CC-213

June, 1963

COMPUTATION CENTER
Massachusetts Institute of Technology
Cambridge 39, Massachusetts

TO: All Programmers

FROM: F. J. Corbató, J. Saltzer, N. Barta, T. Hastings

SUBJECT: An Abbreviated Description of the MAD Compiler Language

A. Introduction

The MAD compiler was designed and prepared originally for the IBM 704 computer by B. Arden, B. Galler and R. Graham at the University of Michigan. The compiler has subsequently been converted to the IBM 7090 by the Michigan group and with their cooperation been adapted by the MIT Computation Center staff to work within the FØRTRAN-FAP Monitor System used on many 7090 machines. The MAD language has many of the features found in FØRTRAN II, ALGØL 58, ALGØL 60 and the soon-to-be-introduced FØRTRAN IV. Since MAD has few restrictions and is highly flexible, it is well-suited as an initial computer language. Moreover, once a programmer has mastered one language, general experience has shown that subsequent languages are much easier to learn.

Only an essential subset of the MAD language is offered here but this subset is sufficient for complete, correct programs. The full language specification is given in the MAD Reference Manual (but with the final definition of the language, of course, implicitly contained in the MAD compiler program of approximately 12,000 instructions.)

B. Types of Constants

1. Integer: of magnitude less than 2^{27} and primarily used for counting and integer arithmetic. Integer constants are written in decimal with an optional plus sign if positive or an obligatory minus sign if negative but without a point. Examples: 4, +9, -100

2. Floating-point: of magnitude zero or in the approximate range 10^{-38} to 10^{38} These numbers are primarily used in arithmetic calculations. Floating-point constants may be written with or without exponents. If written without an exponent, the constant contains from one to eight digits and a decimal point, which must be written, but which may appear anywhere in the number.

   If the number is written with an exponent, it must contain from one to eight digits with or without a decimal point, followed by the letter E, followed by the exponent of the power of 10 that multiplies the number.

   Examples: 3.0, 3. ,+.3 -123.45689
   3.0E+9, +97.5E-30, 100E-5, -.9E3

C. Variables

Each variable may be of one of the two modes: integer or floating-point and only takes on values of the corresponding mode. (Strictly speaking there are 5 modes in MAD but the remaining three are unnecessary here.) Variable names may consist of 1 to 6 letters or digits of which the first is alphabetic. A variable is assumed to be of floating-point mode unless declared otherwise. (See section 3e, below, for mode declaration statements.) Examples of variable names: ALPHA, NAME, ANSWER, L$$P3

D. Arrays

Arrays are sets of variables in which each variable as an element of the array is distinguished by particular integer subscripts. The array name conventions used are the same as for variables. For typographical reasons, subscripts are enclosed in parentheses and separated by commas. Each array has a fixed but arbitrary number of dimensions. Subscripts s_i of the ith dimension are integer mode expressions (see Expressions below) which take on values between 1 and d_i where d_i is the fixed maximum subscript value to be used for the ith dimension. Subscript values of zero (or negative values) are not allowed here with the exception of arrays of one subscript (i.e. vectors) where a subscript value of zero is allowed. Examples: ALPHA(J), TABLE2(K+3, M-N+1), TABLE3(5,7)

E. External Functions

Function names follow the same rules as variable names but must have a distinguishing terminal "." added. (In addition, all variables, arrays, and functions must have unique names.) The value of a function can be of any mode. Examples: FRML. FIRSTF. ABCD.

F. Expressions

Arithmetic expressions may be formed using parentheses and the operations: +, -, *, /, .P., and .ABS. for addition, subtraction, multiplication, division, exponentation, and absolute value respectively. The allowed integer or floating-point constituents of arithmetic expressions are variables, subscripted elements of arrays, constants, and functions. In general, all constituents must be of the same mode but with the exception that if integer and floating-point modes are mixed, the expression is evaluated in the integer mode until conversion to the floating mode becomes necessary. For example ((X.P.N)/Y) + Q*.ABS.P-5.

represents the expression

$$\frac{x^n}{y} + q|p| - 5$$

in the usual algebraic notation (where the value will be of the **floating-point** mode because of the **floating-point** five). All operations must be explicitly stated, e.g. (A+B)C is not correct, but (A+B)*C is correct.

The result of any integer mode calculation is an integer. When two integer mode numbers are divided only the integer portion of the result is kept as the quotient; the fractional portion of the result is discarded e.g. 7/3 = 2

Boolean expressions have the values true, or false. They may be formed using parentheses and the operators .NOT., .OR., .AND., .THEN., and .EQV., for "negation", "inclusive or", "and", "implication", and "equivalence", respectively. The allowed constitutents of boolean expressions are relations which have boolean values. Boolean-valued relations are $>, \geq, =, \neq, \leq$, and which for typographical reasons are written as .G., .GE., .E., .NE., .LE. and .L. respectively; these relations may occur between any two arithmatic expressions and mixed modes are allowed. For example

((A.GE.B).AND.(C.NE.D)).OR.E(I+5).G.A/1000.

represents the boolean expression

$$((a \geq b) \wedge (c \neq d)) \vee (e_1 + 5) > (a/1000.))$$

G. Statements

A program consists of a collection of subprograms. Each subprogram written in the MAD language consists of a sequence of MAD statements. These statements, which compile into segments of machine language instructions, fall into 4 categories:

1. Substitution Statements are of the form

   a = b

   where b is an expression and a is a variable. When executed the new value of the variable a (in the appropriate mode) is computed by evaluating the expression b. Thus it is possible to transform from fixed to floating vice-versa. Note that if the variable a appears in the expression b, the old value of a is used in the computation. Examples:

```
Y= ALPHA/(B+C)-BETA+2.0*S
J= J+1
```

2. Control Statements

   a. TRANSFER TO s is a statement which when executed causes the program to transfer to the statement labeled s in its left-margin (cols. 1-10). Statements are labeled in the same way as variables and each label must be distinct from all other statement labels, variables, arrays, or function names. Example:

```
TRANSFER TO ALPHA
```

b. WHENEVER b,s is a simple conditional statement where b is a boolean expression, and s is any executable statement except: END ØF PRØGRAM, another WHENEVER, THRØUGH or a function ENTRY. If the boolean expression is true the statement s will be executed. Otherwise control will pass to the next statement following the conditional. The comma in this statement must be written. Examples:

```
WHENEVER X.L.100, TRANSFER TØ STØP
WHENEVER Y.G.22..ØR.S.E.X1, W=0
```

c. The sequence

```
WHENEVER b0
...
ØR WHENEVER b1
...
ØR WHENEVER b2
...
ØTHERWISE
...
END ØF CØNDITIØNAL
```

is a compound conditional statement where the b_i are boolean expressions and the dots are any sequence of statements. The compound conditional is a pattern of statements which allows the conditional execution of just one of the program segments bracketed by the indicated statements. The segment executed is determined by the first boolean expression b_i which is true. Any or all of the ØR WHENEVER statements or the ØTHERWISE statement may be omitted. After the conditional execution of one program segment, program control automatically transfers to the statement logically following the required END ØF CØNDITIØNAL statement. This compound conditional is distinguished from the simple conditional by the absence of a comma following the boolean expressions.

Examples:

```
WHENEVER X.LE.0
     Y=0.
     Z=1.
ØR WHENEVER X.G.10.
     Y=10.
     Z=100.
OTHERWISE
     Y=X
     Z=10. *X
END ØF CØNDITIØNAL

WHENEVER W.L.10.
     TRANSFER TØ LØØP2
ØR WHENEVER W.L.20.
     TRANSFER TØ LØØP3
END ØF CØNDITIØNAL
```

d. THRØUGH s, FØR v = e_1, e_2, b is an iteration statement which operates as follows:

1. The variable v is set equal to the expression e_1.
2. The boolean expression b is tested. If b is true, program control passes to the statement after the statement labeled s.
3. If b is false, the statements up to and including s (the scope) are executed.
4. v is incremented by the value of the expression e_2; return to step 2.

A THRØUGH statement may appear within the scope of any other THRØUGH, provided that the scope of the nested THRØUGH lies entirely within the scope of the higher level THRØUGH statement. Note that the statement labeled s may not be a declaration statement.

Example:

```
          THRØUGH LØØP, FØR M=1,2,M.G.48
LØØP      . . .
          . . .
          THRØUGH LØØP1, FØR M=J,K-1, M.G.N+1 .OR.K.E.O
LØØP1     . . .
          . . .
          THRØUGH LØØP2, FØR J=1,1,J.G.M
          THRØUGH LØØP2, FØR K=1,1,K.G.J
LØØP2     . . .
          . . .
```

e. EXECUTE name. (argument list) is a statement which generates a calling sequence to the subprogram name. The listed arguments may be expressions. For example EXECUTE HIM. (.5,X+Y,Z.P.5) creates a calling sequence to the subprogram HIM. with the values of the three expressions as arguments. The names of subprograms follow the same rules as for external functions. Example:

```
EXECUTE SØRT.(A)
```

f. CONTINUE is a dummy statement which when executed causes no action. The statement normally has a label and is used to indicate a joining point in a program, to which another statement may transfer. It is sometimes used as the last statement in the scope of a THRØUGH, to indicate to the reader that the end of the scope has been reached. Example:

```
LØØP3     CØNTINUE
```

3. DECLARATION STATEMENTS

These statements only convey information to the MAD compiler and would not appear on a flow diagram of a program.

a. The following monitor control card (fully defined later) and statement:

```
*       MAD
        ...
        END ØF PRØGRAM
```

bracket a "Main" subprogram; upon subsequent loading of a program composed of several subprograms, the program is started at the beginning of the "Main" subprogram.

b. Similarily the monitor control card and statements

```
*       MAD
        EXTERNAL FUNCTION (argument list)
        ENTRY TØ n.
        . . .
        FUNCTIØN RETURN v
        END ØF FUNCTIØN
```

bracket an external function subprogram which depending on how it is programmed may be used in one of two different ways. In the first case the function named n. has a single value, v, and is used in an expression. An example is the function SIN. in

```
Y=Y +3.5 *SIN.(X-.5)
```

in the second case, the single value v is meaningless (e.g. in a "function" to sort a table) and is omitted from the FUNCTIØN RETURN statement which is still required. This type of external function subprogram may only be used by an EXECUTE statement. In either case, the arguments listed (separated by commas) in an EXTERNAL FUNCTIØN statement must all be of the simple form of variable names, unsubscripted array names or function names (without arguments). These names as they occur in the defining function subprogram act as dummies for compilation purposes only. In the use of the function, the arguments used may be expressions of the appropriate mode.

It also should be noted that, depending on the programming of the function definition, the argument list may not only contain input values but also may contain variable and array names which will contain output values of the external function subprogram after it is executed. Example:

```
EXECUTE SORT.(TABLE3,N)
```

could sort the array named TABLE3, which has N elements.

c. Every array used in a program must have adequate space reserved for it. For example, if the array A(J) may be used with J between 1 and 50, 50 locations must be set aside for the array A. For vectors (one-dimensional arrays) this setting aside may be done by the statement:

```
DIMENSIØN v(d)
```

where v is the name of the vector and d is a constant integer equal to the amount of storage needed. Actually, d+1 locations will be reserved, to allow use of the subscript zero. Several dimension statements may be combined, as in the example:

DIMENSIØN A(50), IARRAY(245), Z1(24)

Arrays of 2 or more dimensions require a slightly more complicated specification, including information about the structure of the array. With every multidimensional array is associated a vector (the "auxiliary vector") containing this information. This vector's name appears in the dimension specification of the multidimensional array. For example, if an array A(J,K,L) may have all three subscripts between 1 and 10, 1000 spaces are needed. The dimension statement may appear as follows:

DIMENSIØN A(1000,AV),AV(3)

Note that the auxiliary vector must have space reserved for it, also, thus its name appears twice in the DIMENSIØN statement. The numbers stored in the auxiliary vector must be as follows:

AV(0) = number of dimensions in the array
AV(1) = location of (1,1,...,1) element, usually 1.
AV(2) = maximum size of second subscript.
AV(3) = " " " third "
AV(4) = " " " fourth "
etc.

In the example AV(3) would be the last entry, since the array had only 3 dimensions. The maximum size of the first subscript is not specified in the auxiliary vector. The auxiliary vector may be filled in by substitution statements at run time, or preset with a VECTØR VALUES statement at compile time. (see below)

d. It is often desirable to preset an array with constant values at the compilation time of a program. This may be done by the statement:

VECTØR VALUES v(s)=$c_s, c_{s+1}, c_{s+2}, \ldots$

where v is the array name and the c_i are constants, all of the same mode. If s is zero, the subscript on v may be omitted.

Examples:

VECTØR VALUES FLØ(23) = 15.5, 21.7, 3E2
VECTØR VALUES AV = 3,1,10,10

It is not necessary to provide a DIMENSIØN statement for vectors preset with a VECTØR VALUES statement, if the element with the highest possible subscript has been preset.

A multidimensional array is stored in consecutive registers in memory. Thus it is possible to calculate an equivalent linear subscript using the following formula:

$$r = d_1 + (\ldots(((s_1-1)\cdot d_2 + (s_2-1))\cdot d_3 + \ldots)\cdot d_n + (s_n-1)$$

where the d_i are the ith elements in the array's auxiliary vector (see above) and the s_i are the values of the subscripts.

An example of multidimensional to linear mapping is the presetting of entries in the middle of a multidimensional array. The multidimensional array a may be preset starting with the rth linear element by:

VECTOR VALUES a(r) = $C_{s_1,s_2,\ldots,s_n}$

where r has been calculated from the s_i's and the auxiliary vector with the above formula. Example:

array B(J,K), 1≤J≤12, 1≤K≤6
to preset B(4,3) = 15.5×10^{12}

```
DIMENSION B(72,BV)
VECTOR VALUES BV= 2,1,6
VECTOR VALUES B(21) = 15.5E12
```

The dimension statement reserves 6X12=72 locations for B, and names the auxiliary vector BV. The first VECTOR VALUES presets the auxiliary vector for B: 2 dimensions, B(1,1) is at 1, Max(K) = 6
Since all entries of BV are preset, it is not dimensioned. The second VECTOR VALUES presets B(4,3). The equivalent linear subscript from the formula is

$$r = d_1 + (s_1-1)\cdot d_2 + s_2 - 1 = 1+(4-1)\cdot 6+3-1 = 21$$

e. All variables are assumed to be of the mode unless explicitly declared otherwise. Mode declarations hold for the entire subprogram and are made by the statement

```
INTEGER list
```

where list consists of one or more variable or array names, separated by commas. More than one mode declaration statement of a given type may be used. Example:

```
INTEGER A,B,IHAT
```

4. Input-Output Statements

Although it is possible to specify elaborate formats by which information can be brought in and out of the computer, for simplicity only three specific formats will be offered here. (The allowed formats of the Computation Center version of MAD are the same as those of FORTRAN II.) In all these statements, magnetic tape is used for input-output efficiency on the 7090 computer; an IBM 1401 computer is used to preprocess input tapes from cards and to post-process output tapes to the printed page.

a. To print out one or more lines of output (or read one or more input data cards) of integers which are right-justified in 4 adjacent fields 18 characters wide, one writes either of the statements:

```
PRINT   FØRMAT   INT,   list
```

or

```
READ    FØRMAT   INT,   list
```

where consecutive numbers on the in/out medium correspond to the variables and array elements specified in the list. A list may contain any number of variables, including none. In addition, whenever integers are printed or read by a subprogram there must be somewhere in the subprogram the format description statement:

Example:

```
VECTØR VALUES INT = $(4I18)$

PRINT FØRMAT INT,A,B(J),ARRAY(23),C,D,E
```

b. Similarly, to print (or read) one or more lines of real numbers which are right-justified in 4 adjacent fields 18 characters wide, the statements used are:

```
PRINT   FØRMAT   REAL,   list
```

or

```
READ    FØRMAT   REAL,   list
```

where list has the same meaning as for integers and somewhere in the subprogram there is also the format statement:

```
VECTOR VALUES REAL = $(4E18.8)$
```

c. To print a one line message of arbitrary text one must use the statement pair:

```
PRINT FØRMAT f

VECTØR VALUES f = $(nHh1h2h3...hn)$
```

where n is an integer such that $1 \leq n \leq 72$ and the successive Hollerith characters h_i contain the message. (The first character h_1 must be a blank.) The format vector name, f, may be any unique name such as HOL1, MESS1, etc. Example:

```
PRINT FORMAT END1
VECTØR VALUES END1 = $(20H END ØF COMPUTATION.)$
```

d. Preparation of Source Subprograms

The statements of a subprogram are called a "source subprogram" and are card-punched on IBM cards using the usual Hollerith codes to represent the letters of the alphabet, digits, and special characters. Statements are punched anywhere in the field of columns 12-72 (blanks are ignored with the obvious exception of the text included between $ quote marks in VECTØR VALUES statements.) If a statement will not fit on one card it may be continued to another

card by punching consecutive digits from 1 to 9 in col. 11 of each continuing card. Statement labels are punched anywhere in the field of columns 1-10. If the letter "R" appears in column 11, that card will be ignored by the compiler, although it will be printed on the compilation listing (with the "R" deleted). This feature is used to make explanatory comments to a person reading the program. If a comment is to be continued "R" must appear in column 11 of each successive card.

e. Processing of a Program

Compiling and execution of a program prepared for the MIT Computation Center is usually done under the control of a system called the FØRTRAN-FAP-MAD Monitor System (FMS). Control of the monitor is by means of special control cards which have an asterisk punched in column 1 and key words in the cols. 7-72 of the card.

MAD source subprograms are compiled by the monitor using the MAD compiler program. The compiler examines each subprogram for various syntactical errors and may give one or more "error diagnostics."

When a subprogram does compile (it may still contain logical errors, however!) it is punched off-line as an "object subprogram" in the form of relocatable binary cards. Off-line printed output also is produced giving the machine language instructions of the compiled subprogram as well as other reference details useful in debugging.

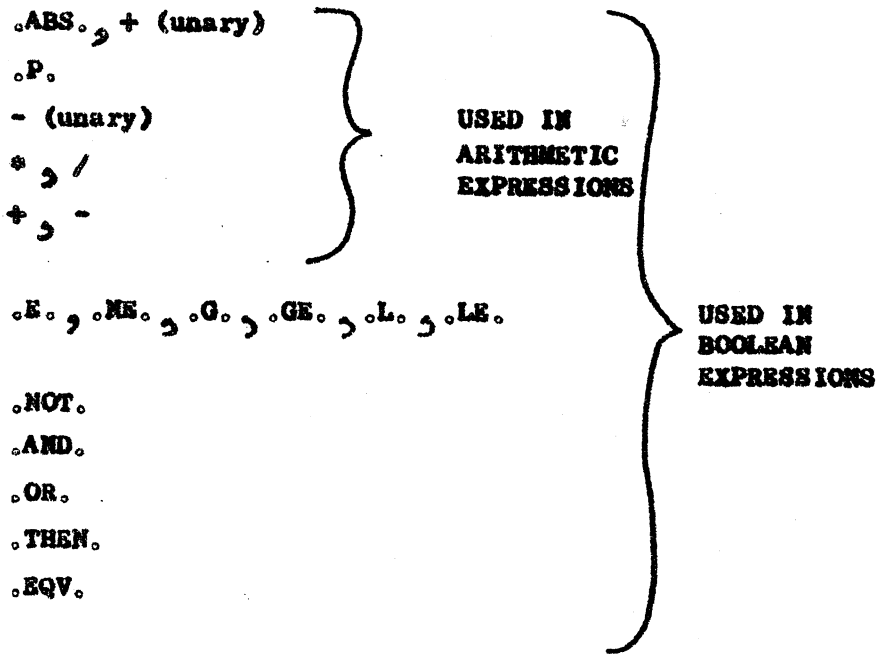
To operate a program in the most elementary manner, the following procedure is used. A run deck is prepared with the following sections in sequence. (See attached example)

1. Identification Card (ID) in a format given by the Center.
2. An FMS control card * XEQ indicating that, if all source programs correctly translate, the program is to be executed.
3. The FMS control card * MAD indicating that the following program is to be compiled by the MAD translator.
4. "Main" subprogram and all other subprograms (in either source or object form) which are explicitly required by EXECUTE statements in these subprograms and which are not on the library tape.
   (The FMS system automatically supplies all needed Library subprograms including those implicitly required by input-output statements such as PRINT, etc., and by external functions such as SQRT, EXP., etc.)
5. The FMS control card * DATA
6. Data cards (if any) to be read from the input tape by the program.

This run deck is written on the off-line input tape (probably along with several other runs) and the FMS system is started. The monitor works on one run at a time, first compiling all the subprograms which are in source language. If there are no compiling diagnostics, the monitor brings in the "BSS loader" which then proceeds to read in each object subprogram, storing them in sequence starting from a lower address of core memory. During loading the loader maintains a storage map giving the location and symbolic name of each subprogram. When the FMS control card "*DATA" is encountered, a second phase of loader processing begins. The loader examines each subprogram to determine what other subprograms are needed by it (listed symbolically in a "transfer vector" at the beginning of each subprogram). If necessary, missing subprograms are obtained from the library tape. Using the completed storage map, the loader converts each symbolic name in a transfer vector to a corresponding transfer instruction to the memory location of the specified subprogram. Thus, the entry links between subprograms are established, and the program is started at the entry point of the "main subprogram.

The MIT version of MAD automatically inserts at an END OF PROGRAM statement the necessary instruction to return to the Monitor system. When the program reaches this point, control returns to the monitor, and it then proceeds to the next job.

LEVEL OF PRECEDENCE OF OPERATORS IN MAD



```
*M2802-9000,FMS,DEBUG,1,1,0,0   JOHN DOE, 18 JUNE, 1963
*           XEQ
*           MAD
           R      JOHN DOE
           R      INTERCHANGE SORT
            VECTOR VALUES INT = $(4I18)$
            DIMENSION KVL (100),KVA(100)
           R      READ IN INPUT
            READ FORMAT INT,N,KVL(1)...KVL(N),
           1                    KVA(1)...KVA(N)
           R      SORT ON KVL
            THROUGH LIST, FOR I=1,1,I.G.N
            KMIN = KVL (I)
            THROUGH SCAN, FOR J=I,1,J.G.N
            WHENEVER KVL(J).LE.KMIN
                 KMIN = KVL(J)
                 IMIN = J
              END OF CONDITIONAL
 SCAN       CONTINUE
            TEMP = KVL(IMIN)
            KVL(IMIN) = KVL(I)
            KVL(I) = TEMP
            TEMP = KVA(IMIN)
            KVA(IMIN) = KVA(I)
 LIST       KVA(I) = TEMP
           R      PRINT RESULTS
            PRINT FORMAT MESS1
            VECTOR VALUES MESS1 = $(7H SORTED)$
            THROUGH PLIST, FOR K=1,1,K.G.N
 PLIST      PRINT FORMAT INT, KVL(K),KVA(K)
            END OF PROGRAM
*           DATA
```