	
A1		scalar A1
A2		codeword for vector A2
A3		primary codeword for matrix A3
A4		codeword for program A4

the control word with symbol

cr 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

cr 00000-4130-0000-00000 cr 40-02-25-25-25

will cause the vector A2 with codeword in A2's VT entry to be printed in decimal form. The control word with symbol

cr 00000-5440-0000-00000 cr 40-03-25-25-25

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

cr 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

cr 00000-4100-0000-00000 cr 40-04-25-25-25

will cause the program A4 with codeword in A4's VT entry to be printed out in octal. The control word with symbol

cr 00001-4030-0000-00000 cr 40-01-25-25-25

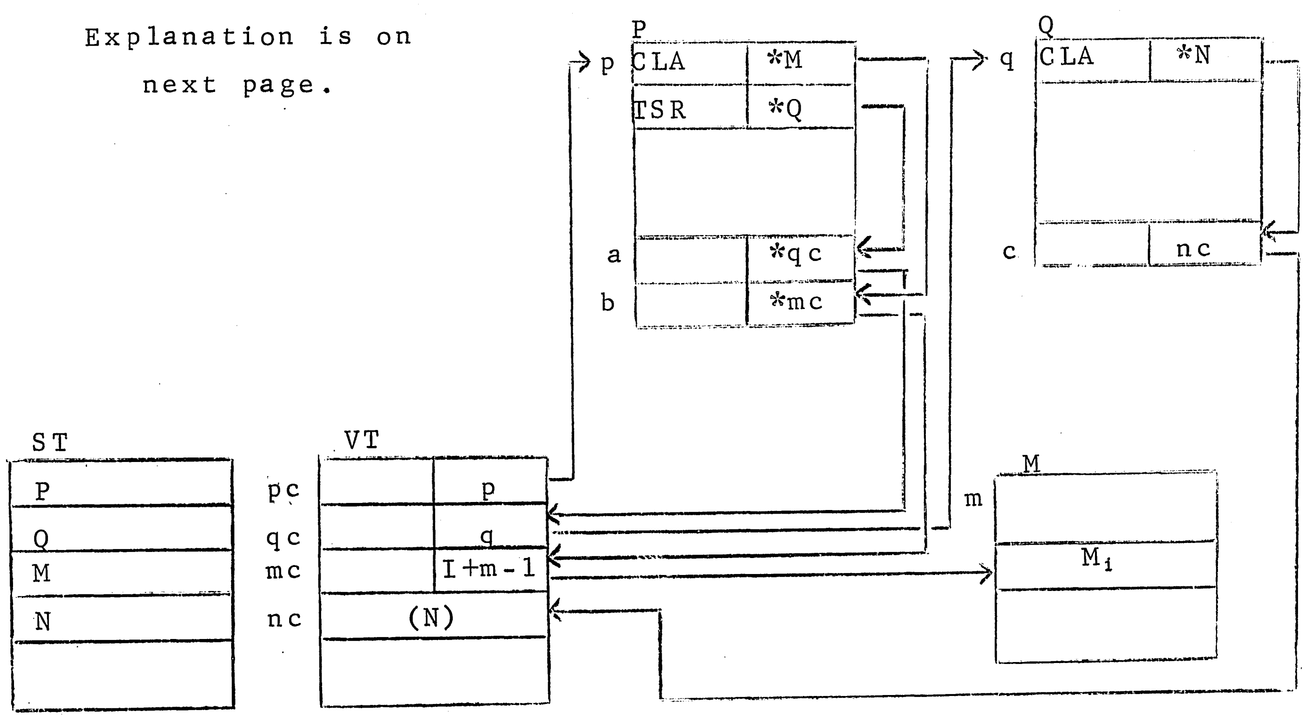
will cause the scalar A1, stored in A1's VT entry, to be printed out in decimal.

SYMBOLIC
CROSS REFERENCES

An absolute Genie program, one that has been generated by the compiler, contains one <u>reference word</u> for each external variable referred to in the program. An order which addresses an external variable does so <u>through</u> the reference word with indirect addressing. At execution time the reference word for a scalar contains the value table (VT) address where the scalar is stored; for a non-scalar it contains an indirect addressing (*) bit and the VT address where the codeword is stored. For any Genie program the output tape is in two sections, the program itself in hexads with no checksum which will be loaded symbolically through SPIREL, and a control word followed by a list which will <u>load symbolic cross references</u> into the program. This operation supplies proper VT addresses in the reference words of the program.

The figure below illustrates symbolic interconnections between two named programs and the named data to which they refer.

Explanation is on
next page.



P and Q are programs, M is a vector, and N is an external scalar. P refers to Q and M through the reference words a and b respectively. Q refers to N through the reference word c. The VT addresses for Q and M are shown as qc and mc respectively, and these are inserted into a and b by loading symbolic cross references into program P. The VT address for N is shown as nc, and this address is inserted into c by loading symbolic cross references into program Q. The paths of addressing from orders of P and Q to the data addressed are shown by arrows in the figure.

Programs written in API language and loaded with numeric codeword addresses rather than names may, with some effort, refer to external quantities whose names are in ST with values or codeword in VT. When writing such a program, a block of reference words should be created within the program. For a scalar named SS the reference word should be written

```
SS                BCD          SS sp sp sp 0 0 0 0
|cr    |1st tab   |2nd tab     |3rd tab
```

For a program named PP the reference word should be written

```
PP                BCD          PP sp sp sp A 0 0 0
```

For a vector named VV the reference word should be written

```
VV                BCD          VV sp sp sp A 0 0 0
```

For a matrix named MM the reference word should be written

```
MM                BCD          MM sp sp sp A 0 0 0
```

The 'A' in the above BCD instructions provides the * bit required in reference words for non-scalars. Within the code the data is always addressed through the reference words with indirect addressing, as

```
                  FAD          *SS
                  TSR'         *PP
                  CLA          *VV
                  STO          *MM
```

Once such an AP1 program is in the machine, proper VT addresses need to be inserted into the address fields of these reference words. Program *173, SXREF, provides a means of filling a block of reference words in the form described above. One "control word" is punched on paper tape for each block of reference words to be operated on by SXREF. The form of this "control word" is

cr nnnnn 0000 rrrr fffff

or

cr nnnnn 0000 rrrr 00000 cr sssss

where nnnnn gives in octal the <u>length</u> of the block of reference words, rrrr gives in octal the relative address within the program of the <u>first word of the block</u>, fffff gives the <u>codeword address</u> of the program if it has been loaded numerically, and sssss gives <u>the 5-hexad name</u> of the program if it has been loaded symbolically. When executed, SXREF will read these "control words" and perform the designated cross referencing until a null word is detected or the end of the paper tape is encountered.

CONTEXT OUTPUT

Once a Genie absolute program is read into the machine and its symbolic cross references have been loaded, the program is in a form that is dependent upon the exact contents and order of ST and VT. It may be desirable to punch with name a single program or a system. To reload such tapes, the ST-VT must first exist in the machine precisely as they did at the time the punching took place.

Program *174, CNTXT, provides for punching of a tape which re-establishes <u>context</u>: the value of 117 (current length of ST and VT), correction of *113 (ST) to its current length, clearing of *122 (VT) to its current length. This tape must then be loaded before any items whose names appear on ST as punched. If sense light 13 is off CNTXT proceeds to punch in hexad with checksum all quantities with names in ST for later symbolic loading.

NUMBER TO NAME CONVERSION

It may be that programs or data which is punched to be loaded at specific addresses or with numbered codeword addresses need to be converted to symbolic loading form for use in a Genie-coded system.

Program *172, SMBLZ, will punch out with the name specified constants loaded into numbered addresses or blocks and arrays loaded with numbered codeword addresses. SMBLZ reads from paper tape the following information about each item to be punched:

cr sssss tab x tab nnn

where sssss is the 5-hexad name which is to be given to the item, x is the digit 0 if the item is a scalar, x is the digit 1 if the item is a program or vector or matrix, and nnn is the three digit address or codeword address where the item is located in memory at the time this punching takes place. If the item is a matrix, all of the array will be punched.

SMBLZ will punch all items described on one tape, exiting only when end of tape is detected. If sense light 13 is on when SMBLZ is executed, tape feed will be supplied between the items punched.

GENIE SPIREL

Genie SPIREL is located on the MT System magnetic tape at block 101.03. This is a full SPIREL and the set of programs which provide support for compiled programs at execution time. The specific contents are listed below.

NAME	CODEWORD ADDRESS	DESCRIPTION
full SPIREL		
	* * * utility programs * * *	
SMBLZ	172	see NUMBER TO NAME CONVERSION
SXREF	173	see SYMBOLIC CROSS REFERENCES
CNTXT	174	see CONTEXT OUTPUT
*** programs whose names may be used in Genie language ***		
SIN	200	
COS	201	
SQR	202	
EXP	203	floating point scalar function
LOG	204	of floating point scalar
ATAN	205	
TAN	206	
COT	207	
LENGTH	210	integer length of vector
ROW	210	integer number of rows in matrix
COL	211	integer number of columns in matrix
VSPACE	213	dynamic creation of vector
MSPACE	214	dynamic creation of matrix
FIX	217	integer nearest floating point input
INV	224	inverse of matrix
TRAN	225	transpose of matrix
EVEN	227	test integer for being even
CONTROL	230	application of SPIREL to named quantity
*** programs which may be used by Genie-generated programs ***		
	212	used for DATA, PRINT, PUNCH, READ command
	215	integer to an integer power

NAME	CODEWORD ADDRESS	DESCRIPTION
	216	floating point number to a floating point power; uses 203,204
	220	copy of vector or matrix
	221	addition of two vectors or two matrices
	222	subtraction of two vectors or two matrices; uses 221
	223	multiplication of vectors or matrices
	226	multiplication of floating point scalar and vector or matrix
	231	floating of an integer vector or matrix

Available for use by the coder are addresses 241-277. The system occupies about 6,000 (octal) words of storage and may be cut down by extracting just those programs necessary to a particular system.

Parameters and restrictions for the named programs are discussed in the section on functions. The remainder of the Genie SPIREL programs are discussed below.

212 operation specified by (B1) on entry:

(B1) = 1, DATA
(B1) = 2, PRINT
(B1) = 3, PUNCH
(B1) = 4, READ

parameters are listed one per word following TRA to the program; word contains name in BCD and addressing information; list terminated by a null word; return to location following null word.

215 $(U)^{(R)} \rightarrow$ U and T7

216 $(U)^{(R)} \rightarrow$ U and T7

220 (B1) = codeword address of copy; (B2) = codeword address of input; (B1) set to 240 before copy if null on entry.

221-223 (B1) = codeword address of first operand; (B2) = codeword address of second operand; (B1) set to 240 before operation if null on entry; codeword for non-scalar result at 240; scalar result in U and T7; storage for first operand freed after operation.

226 (B1) = codeword address of non-scalar operand; (U) = scalar operand; (B1) set to 240 before multiplication if null on entry; codeword for result at 240; storage for non-scalar operand freed after multiplication.

231 (B1) = codeword address of non-scalar operand and result.

RUNNING GENIE
PROGRAMS

The procedure for testing Genie programs should follow an outline similar to the following:

1) load Genie SPIREL from magnetic tape
2) read private programs under SPIREL control
3) activate STEX with control word 00000-3120-0000-00135
4) read data items which are prefixed with SPIREL control words
5) position "run tape" which contains the control word cr 00000-3100-0000-00000 cr PPPPP

where PPPPP is the 5-hexad name of the program to be executed, followed by any data to be read by the program. A "fetch" from location 21 or a CONTINUE to 20 will then cause PPPPP to be executed by SPIREL.

The first version of a Genie program should contain ample PRINT commands that provide display of intermediate results. These may be edited out of the program for production or their execution may be conditional upon 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.

If a progrm stops unexpectedly while it is being checked out, the following information may be of value:

A) dynamic dump of fast registers, obtained by:
   1) type out contents of CC on console typewriter
   2) type 20000 into CC on console keyboard
   3) raise, then depress F0 switch on console
   4) at halt, type saved contents of CC into U on console keyboard
   5) push "CONTINUE" switch on console
   6) output appears on printer, and CC indicates where program stopped, P2 indicates where last transfer occurred, and PF shows where last transfer to subroutine occurred.

B) SPIREL dump of ST-VT, showing values of external scalars and codewords for external non-scalars defined at the time, obtained by:

1) type 20 into CC on console keyboard
2) type SPIREL control word 00000-0500-0000-00000 into U on console keyboard
3) raise, then depress F0 switch on console
4) output appears on printer, and machine stops ready to accept next control word in U.

C) SPIREL dump of any programs in which values of internal variables may be of interest, any external arrays which may be of interest. Note that the codeword address in VT for each item loaded by name appears on the load record for the run. It is easier to use this address, rather than the name, for identification of the item to SPIREL from the console.

Tracing of Genie programs is not advised. If it is done, care must be taken not to trace transfers to programs 136 (SAVE), 137(UNSAVE), 212 (INPUT-OUTPUT).

The program SUBR takes two vectors, V1 and V2, and a scalar, SCLR, as input parameters and returns two more vectors, SCNT and VPRIME, as output.

If V1 and V2 are of the same length, their dot product DPROD is computed and V1 is multiplied by SCLR. If their lengths are different, an indicator is turned on for later testing.

Next, space is taken for the vector SCNT and its elements are evaluated as: $SCNT_j = 0$ if $V1_j$ is within 0.001 of a multiple of $\pi/2$, otherwise $SCNT_j = \sec(V1_j)$.

After SCNT is evaluated, the indicator is tested. If it is off, space is taken for the vector VPRIME and it is evaluated as a function of V2 and SCNT; if the indicator is on, the calculations on VPRIME are skipped.

Finally, the indicator is turned off and the values of SCLR, DPROD, V1, SCNT, and VPRIME are printed.

Notes on Symbolic Listing:

Line	Remark
3,4	All non-scalars, all functions not in the vocabulary of the compiler, and all external scalars must be declared before the SEQ.
5	The one-line definition of function REM is also located before the SEQ: the user must supply a function INT to compute the largest integer contained in a number. External specifications apply to the function REM as well as to the main program.
10	LNG1, LNG2, and j are declared as integers. Since this statement appears after the SEQ, the integers are internal to the program SUBR.
11	HALFPI is defined as 1.570796; this value is used in the code wherever the name appears. Since 'HALFPI' is more than five characters long, it will appear on listings as 'HALFP' and will not be distinguished from any other character beginning 'HALFP'.
12	Several equations separated by commas may appear on one line.

<u>LINE</u> <u>REMARK</u>

13 Since the value of CC is to be unchanged if the condition is not satified, the alternative value is omitted. Note that the Genie lister prints | for ╪ and ← for #.

14,15 Vector V1 is multiplied by vector V2 for a scalar result and each element of V1 is multiplied by SCLR. The use of X to indicate multiplication on line 13 is synonymous with the juxtaposition of the factors on line 14.

20,22,23 Execution of a function may be called for explicitly with an EXECUTE command or implicitly in an arithmetic command, depending on the function.

21,24,25,30,31 These commands control a loop indexed on j. A test is made at the beginning of each pass through the loop to determine which of two calculations is to be performed for the current value of j. At the end of each calculation, j is incremented and control is transferred to the initial test if j ≤ LNG1 or to the first instruction after the loof if j > LNG1.

32 A sense light is tested in Genie language by writing the number of the sense light to be tested as an exponent of SL.

35,40 This is a simpler method of loop control; it is useful for loops with positive increments and a single exit point.

36,37 A statement may extend for more than one line. The case punch for the second line <u>follows</u> the third tab in the 'ct tab tab tab' sequence.

17,27,41 AP2 instructions may be interspersed with GENIE statements; no special indication is necessary. AP2 commands that use SKP, JMP, or otherwise depend on CC should be used with caution. It is difficult to predict the number of machine language instructions which a GENIE command will generate.

43 'END' terminates the command sequence by generating code for return of control to the program at the next higher level.

45 'LEAVE' causes exit from Genie at compilation time.

```
          DEFINE
          VECTORS V1,V2,SCNT,VPRIME
          FUNCTIONS REMND,INT
          REMND(A,B) = A/B - INT(A/B)
SUBR(SCLR,V1,V2,SCNT,VPRIME), =SEQ
          REM    THIS IS A SAMPLE PROGRAM
          INTEGERS LNG1,LNG2,,J
          LET HALFPI=1.570796
          LNG1=LENGTH(V1),LNG2=LENGTH(V2)
          CC=+ARND  ,I,F LNG1 | LNG2 ,,O,R LNG1 ≤ 0
          DPROD=V1 × V2
          V1 := SCLR V1
          CC=+EVN
ARND           SLN             +40000
EVN       EXECUTE VSPACE(SCNT,LNG1)
          ,J=1
COMPA     CC = +LOW  ,I,F REMND(V1 ,J,HALFPI) < 0.001
          SCNT ,J = 1.0/COS(V1 ,,J)
          ,J = ,,J+1
          CC = +COMPA  ,I,F ,,J≤ LNG1, +COMPB
LOW       SCNT ,J = 0.0
               SLN             +40000
          ,J = ,J+1
          CC = +COMPA  ,I,F ,,J ≤ LNG1
COMPB     CC=+OMIT  ,I,F SL'
          REM    SECOND SECTION
          EXECUTE VSPACE(VPRIME,LNG2)
          FOR ,J = 1,1,LNG2
          VPRIME ,J = V2 ,,J  ,I,F SCNT ,,J * V2 ,J, 0 ,,I,F
                                      |V2 ,J| < 0.05, V1 ,J
          REPEAT
OMIT           SLF             +40000
          PRINT SCLR,DPROD,V1,SCNT, VPRIME
END
          DEFINE
LEAVE
```

```
REMND   START NEW PROGRAM,
←BGIN   PROGRAM SEQUENCE,
END     PROGRAM SEQUENCE,
```

```
REMND ,=  0   ←BGIN    1    10 01000 02 4400 00136
          1            2    01 40007 00 4100 77763
          2            3    47 21641 00 0001 00010   END   +   1
          3            4    01 21700 06 0600 00001   B
          4            5    01 21700 05 0600 00000   A
          5            6    05 12700 00 0000 00006   T6
          6            7    01 40000 07 4401 00006   INT
          7           10    01 40007 04 4401 00003   END   +   1
         10           11    05 12700 00 0000 00006   T6
         11           12    01 10400 07 1000 00004   T4
          0   END     13    01 01000 00 4400 00137
          1           14    01 40006 00 4000 00000
          2           15    07 01000 00 4200 00000
```

```
   REMND    SYMBOL TABLE,
111         A       102    0   0                    0   0
112         B       102    1   0                    0   0
113         ←BGIN   100    1   3                    0   0
```

```
SUBROUTINES REFERENCED
   INT
```

```
SUBR     START NEW PROGRAM
*BGIN    PROGRAM SEQUENCE,
ARND     PROGRAM SEQUENCE,
EVN      PROGRAM SEQUENCE,
COMPA    PROGRAM SEQUENCE,
LOW      PROGRAM SEQUENCE,
COMPB    PROGRAM SEQUENCE,
*FOR1    PROGRAM SEQUENCE,
*RPT1    PROGRAM SEQUENCE,
OMIT     PROGRAM SEQUENCE,
END      PROGRAM SEQUENCE,
```

```
SUBR  .=  0   *BGIN    1   10 01000 02 4400 00136
          1            2   01 40007 00 4100 77760
          2            3   47 21641 00 0001 00205    END    +    1
THIS IS A SAMPLE PROGRAM
          7            4   01 21700 07 0200 00001    V1
         10            5   01 40000 00 4400 00210    LENGT
         11            6   01 40007 00 4401 00202    END    +    1
         12            7   01 20001 00 4001 00203    LNG1
         13           10   01 21700 07 0200 00002    V2
         14           11   01 40000 00 4400 00210    LENGT
         15           12   01 40007 00 4401 00176    END    +    1
         16           13   01 20001 00 4001 00200    LNG2
         17           14   01 21700 06 0001 00176    LNG1
         20           15   06 02050 00 0001 00176    LNG2
         21           16   01 01000 00 4001 00001
         22           17   01 01000 00 4001 00002
         23           20   06 02510 00 4000 00000
         24           21   01 01000 00 4001 00002
         25           22   01 21700 00 4001 00023    ARND
         26           23   01 20040 40 0000 00000
         27           24   01 21700 41 0200 00001    V1
         30           25   01 21700 42 0200 00002    V2
         31           26   01 40000 00 4400 00223
         32           27   01 40007 00 4401 00161    END    +    1
         33           30   01 20001 00 4001 00164    DPROD
         34           31   01 21700 42 0200 00001    V1
         35           32   00 40000 41 4400 00220
         36           33   01 40007 00 4401 00155    END    +    1
         37           34   01 50400 00 0600 00000    SCLR
         40           35   02 40000 00 4400 00226
         41           36   01 40007 00 4401 00152    END    +    1
         42           37   01 21700 41 0200 00001    V1
         43           40   00 40000 42 4400 00135
         44           41   01 40007 00 4401 00147    END    +    1
         45           42   00 50401 52 0000 00240
         46           43   02 20001 00 4002 00000
         47           44   41 21641 00 0004 00000
         50           45   01 21700 40 4001 00001    EVN
          0   ARND    46   01 42000 00 4000 40000
          0   EVN     47   01 21702 26 0200 00003    SCNT
          1           50   00 20102 26 4001 00142    LNG1
          2           51   01 40000 00 4400 00213    VSPAC
          3           52   01 40007 00 4401 00136    END    +    1
          4           53   01 21700 00 4000 00001
```

```
        5                 54     01  20001 00 4001 00141    ,J
        0   COMPA         55     01  21740 41 0001 00140    ,J
        1                 56     01  21740 00 0600 00001    V1
        2                 57     01  20001 71 4001 00137    *P1
        3                 60     41  21602 26 4000 00000
        4                 61     00  20102 26 4001 00136    HALFP
        5                 62     01  40000 00 4401 00136    REMND
        6                 63     01  40007 00 4401 00125    END     +      1
        7                 64     01  06550 00 0001 00135    *NUMB
       10                 65     01  01000 00 4001 00002
       11                 66     01  21700 00 4001 00024    LOW
       12                 67     01  20040 40 0000 00000
       13                 70     01  21740 41 0001 00125    ,J
       14                 71     01  21740 00 0600 00001    V1
       15                 72     01  40000 07 4400 00201
       16                 73     01  40007 00 4401 00115    END     +      1
       17                 74     01  16700 00 0001 00126    *ONEF
       20                 75     01  20001 26 4100 00000
       21                 76     01  21700 41 0001 00117    ,J
       22                 77     01  21700 66 0100 77776
       23                100     01  20001 00 4600 00003    SCNT
       24                101     01  21740 00 4000 00001
       25                102     01  10000 00 0001 00113    ,J
       26                103     01  20001 00 4001 00112    ,J
       27                104     01  21740 00 0001 00111    ,J
       30                105     01  02510 00 0001 00105    LNG1
       31                106     01  01000 00 4001 00002
       32                107     01  21700 00 4001 77744    COMPA
       33                110     01  01000 00 4001 00001
       34                111     01  21700 00 4001 00014    COMPB
       35                112     01  20040 40 0000 00000
        0   LOW          113     01  21700 41 0001 00102    ,J
        1                114     00  20001 00 4600 00003    SCNT
        2                115     01  42000 00 4000 40000
        3                116     01  21740 00 4000 00001
        4                117     01  10000 00 0001 00076    ,J
        5                120     01  20001 00 4001 00075    ,J
        6                121     01  21740 00 0001 00074    ,J
        7                122     01  02510 00 0001 00070    LNG1
       10                123     01  01000 00 4001 00002
       11                124     01  21700 00 4001 77727    COMPA
       12                125     01  20040 40 0000 00000
        0   COMPB        126     01  21700 00 0000 77770
        1                127     01  45015 00 4000 00016
        2                130     01  50010 00 4000 77776
        3                131     01  01060 00 4001 00001
        4                132     01  01000 00 4001 00002
        5                133     01  21700 00 4001 00043    OMIT
        6                134     01  20040 40 0000 00000
SECOND  SECTION
       12                135     01  21702 26 0200 00004    VPRIM
       13                136     00  20102 26 4001 00055    LNG2
       14                137     01  40000 00 4400 00213    VSPAC
       15                140     01  40007 00 4401 00050    END     +      1
        0   *FOR1        141     01  21700 00 4000 00001
        1                142     01  20001 00 4001 00053    ,J
        2                143     01  21700 00 0001 00050    LNG2
        3                144     01  02110 00 0001 00051    ,J
        4                145     01  01000 00 4001 00031    *RPT1  +      3
        5                146     01  21700 06 0001 00047    ,J
        6                147     01  21740 41 0000 00006    T6
        7                150     01  21740 04 0600 00003    SCNT
       10                151     01  21740 41 0000 00006    T6
       11                152     01  21740 00 0600 00002    V2
       12                153     01  02510 00 0000 00004    T4
       13                154     01  01000 00 4001 00003
```

```
   14            155    01 21740 41 0000 00006      T6
   15            156    01 21740 00 0600 00002      V2
   16            157    01 01000 00 4001 00010
   17            160    01 21740 41 0000 00006      T6
   20            161    01 21740 00 0600 00002      V2
   21            162    21 06550 00 0001 00041      *NUMB
   22            163    01 01000 00 4001 00002
   23            164    01 21700 00 4000 00000
   24            165    01 01000 00 4001 00002
   25            166    01 21740 41 0000 00006      T6
   26            167    01 21740 00 0600 00001      V1
   27            170    01 20001 26 4100 00000
   30            171    06 20040 41 0000 00000
   31            172    01 21700 66 0100 77776
   32            173    01 20001 00 4600 00004      VPRIM
    0   *RPT1    174    01 21700 00 4000 00001
    1            175    01 10401 00 0001 00020      IJ
    2            176    01 01000 00 4001 77743      *FOR1  +    2
    0   OMIT     177    01 42004 00 4000 40000
    1            200    01 40001 00 4000 00002
    2            201    01 01000 00 4400 00212
    3            202    62 42536 12 5600 00000      SCLR
    4            203    43 57615 64 3001 00011      DPROD
    5            204    65 01252 52 5200 00001      V1
    6            205    62 42556 32 5200 00003      SCNT
    7            206    65 57615 05 4200 00004      VPRIM
   10            207    00 00000 00 0000 00000
    0   END      210    01 01000 00 4400 00137
    1            211    01 40006 00 4000 00000
    2            212    07 01000 00 4200 00000


 SUBR      SYMBOL  TABLE.
 112        SCLR    102     0    0                          0    0
 113        *BGIN   100     1    3                          0    0
 114        LNG1    200   213    3                          0    0
 115        LNG2    200   214    3                          0    0
 116        IJ      200   216    3                          0    0
 117        HALFF   100   220    3     1001444176576 2130    0
 120        ARND    100    46    3                          0    0
 121        DPROD   100   215    3                          0    0
 122        EVN     100    47    3                          0    0
 123        COMPA   100    55    3                          0    0
 124        LOW     100   113    3                          0    0
 125        *NUMB   100   222    3     76101422335136 1524    0
 126        *PI     100   217    3                          0    0
 127        COMPB   100   126    3                          0    0
 130        OMIT    100   177    3                          0    0
 131        *FOR1   100   141    3                          0    0
 132        *RPT1   100   174    3                          0    0
 133        *NUMB   100   224    3     77014631463146 3146    0


        SUBROUTINES REFERENCED
                        201
         REMND
         VSPAC          213
                        135
                        226
                        220
                        223
         LENGT          210
```

END OF DEFINITION SET. EXTERNAL SYMBOLS.

103	V1	121	1	0	0	0
104	V2	121	2	0	0	0
105	SCNT	121	3	0	0	0
106	VPRIM	121	4	0	0	0
107	REMND	110	221	0	0	0
110	INT	110	16	3	0	0
111	SUBR	10	5	0	0	0

```
00000              ORG
                   REM       BACK-TRANSLATION
L1        -Z       TRA       ,A*136,U+R
L2                 SPF       ,AB6+77763
          PF       RWT       L14
                   CLA       *PF+1,U+T4
                   CLA       *PF+Z,U+T5
          T5       12700     6
                   TSR       ,A*L16,U+T7
                   SPF       ,A*L14,U+T4
          T5       12700     6
                   FAD       -4,U+T7
                   TRA       ,A*137
L14                SB6       ,AZ
          T7       TRA       ,APF+Z
L16       Z        00000     *Z
                   END
```

```
00000           ORG
                REM            BACK-TRANSLATION
L1      -Z      TRA            ,A*136,U-R
L2              SPF            ,AB6+77760
        PF      RWT            L211
                CLA            PF+1,U-T7
                TSR            ,A*210
                SPF            ,A*L211
                STO            ,AL213
                CLA            PF+2,U-T7
                TSR            ,A*210
                SPF            ,A*L211
                STO            ,AL214
                CLA            L213,J-T6
        T6      IF(NZE)SKP     L214
                TRA            ,AL20
L17             TRA            ,AL22
L20     T6      IF(NEG)SKP     ,AZ
                TRA            ,AL24
L22             CLA            ,AL46
                NOP            Z,U-CC
L24             CLA            PF+1,U-B1
                CLA            PF+2,U-B2
                TSR            ,A*223
                SPF            ,A*L211
                STO            ,AL215
                CLA            PF+1,U-B2
        Z       TSR            ,A*220,U-B1
                SPF            ,A*L211
                LDR            *PF+Z
        R       TSR            ,A*226
                SPF            ,A*L211
                CLA            PF+1,U-B1
        Z       TSR            ,A*135,U-B2
                SPF            ,A*L211
        Z       LDR-           240,R-B2
        R       STO            ,AB1+Z
        B1      RWT            B2+Z
                CLA            ,AL47,U-CC
L46             SUN            ,A40000
L47             21702          PF+3,B6+1
        Z       20102          ,AL213,B6+1
                TSR            ,A*213
                SPF            ,A*L211
                CLA            ,A1
                STO            ,AL216
L55             21740          L216,U-B1
                21740          *PF+1
                STO            ,AL217,I-B1
        B1      21602          ,AZ,B6+1
        Z       20102          ,AL220,B6+1
                TSR            ,A*L221
                SPF            ,A*L211
                IF(NNZ)SKP     L222
```

```
            TRA           ,AL70
L66         CLA           ,AL113
            NOP           Z,U-CC
L70         21740         L216,U-B1
            21740         *PF+1
            TSR           ,A*201,U-T7
            SPF           ,A*L211
            VDF           L223
            STO           ,AB6+Z,B6+1
            CLA           L216,U-B1
            CLA           B6+77776,B6-1
            STO           ,A*PF+3
            21740         ,A1
            ADD           L216
            STO           ,AL216
            21740         L216
            IF(NEG)SKP    L213
            TRA           ,AL111
L107        CLA           ,AL55
            TRA           ,AL112
L111        CLA           ,AL126
L112        NOP           Z,U-CC
L113        CLA           L216,U-B1
        Z   STO           ,A*PF+3
            SUN           ,A40000
            21740         ,A1
            ADD           L216
            STO           ,AL216
            21740         L216
            IF(NEG)SKP    L213
            TRA           ,AL126
L124        CLA           ,AL55
            NOP           Z,U-CC
L126        CLA           77770
            LRS           ,A16
            ORU           ,A77776
            IF(ODD)TRA    ,AL133
            TRA           ,AL135
L133        CLA           ,AL177
            NOP           Z,U-CC
L135        21702         PF+4,B6+1
        Z   20102         ,AL214,B6+1
            TSR           ,A*213
            SPF           ,A*L211
            CLA           ,A1
            STO           ,AL216
L143        CLA           L214
            IF(POS)SKP    L216
            TRA           ,AL177
L146        CLA           L216,U-T6
            21740         6,U-B1
            21740         *PF+3,U-T4
            21740         6,U-B1
            21740         *PF+2
            IF(NEG)SKP    4
```

```
                TRA         ,AL160
L155            21740       6,U+B1
                21740       *PF+2
                TRA         ,AL170
L160            21740       6,U+B1
                21740       *PF+2
        IIU1    IF(NNZ)SKP  L224
                TRA         ,AL166
L164            CLA         ,AZ
                TRA         ,AL170
L166            21740       6,U+B1
                21740       *PF+1
L170            STO         ,AB6+Z,B6+1
        T6      NOP         Z,U+B1
                CLA         36+77776,B6-1
                STO         ,A*PF+4
                CLA         ,A1
                FAD+        L216
                TRA         ,AL143
L177            SLF         ,A40000Q
                SB1         ,A2
                TRA         ,A*212
L202    1B21    42536       -,A*PF+Z,R+R
        B3      57615       -(L215),B4-1
        1B51    01252       -,APF+1,R+B2
        1B21    42556       -,APF+3,B2+X
        1B51    57615       ,APF+4,U+T5
        Z       00000       Z
L210            TRA         ,A*137
L211            SB6         ,AZ
        T7      TRA         ,APF+Z
L213            OCT         000000000000000000
L214            OCT         000000000000000000
L215            OCT         000000000000000000
L216            OCT         000000000000000000
L217            OCT         000000000000000000
L220            OCT         010014441765762130
L221    Z       00000       *Z
L222    -1B61   10142       -(*B1+B3+B6+L61747),B3+1
L223            IF(PSN)HTR  Z
L224    -1PF1   01463       (,APF+B2+B3+B6+631461),R+T4
                END
```

The program NEWTN(COEF,GUESS) uses a variant of Newton's method to obtain the roots of a polynomial

$$P(X) = X^n + A_{n-1}X^{n-1} + --- + A_1X + A_0$$

INPUT: Vector COEF, of length n, the coefficients $(A_{n-1}, A_{n-2}, ---, A_0)$

Vector GUESS, length n, containing the approximate roots of P.

OUTPUT: COEF: unchanged

GUESS: contains the refined values of the roots

Vector POFR, length n, which contains the value of P at the next to last iteration for each root.

METHOD: Let X_K denote the value obtained for a certain root at the K-th iteration. Then

X_1 = (value obtained from GUESS)

$X_2 = 1.001X_1$

$$X_{K+1} = X_K - P(X_K) \frac{X_K - X_{K-1}}{P(X_K) - P(X_{K-1})} \quad \text{if } K>2$$

At most twenty iterations are performed.

```
            DEFINE
            VECTORS COEF,GUESS,POFR
NEWTN(COEF,GUESS), = SEQ
            INTEGERS J,K,L,M
            L = ROW(COEF)
            EXECUTE VSPACE(POFR,L)
            FOR J = 1,1,L
            GA = GUESS_J
            FOR K = 1,1,20
            FN = 1.0
            FOR M = 1,1,L
            FN = COEF_M+FN×GA
            REPEAT
            CC = ←INIT ., IF 1<K
            FO = FN, GO = GA
            GA = 1.001GA
            CC = ←LOOP
INIT        GS = GA, DELF = FN-FO
            CC = ←QUIT ., IF DELF = 0
            GA = GA-FN(GA-GO)/DELF
            GO = GS, FO = FN
LOOP        REPEAT
QUIT        GUESS_J = GA
            POFR_J = FN
            REPEAT
END
            DEFINE
LEAVE
```

EPILOGUE

The Genie compiler is the invention of John K. Iliffe, now with Ferranti, Ltd. in London. Major contributions to its realization have been made by Jane G. Jodeit, T.A. Kitchens, Jr., and Jo Kathryn Mann.

Programming development has been supported by the National Science Foundation under grant NSF G-17934. Construction of the Rice University Computer was supported by the Atomic Energy Commission under contract AT-(40-1)-1825, further development under contract AT-(40-1)-2572.

Genie may well be improved and extended by future efforts in a number of areas:

*(i) Notation for sense light iterrogation would be very useful.

*(ii) The case of a program with no command labels should be handled properly.

(iii) Function names should be allowed as parameters.

*(iv) The machinery for Boolean variables exists but needs to be checked out and made available.

*(v) At compilation time a list of programs referred to in the compiled code should be provided.

(vi) Compound conditional commands should be permitted.

(vii) Checks on overflow of size limits and various other compiler diagnostics should be provided and documented.

(viii) A major effort would be required to allow programs to use themselves, but this might be interesting and worthwhile.

(ix) More elaborate input-output facilities would be useful.

Jane G. Jodeit
April, 1963

Rice University
Houston, Texas

*provided by October, 1963

Since April, 1963 Genie has been subjected to considerable use at Rice, and the system has been improved in various areas:

(i) Boolean arithmetic is available.

(ii) Notation for sense light interrogation is available.

(iii) A program with no command labels is compiled properly.

(iv) Programs referred to in compiled code are listed on compilation output.

(v) The machine index registers are addressable in Genie language.

(vi) Elements of general arrays of more than two dimensions may be referred to in Genie language.

(vii) Genie generated code for loops and for matrix operations is more efficient.

(viii) Numbers may be specifically placed within programs.

(ix) Matrix operations are extended to matrices of integers where meaningful.

(x) The program name may appear at the left margin for ease of identification, and the need for many 'tabulate' punches has been eliminated.

(xi) Iteration parameters for loops may decrease or increase from initial to final value.

(xii) Provision has been made for explanatory remarks within programs.

(xiii) Simple uniform notation has been introduced for designation of the result of a function to be implicitly executed.

(xiv) Genie PLACER has been extended to include the Genie compiler itself and the newly developed translator from machine code to assembly language. Magnetic tape handling provides the system access to two full 8K memory loads.

(xv) These NOTES on Genie have been improved and augumented. A separate document on the assembly language is available, and one on SPIREL is forthcoming.

Mary M. Shaw

October, 1963

Rice University
Houston, Texas

APPENDIX

GENIE CODING CONVENTIONS

This appendix 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.

Program initialization and termination

The 'SEQ' 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' there is an 'END', so there is an "END code sequence" corresponding to each ←BGIN code sequence. The form of these code sequences depends on the number of parameters (k) listed for the program and, in some cases, the type of parameters. 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 difinition, 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 in T7 is stored at internal location '←T7ST'; a non-scalar parameter is stored on the B6-list. All fast registers are saved; if there are parameters on the B6-list (k>1 or K=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:

```
k=1          ←BGIN     PF     RPA,WTG    END
fast                            ⋮
             END              TRA        Z

k=1          ←BGIN     -Z     TRA        *+136, U→R
scalar                 T7     STO        ←T7ST
                                ⋮
             END              TRA        *+137
                       T7     TRA        PF

k=1
non-scalar   ←BGIN     T7     STO        B6, B6+1
                       -Z     TRA        *+136, U→R
                              SPF        B6-10
                       PF     RPA,WTG    END+1
                                ⋮
             END              TRA        *+137
                              SB6        Z
                       T7     TRA        PF

k>1          ←BGIN     -Z     TRA        *+136, U→R
                              SPF        B6-k-9
                       PF     RPA,WTG    END+1
                                ⋮
             END              TRA        *+137
                              SB6        Z
                       T7     TRA        PF
```

RESULT for implicit execution

A program which is single valued may be executed implicitly; that is, it may be mentioned within the formula on the right hand side of an equation in Genie language. A scalar result must be in U upon exit from the program, a non-scalar result in the non-scalar accumulator whose codeword is by definition at location +240 during execution. The name 'RESULT' is interpreted by the compiler as T7 for a scalar and as codeword address +240 for a non-scalar. 'RESULT' may appear only on the left hand 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 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, the values being stored within the program. External variables may be scalar or non-scalar, the address or * codeword address respectively being stored in a 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, the k^{th} parameter being addressed at (PF)+k-1 after execution of the '←BGIN' code sequence. 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 chart summarizes addressing conventions for variables.

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

B6-list, working storage

The SPIREL system reserves machine locations from 17600_8 upward 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. In addition, the B6-list is 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. 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      A
        STO      B6, B6+1
        .
        .
        CLA      B
        STO      B6, B6+1
        .
        .
        CLA      C
        STO      B6, 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
        .
        .
```

Parameter set-up for program execution

Execution of a program with a single 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 k^{th} parameter is a scalar SP, then SP→B6, B6+1; if the k^{th} parameter is a non-scalar NSP, then *NSP→B6, B6+1. 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. are generated by the compiler. The quantity is stored at ←Pn, and then ←Pn for a scalar or *←Pn for a non-scalar is stored on the B6-list. The execution of program PROG is accomplished by TSR *PROG where PROG is a location within the program doing the execution which contain * codeword address for PROG; the codeword for PROG is in the Value Table (*+122). Thus, PROG is an external variable with respect to the program which executes it.

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. In face, 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 activated
2) indexed by B1
3) initial index = 1

standard form of two dimensional array, matrix

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

Arithmetic operations involving standard non-scalars parallels scalar arithmetic quite closely. By convention, codeword +240 is used as a non-scalar accumulator, commonly called 'U*'. 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 U*. The non-scalar result of such an operation is placed in U*. The creation of a new U* causes the storage previously addressed by that "name" to be freed. If a non-scalar in U* needs to be temporarily saved, this is done on the B6-list; that is, a word on the B6-list is taken as the codeword for the storage addressed as U*, and the U* codeword is cleared. Note that this storage also involves adjustment of the STEX back-reference to address the new codeword.

The code sequence generated by the compiler for matrix storage A → B is as follows:

```
          CLA       A,U→B2          ┐
     Z    TSR       *+220,U→B1      │copy A→U* only if AǂU*
ǂ         SPF       *END+1          ┘
          CLA       B,U→B1          ┐
     Z    TSR       *+135,U→B2      │free storage addressed as
ǂ         SPF       *END+1          │B only if BǂU* and not on
                                    ┘B6-list
     Z    LDR→      +240,R→B2       ┐clear U* codeword        ┐
     R    STO       B1              │store new codeword       │if BǂU*
                                    │for B                    │
     B1   RPA,WTG   B2              ┘update back-reference    ┘
```

ǂ(PF) reset after destruction by TSR only if program using (PF) for reference to parameters.

<u>Assignment of type to variables</u>

In the Genie language each scalar, vector, matrix, and function (result) has a type: integer, floating point, or Boolean. The type of a variable may be explicitly specified in a declaration: INTEGER for integer, SCALAR for floating point, and BOOLEAN for Boolean. 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 hand side of an equation, the variable is assigned floating point type.
2) If a variable name first appears on the left hand side of an equation, the variable is assigned the 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 left and right hand 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 right hand side, but the type of the quantity on the left hand side will be unaffected.

Arithmetic combination of variables of different types

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 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 numerically 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. Otherwise, the floating of an integer scalar A is accomplished by the following generated code:

```
+53100    -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 *+231.

Boolean variables and operations

A Boolean variable may take on the value "TRUE' or 'FALSE', these being represented in the computer by full length quantities

```
TRUE  = +0077777777777777
FALSE = +0077777777777776
```

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
AND    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
        SYS     B
        AND     #+77777
```

The only meaningful unary operation on a Boolean variable is complementation, not A, true if A is false

```
        CLA     A
   -U   ORU     #+77776
```

The machine registers sense lights (SL) and indicator lights (IL) are each 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 variables SL and IL are 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
        LUR     *R       ⎪ or
        ORU     #+77776  ⎭ an expression
```

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 fifteen bits (the rightmost in a machine word) A^1, A^2, ..., A^{15} each with a value of TRUE or FALSE.

APPENDIX
11

Loop coding

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 K^{th} 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
             skip
             storage for increment
             compute increment                                    A
             store increment
             skip
             storage for final
             compute final                                        A
             store final
[←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 operations on vectors and matrices.

B1 is used in input-output commands to specify to program *+212 the operation to be performed.

B6 always addresses the push-down B6-list which is used for temporary storage of scalars and non-scalars and for multiple parameter storage.

PF is used within a program to address its own parameters if there are more than one or if there is only one but that is a non-scalar. The appropriate value of (PF) is, in such cases, stored in the address portion of END+1 so that resetting is easily accomplished by

SPF   *END+1

P2 is used in transfers (TRA, and not TSR) to

*+212, the input-output program

*+136, SAVE used in the ←BGIN code sequence

*+137, UNSAVE used in the END code sequence

Therefore, these orders must not be traced.

Rearrangemnet 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, therebe giving each a name. Then the original formula will involve only simple variable names, and rearrangement will not take place.