GENIE

GENIE

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

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

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

The unit of definition to the Genie compiler is the definition set, which has the form

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

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

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

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

1) a single lower case Roman letter;

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

By rule 1) the following are examples of names:

a i p x

By rule 2) the following are examples of names:

A CAT Fn DDxy I2 PQ29 Dog3

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

ab A B38 Pt4p M5ef w10

They are interpreted respectively as:

a×b A×B38 Pt4×p M5×e×f w×10

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

The names

m Man

are stored as

.M M.AN

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

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

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

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

In alphabetical order, vocabulary names are:

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

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

B6 I PF R S U X

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

and if or not

A string of decimal numerals

$$DDD < 2^{14}$$

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

$$A.B*C$$

which is interpreted as

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

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

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

$$+\phi\phi\phi$$

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

The following numbers will be understood as shown:

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

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

3_641_209.4_*_-8

is exactly equivalent to

3641209.4*-8

and

+00200_0130_0004_00257

is exactly equivalent to

+002000130000400257

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

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

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

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

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

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

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

A more general form is illustrated by:

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

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

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

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

A <u>scalar</u> is described by a <u>type</u> declaration:

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

A <u>non-scalar</u> is described by a <u>shape</u> declaration:

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

<u>and</u> a type declaration which applies to its elements.

A <u>function</u> is described the <u>mode</u> declaration:

FUNCTION or FUNCTIONS	for a private program name

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

Not all variables need be described by declarations. When

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

type	real floating point
shape	scalar
mode	non-function

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

$$\text{INFER} \left\{\begin{matrix}\text{INTEGER}\\ \text{REAL}\\ \text{SCALAR}\\ \text{COMPLEX}\\ \text{BOOLEAN}\end{matrix}\right\} \left\{\begin{matrix}\text{VECTOR}\\ \text{MATRIX}\\ \text{ARRAY}\end{matrix}\right\}$$

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

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

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

Thus,

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

is a legal statement, but

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

is not.

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

```
y=a+F(P)+b
```

or for explicit execution as 'G' in the command

```
EXECUTE G(Q)
```

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

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

```
RESULT=scalar or non-scalar arithmetic expression
| cr      | 1st tab
```

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

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

```
f(x,y)=3ax+a^2y,  a=2+x
| cr
```

The function f may then be executed implicitly within the command

sequence of a program,

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

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

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

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

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

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

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

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

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

...F1(F2)...

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

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

if a single parameter is a function.

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

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

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

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

An integer value may be specified, as

```
LET K=3
```

The variable K takes on integer type.

A complex value may be specified, as

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

or

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

The variables CVAL and POLE take on complex type.

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

```
LET t=TRUE
```

or

```
LET No=FALSE
```

The variables t and No take on Boolean type.

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

```
LET MASK=+777777077
```

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

A fixed address or codeword address may be specified, as

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

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

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

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

```
LET ALPHA = BETA
```

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

```
LET COUNT = B5
```

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

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

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

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

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

```
REM COMPUTE_FIRST_VALUE
```

or

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

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

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

All statements of a program from the '.=' to and including the 'END', except 'LET's, remarks, and declarations, cause code to be generated. Such statements are called 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 four categories; arithmetic, control, input-output, or data. These will be discussed in separate sections.

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

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

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

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

unary inflections: -, |...|, and 'not'
subscription
exponentiation
× and /
+ and binary -
relations: =, ≠, <, ≮, ≤, ≯

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

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

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

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

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

-/ between integer or real floating point scalar operands.

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

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

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

+ FALSE if both operands FALSE, otherwise TRUE
- TRUE if operands differ, FALSE if operands the same
× TRUE if both operands TRUE, otherwise FALSE
/ TRUE if operands the same, FALSE if operands differ

The octal representations for the Boolean values are

TRUE 007777777777777777
FALSE 007777777777777776

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

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

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

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

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

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

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

with a complex operand, and the result is complex.

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

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

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

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

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

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

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

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

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

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

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

13) Relations $=$, $\neq$, $<$, $\not<$, $\leq$, $\not\leq$ between integer or real floating point scalar operands.

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

is required:

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

Note that the relations > and ≥ are not available, but > is equivalent to ≯ and ≥ is equivalent to ≮.

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

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

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

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

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

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

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

The form of a simple arithmetic command is illustrated by:

```
        A=arithmetic expression
| cr      | 1st tab
```

The form of a compound arithmetic command is illustrated by:

```
        A=arithmetic expression, B=arithmetic expression, ...
| cr      | 1st tab
```

where more than one equation appears in the command.

If there are no interdependencies among the equations of a command, the equations are coded by Genie in the order given. If there are interdependencies, the first equation will be coded last and preference will be given to coding the remaining equations from right to left; for the second and any following equations, if the 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.

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

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

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

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

CONDITIONAL ARITHMETIC COMMANDS

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

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

| cr | 1st tab

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

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

$$K = 1.0 \text{ if } B \leq C, 2.0 \text{ if } x < -12.9, 3.0$$

$$K = 1.0 \text{ if not } (SL^n), 3.0$$

$$K = 1.0 \text{ if } SL^5 + \text{not } (SL^n)$$

| cr | 1st tab

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

$$K = 1.0 \text{ if } (B \leq C \text{ or } |C + D| \neq 3.72) \text{ and } SL^5 + \text{not } (SL^n),$$
$$2.0 \text{ if } x < -12.9, 3.0$$

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

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

arithmetic operations

'and'

'or'

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

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

F_1 r F_2 and F_2 r' F_3

or $(F_1 \text{ r } F_2) \times (F_2 \text{ r' } F_3)$

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

light in the indicator register off.

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

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

```
        CC = #LOOP or GO TO LOOP
| cr    | 1st tab
```

Here 'CC' is the mnemonic for the control counter which is normally stepped sequentially through the orders of the code. 'LOOP' is a label on a command of the program, the command to which control will be passed by this transfer command. Note that 'END' is a label in every program and may be transferred to for exit from the program. The inflection '#' is required in this context to indicate that the address corresponding to LOOP, and not the contents of the location whose address is LOOP, is to be calculated on the righthand side. The '#' inflection is analagous to the 'a' bit in AP1.

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

```
        CC = #A1 if P1 , #A2 if P2 , ... , #An if Pn , #An+1 or
| cr    | 1st tab    GO TO A1 if ...etc.
```

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

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

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

The elements of the FOR command are

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

The command

EXECUTE PROG(PARAM)

| cr | 1st tab

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

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

| cr | 1st tab

And a compound unconditional EXECUTE command is allowed, such as

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

| cr | 1st tab

The input-output commands are:

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

| cr | | lst tab

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

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

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

†Blank or numbers 1 through 7 only

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

```
          DATA  NAME
| cr      | lst tab
```

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

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

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

at the time of execution are 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 that scalars internal to some program are not punched or read. A name which has been assigned a machine address in a LET statement may appear in the list for PUNCH or READ. PUNCH provides, for each variable listed, as many control words as are necessary to recreate the form of the variable at a later read time. The content of the variable is punched in hexad with checksum format. These output tapes may be loaded through SPIREL or they may be read with a READ command. The READ command will read any tape produced by PUNCH. Also, READ will read any scalar, standard vector, or standard matrix punched with name by use of SPIREL directly.

The DISPLAY command provides decimal display of named scalars on the storage scope at the console. These are labelled by the name given in the argument list. Any name may have been assigned a machine address in a LET statement. As many as eight lines will be displayed by a single command. Real scalars are displayed one per line. Complex scalars are displayed on two lines, real part with the name of the variable followed by imaginary part with the name "ditto" (written '←←←←←'). Non-scalars may not be displayed.

The INPUT command and the OUTPUT command provide input and output of named scalars and non-scalars through programs supplied by the user. Any name may have been assigned a machine address by a LET command. For each variable named in an INPUT command control is passed to the program named INPUT; for an OUTPUT command, the program named OUTPUT is used. A complex variable is handled as two variables, the real part with the name of the complex variable and the imaginary part with the name "ditto". Details about the INPUT and OUTPUT programs are given in the library literature.

Formatted printer output may be obtained by use of the command

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

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

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

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

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

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

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

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

```
          TITLE   PRINT ONE LINE HERE
          TITLE
PRINT ANOTHER LINE HERE ALSO
| cr      | lst tab
```

The above would cause the following to be printed:

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

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

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

If the list is blank, the page is restored.

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

```
ACCEPT A,B,C
```

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

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

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

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

typewriter input: →, T, +002345 cr

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

```
ACCEPT NAME
```

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

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

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

|cr |lst tab

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

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

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

```
MESS1     BCD _ _TEMPUS_FUGIT
```

or

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

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

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

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

In the program Genie generates, in this case,

floating point 36.5 at CONST
floating point -2.0×10^{8} at CONST+1
integer 6 at CONST+2
octal 774777 (right-adjusted) at CONST+3

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

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

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

```
LINE      FORMAT ddd   ITERATIONS,   CASE aa,   K=bb,   T=-d.ddce+d
```

or

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

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

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

T7 should never be used in the Genie language.

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

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

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

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

T4	T6	CC	B2	B4
T5	T7	B1	B3	B5

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

R	B6	U	I
S	PF	X	

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

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

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

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

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

Then the instruction

```
                    QSR       #15
```

could be used instead.

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

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

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

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

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

```
END       TRA       Z
```

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

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

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

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

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

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

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

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

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

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

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

≠ is punched '= backspace uc |'

≮ is punched '< backspace uc |'

≰ is punched '≤ backspace |'

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

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

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

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

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

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

COMPILATION PROCEDURE

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

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

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

Program listing. Four columns are printed, giving:

(a) The symbolic location (if any).

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

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

(d) The symbolic address (if any).

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

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

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

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

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

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

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

Compilation options. See PLACER-TRANSLATE.

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

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

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

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

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

Arithmetic error tracing may help to locate mathematical problems.

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

- Least Squares

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

Lines 6 to 13:

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

Lines 14 to 45:

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

Lines 46 to 55:

The normal matrix is computed taking the weights into account.

Lines 56 to 67:

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

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

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

```
                                                                    1
        DEFINE                                                      2
        MATRIX XDATA, SIGMA                                         3
        VECTOR YDATA, COEFS, YCALC, RESID, WGHTS                    4
PFIT(XDATA, YDATA, WGHTS, COEFS, YCALC, RESID, SQSUM, SIGMA).=SEQ   5
        INTEGER .N, .M, .H, .J, .I                                  6
        .N = LENGTH(YDATA)                                          7
        .M = LENGTH(COEFS)                                         10
        .H = ROW (XDATA)                                           11
        .J = .M - .H                                               12
        CC = * WATE .IF .J = 0                                     13
        7       BAU             XDATA, U*B1                        14
                LRS             27                                 15
                CLA             *+1150                             16
                LRS             18                                 17
                CLA             .J, U*B2                           20
                LRS             15, R*T7                           21
                TSR             *+126                              22
                SPF             *END+1                             23
                CLA             B1, U*B4                           24
                CRL             15, R*B3                           25
        -B2     ADD             *B4+B3+1, U*B3                     26
        B3      BNA             B4+1, U*R                          27
                CRL             15                                 30
                LRL             12                                 31
                CLA             *+0120                             32
                LRS             12                                 33
        R       CRR             15, U*T6                           34
LOOP    B3      RPA             T6, B2=1                           35
        T6      TSR             *+126, U*T7                        36
                SPF             *END+1                             37
        B2      IF(NZE)TRA      *LOOP, B3+1                        40
        FOR     .J = 1, 1, .N                                      41
        FOR     .I = .H+1, 1, .M                                   42
        XDATA.I,.J = XDATA.I-1,.J*XDATA.H,.J                       43
        REPEAT                                                     44
        REPEAT                                                     45
WATE    EXECUTE VSPACE(RESID, .N)                                  46
        SIGMA = TRAN(XDATA)                                        47
        FOR .I = 1, 1, .N                                          50
        RESID.I = WGHTS.I * YDATA.I                                51
        FOR  .J = 1, 1, .M                                         52
        SIGMA.I,.J = SIGMA.I,.J * WGHTS.I                          53
        REPEAT                                                     54
        REPEAT                                                     55
WORK    COEFS = XDATA * RESID                                      56
        SIGMA = XDATA * SIGMA                                      57
        COEFS = SOLN(SIGMA, COEFS)                                 60
```

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

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

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


PFIT  .=
                ←BGIN      1    10 01000 02 4400 00136
                LOOP      51    43 21601 62 0000 00006
                ←FOR1     55    20 20001 00 4001 00236    .J
                ←FOR2     61    20 10000 00 0001 00231    .H
                ←RPT2    103    20 10401 00 0001 00211    .I
                ←RPT1    105    20 10401 00 0001 00206    .J
                WATE     107    01 21703 26 0200 00005    RESID
                ←FOR3    124    20 20001 00 4001 00170    .I
                ←FOR4    137    20 20001 00 4001 00154    .J
                ←RPT4    155    20 10401 00 0001 00136    .J
                ←RPT3    157    20 10401 00 0001 00135    .I
                WORK     161    01 21701 41 0200 00000    XDATA
                SUMM     245    00 20001 00 4600 00006    SQSUM
                ←FOR5    246    20 20001 00 4001 00046    .I
                ←RPT5    262    20 10401 00 0001 00032    .I
                END      306    01 01000 00 4400 00137
                         307    01 40006 00 4000 00000
                         310    07 01000 00 4200 00000
REFERENCE WORDS...
                SMMPY 77770    625454577040000000
                MPOWE 77771    545756644400000000
                MSUB  77772    546264412540000000
                SOLN  77773    625453552540000000
                MMPY  77774    545457702540000000
                TRAN  77775    636140552540000000
                VSPAC 77776    656257404240000000
                LENGT 77777    534455466340000000
INTERNAL STORAGE..
```

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

PARAMETERS AT FF +

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

SUBROUTINES REFERENCED

		137
GENIF...	SMMPY	
GENIF...	MPOWE	
GENIF...	MSUB	
GENIF...	SOLN	
GENIF...	MMPY	
		135
GENIE...	TRAN	
GENIF...	VSPAC	
		126
GENIF...	LENGT	
		136

```
                                                                    1
PFIT                ORG                                             2
                    REM     BACK-TRANSLATION                        3
L77770              REF     *SMMPY                                  4
L77771              REF     *MPOWE                                  5
L77772              REF     *MSUB                                   6
L77773              REF     *SOLN                                   7
L77774              REF     *MMPY                                   10
L77775              REF     *TRAN                                   11
L77776              REF     *VSPAC                                  12
L77777              REF     *LENGT                                  13
L1          -Z      TRA     *136,U→R                                14
L2                  SPF     B6=22                                   15
            PF      RWT     L307                                    16
                    CLA     PF+1,U→T7                               17
                    TSR     *L77777                                 20
                    SPF     *L307                                   21
                    STO     L311                                    22
                    CLA     PF+3,U→T7                               23
                    TSR     *L77777                                 24
                    SPF     *L307                                   25
                    STO     L312                                    26
                    CLA     PF,U→T7                                 27
                    TSR     *L77777                                 30
                    SPF     *L307                                   31
                    STO     L313                                    32
                    21740   L312                                    33
                    SUB     L313                                    34
                    STO     L314                                    35
                    21740   L314                                    36
                    IF(ZER)SKP      a2                              37
                    TRA     L30                                     40
                    CLA     aL107                                   41
                    NOP     Z,U→CC                                  42
L30         7       BAU     PF,U→B1                                 43
                    LRS     33                                      44
                    CLA     a1150                                   45
                    LRS     14                                      46
                    CLA     L314,U→B2                               47
                    LRS     17,R→T7                                 50
                    TSR     *126                                    51
                    SPF     *L307                                   52
                    CLA     B1,U→B4                                 53
                    CRL     17,R→B3                                 54
            -B2     ADD     a33+B4+1,U→B3                           55
            B3      00204   B4+1,U→R                                56
                    CRL     17                                      57
                    LRL     14                                      60
                    CLA     a120                                    61
                    LRS     14                                      62
            P       CRR     17,U→T6                                 63
L51         B3      RPA     T6,B2-1                                 64
            T6      TSR     *126,U→T7                               65
                    SPF     *L307                                   66
            B2      IF(NZE)TRA      L51,B2+1                        67
            1       STO     L314                                    70
L56                 CLA     L311                                    71
                    IF(POS)SKP      L314                            72
                    TRA     L107                                    73
```

```
                I       ADD         L313                    74
                        STO         L315                    75
L63                     CLA         L312                    76
                        IF(POS)SKP  L315                    77
                        TRA         L105                    100
                        CLA         L314,U→T6               101
                I       BUS+2       L315,B6+1               102
                T6      NOP         Z,U→B2                  103
                        21740       B6-1,U→B1               104
                        21740       *PF,B6=1                105
                        NOP         Z,J→T4                  106
                T6      NOP         Z,U→B2                  107
                        21740       L313,U→B1               110
                        21740       *PF                     111
                        10620       T4,U→P                  112
                T6      NOP         Z,J→B2                  113
                        CLA         L315,U→B1               114
                P       STO         *PF                     115
                I       FAD→        L315                    116
                        TRA         L63                     117
L105            I       FAD→        L314                    120
                        TRA         L86                     121
L107                    CLA+2       PF+5,B6+1               122
                Z       BAU+2       aL311,B6+1              123
                        TSR         *L77776                 124
                        SPF         *L307                   125
                        CLA         PF,U→T7                 126
                        TSR         *L77775                 127
                        SPF         *L307                   130
                        CLA         PF+7,U→B1               131
                Z       TSR         *135,U→B2               132
                        SPF         *L307                   133
                Z       LDR→        10,R→B2                 134
                P       STO         B1                      135
                B1      RWT         B2                      136
                I       STO         L315                    137
L125                    CLA         L311                    140
                        IF(POS)SKP  L315                    141
                        TRA         L161                    142
                        21740       L315,U→B1               143
                        21740       *PF+2,U→T4              144
                        21740       L315,U→B1               145
                        21740       *PF+1                   146
                        10620       T4,U→P                  147
                        CLA         L315,U→B1               150
                P       STO         *PF+5                   151
                I       STO         L314                    152
L147                    CLA         L312                    153
                        IF(POS)SKP  L314                    154
                        TRA         L157                    155
                        CLA         L315,U→T6               156
                        CLA         L314,U→B2               157
                        21740       T6,U→B1                 160
                        21740       *PF+7,U→T4              161
                        21740       T6,U→B1                 162
                        21740       *PF+2                   163
                        10620       T4,U→P                  164
                        CLA         L314,U→B2               165
                T6      NOP         Z,J→B1                  166
```

```
                R       STO     *PF+7                           167
                I       FAD*    L314                            170
                        TRA     L140                            171
L157            I       FAD*    L315                            172
                        TRA     L125                            173
L161                    CLA     PF,U*B1                         174
                        CLA     PF+5,U*B2                       175
                        TSR     *L77774                         176
                        SPF     *L307                           177
                        CLA     PF+3,U*B1                       200
                Z       TSR     *135,U*B2                       201
                        SPF     *L307                           202
                Z       LDR*    10,R*B2                         203
                R       STO     B1                              204
                R1      RWT     B2                              205
                        CLA     PF,U*B1                         206
                        CLA     PF+7,U*B2                       207
                        TSR     *L77774                         210
                        SPF     *L307                           211
                        CLA     PF+7,U*B1                       212
                Z       TSR     *135,U*B2                       213
                        SPF     *L307                           214
                Z       LDR*    10,R*B2                         215
                R       STO     B1                              216
                R1      RWT     B2                              217
                        CLA+2   PF+7,B6+1                       220
                        CLA+2   PF+3,B6+1                       221
                        TSR     *L77773                         222
                        SPF     *L307                           223
                        CLA     PF+3,U*B1                       224
                Z       TSR     *135,U*B2                       225
                        SPF     *L307                           226
                Z       LDR*    10,R*B2                         227
                R       STO     B1                              230
                R1      RWT     B2                              231
                        CLA     PF,U*T7                         232
                        TSR     *L77775                         233
                        SPF     *L307                           234
                        CLA     PF+3,U*B2                       235
                Z       TSR     *L77774,U*B1                    236
                        SPF     *L307                           237
                        CLA     PF+4,U*B1                       240
                Z       TSR     *135,U*B2                       241
                        SPF     *L307                           242
                Z       LDR*    10,R*B2                         243
                R       STO     B1                              244
                R1      RWT     B2                              245
                        CLA     PF+1,U*B1                       246
                        CLA     PF+4,U*B2                       247
                        TSR     *L77772                         250
                        SPF     *L307                           251
                        CLA     PF+5,U*B1                       252
                Z       TSR     *135,U*B2                       253
                        SPF     *L307                           254
                Z       LDR*    10,R*B2                         255
                R       STO     B1                              256
                R1      RWT     B2                              257
                Z       STO     *PF+6                           260
                I       STO     L315                            261
```

```
L247                CLA             L311                        262
                    IF(POS)SKP      L315                        263
                    TRA             L264                        264
                    CLA             L315,U→T6                   265
                    21740           T6,U→B1                     266
                    21740           *PF+5                       267
                    FMP             U,U→T4                      270
                    21740           T6,U→B1                     271
                    21740           *PF+2                       272
                    FAD             T4                          273
                    FAD•            *PF+6                       274
                I   FAD•            L315                        275
                    TRA             L247                        276
L264                21740           L311                        277
                    SUB             L312                        300
                    53100           -1                          301
                    FMP             L316                        302
                    VDF             *PF+6,U→T4                  303
                    LDR             -81                         304
                    CLA             PF+7                        305
                    TSR             *L77771                     306
                    SPF             *L307                       307
                    SB1             10                          310
                T4  TSR             *L77770                     311
                    SPF             *L307                       312
                    CLA             PF+7,U→B1                   313
                Z   TSR             *135,U→B2                   314
                    SPF             *L307                       315
                Z   LDR•            10,R→B2                     316
                P   STO             B1                          317
                B1  RWT             B2                          320
                    TRA             *137                        321
L307                SB6             Z                           322
                T7  TRA             PF                          323
L311                OCT             000000000000000000          324
L312                OCT             000000000000000000          325
L313                OCT             000000000000000000          326
L314                OCT             000000000000000000          327
L315                OCT             000000000000000000          330
L316                OCT             062000000000000000          331
                    END                                         332
                                                                333
                                                                334
```

• Numerical Integration

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

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

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

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

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

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

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

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

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

```
                                                                        1
        DEFINE                                                          2
        SCALARS X0,Y0,Z0,XF,H,X,Y,ZED                                   3
        INTEGER M                                                       4
        VECTORS B,C,W                                                   5
        FUNCTIONS F,ADAMS                                               6
                                                                        7
EXAMPLE3(7),=SEQ                                                        10
        REM                                                             11
THIS IS THE DRIVER PROGRAM.  IT CONTROLS INPUT, INTEGRATION, AND OUTPUT
                PAG             +7                                      13
                SLN             +00002                                  14
                LT7             +00000 3120 0000 00135                  15
                TSR             *+126                                   16
        EXECUTE SCRIBE(HEADER)                                          17
        M=6, H=0.01                                                     20
LOOP    DATA X0,Y0,Z0,XE                                                21
        EXECUTE ADAMS(X0,Y0,Z0,XE)                                      22
        EXECUTE SCRIBE(M,H,X0,Y0,Z0,IN)                                 23
        EXECUTE SCRIBE(X,Y,ZED,OUT)                                     24
                SPA             +7                                      25
        CC=+LOOP                                                        26
HEADER  FORMAT                                                          27
M       H               Y0              Y0              Z0              30
IN      FORMAT                                                          31
d    d.ddddd    -ddddd.ddddd    -ddddd.ddddd    -ddddd.ddddd
OUT     FORMAT                                                          33
                -ddddd.ddddd    -ddddd.ddddd    -ddddd.ddddd
END                                                                     35
                                                                        36
F(XX,YY,ZZ)=ZZ(SIN(TMP)+COS(TMP)+YY**2+2), TMP=XX+YY+ZZ                 37
                                                                        40
ADAMS(XX,YY,ZZ,EE).=SEQ                                                 41
        REM     XX,YY,ZZ ARE THE INITIAL VALUES FOR X,Y,Y PRIME
        REM     M IS THE ORDER OF THE METHOD (≤6)                       43
        REM     EE IS THE END OF THE INTEGRATION                        44
        REM     H IS THE INTEGRATION STEP                               45
        REM     W0 IS THE SECOND DERIVATIVE, WK THE KTH BACK DIFF.
        EXECUTE VSPACE(B,7)                                             47
        EXECUTE VSPACE(C,7)                                             50
        EXECUTE VSPACE(W,M+1)                                           51
        B1=1.,          C1=0.5                                          52
        B2=0.5,         C2=1./6.                                        53
        B3=5./12.,      C3=1./8.                                        54
        B4=3./8.,       C4=19./180.                                     55
        B5=251./720.,   C5=3./32.                                       56
```

```
        B6=95./288.,          C6=963./10080.                    57
        B7=19087./60480.,     C7=275./3456.                     60
        WE=1*10, R=M, DECIDE=TRUE                               61
LOOP    CC=*ADMINT                                              62
RLINT   BETA=1*-7<1W_M-WE1                                      63
        WE=W_M                                                  64
        FOR J=M,-1,1                                            65
        FOR V=2,1,M                                             66
        W_V=W_V-W_V+1                                           67
        REPEAT                                                  70
        REPEAT                                                  71
        CC=*LOOP .IF BETA                                       72
        R=FIX((XE-XX)/H), DECIDE=FALSE                          73
ADMINT  X=XX, Y=YY, ZED=ZZ                                      74
        FOR J=1,1,R+1                                           75
        CC=*L14 .IF J=1                                         76
        FOR V=M,-1,1                                            77
        W_V+1=W_V                                               100
        REPEAT                                                  101
L14     W_1=F(X,Y,ZED)                                          102
        REM    F IS THE FUNCTION DEFINING THE                   103
        REM    DIFFERENTIAL EQUATION                            104
        CC=*NSHIFT .IF J=0                                      105
        FOR V=2,1,M+1                                           106
        W_V+1=W_V                                               107
        REPEAT                                                  110
NSHIFT  P=Z, Q=Z                                                111
        FOR V=1,1,M+1                                           112
        P=P+B_V W_V                                             113
        Q=Q+C_V W_V                                             114
        REPEAT                                                  115
        X=X+H                                                   116
        Y=Y+H(ZED+Q H)                                          117
        ZED=ZED+P H                                             120
        REPEAT                                                  121
        CC=*RLINT .IF DECIDE                                    122
        ERASE B,C,W                                             123
END                                                             124
        DEFINE                                                  125
LEAVE                                                           126
                                                                127
```

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

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


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

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

               HEADE     51     00 00000 00 0000 00007

               IN        61     00 00500 00 0000 00010

               OUT       72     00 00300 00 0000 00010

               END      103     01 01000 00 4000 00000
REFERENCE WORDS...
               ZED   77764     7144432525000000000
               Y     77765     7025252525000000000
               X     77766     6725252525000000000
               ADAMS 77767     4043405462400000000
               XE    77770     6744252525000000000
               ZO    77771     7100252525000000000
               YO    77772     7000252525000000000
               XO    77773     6700252525000000000
               .INOU 77774     7555555644000000000
               H     77775     4725252525000000000
               M     77776     5425252525000000000
               SCRIB 77777     6242615041400000000
INTERNAL STORAGE..
               .NUMB   104           3130000000135
               .NUMB   105     7700243636075075341


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

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

←BGIN   PROGRAM SEQUENCE
END     PROGRAM SEQUENCE



F     .=
            ←BGIN       1     10 01000 02 4400 00136

            END        25     01 01000 00 4400 00137
                       26     01 40006 00 4000 00000
                       27     07 01000 00 4200 00000
REFERENCE WORDS...
            COS    77776      4256622525400000000
            SIN    77777      4250552525400000000
INTERNAL STORAGE..
            TMP        30                          0
            ←NUMB      31      1002000000000000000

    PARAMETERS AT PF +
            XX              0
            YY              1
            ZZ              2



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

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

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

ADAMS .=
            ←BGIN      1    10 01000 02 4400 00136

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

            RLINT   129    01 21740 41 0401 77650    M

            ←FOR1   134    01 21700 00 0401 77636    M

            ←FOR2   141    01 21700 00 4000 00002

            ←RPT2   155    20 10401 00 0001 00230    V

            ←RPT1   157    30 10401 00 0001 00225    J

            ADMIN   175    01 21700 00 0600 00000    XX

            ←FOR3   209    20 20001 00 4001 00201    J

            ←FOR4   215    01 21700 00 0401 77555    M

            ←RPT4   230    30 10401 00 0001 00155    V

            L14     239    00 20102 26 4401 77534    X

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

            ←RPT5   263    20 10401 00 0001 00122    V

            NSHIF   265    00 20001 00 4001 00122    P

            ←FOR6   267    20 20001 00 4001 00116    V

            ←RPT6   313    20 10401 00 0001 00072    V

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

```
                END         347     01 01000 00 4400 00137
                            350     01 40006 00 4000 00000
                            351     07 01000 00 4200 00000
REFERENCE WORDS...
                F       77764     4525252525400000000
                ZED     77765     7144432525000000000
                Y       77766     7025252525000000000
                X       77767     6725252525000000000
                FIX     77770     4550672525400000000
                H       77771     4725252525000000000
                XE      77772     6744252525000000000
                M       77773     5425252525000000000
                W       77774     6625252525400000000
                C       77775     4225252525400000000
                VSPAC   77776     6565257404240000000
                B       77777     4125252525400000000
INTERNAL STORAGE..
                +P1       352                         0
                +UNEF     353       1001000000000000000
                +NUMB     354     7720000000000000000
                +NUMB     355       1006000000000000000
                +NUMB     356       1005000000000000000
                +NUMB     357       1014000000000000000
                +NUMB     360       1010000000000000000
                +NUMB     361       1003000000000000000
                +NUMB     362       1023000000000000000
                +NUMB     363       1264000000000000000
                +NUMB     364       1373000000000000000
                +NUMB     365       2002640000000000000
                +NUMB     366       1040000000000000000
                +NUMB     367       1137000000000000000
                +NUMB     370       2001100000000000000
                +NUMB     371       2003276000000000000
                +NUMB     372       2047300000000000000
                +NUMB     373       2112436000000000000
                +NUMB     374       2354200000000000000
                +NUMB     375       2001046000000000000
                +NUMB     376       2015400000000000000
                +NUMB     377       5002250057440000000
                WE        400                         0
                R         401                         0
                DECID     402                         0
                +NUMB     403     7500153277455152745
                BETA      404                         0
                J         405                         0
                V         406                         0
                +P2       407                         0
                P         410                         0
                Q         411                         0

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

SUBROUTINES REFERENCED

137
135

F
GENIF... FIX
GENIE... VSPAC

136

- Complex Matrix Inverse

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

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

Lines 11 to 13:

Working storage defined.

Line 14:

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

Lines 20 to 27:

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

Lines 30 to 64:

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

Lines 66 to 103:

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

Line 104:

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

functions are stored. Working storage is freed.

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

```
                    DEFINE                                          2
                    COMPLEX MATRIX B,A,NEW                          3
                    INTEGER VECTOR ROWW,COLL                        4
INVERT(B),=SEQ                                                      5
                    INTEGER GG,HH,L,C,D,E                           6
                    COMPLEX G                                       7
                    L=ROW(B)                                        10
                    EXECUTE VSPACE(ROWW,L)                          11
                    EXECUTE VSPACE(COLL,L)                          12
                    EXECUTE MSPACE(NEW,L,L)                         13
                    A=B                                             14
                    FOR C=1,1,L.                                    15
                    GMOD=0.                                         16
                    GG=C,HH=C                                       17
                    FOR D=C,1,L                                     20
                    FOR E=C,1,L                                     21
                    KMOD=MOD(A D,E)                                 22
                    CC=←MORE ,If KMOD≤GMOD                          23
                    GG=D,HH=E                                       24
                    GMOD=KMOD                                       25
MORE                REPEAT                                          26
                    REPEAT                                          27
                    CC=←BYE ,If GMOD≤1.0*-12                        30
                    COLL C=GG                                       31
                    ROWW C=HH                                       32
                    FOR D=1,1,L                                     33
                    G=A C,D                                         34
                    A C,D=A GG,D                                    35
                    A GG,D=G                                        36
                    REPEAT                                          37
                    FOR D=1,1,L                                     40
                    G=A D,C                                         41
                    A D,C=A D,HH                                    42
                    A D,HH=G                                        43
                    REPEAT                                          44
                    NEW C,C=(1.+/0.)/A C,C                          45
                    FOR D=1,1,L                                     46
                    CC=←SOME ,If D=C                                47
                    NEW D,C=A D,C/A C,C                             50
SOME                REPEAT                                          51
GENIE               FOR D=1,1,L                                     52
September, 1966     CC=←TOME ,If D=C                                53
                    NEW C,D=-(A C,D/A C,C)                          54
```

```
TOME        REPEAT                                                  55
            FOR D=1,1,L                                             56
            FOR E=1,1,L                                             57
            CC=→VOME ,IF D=C ,0 ,R E=C                              60
            NEW_{D,E}=A_{D,E}+(NEW_{C,F})(A_{D,C})                  61
VOME        REPEAT                                                  62
            REPEAT                                                  63
            A=NEW                                                   64
            REPEAT                                                  65
            FOR C=L,-1,1                                            66
            HH=COLL_C                                               67
            GG=ROWW_C                                               70
            FOR D=1,1,L                                             71
            G=A_{D,C}                                               72
            A_{D,C}=A_{D,HH}                                        73
            A_{D,HH}=G                                              74
            REPEAT                                                  75
            FOR D=1,1,L                                             76
            G=A_{C,D}                                               77
            A_{C,D}=A_{GG,D}                                        100
            A_{GG,D}=G                                              101
            REPEAT                                                  102
            REPEAT                                                  103
            RESULT=A                                                104
            ERASE COLL,ROWW,NEW,A                                   105
            CC=→END                                                 106
BYE         EXECUTE SCRIBE(MESS)                                    107
            CC=→END                                                 110
MESS        FORMAT                                                  111
NO INVERSE  DUE TO SINGULARITY                                      112
END                                                                 113
            DEFINE                                                  114
LEAVE                                                               115
                                                                    116
```

GENIE
September, 1966

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

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


INVER .=
                ←BGIN     1    10 01000 02 4400 00136
                ←FOR1    47    20 20001 00 4001 00511    C
                ←FOR2    6A    01 21700 00 0001 00500    C
                ←FOR3    6E    01 21700 00 0001 00473    C
                MORE    11E    20 10401 00 0001 00450    E
                ←RPT2   117    20 10401 00 0001 00445    D
                ←FOR4   134    20 20001 00 4001 00430    D
                ←RPT4   162    20 10401 00 0001 00402    D
                ←FOR5   164    20 20001 00 4001 00400    D
                ←RPT5   212    20 10401 00 0001 00352    D
                ←FOR6   232    20 20001 00 4001 00331    D
                SOME    26E    20 10401 00 0001 00277    D
                ←FOR7   267    20 20001 00 4001 00275    D
                TOME    321    20 10401 00 0001 00243    D
GENIE
September, 1966 ←FOR8   323    20 20001 00 4001 00241    D
                ←FOR9   327    20 20001 00 4001 00236    E
```

```
                    VOME     375     20 10401 00 0001 00170     E

                    ←RPT8    377     20 10401 00 0001 00165     D

                    ←RPT1    420     20 10401 00 0001 00140     C

                    ←FORa    42?     01 21700 00 0001 00133     L

                    ←FORb    43=     20 20001 00 4001 00127     D

                    ←RPTb    463     20 10401 00 0001 00101     D

                    ←FORc    46=     20 20001 00 4001 00077     D

                    ←RPTc    513     20 10401 00 0001 00051     D

                    ←RPTa    515     30 10401 00 0001 00043     C

                    BYE      541     01 21703 26 4001 00004     MESS

                    MESS     546     00 00000 00 0000 00004

                    END      553     01 01000 00 4400 00137
                             554     01 40006 00 4000 00000
                             555     07 01000 00 4200 00000
REFERENCE WORDS...
                    SCRIB 77755      5242615041400000000
                    CADD  77756      4240434325400000000
                    CMPY  77757      4254577025400000000
                    CDIV  77760      4242506525400000000
                    CMPLX 77761      4254575367000000000
                    ←←←←← 77762      7575757575000000000
                    MOD   77763      5454432525400000000
                    CSTAR 77764      4262634061400000000
                    ←←←←← 77765      7575757575400000000
                    A     77766      4025252525400000000
                    ←←←←← 77767      7575757575400000000
                    CMCPY 77770      4254425770400000000
                    CMSPA 77771      4254625740400000000
                    NEW   77772      5544662525400000000
                    ←←←←← 77773      7575757575400000000
                    COLL  77774      4254535325400000000
                    VSPAC 77775      6562574042400000000
                    ROWW  77776      6154666625400000000
                    CROW  77777      4261566625400000000
INTERNAL STORAGE..
                    L        556                        0
                    ←P1      557                        0
                    ←P2      560                        0
                    C        561                        0
                    GMOD     562                        0
                    GG       563                        0
                    HH       564                        0
                    D        565                        0
                    E        566                        0
                    ←Q1      567                        0
                    ←←←←←    570                        0
                    KMOD     571                        0
                    ←NUMB    572     7300106274630045570
                    G        573                        0
                    ←←←←←    574                        0
GENIE               ←ONEF    575      1001000000000000
September, 1966
          PARAMETERS AT PF +
                    B                 0
                    ←←←←←             1
```

SUBROUTINES REFERENCED

137

GENIE... SCRIB
GENIE... CADD
GENIE... CMPY
GENIE... CDIV
GENIE... MOD

135

GENIE... CMCPY
GENIE... CMSPA
GENIE... VSPAC
GENIE... CROW

136

CODING CONVENTIONS

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

• Programs initialization and termination

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

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

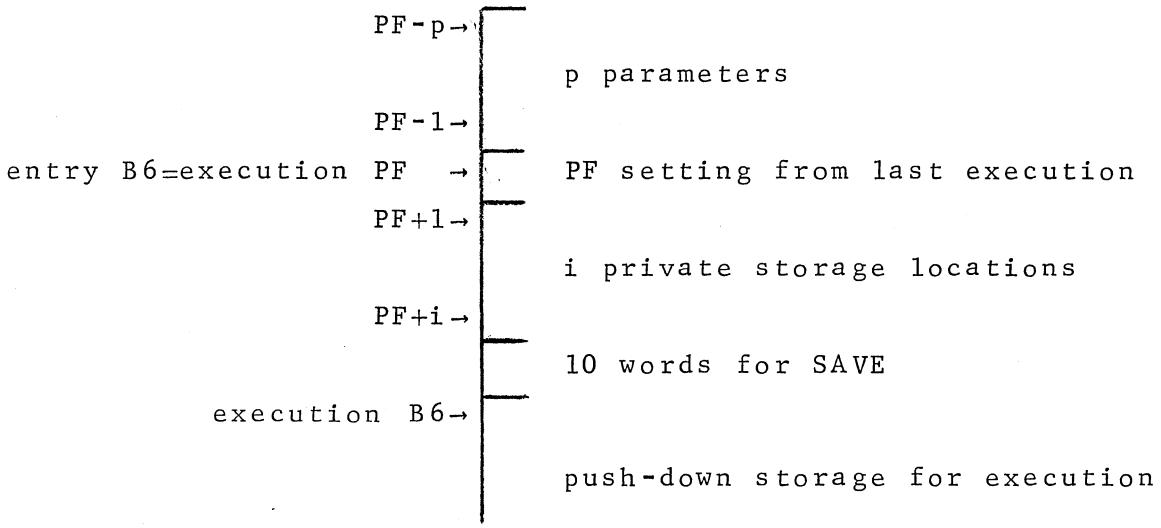
For a non-recursive program -- one begun with 'SEQ' or a one statement function ... A single fast parameter in the definition of a program is a special case which causes only PF to be saved and assumes no parameter addressing in Genie language within the program. Otherwise, fast register names should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

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

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

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

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

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

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

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



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

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

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

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

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

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

- Addressing of variables

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

For a non-recursive program --

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

For a recursive program --

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

- B6-list, working storage

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

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

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

- Parameter set-up for program execution

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

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

- Representation of Complex Variables

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

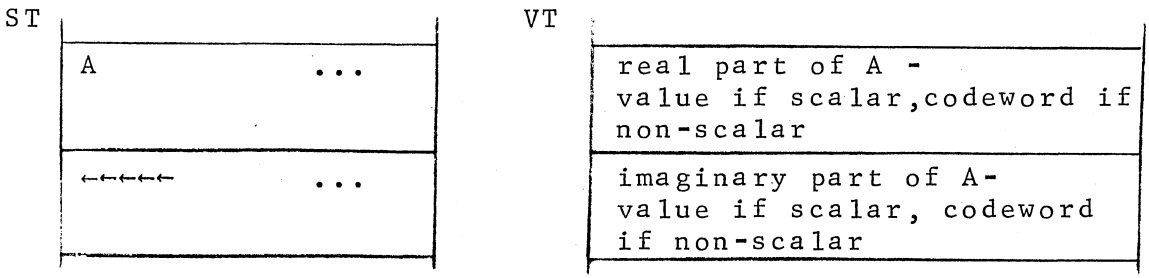
Genie generates internal storage for the complex scalar A as

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

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

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

Then while running the corresponding ST-VT configuration is



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

- Subscription

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

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

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

• Operations on standard forms of non-scalars

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

standard form of one dimensional array, vector

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

standard form of two dimensional array, matrix

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

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

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

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

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

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

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

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

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

- Assignment of type and shape to variables

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

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

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

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

```
-LDU     -A
FMP      ←TW47
```

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

Integers and real floating point scalars and non-scalars may be combined with complex scalars and non-scalars in any mathematically meaningful way. In all cases except exponentiation of a complex scalar by an integer or floating point scalar the non-complex quantity is made complex before the combination takes place. A floating point quantity is made complex with real part equal the floating point quantity and zero imaginary part; an integer quantity is floated then made complex as a floating point quantity.

- 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  = +007777777777777777
FALSE = +007777777777777776
```

The binary operations between Boolean variables to yield a Boolean value cause code to be generated as follows:

or, A+B, true if either A or B is true

```
CLA   A
ORU   B
```

and, A×B, true if both A and B are true

```
CLA   A
ORU   B
```

symmetric difference, A-B, true if A and B have different values

```
CLA   A
SYD   B
ORU   #77776
```

symmetric sum, A/B, true if A and B have the same value

```
CLA   -A
SYD   B
```

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

```
-I   ORU   -A
```

The machine register sense lights (SL) is a collection of 15 bits, any one of which may be individually meaningful and may be in an on or off (1 or 0) state at any time. The variable SL is Boolean and exponentiation to an integer power is defined

A^B, true if bit B of A is on (1) where the bits of A are numbered from 1 to 15, from left to right

```
CLA     A          ┐
LUR     15-B       │ if B is a number
ORU     #+77776    ┘
CLA     B          ┐
BUS     #15,U→R    │ if B is
CLA     A          │ a name
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 15 bits (the rightmost in a machine word) A^1, A^2, ..., A^{15} each with a value of TRUE or FALSE.

- 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
            compute increment                                    ┐
            store increment                                      A
            compute final                                        ┐
            store final                                          A
[←FORk+m]       LT7                 final                        B
            Z   IF(POS)SKP          increment                    ┐
            T7  IF(POS)SKP          iteration parameter, CC+1    C
            T7  IF(NEG)SKP          iteration parameter          ┘
                TRA                 ←RPTk+n
                 ⋮
              orders of loop
                 ⋮
←RPTk           CLA                 increment                    ┐
                FAD→                iteration parameter          D
                TRA                 ←FORk+m                      ┘
[←RPTk+n]        ⋮
```

Seldom is the full generalized code necessary, and the following notes pertain to condensations which are provided in various specific cases.

(A) The increment and the final value are computed and stored only if they are given by expressions, that is, not simple variable names or explicit numbers.

(B) The final value will be stored in the address field of the order if it is given by an explicit integer.

(C) If the increment is given by an explicit integer, it will not be tested for being positive or negative and only the appropriate comparison of iteration parameter to final value will be generated.

(D) If the iteration parameter is a long fast register F, the ←RPTk code sequence will be

```
←RPTk     F     FAD       increment, U→F
                TRA       ←FORk+m
```

If the iteration parameter is an index register Bi and the increment is an explicit integer +1 or -1, the ←RPTk code sequence will be

```
←RPTk           TRA       ←FORk+m,Bi±1
```

- Use of fast registers in Genie generated code

Fast registers may be used in the Genie language and in assembly language coding to be used in a Genie context if there is no conflict with usage generated by the compiler:

T7 is always subject to use for special purpose temporary storage.

T7 is used for storage of a single parameter when a function is executed implicitly or explicitly.

T4, T5, T6 are subject to use in any arithmetic command for scalar temporary storage and for storage of scalars mentioned two or more times in one equation if these fast register names are not mentioned explicitly in the command.

B1 is used when loading parameters onto the B6-list if a name ←Pn is used.

B1, B2, B3, B4, B5 are used for subscripts in addressing elements of arrays. The first k are used to address an element of an array of k dimensions.

B1 and B2 are used in complex scalar arithmetic.

B1, B2, and PF may be used in operations on vectors and matrices.

B1 is used in input-output commands to specify to the program ←INOUT the operation to be performed.

B1 is used in raising an integer or a real floating point scalar to an integer power ≤ 7.

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

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

```
SPF          *END+1
```

PF is used within every recursive program to address parameters and private storage locations. The appropriate value of (PF) is stored in the address portion of END+1 so that resetting is easily accomplished by

```
SPF          *END+1
```

- Rearrangement of arithmetic formulae for efficient evaluation

The compiler has the ability to rearrange the terms in addition (or subtraction) and multiplication (or division) strings. Constant terms are shifted to the left in the formula. Terms which are themselves expressions, rather than simple variable names or numbers, are shifted to the left to save temporary stores that would be required were such complex terms to appear to the right in a string. The ordering of the complex terms is determined by the number of temporary stores required to evaluate each; the complex term requiring the most temporary stores will be shifted farthest to the left.

If the order of evaluation within a formula is of importance, this rearrangement may be avoided by defining each complex term in a separate equation, thereby giving each a name. Then the original formula will involve only simple variable names, and rearrangement will not take place.