To: MSPM Distribution

From: Art Evans

Subject: MSPM BZ.3.01

Date: February 20, 1968

Attached is MSPM section BZ.3.01 describing the language features of MAD as it will be implemented on Multics. Coming soon will be BZ.3.00, an overview, which will emphasize that

- MAD will be Multics compatible, creating pure procedure and obeying the usual call-save-return conventions. (Thus, MAD-compiled procedures may call those compiled by EPL, and vice versa.) The compiler will honor the usual options, etc.

- Inter-segment referencing will be provided, by changes to the MAD language.

Published: 02/20/68

Identification

The Multics MAD Language
E. I. Ancona

Purpose

The purpose of this document is to describe those features of the MAD language which the Multics translator will eventually compile, those features which will be implemented in the initial version and the changes which will be incorporated into the language.

It should generally be safe to assume that the translator will eventually compile the language described in the August 1966 version of the MAD manual except only where differences are indicated in this document. Nonetheless, it should be realized that this is a preliminary document, and that changes may have to be made. Several statements mentioned here are not on CTSS MAD. They are referred to here for the sake of completeness. It is assumed the reader is familiar with EPL.

I. Deletions from the Language:

Following are the statement types which will not be incorporated in the Multics MAD compiler. Section references are to the August 1966 edition of the MAD manual.

(a) Pause statement (section 2.7)
(b) transmit statement (section 3.10.4)
(c) symbol table statements (section 3.11)
(d) listing on and off declarations (section 3.12)
(e) references on and off declarations (section 3.13)
(f) features described in appendix B, C, D, should not be used.
(g) erasable declaration (section 3.5)
(h) list manipulation statements. These are not necessary since all Multics MAD external functions will be recursive. (section 2.13)
(i) the program common declaration will be replaced by the common declaration--see part III of this document (section 3.6)
(j) there will be no defined operators.

II. Changes in the Language:

The following changes will be implemented in Multics MAD. Numbers in brackets refer to sections in the MAD manual most pertinent to the change:

A. Alphabetic constants:

An alphabetic constant is written as from one to four characters preceeded and followed by the character " (double quote). Admissible characters are all of the characters in the ascii code. Special conventions will be developed to allow the use of " (double quote) as well as non-printing characters. [1.1.3]

B. Names:

Names of variables or labels consist of from one to 31 characters, the first being a letter. The admissible characters are the upper and lower case alphabet, the digits and the character _ (underscore). Note that A and a are different names. Furthermore, a name may contain exactly one dollar sign. In that case, the part before the $ is the segment name and the part after the $ is the entry name. Thus NAME1$NAME2. is entry NAME2, in segment NAME1. Note that the name of a variable or an identifier may not be chosen to be that of a keyword. [1.2, 1.3]

C. Keywords:

A keyword, e.g., dimension or whenever, consists of lower case letters only. The abbreviations described in Appendix A may be used except, of course, that they should be lower case.

D. Interseqment referencing:

a. Defining a function:

Both external and internal functions in Multics MAD will be defined as specified in sections 3.10-3.10.3 of the Manual. However, for external functions only, the restriction that within one one function definition, all functions and procedures defined must use the same set of dummy variables, is relaxed if one obeys the following conventions:

(i) Each entry point defined, including the primary entry point must have a list of dummy variables associated with it. For example, if cos. has (X, Y) as dummy arguments, sin. has W and tan. has (Z, Y), then the definition might be

```
external function
entry to cos. (X, Y)
.
.
.
.
entry to sin.(W)
.
.
.
function return alpha + 4
entry to tan.(Z,Y)
.
.
function return Beta = 9
end of function.
```

(ii) If several entry points wish to use the same dummy argument then it must appear in the same position in the argument list for every entry point.

b. Calling a function:

If a function name does not contain a $, e.g., NAME1. (argument list), the following procedure is used to locate it and generate the proper calling sequence:

(i) Determine whether it is an internal function

(ii) If not, check whether there is an entry point called NAME1. within this segment.

(iii) If not, then the name of the segment being referenced is assumed to be NAME1 and its entry point also NAME1., i.e., it is equivalent to NAME1$NAME1.

c. External data references:

Reference to data in other segments is achieved in the following way:

NAME1$NAME2 (subscripts)

where NAME1 is the name of the segment

NAME2 is the name of the data in NAME1

(subscripts) is the list of subscripts if NAME1$NAME2 is an array.

E. Field specification:

In order to allow a free field format,

a. statement labels must be followed by a colon.

b. a comment must be enclosed by two slashes // and the new line symbol.

c. Statements which do not fit on one line must contain a % as the first character of the next line, indicating that it is a continuation of the previous line. The compiler will in all cases just delete the % and the new line symbol.

d. However, a new statement must always begin on a new line.

III. Additions to the Language:

A. Storage classes:

Three storage classes will be defined:

(1) Common
(2) Automatic
(3) Static

Default storage class for all variables will be automatic. If another default class is desired, it may be effected by a statement such as

normal storage is <class>

where <class> may be common, automatic or static.

Variables whose class should be other than that normally assigned may be so declared by statements of the form

<class> variable1, variable2, variable3

where <class> is the storage class.

a. Common storage:

Common storage is similar to the old program common declaration. Storage for a variable declared common is allocated before execution of the program and is never released during execution. The storage will be in the <common_> segment.

Variables declared common may be referenced by several procedures.

b. Static storage:

Static storage is also allocated before execution and never released during execution. However, variables declared static are local to the procedure declaring them. They will be stored in the linkage segment.

c. Automatic storage:

Automatic storage is allocated when the procedure is called. When the procedure returns, the storage is released. Variables declared automatic will be stored in the stack.

Notes: (1) Automatic ≡ automatic in PL/I

Static ≡ internal static in PL/I

Common ≡ external static in PL/I

(2) Default storage class in the old MAD was internal static.

(3) Putting a $ into a name makes it an external symbol.

B. Pointer data:

An additional mode will be available: pointer. A variable is assigned pointer mode by a declaration of the type

pointer variable1, variable2, variable3

a. If X is declared of pointer mode, then statements such as X = .addr. Var1 will cause X to point to the location of Var1. The .addr. function will be a built-in function.

b. Two operations are permitted with pointers: .E. and .NE.

   Thus if X and Y are pointers, then X.E.Y will check whether X and Y point to the same location.

c. Note that pointers may be passed as arguments.

d. There will be no based storage, in the sense of PL/I.

e. Pointers will not be implemented in the original version.

IV. Input-Output:

All input-output statements will be compiled into calls to appropriate subroutines. In the initial implementation, the following statements will be allowed:

```
read data
read and print data
print results
print comment "<string>"     [2.16]
```

The corresponding subroutines will be specified but not implemented at this time.

V. Following is a list of all keywords in the MAD manual except I/O. Their status for the Multics MAD compiler will be described according to

(a) they will be implemented in the initial version

(b) they might be implemented "later"

(c) they have been deleted from the language.

	Now	Later	Never
transfer to <label>	X		
whenever B, Q	X		
whenever B	X		
or whenever B	X		
end of conditional	X		
otherwise	X		
continue	X		
through <label>, for values of V=E(1), E(2),...,E(m)		X	
through <label>, for V=E(1),E(2),B	X		
pause no.n			X
execute	X		
error return		X	
end of program	X		
function return E	X		
entry to <function_name>	X		
set list to μ, E			X
save data <list>			X
save return			X
restore data <list>			X
restore return			X
(I=E(1),E(2),B,S(1),...,S(n)) iterated		X	
remark (altered)	X		
<mode> <list>	X		
mode number n <list>		X	
dimension	X		
normal mode is <mode>	X		
equivalence		X	
erasable			X
program common (replaced by common)	X		
vector values A(n)=C(0),C(1),...,C(n)	X		
vector values A(m)...A(n)=k		X	
parameter		X	
internal function	X		
external function	X		
end of function	X		
transmit			X
symbol table vector v			X
full symbol table vector			X
listing on			X
listing off			X
references on			X
references off			X

VI. Operations:

All operations described in the Manual (both arithmetic and Boolean) will be available, with the possible exception of .P. which will (ultimately) compile into a math library subroutine.

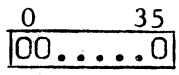
VII. Data representation:

(i) Integers are stored as GE 645 single-word integer quantities. Their values must be $\leq(2(35) - 1)$ in magnitude. Precision $\cong$ 10 decimal digits. [1.1.1]

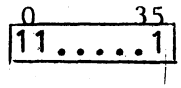
(ii) Floating point numbers are stored as GE 645 single-word floating-point numbers. Precision $\cong$ 8 significant decimal digits. [1.1.2]

(iii) Alphabetic characters are coded in 7 bits, right justified in 9-bit bytes with leading zeros. An alphabetic value will be four characters in length. If a constant written by the programmer is less than four characters long, it will be extended to four characters by adding blanks on the right. [1.1.3]

(iv) Boolean values are stored as GE 645 single-word fixed-point numbers.



[1.1.4]



(v) Octal quantities are stored as GE 645 single-word integer quantities. Each octal number consists of 12 digits. [1.1.5]

(vi) Pointers are stored as its pairs.

VIII. Additional restrictions and comments:

The following are restrictions which should be remembered when reading the manual.

[1.1.6] Modes 5, 6, 7 may neither be used nor defined.

[1.12] Block notation will not be available in the initial implementation.

[2.17] Iterated expressions and symbols may not be used.

[3.3.1] Setdim. will not be available in the initial implementation.