Genie

An Intermediate Range Assembly System

1. Introduction

This may be regarded as a trial version of Genie. It would be unduly optimistic to start using it immediately for urgent coding, but potential users are urged to try the formalism, which seems to be new in some respects, and let us know how it behaves.

Some technical details are given in Programming Memorandum #4, in which general translation processes are formalised. The present memorandum consists of a user's guide to F1, the formula language, and AP2, the revised version of the 'machine symbolic' code. One of the main objectives of Genie is to assist in writing more elaborate automatic coding systems, and in carrying out experiments where the ability to treat complete programs as objects of study is essential. This facility is provided by other parts of Genie which are not described here, but it is felt that some of its flexibility may also be useful in experimental numerical investigations.

2. The Formula Language, F1

2.1 The Assembly Process

A well defined problem requires both a procedure which will lead to its solution, and data to which the procedure will be applied. It is convenient to use the symbols and conventions of algebra to describe a procedure, and F1 is designed to be used when the data consists of 'ordinary' numerical operands: numbers, vectors and matrices.

A procedure is represented by a program, which consists of a sequence of commands, followed by a set of definitions. The commands are written down in vertical arrangement on the coding sheet (Figure 1), and when the program is applied they are normally obeyed in succession, starting with the first one on the sheet. If the commands are numbered 1,2,3,... then on completion of a command a special register in the machine, the Control Counter (CC), contains the number of the next one to be obeyed. Since the Control Counter may itself be directly altered by a command, the 'normal' sequencing of operations may be interrupted at any time: this corresponds to the usual 'transfer of control'. For such purposes, the commands may be identified by names which are placed in the left-hand margin of the coding sheet, under the heading LOCN. The commands themselves start at the first 'tab' position and extend across the line; they may be continued by indenting to the third 'tab' position on the next line.

LOCN	SETU	OPN	AUX,ADDR	REMARKS
Figure 1. An example of coding in F1. The evaluation of the integral F(x) between limits a and b to preassigned accuracy Δ. V is a given number which is greater than F(x) in the range (a,b). This heading is punched with the 7th hole punch on the paper tape, so it is ignored by the computer. Now it is turned off →				
	S(F, a, b, Δ, V)$_o$ = SEQ			
	Function F			
	IA = V(b - a)			
	n = 1			
	h = (b - a)/2			
	J = h(F(a) + F(b))			
A	I = J + $4\Sigma_{i=1,n}$F(a + 2(i - 1)h)			

(Figure 1 - continued)

LOCN	SETU	OPN	AUX,ADDR	REMARKS
	CC = #B if $\Delta \nless$ |I - IA|			
	IA = I, J = (I + J)/4, n = 2n, h = h/2			
	CC = #A			
B	T7 = I/3			
	return			
		END		

It will be seen that a command looks much like any equation or set of equations of algebra: it includes the usual operations, function names, etc., although some care has to be taken where the usual notation is ambiguous, or where the mechanics of the Flexowriter permit the same printed document to be achieved by different sequences of key strokes. Genie looks at a program as a linear sequence of character codes, and not as a printed page.

Generally speaking, a command 'defines' just one object: its principal variable. The coder must see that the quantities on which it depends have been defined before the command is executed. This can happen in two ways: (a) by appearing as principal variables of other commands, previously executed, or (b) by appearing as 'definitions' outside the command sequence. An example should make the distinction clearer. A command such as "y = 3.06" causes the generation of code which transfers the number '3.06' from one storage cell to another, which has been assigned to 'y'. A definition such as "y = 3.06" generates no code, but ensures that in all commands the value of 'y' is taken as '3.06'. Definitions generally have the form of programs which define 'functions' although (as in the above case) they may degenerate into single commands defining single numbers.

A program with one or more 'input' parameters and a single 'output' value may be used in formulae in the usual algebraic sense. In its definition, the name of the program is followed by the names of its 'input' parameters, i.e., the form:

$f(x,y) = x^2 + y^2 - 2kxy$

constitutes a definition of the single valued function f. Its parameters are 'x' and 'y', and 'k' is a quantity which must be defined elsewhere in the program.

A program may also be used in a more general way, as a many-valued function of several arguments. It cannot then be used in a formula, but it may appear as a single name, followed by the names of its input and output parameters, i. e., if P(a,b,#X,#Y,#Z) is a program which computes X,Y,Z from the input quantites a,b, then:

P(U,V,#L,#M,#N)

is a command causing L,M,N to be calculated from U and V. Note the use of "#" to indicate a quantity computed by the routine.

A program may still use names which are undefined when it is being coded: the coder must be aware of this and do one of two things about it. If he intends to use the program as a function in one of the above two senses, then any of the undefined quantities which are input or output parameters must be listed as such after the name of the program. On the other hand, there may be quantities which he is unable or unwilling to define when the code is written. In this case their names must be written as part of the heading of the program as shown in Figure 2. Here a,b, and c are the parameters of program P, and G is an undefined name in Q. P has no undefined names and Q has no parameters. A sequence of commands is

parenthesised by the "SEQ...END" construction. A program is parenthesised by the "(...)" construction. These may be omitted where there is no ambiguity.

LOCN	SETU OPN AUX,ADDR	REMARKS
Figure 2. An example of program nesting. (See text).		
	Q = (SEQ G	Start of Q
	y = (x + 3.075)x	Commands
	read G	of Q
	u = G sin y + P(G, 2G, y)	
	print G,u	
	return	
	END	
	x = cos(4π/3) + log 3	Def of x
	P(a,b,c) = SEQ	Start of P
A1	$T4 = (a - b)^2 + (b - c)^2 + x(c-a)^2$	
	CC = #A2 if 51.75 < T4	Commands
	T7 = sqrT4	of P
	return	Logical end of P
A2	T4 = T4 - 51.75	
	CC = #A1	
	END)	End of P and Q.

Assembly is sufficiently fast to permit frequent re-runs in the course of program checking. If this is not desirable, a code may be partitioned into subroutines which may be tested individually and combined together at a later stage. Provision is also made for inserting corrections in the form of short command sequences.

Genie is basically a two-phase assembly system. In the first phase, a single address code is written, similar to machine code in most respects, except that it contains indirect references to its operands via a 'symbol table', and a number of pseudo-orders which affect the second phase. Initially the symbol table contains only

some fixed names. Commands are processed one after another: if a command contains some new symbols, they are added to the table. The code thus formed, together with its table, may be retained if corrections are to be made to the program later. It is called 'δ-code'. At each stage of assembly, the names presently on the symbol table may be identified with the names appearing in a command, and it is this facility, of course, which makes the program into a coherent whole. At the end of a program all its symbols are removed from the table, and this has the following important effect: a sub-program P may share some symbols with the program Q in which it appears, and also have its own private symbols which do not appear in Q. The same remarks apply to any other sub-program P' in Q. However, the private symbols of P and P' are distinct in meaning although some of them may be identical in form. This is probably the only tricky device in Genie, and it should be clearly understood if programs are nested within one another. It is also applied to commands, which may have their private 'auxiliary variables' distinct from quantities similarly named in other commands.

In the second phase of assembly, machine code is written. Generally, this is done automatically, and stored in the machine ready for execution. Under sense switch control, both the δ-code and final codes can be punched out and printed for reference purposes.

A program which is punched out in binary form contains symbolic references to its undefined names. It may be reloaded at any time, together with definitions of these quantities, prior to execution. This is the normal way of adding library routines to a program: in the present terms, a 'library' program is one which con-

tains no names other than those of its parameters, its own internal variables, certain fixed registers in the machine, and other library routines. Sets of interdependent library programs may be read into the machine in any order, prior to execution. (In Figure 2, Q is a library program, but P is not.)

Storage space is not reserved for vectors, matrices, subroutines etc., unless they are completely defined at assembly time: it is obtained during execution from a 'bank' of stores which is controlled by an independent routine. It is necessary to know the nature of the undefined quantities, if they are not single precision floating point numbers, and this must be given by a declaration. In one other respect Genie does not observe the traditional distinction between 'assembly' and 'execution'; it will recognise a program which depends on no external quantities and execute it automatically: this applies at all levels of coding.

2.2 Details of F1

2.2.1 Names

Symbolic names are used for operands and certain special purposes described below. The operands of F1 are assumed to be single precision floating point numbers unless, by declaration or implication, they are stated to be otherwise.

Type 1 Name: This is either a single lower case letter,[1] or a sequence consisting of some upper case letters, followed by some (possibly none) lower case letters, followed by some (possibly none) decimal digits. Any number of characters may appear in a

[1] 'Lower case letters' include π, Δ, α, β, γ, λ

name, but two names of more than four characters will be considered identical if their first four characters are similar. A name may not contain a 'space' (blank character).

Examples: K6, Mass, Time, p, Q

Type 2 Name: In order to make use of the 'fast' registers on the machine as temporary stores, and for specialised purposes, 17 registers are identified by symbolic names, which should not be used for any other purpose. These are:

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

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

Special purpose: X

Type 3 Name: Since F1 can be used in conjunction with AP2, the symbolic operation codes of the latter may not be used as operand names in F1. There are about 100 of these, and they are listed in Programming Memorandum #3, Section 3. They are all, with the exception of "IF", either three upper case Roman letters or two upper case Roman letters followed by a single digit.

Example: AB1, CLA, NZE, SEQ

Special Symbols: Certain combinations of letters have special meanings which will be given in later paragraphs. They fall under the following headings:

(i) Boolean operations: "and" and "or"

(ii) System subroutine names: "sin", "cos", "tan", "log", "exp", "sqr"

(iii) Control words: "if", "for", "repeat", "dim", "return"

(iv) Declarations: "Function", "Vector", "Matrix", "Integer"

(v) Input-Output: "print", "read", "punch"

(vi) Others: The sense light register, σ

2.2.2 Numbers

Any string of digits not forming part of a name is treated as a number. In F1, this is assumed to be a decimal integer, or a floating point decimal number, according as to whether a decimal point does not or does appear in the string. It is converted to the appropriate binary form. A number may be of any length. Integers are evaluated modulo 32,768; floating point numbers are significant to about 13 decimal places.

Examples: 122, 87645, .000324, 895.2, 3., .5

2.2.3 Subscripts

A name may have one or two subscripts. If two, they must be separated by a comma. They may have any form permissible to a formula (Section 2.2.5), and they are evaluated as integers modulo 512. All arrays are indirectly addressed through 'codewords' (Programming Memorandum #3, Section 2). Subscripting is indicated by a half-line shift down on the typewritten sheet. To avoid ambiguity, parentheses must be used in cases where subscripts appear in exponents, and vice versa.

2.2.4 Operations

(i) The binary operations of F1, in order of decreasing 'precedence' are: "ⓔ" (exponentiation), "x", "/", "+" and "-" (with the same precedence) and "," (list formation). These all have the usual meanings with regard to floating point quantities, and

floating point operations are followed by rounding and normalization. Associative operations are obeyed from right to left in the formula. A function name operates on the value of the expression to its right, which includes all terms which are connected by the operation of multiplication. Where no possible confusion can arise, the "x" sign between operands may be omitted.

Example: $abc/4kt^2$ is evaluated as: (a x(b x c))/(4 x (k x (t ⓔ 2)))

sin4p/3 is evaluated as: (sin (4 x p))/3

Exponentiation is indicated by a half line shift up on the typewritten sheet.

Operands are normally assumed to be single precision floating point numbers. They may be determined otherwise by implication (Section 2.2.7) or declaration (Section 2.2.8). Matrices and vectors must always be in single precision floating point form.(1) Expressions involving 'mixed' operands are permitted where an unambigious interpretation can be given to each operation. In Figure 3, the permitted operations for given operands are indicated, together with the type of the resultant operand.

A ⓡ B		Second operand B			
		Integer	Fl.pt. No.	Vector	Matrix
First Operand A	Integer	All(Integer)	All(Fl.pt.No.)	x(Vector)	x(Matrix)
	Fl.pt.No.	All(Fl.pt. No.)	All(Fl.pt.No.)	x(Vector)	x(Matrix)
	Vector	x (Vector)	x (Vector)	x(Scalar) +-(Vector)	x(Vector)
	Matrix	ⓔ x(Matrix)	x(Matrix)	x (Vector)	±x(Matrix)

Figure 3: Permitted operations, and resultant operands.

(ii) The unary operations of Fl are "-" (when it is not immediately preceded by an operand and "@" (absolute value). The

(1) i.e., when they occur as operands. If elements are addressed individually, they may be all integers or all floating point numbers.

latter is inferred by the "| ... |" construction, in which succeeding "|" characters at the same parenthetic 'depth' are taken as left and right members of the construction. Ambiguity must be removed by using ordinary parentheses.

Example: |a - P|x - y|| must be written: |a - P(|x - y|)|

Unary operations, when applied to vectors or matrices, affect each element of the array.

(iii) The relational operations are "=", "<", "≤", "≠", "≮" and "≰". These are used in the construction of predicates.

Relational operations do not apply between vector or matrix operands.

2.2.5 Formulae

Any name or number is a formula of F1. Let 'f' and 'g' stand for formulae. Then if 'B' stands for a binary operation, 'fBg' is a formula, and so are '(f)' and '(g)'. Also, if 'U' stands for a unary operation, then 'Uf' and 'Ug' are formulae. All the formulae of F1 may be constructed in this way.

2.2.6 Predicates

Any two formulae connected by a relational operation constitute a predicate. A predicate assumes the value '1' if it is true, and '0' if it is false, i.e., it is evaluated in terms of its characteristic function. The usual Boolean operations between predicates are admitted and denoted by "and" and "or" (the binary operations, in which "and" takes precedence.)

The special form "A r B s C" is admitted and interpreted as

"A r B and B s C", where A, B and C are formulae and r, s are relational operations.

A special set of predicate or sense lights is recognised in F1, corresponding to the sense switches (lights) on the machine console. These are denoted by $\sigma^1, \sigma^2, \ldots, \sigma^{15}$. When a switch is in the 'on' position, it is in the 'true' state, and has value 1.

2.2.7 Equations

A formula or predicate may be evaluated, subject to the usual rules of computing machinery, provided a value has been assigned to each variable name appearing within it. In a calculation such a 'value-assignment' is made by means of an equation:

$$V = F$$

where "V" is a (possibly subscripted) name, and "F" is a formula.

The "=" sign used here is the same as that in the relational operation: the context distinguishes the use to which it is put. A subset (x) of the names which appear in F may be designated as the parameters of V, which may then be regarded as a 'function' name and appear in formulae in the usual way.

Predicates are introduced in an equation by the special name "if". They occur in conditional equations:

$$V = F^{(1)} \text{ if } C^{(1)}, F^{(2)} \text{ if } C^{(2)}, \ldots, F^{(n)} \text{ if } C^{(n)}, F^{(0)}$$

where "V" is a (possibly subscripted) name, $F^{(i)}$ $(i = 0,1,\ldots n)$ is a formula, and $C^{(i)}(i = 1,2,\ldots,n)$ is a predicate. The equation is scanned from left to right, and V assumes the value of the first

formula for which the corresponding predicate is true. If no predicate is true, then V assumes the value of $F^{(0)}$. The last formula may be omitted if all the preceding predicates are never simultaneously false, of if V has been defined previously, and is to remain unchanged if none of the preceding predicates is true.

Example: $f(x) = 0$ if $x<0$, 1 if $0 \leq x \leq 1$, 0

Besides making a value assignment at execution time, an equation is used during assembly to indicate the type of variable which it defines. The value of F may be determined (during assembly) to be an integer, floating point number, vector or matrix. If V has not occurred previously in assembly, this is sufficient to fix the type of V from this equation onwards. If V has occurred previously, then some assumption regarding its type has been made. If F is not of this type, then an error has occurred; the only exception to this rule is that F may be converted to an integer (from floating point form) or to a floating point number (from integer form) to conform with V.

Examples:

$x = 3gt^2 + 6g/2$	(x is a 'simple' operand)
$G(r) = 32r - 4p_i + 10$	(G is a function)
$K = (1.5,2.5,-7.0)$	(K is a vector)
$M = ((1,1,0),(1,0,1),(0,1,1))$	(M is a matrix)

2.2.8 Declarations

In a program, the name of a variable may occur in an equation before it is formally defined. In this case, a declaration must be made if it is to be treated as anything but a single precision floating point number. The declaration has the form:

D L,M,N...

where D is one of the declaration names 'Integer', 'Function', 'Vector' and 'Matrix', and L,M,N,... are the names of the subjects of the declaration.

The dimension of an array (vector or matrix) may be used as any other integer operand. The special name "dim" is provided for this purpose, and it may be used in two ways, illustrated by the following examples. Let A be the name of a vector.

(i)	dim A = 10	Causes space to be reserved for 10 elements, and a codeword belonging to A, during assembly.
(ii)	dim A = n	Causes code to be generated which will obtain storage space for A (and its codeword) at execution time, depending on the current value of n.
(iii)	m = dim A	Causes the integer m to be assigned the current value of the dimension of A (at execution time).

If B is the name of a matrix, then its dimensions are $dim_1 B$ (no. of rows) and $dim_2 B$ (no. of columns). They may be used in a similar way to (i), (ii) and (iii).

2.2.9 Commands

A command in F1 may be composed of any number of equations separated by commas. The first is the principal equation of the command; on the left hand side of the principal equation is the principal variable, which may be a subscripted name. Following the principal equation are auxiliary equations; on the left hand

side of an auxiliary equation is an auxiliary variable. Auxiliary variables are of two types: simple (non-subscripted) and subscripted. Subscripted auxiliary variables appear on the left hand side of two types of auxiliary equations: recurrence relations and initialization equations. The auxiliary equations need appear in no particular order: they are used to define quantitites appearing on the right hand side of other equations, and must be consistent and free from circularity.

As a program is being assembled, a list of principal variables is formed. In processing a command, the names occurring there may or may not appear on this list. A name which is not on this list, and which does not appear as an auxiliary variable, is added to the list. A name appearing as an auxiliary variable, but not on the list, is defined only within the command. Thus, the same name may be used for different auxiliary variables in any number of different commands. The auxiliary equations are not completely general, and we now illustrate the forms which they may take.

(1) Simple auxiliary equation. The auxiliary variable is not subscripted.

Example: $y = a + b_u c_v/d_w - e_w(f(a) - f(t)) - g, a = 2g, u = t + r,$

$g = (t/2 \text{ if } u < 3,0)$

Note: Here a, u, and g are defined by simple auxiliary equations. The right hand side of the conditional equation must be entirely enclosed in parentheses to distinguish its comma(s) from those appearing between auxiliary equations.

(2) Preceding Values Recursion. The auxiliary variable has a single name as subscript, and the auxiliary equation constitutes

a recurrence relation, the initial values of which are given by auxiliary initialization equations. The subscript is considered a 'dummy' variable. Apart from the recurrence relation and initialization equations, the variable thus defined may appear only once elsewhere in the command, as follows: on the right hand side of another equation of the command, with a subscript consisting of a number or a single variable name.

Example: $y = a + b_u c_v/d_w - e_w(f(a) - f(t) - g,\ b_i = 3b_{i-1} + 4,$
$b_0 = 1,\ c_i = 5c_{i-1}/c_{i-2} - 6r,\ c_0 = 1,\ c_1 = 2$

Note: Here b and c are defined by preceding values recursion. In a recurrence relation of order r, the initial values range from subscript 0 to r-1. Any initial values which are not given are assumed to be zero.

(3) Simultaneous preceding values recursion. This is a generalization of the above case, in which the definition of one subscripted auxiliary variable may involve another auxiliary variable which is also subscripted. Such dependence is restricted only by the fact that a circular definition may not arise. Apart from their recurrence and initialization equations, simultaneously recursive auxiliary variables may appear elsewhere in the command only once, and only with the same numeric or single name subscript.

Example: $y = a + b_u c_v/d_w - e_w(f(a) - f(t)) - g,\ d_i = 4d_{i-1} +$
$e_i e_{i-2},\ d_o = 1,\ e_i = 2d_{i-1}/e_{i-1},\ e_0 = 5$

Note: This command involves a simultaneous recursion on d and e. In execution, initial variables which appear with negative subscripts are taken to have value zero.

During assembly, each command is dealt with as a whole, and

there is a limit to the size of command which can be handled, just as there is a limit to convenient human comprehension of a single recursive definition. An upper limit of six lines for the size of a command should be borne in mind.

(4) Other Forms of Auxiliary Equation. As indicated in section 2.2.7, an equation may be used to define a function name by listing after it the parameters which are used in its definition. These parameter names may not appear anywhere else in the program. By convention, the value of a function defined by an equation is placed in fast register T7 after it is calculated. For this reason, T7 should not be used for storage by the coder.

Example: $y = a + b_u c_v / d_w - e_w(f(a) - f(t)) - g,\ f(x) = 2x + r^2$

Note: This command contains a definition of the function f. Here it is single valued, although the 'value' may be a vector of results, whose codeword is stored in T7. A vector is formed by the "," operation.

Example: $v = A \exp(f_1(x)) + B \exp(f_2(x)),\ f(y) = ((-b + T4)/2a,$
$(-b - T4)/2a),\ T4 = sqr(b^2 - 4ac)$

Note: The term "$f_1(x)$" causes only the first element of f to be calculated. However, an equation such as "G = f(x)" would cause both elements to be calculated and assigned to the vector G. The use of a fast register as an auxiliary variable is illustrated here; by a choice such as this the resultant machine code is slightly more efficient: only T4, T5 and T6 are available for such purposes.

It is evident that assembly does not necessarily take place in the order in which commands are executed. It is the coder's re-

sponsibility to ensure that when a command is executed all the variables which are not defined by auxiliary equations have been previously calculated. An assumption to this effect is in Genie.

2.2.10 Iterative Operators

Repetitive cycles of commands with a parameter change may be introduced in two ways in F1. The first is the 'for' construction which has the general form:

```
for h = p,p ± q,...,r
Q
repeat
```

where h is a (possibly subscripted) name, p,q,r are names or numbers, and Q stands for a sequence of commands.

Example:
```
for x = 2.5, 2.5 + A,...,Xmax
CC    = #K if (sinx)(exp(x/2))< Z5
repeat
```

Note: The use of "#" to obtain the equivalent assigned to K, rather than its contents.

The 'for' loop is coded in a way precisely equivalent to the sequence:

```
        h  = p
J       CC = #I if h ≤ r          (or h < r in the case
        Q                                  "p - q" )
        h  = p + q                (or p - q)
        CC = #J
I       (continue)...
```

Transfers may be made to and from the 'for' loop consistent with this interpretation.

The second form of iteration can occur within a formula, and is initiated by the "Σ" operator. Let F be a formula. Then the construction " $\Sigma_{i=m,n}F$", where m and n are integers or variable names and i is a variable name, is taken to have the value of S obtained by the code:

```
S = 0
for i = m, m + 1,...,n
S = S + F
repeat
```

It will be observed that much inefficient code may be constructed inadvertently by using Σ. In Genie, no attempt is made to devise efficient recurrence relations which would properly be used, for example, in evaluating a polynomial. These should be given by the coder.

Example: $G = u_{10}$, $u_i = xu_{i-1} + A_i$, $u_0 = 0$

The 'range' of a Σ is the same as that of a function name.

Example: $\Sigma_{i=1,n}ab + c$ is coded as $((\Sigma_{i=1,n}(axb)) + c)$

2.2.11 System Subroutines

The six elementary function names given in section 2.2.1 are available at execution time and may be used in formulae without being defined in any other way. They may also be used at assembly time where they occur with a constant argument, which is sometimes convenient for the coder, and does not give rise to inefficient code.

Example: y = π/4 - sqr7.55 would be assembled as

y =-1.962328169409

(assuming 'π' has been given its usual numerical value).

undefined

It is generally true that a command which depends on no/external quantities or parameters is evaluated during assembly, and replaced by a simpler equation.

2.2.12 Subroutines in General

As indicated in Section 2.1, any single valued function of several variables may be used in a formula in the usual algebraic way.[1] Such functions are encoded as closed subroutines, and are terminated logically (as opposed to physically) by the "return" command. By convention, T7 is used to contain the value of the function, although it is within the scope of F1 to compute functions whose values are vectors or matrices. If the function is defined by a program, the last executed command must set T7 to the appropriate value. If the value is a vector, for instance, then a command of the type: "T7 = (a,b,c,d)" would cause a vector to be stored in memory, and the codeword referring to it in T7.

In other cases, a program may be defined and used in a more conventional way. As a trivial example, consider the following definition:

```
P(a,b,c,#d,#e) = SEQ
T4 = b^2 - 4ac
T4 = sqrT4 if T4≠ 0,0
 d = (-b - T4)/2a
 e = (-b + T4)/2a
return
        END
```

When this program is assembled, all quantities appearing in the para-

---

[1] Which permits a formula to appear as an argument, as in $\cos(4x+t_j)$

meter list are addressed through a calling sequence. Consequently, a command of the form: " P(U,t,MG6,#R1,#R2) " would result in the roots of the equation '$Ux^2 + tx + MG6 = 0$' being stored in locations R1 and R2.

Finally, one or more of the fast registers T4, T5, T6 may be used to contain input or output quantities: the coder then has the responsibility of seeing that they are used correctly, since they are not included in the calling sequence which is generated. For another trivial example, if CMPY is the name of a program which treats its two input parameters as the real and imaginary parts of a complex number, and multiplies the number into the complex pair (T4,T5), then a complex multiplication (A,B) X (C,D) → (T4,T5) would be written:

```
T4 = A
T5 = B
     CMPY (C,D)
```

2.3 Input and Output in F1

2.3.1 General Rules

It should be first noted that Genie itself is a type of input routine, the purpose of which is to read in definitions from paper tape, construct machine code from them, and, where possible, to replace them by simpler definitions. In this section, we are concerned with the type of input or output command which can be given during the execution of a program. Secondly, according to the rule given in Section 2.1, a named quantity which is to be input during the execution of a program is undefined during assembly,

and must therefore be listed as such in the program heading.

The most convenient way of referring to data is by name. If an order such as "read x" is given, some equation such as "x = 3.00275" is expected from the paper tape reader. Conversely, a command "print x" will print out an equation defining x. In F1, only numerical information may be handled on the right hand side of an equation.

A more direct way of transmitting data consisting of single numbers is by 'number'. This may be applied to unnamed quantities which are placed in T7 by the coder prior to output, or by the machine after input.

All numbers are converted to and from decimal form, with the convention that floating point numbers contain a decimal point (in decimal form) or a non-zero exponent (in binary form). For input purposes, numbers may be separated from one another by commas, 'tab' or 'cr' punches. Vectors must be enclosed in parentheses, and so must matrices, which are written down (and stored) by row.

Two output forms are permitted: a full precision conversion (#1) in which a number is printed or punched to 13 decimal figures, and a 'half precision' (#2), in which a number is printed or punched to 8 significant decimal figures. More elaborate output formats may be achieved by recourse to the output routines which are used by Genie and admit practically any desirable form, but the control of these is not included in F1: subroutines to use them may be written in machine symbolic code.

2.3.2 Paper Tape Input

Input by equation is initiated by the command:

read L,M,...,N

where L,M,...N are names of undefined quantites which appear in the heading of the main program. Equations defining these must appear in the given order on the paper tape reader. The given names may not be subscripted. They may not include fast register names.

Input by 'number' is initiated by the command:

read

which causes the next number on tape to be read, converted to binary form, and placed in T7.

2.3.3 Paper Tape and Printer Output

Output by equation is initiated by the command:

{punch / print} #n, L,M...N

where n is 1 or 2, and L,M...N are names appearing in the program. This command causes equations to be printed giving the current values of L,M,...N in full (#1) or half (#2) precision. An array A of subscripted elements A_i is permissible as output when referred to as a whole by the unsubscripted name A. They may not include fast register names.

Output by 'number' is initiated by the command:

{punch / print} #n

which causes T7 to be converted to integer or floating point decimal form and printed on a new line, or punched on paper tape, followed by a 'tab' character. The value of n determines whether the number is printed or punched to full or half precision.

3. The Machine Symbolic Code, AP2

3.1 General Structure

It is possible that within F1 the form of commands is inadequate for describing certain operations concisely. In this case, the coder may use machine commands written in a symbolic form. They may be mixed freely with commands of F1, and each symbolic command will be translated into a single machine order. The names used for operands in a command of AP2 will be identified with the names used in F1 commands. It should be remembered that code generated by F1 may use any of the fast registers except T4,T5 and T6 (unless matrix operations are involved); on the other hand, commands are independent and do not make use of fast registers for any purpose but internal storage, except as directed by the coder.

Genie is rather particular about choosing where to put things in general storage, and allows only a few fixed locations which the coder can use for manual control purposes. Otherwise, space is taken as required from a storage 'bank', and automatically returned there after a program has been executed. In particular, the coder must observe the rule of the 'B6 list' whenever using F1 commands or subroutines: that working storage space will always be taken from a block of cells starting with the address given in B6. The contents of B6 are unchanged after the command or subroutine.

3.2 Details of AP2

3.2.1 Names

The Type 1 names are the same as those of F1, with the exception

of "a" and "d" which have special purposes described later. The Type 2 fast register names are the same as in F1, and the Type 3 names (the operation codes and pseudo-orders) are those given in Programming Memorandum #3.

Special Symbols: The special symbols of AP2 are: *,a,d,+,-,→,|,),(,tab, cr, and , (comma). The use of these is given later.

3.2.2 Numbers

Any string of digits not forming part of a name is treated as a number. In AP2 this is treated as an octal integer, unless it is preceded by the special character "d", in which case it is treated as a decimal integer or floating point number, as in F1.

3.2.3 Subscripts

Subscripting is not permitted in AP2. In order to obtain an element of an array which has been defined in F1, it must be indirectly addressed in the following way. Let A be the name of a vector. Then the element A_i is obtained by:

```
SB1     i
CLA     *A
```

Let B be the name of a matrix. Then the element $B_{i,j}$ is obtained by:

```
SB1     i
SB2     j
CLA     *B
```

3.2.4 Operations

Six operation signs are used, and these are interpreted in ways which vary with the field in which they appear. They are "*"

(ADDR), "a"(ADDR),"+"(OPN,ADDR, and AUX), "-"(SETU, OPN, ADDR, and AUX), "→"(AUX), and "|" (SETU and ADDR).

3.2.5 Commands

The reader will recall the subdivision of the 54-bit instruction word in the Rice Institute computer into four distinct fields, and the order in which these are decoded:

(i) A fast register named by SETU (6 bits) is brought to U with appropriate sign modification.

(ii) The ADDR field (27 bits) is decoded, and a second operand brought to S with appropriate sign modifications.

(iii) The machine order designated by the OPN field (15 bits) is applied to U, R and S, and possibly one other memory cell.

(iv) A second operation, designated by AUX (6 bits) is executed.

A command of AP2 represents a single machine order. It may be named, as in F1, for control purposes. The command is typed across the coding sheet starting at the first 'tab' position (SETU), followed by OPN (second 'tab'), and ADDR (third 'tab'). The AUX field is separated from ADDR by a comma. Some remarks may be made after a command, separated from the ADDR, AUX field by a 'tab' punch; these are ignored by Genie.

The acceptable symbols in each field are as follows:

LOCN may be blank, or a symbolic name, not of Type 2 or 3.

SETU may be blank, or a number (which is evaluated modulo 64), or a Type 2 symbol of the A- or B-series. If "f" is such a symbol, then it may appear in the forms "f", "-f", "|f|",

or "-|f|". A blank field is interpreted as "U".

OPN may not be blank. It may be numeric (modulo 32,768), or a Type 3 symbol, or any other symbol whose equivalent has been assigned by the coder by means of the EQU pseudo-order.

ADDR may be blank, or numeric (modulo 32,768), or a combination of symbols and numbers. Any B-series Type 2 symbols are taken to be address modifiers at execution time. Other symbols are replaced by their 15-bit equivalents. If several symbols and numbers appear, their equivalents are combined by the binary "+" and " -" operations. The sign "*" is used to denote indirect addressing. The sign "a" is used to denote immediate addressing. If "m" is an allowed address symbol (or string of symbols), then it may be finally modified to give "m", "-m", "|m|" or "-|m|". An initial "-" sign acts as an inflection on the whole field rather than as the modifier of some symbol.

Examples: "L5 - d18" means 'the contents of cell ((equivalent of L5) minus 22 (octal))'

"-d18 + L5" means 'the negated contents of cell (22 + (the equivalent of L5))'

On certain commands, the bit corresponding to "a" is automatically turned 'on', whether or not it has appeared in the field, thus anticipating the coder's intentions. This happens in the following situations:

(a) when the STO, SLN, SLF, TRA, or HTR orders are used.

(b) when ADDR consists only of one or more B-series symbols.

(c) when a Class 4 OPN is used (shift, set B-registers etc.), and ADDR is purely numeric.

Examples:
```
T7    STO    K + B5
B2    ADD    B6, U + B2
      LLS    d24
```

Any symbols appearing in parentheses in ADDR are ignored.

An alternative form of the ADDR field is simply a floating point decimal number, which is to be used as an operand:

Examples:
```
      CLA    d2.0092071, U + T7
T4    FMP    d102993.454
```

Storage space is reserved for the operand at the end of the program.

AUX may be blank or numeric (modulo 64), or one of the forms: "U+f", "R+f", "I+b", "b + 1", "b - 1" or "b + X", where 'f' stands for any A- or B-series symbol, and 'b' stands for any B-series symbol.

3.2.6 Pseudo-Orders

Pseudo-orders appear in the OPN field. They are not translated into machine orders, but have the following functions:

(i) SEQ...END. These delimit a string of commands, as in F1, and they may be followed by definitions of subroutines, constants, etc.

(ii) BSS (Block started by symbol) and BES (Block ended by symbol). These cause a block of consecutive stores to be reserved in the generated program. The length of the block is given by the

symbol(s) appearing in the ADDR field. The symbol in the LOCN field is assigned an equivalent which is the address of the first (BSS) or last (BES) cell in the block.

Example: GJ4 BSS M + N - d101

(iii) EQU (Equivalence). The Type 1 name in the LOCN field is given the equivalent currently assigned to the symbol(s) in the ADDR field.

(iv) REM (Remarks). All characters and symbols following this pseudo-order are reproduced in the printed program listing without affecting the assembly process. Remarks may be continued at the third 'tab' position on successive lines.

(v) Printer control. PRI (Print assembled code on-line); SUP (Suppress printing); SGL (Single space); DBL (Double space). These apply to the final output listing only. This may also be controlled by sense switches, which can over-ride the pseudo-orders.

(vi) Data Input. In each of the following pseudo-orders a set of numbers or character codes is read into consecutive cells in the generated program. Each number occupies one cell. Nine character codes are stored to one cell.

DEC (Decimal input) Number format is the same as F1, but each number may be followed by a decimal integer scale factor, which is preceded by the character "e": "3.40829e-12" stands for the number '3.40829×10^{-12}'. Integers are evaluated modulo 2^{48}.

OCT (Octal input) Numbers may be up to 18 digits long. An initial "-" sign complements the final number after conversion and shifting. A "b" character followed by a decimal integer 'n' causes the converted number to be shifted n places left ($n>0$) or right ($n<0$) after conversion.

If any number is followed by "t" and a digit (1,2,or3) it is stored with the appropriate tag.

BCD (Binary coded data). All characters which follow are converted to printer codes and stored in successive words, terminated by a 'cr' code.

FLX (Flexowriter codes). All the following characters are read and stored as Flexowriter code (i.e., without conversion). These are also terminated by a 'cr' code.

Note: BCD and FLX data may be continued at the third 'tab' position on successive lines.

****

The example in Figure 1 was taken from A. J. Perlis and K. Samelson, "Report on the Algorithmic Language ALGOL", Num. Math. 1, p.41 (1959).

Genie was devised by Jane Griffin, Ann Heard, J. K. Iliffe, Jo Kathryn Mann and C. McGehee.

****