MASSACHUSETTS INSTITUTE OF TECHNOLOGY

PROJECT MAC

Artificial Intelligence
Memo. No. 90
Revised: October 1968

MAC-M-279
October 1965

MIDAS

Peter Samson

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Project MAC

Memorandum MAC-M-268

January 31, 1966

TO: PDP-6 USERS

FROM: Peter Samson

Attached to A.I. Memo. 90 "MIDAS" (MAC-M-279)

SUBJECT: Linking Loader for MIDAS

The MIDAS Linking Loader is a PDP-6 program to load relocatable-format output from the MIDAS assembler, with facilities to handle symbolic cross-references between independently assembled programs. Although it is arranged primarily to load from DECtape, the loader is able also to load paper-tape relocatable programs.

To use the loader, load it off the MACDMP SYSTEM tape as the file STINK . (A file STINK NEW may exist, repairing old bugs or introducing new features.) Then the loader expects commands to be typed in on the on-line Teletype; two successive ALT MODE characters terminate the string of commands. The commands in a string are not performed until the string is thus terminated. While a command string has not been terminated, RUBOUT will erase the last typed-in character (and type it out again as a reminder). A command string may contain any number of commands, and the effect is the same whether the commands are together in one string or are in successively typed-in strings each delimited by two ALT MODEs.

The loader maintains two tables whose contents may change as programs are loaded: (a) the Loader Table, which contains definitions of global symbols and unresolved virtual usages; (b) the local symbol table, containing all program names, and the local symbols for each program for which their loading was requested.

In the following command descriptions, n is an octal number, or one of the command characters said to have a value. (§) is ALT MODE, which echoes out as $. ␣ represents SPACE .

command form	Meaning
P	Set to read from the paper tape reader. (Tape must be in the reader, and the reader must be on when this command is performed.)
nMname1␣name2 (§)	Set to read from beginning of file name1 name2 on DECtape unit n . (If n is omitted, the last DECtape mentioned is assumed.)
N	Load selected input file without local symbols.

Load selected input file, saving local symbol definitions for DDT. (N and L set the Current Starting Address to that specified in the program loaded if that is not Ø.)

Copy all defined global symbols in the Loader Table into the local symbol table (for DDT); then delete same from the Loader Table.

Read in the relocatable version of DDT from DECtape unit 1, tell it of all symbols in the local symbol table; wipe out the loader and transfer control to DDT.

List files of DECtape unit n . (As in the M command, the argument n may be omitted.)

Transfer control to the Current Starting Address.

Set the Current Starting Address to n and transfer control thereto.

Print the value of n as an octal integer.

Has the value of the Current Starting Address.

Has the value of the lowest address currently used by the loader.

Delete all local and global symbols from the local symbol table and the Loader Table.

Zero core except registers 2Ø through 37 and the loader (from E up).

Print contents of location n .

Has the current value of program relocation.

Set program relocation to n .

Set common relocation to n .

Print storage map: Each program in core has one line in the map. At the left is the program name, and at the right in octal is a word whose right half is the first location used by the program and whose left half is the last location used by the program.

?	Print storage map and missing list (short form). Each program loaded appears as follows: one line with the program name at the left and the first address used by the program at the right; any number of lines indented one space, each listing an undefined symbol used in that program, with the address of its first use therein. Symbols are global unless preceded by * meaning local.
n?	Print storage map and missing list (long form). (Here the value of n is immaterial, but an argument must be given.) Like ? with the following changes: (a) following the program name is a 36-bit word in octal with first and last addresses as for the S command; (b) the address is given of each reference to each undefined symbol.
n <sym>	Define symbol sym with the value n . The symbol will be global unless a * is typed somewhere between < and > .

Error message	meaning
SCE adr	Storage capacity exceeded. The program being Gloaded collided with the loader at adr .
UGA adr sym	Undefined global assignment. The global symbol sym was undefined when needed by the loader to perform a parameter assignment or location assignment. The current loading address is adr .
MDG adr sym	Multiply defined global. A defined global appeared to the left of a : when adr , the current location, did not equal the value of the global. It was not redefined.
CKS	Checksum error.
FNF	File not found on DECtape specified.
TMS num	Too much symbols: occurs when loading DDT, and means that DDT + symbols + program exceeds storage available by num registers.
ILM	Illegal memory reference: an error by the loader.

A tape labelled LIBRARY is available, containing various useful subroutines in the file LIBRAR 1 . Up-to-date details are posted in the PDP-6 room. Each program in the library file was assembled with the .LIBRA pseudoinstruction, and so will be loaded only if in the Loader Table is a request for a global symbol defined in that program. Therefore the library file should not be loaded until all programs have been loaded which make reference to the library subroutines.

Command string Examples.

(a1) To load the program APLHA RALPHA from DECtape unit 3, the program SUBR 1 from unit 2, and the program BR from unit 2; then to get a storage map and missing list:

3MALPHA␣RALPHA (S) L2MSUBR␣1 (S) LMSUBR␣2 (S) L? (S)(S)

In this example, L was used for each program to load its local symbols. The N command could have been used in each case instead not to load local symbols.

(a2) Then to go to DTT:

TD (S)(S)

(b) To load the program in the paper-tape reader and transfer to its starting address:

PNG (S)(S)

The L command was not used here because DDT was not requested.

(c) To load PROG REL from DECtape unit 1 and the requested library routines from unit 4:

1MPROG␣REL (S) L4MLIBRAR␣1 (S) N (S)(S)

MIDAS is a PDP-6 assembly program. Its input is a symbolic-language file either of paper tape punched in ASCII code, or a DECtape ASCII-mode file according to the MAC file format (see MAC-M-249). The output of MIDAS is a binary file -- either a paper tape or a SBLK or RELOC-mode file on DECtape. The input language is format-free, meaning that the value or use of a word in the input does not depend on its position in a line or on the page, but instead upon the characters (such as colon, comma, and carriage return) which delimit it.

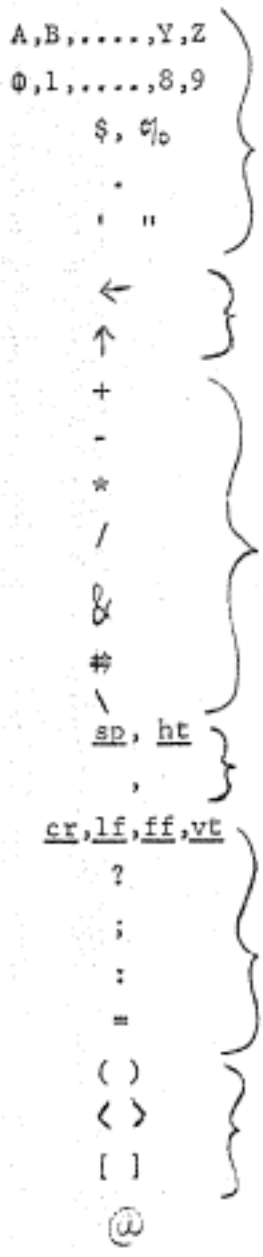
The output of MIDAS consists primarily of storage words: 36-bit quantities intended to be placed in memory locations when the file is loaded. Also in the output file usually a symbol table -- a list of names and values of all symbols defined by the programmer - is included for use with DDT (see DEC-6-Ø-UP-DDT-UM-FP-ACTØØ and A.I. Memo. No. 147, "A Multiple Procedure DDT")

The programmer may select one of several formats for his binary output: one of them, relocatable, allows him to make symbolic cross-references between independently assembled programs.

The MIDAS symbolic language provides a wide variety of ways the programmer may express quantities of interest to him: as PDP-6 machine instructions, integer to any of several radices, floating point numbers, half-words, character strings, etc. In addition it has a variety of features (e.g. macroinstructions and indefinite repeat) for substitution and repitition of character strings comprising input to MIDAS.

The original specifications for MIDAS were undertaken in the winter of 1963-4 by members of the Tech Model Railroad Club, including Messrs. Frazier, Greenblatt, Gross, Holloway, Kotok, Nelson, Eggers and Samson. Since then, most of the work on the assembler has been done by Mr. Greenblatt who is now managing the maintenance and development of MIDAS. The corresponding position regarding the MIDAS linking loader is held by Mr. Holloway.

SYNTAX OF CHARACTERS
Appearing in Value Words



ignored except to separate dummy symbol names

syllable constituents

intra-syllable operators

syllable separators

field separators

the symbol Δ will be used to stand for any single field separator

the symbol ↲ will be used to stand for any one character of the set cr,lf,ff,vt

word terminators

delimit syllable from without;
delimit word within

indirect bit; @ may appear anywhere in a word

SYLLABLES

The elementary unit of meaning in MIDAS is the syllable. Depending upon its form and upon how it was defined, a syllable may be a number, a symbol, a macro name, a pseudoinstruction, a quoted character, or one of several kinds of bracketed word. A pseudoinstruction directs some action on the part of MIDAS; a macro name stands for a string of characters; other syllables (termed valued syllables) represent numeric quantities, which are 36 bits in size. Such a quantity is the value of the syllable. Associated with each valued syllable is a relocation: a quantity which is either Ø or 1 . The value of some syllables may be virtual, meaning the value is not known at assembly time but will become known as the program is loaded. Virtual quantities have a relocation of Ø.

NUMBERS

A string of digits forms an integer with its expressed numeric value and Ø relocation. The number is interpreted in the current radix. An integer ended by ' (single quote) is however taken as in base 8; and an integer ended by . (period) is taken in base 1Ø. An integer may be followed by ↑ followed by an integer:

A ↑ B, where A and B are integers, means the integer

$A * R_A^B$ with R_A being the radix in which A is expressed.

A string of digits with . (period) to the left of some digit is a floating-point number; decimal radix is assumed. A floating-point number may be followed by ↑ integer, and the result is a floating point number: e.g. 3.5 ↑ 4 means the

same as 35ØØØ.Ø . Any of these numeric formats may be followed by ← integer, which multiplies the current value by 2^{INTEGER} . (The exponent integer may be terminated by ' or . to force its radix; otherwise it used the current radix.) The result is a fixed-point integer! For instance, 1.5 ← 3 = 14'; and 17' ← 3 = 17Ø'.

SYMBOLS

A string of syllable constituents, which includes at least one letter, or at least one $ or %, or at least two . (periods), or which consists of the single character . (period) is a name. Only the first six characters of a name are used by the assembler. A name may be defined as a symbol, a macro name, or a pseudoinstruction. A symbol is a valued syllable.

One may write syllable1 ← syllable2 which takes syllable2 as an integer which may not be virtual, and must have Ø relocation; the expression means $\text{syllable1} * 2^{\text{syllable2}}$. If either syllable contains any operators whatsoever it must be a bracketed word. If syllable1 is virtual, or has (1) relocation, the value of syllable2 must be less than $2^{1Ø}$.

If the character ' (single quote) appears to the right of a character in a symbol, it declares that symbol to be a variable. MIDAS forms a list of distinct variables, which become defined, at the first subsequent use of the pseudoinstruction VARIAB or END, as unique memory locations.

If " (double quote) follows any character in a symbol, it declares that symbol global. If a global symbol is NOT defined in the program where it appears its

value is virtual, to be supplied by the loader. If a global symbol is defined in the program, its value is passed on to the loader and there may resolve virtual quantities in other programs.

. AND $.; THE CURRENT LOCATION

MIDAS maintains an 18-bit quantity (with 1-bit relocation) called the current location counter. Its value, is that address into which the next storage word will be assembled. The current location may be set by use of the pseudoinstruction LOC; it is advanced by 1 with each storage word assembled, and by some amount at each use of the pseudoinstructions BLOCK, CONSTA, VARIAB and END. By use of LOC the current location may become virtual.

MIDAS also maintains the offset: a 36-bit-&-relocation quantity. This normally has value Ø, relocation Ø but may be set otherwise by the OFFSET pseudoinstruction.

The symbol . (period) always has the value (and relocation) of Current Location plus Offset. For instance (assuming offset = Ø), in ADD [SUB .] the . refers to the location of the ADD, not that of the SUB.

The symbol $." is virtual. It is the address where the loader is about to put a storage word. So in ADD [SUB $."] (regardless of any offset) the $." has the value of the location where the SUB is to be put.

The virtual symbol $R." stands for the loader relocation, i.e. the amount that the loader adds to the value of each quantity (including the location counter) which has a relocation of 1.

QUOTED CHARACTERS

If '(single quote) or "(double quote) or ↑(up arrow) appears without a syllable constituent immediately to its left, the quote (or arrow) and the character immediately to its right are taken as a syllable with the value of the sixbit, or ASCII code, or ASCII anded with 77°, for the character, respectively. e.g., 'A means 41'; "+ means 53', ↑Q means 21'.

BRACKETED WORDS

<word> is a syllable with the value of word, where word is a word.

(word) If the (is immediately preceded by a syllable-separating character, is a syllable whose value is $word_s$, the value of word with left and right halves swapped.

[word] is a constant syllable, and word in this case is called a constant word. As MIDAS assembles a program it forms a list of distinct constant words, and at the appearance of the pseudoinstruction CONSTA or END a number of locations is reserved equal to the number of constant words outstanding. Each constant word is assembled to appear in a unique one of these locations. Then, each constant syllable which refers to a certain constant word is given the value of the address of that constant word. Last, the table of constant words is reset to contain none.

FIELDS

A valued syllable is a field; and two or more valued syllables may be combined to form a field by means of the syllable separators + - * / \ # and &. These perform respectively 36-bit integer operations of addition, subtraction, multiplication, division (will truncate quotient) and bitwise Boolean OR, XOR, and AND. All &'s are performed first, then all #'s, then \'s, then all *'s and /'s; last, + and - . Operators of the same hierarchy are performed in order from left to right.

WORDS

One or more fields connected by field separators form a word. The two field separators are space (or horizontal tab), and comma. (Space and h.tab are synonymous and will both be called spaces.) Spaces before and after a word are ignored. More than one space in a row are treated as one. Spaces adjacent to a comma are ignored.

The values of the fields are combined to form the value of the word according to the number of fields and the pattern of field separators, as detailed in the following chart.

FORMAT TABLE

(A, B and C stand for fields)

format number in octal	pattern	value
13	,,C	unassigned
14	,A	A & 777777
15	,A ◡ C	unassigned
16	,A,	
17	,A,C	
2Ø	A	A
21	not possible	
22		
23		
24	A ◡ B	A ⊕ B & 777777
25	A ◡ B ◡ C	A ⊕ B & 777777 ⊕ C & 777777
26	A ◡ B,	A + ⟨B & 17⟩ ← 23.
27	A ◡ B,C	A + ⟨B & 17⟩ ← 23. ⊕ C & 777777
3Ø	A,	A
31	not possible	
32	A,,	⟨A & 777777 ⟩ ← 18.
33	A,,C	⟨A & 777777 ⟩ ← 18. + C & 777777
34	A,B	A ⊕ B & 777777
35	A,B ◡ C	unassigned
36	A,B,	
37	A,B,C	

The sign ⊕ means addition with carry suppressed from bit 18 to bit 17.

In the case of four or more fields in a word, the first three are treated according to the chart and all subsequent fields, regardless of separators, are treated like field C.

If in a word appears (word) with the character to the left of the (other than + - * / \ # or &, the swapped value of the word within the parentheses is saved and at the end of the outer word is added into the word being formed.

Anywhere in a word may appear the character @ . The ONLY effect of this is that when the word has been evaluated, the indirect bit, 1 ← 22., is ORed into the value. The @ does not terminate syllables or fields, nor is it taken part of a syllable or field.

Certain symbols form the MIDAS initial symbol table. Any of these which represent PDP-6 machine instructions (except the 8 i-o instructions) may be independently defined by the programmer, in which case both values are available to MIDAS. When the symbol appears in the leftmost field of a word it assumes the initial value; otherwise, the new value.

WORD USES

name:	makes name a symbol with value, relocation and virtuality equal to those of the Current Location.
name=word↵	makes name a symbol with value, relocation and virtuality of word.
name==word↵	same as name=word↵, but also has the effect, when the symbol table is read by DDT, of "half-killing" the symbol name in DDT's symbol table.

word↵ }
word? }

word is taken as a storage word.

(a series of ↵ characters with no intervening word creates no storage words and does not change the Current Location Center.

word;text lf

word is taken as a storage word, and the text from the ; to the first line feed is ignored.

NOTE: If ↵ or ; terminates a word it automatically closes all open ([and < groupings. Hence ADD␣[(3↵ is the same as ADD␣[(3)]↵.

PSEUDOINSTRUCTIONS PART I

Any arguments supplied to the following pseudoinstructions in excess of those needed are ignored.

LOC Δ word↵

sets the Current Location Counter to the right 18 bits of the value of word, with the relocation and virtuality of word.

BLOCK Δ word↵

sets the Current Location Counter to the right 18 bits of the sum of its previous value and the value of word.

END Δ word↵

This marks the end of the program, and the right 18 bits of the value of the word, and its relocation, are saved as the program starting address. The END also acts as VARIAB and CONSTA, in that order, if there are any constant words or variables undefined.

XWORD Δ field1 Δ field2↵ }
XWD Δ field1 Δ field2↵ }

Same as (,field1),field2

EXP Δ word↵

Same as word↵

.OP Δ field1 Δ field2 Δ field3Δ

This is a syllable and so may have syllable-combining operators to its left and at the right of the lsst Δ . The value of the syllable is the result of performing field1 as a PDP-6 instruction with field2 in the

specified accumulator and field3 in the specified memory location. The value of .OP is the resulting contents of the accumulator.

OCTAL — Sets current radix to 8

DECIMAL — Sets current radix to 10.

RADIX △ word ↵ — Sets current radix to value of word.

CONSTA ↵ — Reserves space where it appears for all unassigned constant words.

VARIAB ↵ — Reserves space where it appears for all undefined variables.

NULL ↵ — has no effect. Any number of arguments may be given and will have no effect.

EQUALS △ name1 △ name2 ↵ — name1 is made a synonym of name2 and may be used anywhere instead of name2. This pseudoinstruction is generally used upon macros and pseudoinstructions rather than symbols.

TITLE △ string lf — The string of characters through the first line feed is the title of the program, and is printed out when the TITLE pseudoinstruction is encountered on each pass of the assembly. The program name, which may be used by DDT, is set to the first 6 characters of the first syllable in the title.

OFFSET △ word ↵ — The Offset is set to the value and relocation of word.

SQUOZE △ field, symbol — Is a syllable with value equal to the radix-50 representation of symbol plus ⟨ field & 17 ⟩ ← 30.

ASCII ␣ jtextj — where j is any ASCII character, and text is any string of ASCII characters not containing j: generates one or more storage words, containing each five 7-bit characters of the text, from left to right left-justified in successive words (in order that can be read with ILDB). If j is a syllable, field, or

word-separator other then space, the space after ASCII can be omitted. The entire usage is treated as a syllable: any syllable-combining operators at its left act upon the first storage word it generates, and any syllable-combining operators to its right act upon the last storage word it generates.

ASCIZ␣jtextj — like ASCII, but if number of characters in text is divisible by 5, assembles final additional storage word of Ø.

.ASCII jtextj — like ASCIZ; but also when the character ! is encountered in the text takes the field following it, evaluates that field, creates the character string representing that value in octal digits, and puts that character string into the assembled text storage words instead of the ! and field.

SIXBIT␣jtextj — Same action as ASCII, but puts six sixbit characters to the word.

.FNAM1
.FNAM2 — In time-sharing version only .FNAM1 and .FNAM2 correspond respectively to the sixbit code for the first and second subnames of the source file (see Operating Instructions). For example, if the source file name is RANDOM PROG, .FNAM1 has the same numeric value as SIXBIT /RANDOM/ .

PRINTC␣jtextj — Takes a text argument identical in form to that of ASCII, and prints the text on-line when the pseudoinstruction is met on each pass of the assembly. No storage words are created.

PRINTX␣jtextj — Same as PRINTC, but suppresses type-out of the character ! .

EXPUNG ␣name1 ␣name2 ␣...␣namen — Each name is expunged, i.e., deleted from the symbol table; and effectively forgotten by MIDAS.

.BEGIN␣
.END␣

The code appearing between .BEGIN and its matching .END is termed a program block (not to be confused with a DECtape block, a logical block of assembler output, or a group of locations spaced over by the BLOCK pseudoinstruction). All symbol definitions effected within a program block are "undone" at the .END, and such symbols regain the values they had at the .BEGIN. Program blocks may be nested.

CONI field1, field2␣
CONO field1, field2␣
CONSZ field1, field2␣
CONSO field1, field2␣
DATAI field1, field2␣
DATAO field1, field2␣
BLKI field1, field2␣
BLKO field1, field2␣

The eight PDP-6 i-o instructions are defined as pseudoinstructions to MIDAS. If they appear in the leftmost field of a word, then for that word the evaluation rules for certain word formats changes according to the following table; and the i-o instruction otherwise acts as a symbol with the value shown in the Initial Symbol Table.

format number in octal	pattern	value
26	A ␣ B,	A + <B & 177> ← 26.
27	A ␣ B, C	A + <B & 177> ← 26. ⊕ C & 777777

.FORMA Δ no, fs

Inserts an entry in the format table (i.e., replaces old entry) for format number no (see format table on Page 9). The numeric value of field fs is taken as three 12-bit bytes referring to the (up to) three distinctly handled fields in a word: the left 12 bits refer to the right most field, the middle 12 bits to the field next to the right most (if any) and the right 12 bits to the field 2 from the right and any additonal fields. (If there is only one field in a given format it is the right most regardless of punctuation which may be required after it.)
A 12-bit byte describing a particular field is in turn treated as two 6-bit bytes. The right 6-bit byte specifies a mask and the left 6-bit byte specifies a shift. The mask number (say m) directs that only the right m bits of the field value be taking the shift number (say s) directs that the bits remaining after masking be shifted left s bits. The fields, after this masking and shifting are added to give the value of the word. (Example: the 12-bit specification $27\emptyset4_8$ describes an accumulator field:

<f&17> ← 23.

There are three exceptions to the above procedure. (1) If a field is specified as 00022_8 (right half, not shifted) the carry out of bit 18 is suppressed as the field is added into the word. (2) A virtual quantity may only occur in a field specified 00044_8, 00022_8, 2222_8, 05004_8 or 27004_8. (3) If as a syllable in the left-most field of a word appears any of the eight i-o instructions (DATAO, DATAJ, CONO, CONI, BLKC, BLKJ, CONSZ, CONSO) then any field in that word specified 27004_8 is instead taken as if specified 32111_8 (i-o device field).

.LENGT jtextj

(text argument as for ASCII) this is a syllable with the value of the number of characters in the text.

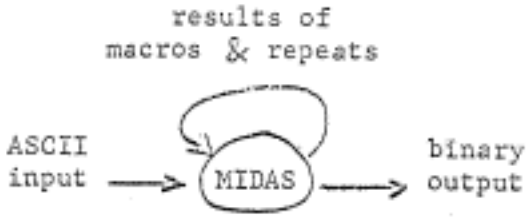
.TYPE name

is a syllable whose value depends on the nature (at that point) of the name name.

value (octal)	if name is a
1	pseudoinstruction or macroname
2	defined symbol (not global, not a variable)
3	undefined local symbol
4	defined local variable
5	undefined local variable
6	defined global variable
7	undefined global variable
10	defined global symbol (not a variable)
11	undefined global symbol (not a variable)
17	unseen (except in .TYPE)

PSEUDOINSTRUCTIONS PART II.

The foregoing sections described those features of the MIDAS language which (generally) handle numeric values; these features should be sufficient to the casual user, and adequate for a majority of MIDAS-language programs. The following sections describe the character-string-handling feature of MIDAS, primarily macroinstructions, REPEAT and IRP (indefinite repeat). MIDAS takes each usage of such pseudo- or macroinstructions and translates that string of characters into a new character string (according to the rules below) which is fed back as input to MIDAS. (See sketch)



REPEAT Δ field Δ text cr

(where text is any character string whose first character is not [and which contains no cr) MIDAS detects this pseudoinstruction, determines the numeric value of field (as an integer), say n; and passes text back as input a total of n times, followed each time by cr-lf.

REPEAT Δ field Δ [text]

Finds n, the value of field; then passes text to the input n times. The text may contain any characters including [and]; the] which delimits the text is the first that matches the opening [. No extra characters are passed on. The (outermost) brackets surrounding text are not passed on.

.RPCNT — This is a symbol whose value depends on the innermost repeat in which it is contained: the first time that repeat is processed, .RPCNT has the value ∅; the second time, it has the value 1; etc.

The following pseudoinstructions, whose names begin IF, are termed conditional assembly pseudoinstructions.

IF1 Δ textcr
IF1 Δ [text] — (take text argument like that of REPEAT.) The text is passed back as input if the assembler is performing Pass 1. Otherwise this usage gives no characters out.

IF2 Δ textcr
IF2 Δ [text] — Like IF1, but gives text back to input only if performing Pass 2.

IFE Δ field Δ textcr
IFE Δ field Δ [text] — If the numeric value of field is equal to ∅, the text is reprocessed as input; otherwise not.

IFG
IFGE
IFN
IFL
IFLE — Identical to IFE in form; pass text back again according as the value of their first argument is: greater than; greater than or equal to; not equal to; less than; less than or equal to: ∅, respectively.

IFSE Δ string1 Δ string2 Δ textcr
IFSE Δ string1 Δ string2 Δ [text] — (text argument as in REPEAT.) The strings are compared character-for-character. Each string may contain spaces and commas if its first character is open bracket ([), in which case that string is ended by the matching close bracket (]). Such outermost brackets, if they exist, are not used in the string comparison. The text is then processed only if the two fields are identical character strings.

IFSN — Like IFSE, but passes text on only if the two fields are not identical strings.

IRP Δ A, B, [P, Q, R]↵ — The IRP pseudoinstruction takes a triplet of arguments, separated by commas: the first two are names, called dummy symbols; the third is termed a list. Each of the dummy symbol names need not be given. The list consists of an open bracket, any number of elements

separated by commas, and a closed bracket. Each element is a string of zero or more characters and lists. The only character excluded is comma; and that may appear if in a list, i.e. inside [ ]. For instance: IRP N1, ,[AX,BY,[C,D,E],,] has N1 its first dummy symbol, has no second dummy symbol, and has a list of five elements: AX; BY; [C,D,E]; and two null arguments.

TERMIN

The range of an IRP is the string of characters from that IRP to the next matching TERMIN, exclusive. The effect of IRP-TERMIN is to pass the range of the IRP on as input again a number of times equal to the number of elements in the list of the IRP. The first time through the range, the first dummy symbol of the IRP has the string value of the first element of the list, meaning that: appearance of that dummy symbol inside the range (terminated at each end by ! or any non-syllable-constituent) is replaced (in effect) by the current string value of the dummy symbol. The second time through the range, the first dummy symbol has the string value of the second element of the list, etc.

Example:
```
IRP  A,,[W1,W2,W4]
ADD  3, A
TERMIN
```

will recirculate the string

```
ADD 3, W1
ADD 3, W2
ADD 3, W4
```

The second dummy symbol of IRP has, on each iteration through the range, the string value of the remainder of the list yet unused: for instance, on the third iteration of IRP Q,R,[AND,IOR,XOR,EQV, ANDCA] for R would be substituted EQV, ANDCA.

IRP A,B,[X,Y,Z]C,D,[P,Q,R]E,F,[S,T,U] ...

The IRP pseudoinstruction may be followed by any number of triplets. At each iteration,

all the dummy symbols are advanced in their respective lists. The IRP iterates a number of times equal to the number of elements in the longest list; if one list is exhausted before another, both its dummy symbols receive null string values.

IRPC Δ A, B, [string] ↵

Here A and B are dummy symbols, and string is a string of characters ended by the first matching]. On each iteration of the IRPC the first dummy symbol takes the string value of each successive character in string, and the second dummy symbol has the string value of the string to the right of that character. In all other respects IRPC is used identically to IRP.

IRPS Δ A,B,[syl1 ▽ syl2 ▽... ▽ syln]

Again A and B are dummy symbols; ▽ is any field or syllable separator. On each iteration the first dummy symbol has the strin value of successively each of the sylᵢ, and th second dummy symbol has the string value of th ▽ to the right of the syllable which is the current value of the first dummy symbol. Othe wise IRPS is like IRP and IRPC.

MACROINSTRUCTIONS

A macroinstruction definition is a syntactic element consisting of: a define line; a definition body; and the pseudoinstruction TERMIN.

A define line contains in order: the pseudoinstruction DEFINE; a name, which is to be defined, called the macro name; and zero or more names which are dummy symbols; all separated by spaces or commas, except that a dummy symbol may be preceded or terminated by / or \ instead; the whole extending through the first ↵ character, but if that character is cr and the next character is lf, the lf also is taken as part of the define line.

The definition body is all text following the define line, up to but not including the first unmatched use of TERMIN. (The TERMIN pseudoinstruction is matched by IRP, IRPC, IRPS and DEFINE.)

The effect of a macroinstruction definition is that when its TERMIN is encountered the macro name becomes defined as a macroinstruction with the string value of the definition body.

As this happens, the characters to the left and right of each dummy symbol occurrence in the body are examined: each one which is an exclamation point is deleted and its place taken by a mark which is invisible; except that it deliminates the dummy on that side when dummy symbols are looked for in (II) below.

Subsequent to its definition, if a macro name is encountered as a syllable (i.e. not part of any text argument nor in a macro definition) the following steps are performed:

(I) the characters following the separator which ends the macro name are read as a list of arguments, separated by comma and the list ended by carriage return or semicolon. This process, termed the argument scan, associates in order each dummy symbol in the define line of the macroinstruction with the corresponding argument in the argument list. If in the define line a dummy symbol is ended by /, the scan for its argument is not terminated by

comma (which in this case is treated simply as another character in the argument), but by any ↵ character; if there are dummy symbols to the right of the /, the scan assumes that their arguments are not expressed. The number of arguments read will in any case not exceed the number of dummy symbols in the macro definition. Furthermore, if the first character where an argument is expected is [(open bracket), that argument will consist of all characters following the [, up to the matching]; the next argument (if any) should follow the] immediately without an intervening comma. (Right bracket is matched only by left bracket.) Neither outermost bracket is passed on as part of the argument. Also if the first character where an argument is expected is \ (backslash), the subsequent characters up to but not including the first space or comma are interpreted as a field, and for the argument is supplied a string of digits expressing the value of the word (as an integer) in the Current Radix;

the next argument immediately follows the space or comma which ended the field. If fewer arguments are expressed than there are dummy symbols, the arguments given are associated with the leftmost dummy symbols, and the unsatisfied dummy symbols are given string values according to the following rules:

1) If there is no / or \ in the list of dummy symbols in the define line, all unsatisfied dummy symbols get the null string value.

2) If there is one or more / or \ in the list of dummy symbols in the define line, all unsatisfied dummy symbols which appear to the left of the first / or \ get the null string value, and all unsatisfied dummy symbols to the right of the first / or \ are given a string value of the form G@@@@1 or G@@@@2, etc. (called a <u>generated symbol</u>; a unique generated symbol being assigned to each such dummy symbol of each use of any such macroinstruction.

(II) The effect to the assembler is that the macro name and argument list are replaced by the body of the definition of that macroinstruction, with each dummy symbol in the body receiving the string value of the corresponding argument.

For example, if MAC is defined as follows

```
DEFINE ␣MAC ␣E,F↵
→| MOVEI␣ E,3 ↵
→| IMULB␣ E,F ↵
TERMIN
```

then MAC 1 , LOC will be seen by the assembler as

```
→| MOVEI ␣1,3↵
→| IMULB ␣1,LOC↵
```

If the argument scan is terminated by the presence of a semicolon, that semicolon (as well as what follows it) is passed on to the assembler after the macroinstruction body.

The character ! may be used to delimit dummy symbols when the effect of any other syllable separator is not wanted. The definition

```
DEFINE   HAC  M,E,F↵
 →| MOVEI ␣E,3 ↵
 →| IMUL!M ␣E,F↵
TERMIN
```

enables one to then say HAC ␣B, 1 ,LOC to generate

```
→| MOVEI ␣1 ,3↵
 →| IMULB ␣1 ,LOC ↵
```

or HAC ␣ , 1 ,LOC to give

```
→|MOVEI ␣ 1 ,3
→| IMUL ␣ 1 ,LOC
```

since the first argument, which extends from the space ending HAC to the next comma, is null. Notice that only the single character just to the right of the macro name is lost; HAC␣␣,1,LOC has a first argument consisting of one space.

A common use of \ preceding an argument is as follows:

```
DEFINE␣DISPATCH'N,TAB
    JRST␣TAB!N
TERMIN
..=1
REPEAT 3Ø,[DISPATCH \.RPCNT+1,DT
]
JRST DT1
JRST DT2
  :
  :
JRST DTØ
```

producing

The argument brackets [] are often used to pass on argument lists to a macro or IRP within the definition of the macro being called:

```
DEFINE   MAPLIST  FN,L
    IRP     A,,[L]
    MAPONE    FN,A
    TERMIN
TERMIN
```

```
DEFINE MAPONE F,AL
    F␣AL
TERMIN
```

```
MAPLIST  MAC,[[1,LOC],[1,LOK],[2,LOG]]
  ⇩
MAPONE  MAC,[1,LOC]
MAPONE  MAC,[1,LOK]
MAPONE  MAC,[2,LOG]
  ⇩
  MAC  1,LOC
  MAC  1,LOK
  MAC  2,LOG
```

The pseudoinstruction .QUOTE takes the symbol following and passes it on unchanged, protecting it from being taken as a dummy symbol or as either of the pseudoinstructions DEFINE and TERMIN. In all other cases, a name which is currently a dummy symbol will appear to be replaced by its string value whenever it appears bounded on each side by non-syllable constituents. For example, if A is an accumulator as well as a dummy symbol within a macro definition, one might need to say

ADD ␣ .QUOTE␣A,A

to differentiate the two meanings.

.TAG␣name▽
.GO ␣ name▽

may be used within the range of IRP, IRPS or IRPC; or in the body of a macro definition. Only the first 6 characters of the name are used. The .TAG has no effect except to mark a place in the IRP etc. or macro which may be referred to by a .GO . As one of these repeats or a macro is being processed, sending characters to the assembler, when a .GO is encountered it is not output nor does outputting resume right after the .GO ; instead, the range of the repeat (or body of the macro) in which the .GO occurred is searched for a .TAG with the matching name. If it is found, outputting to the assembler resumes to the right of the tag. If it is not found, processing of the current repeat or macro is terminated and processing returns to the point just beyond the range of the repeat or just after the macro usage; if this is within an outer repeat or macro body, searching for the missing tag is done therein, etc., if a tag is not found anywhere until processing reaches the top level (outside all repeats and macro calls), assembling then resumes from the point just after the outermost repeat or macro.

.GSSET Δ no↵ Sets generated-symbol counter to value of no.

PSEUDOINSTRUCTIONS PART III

Binary Output Formats.

Storage words are output in one of two modes: absolute or relocatable. There are three possible formats for absolute output, named Read-in mode, Half-word read-in mode, and simple block formats; there is one relocatable format. The Read-in mode format, if punched on paper tape, can be read by the RIM Loader (at 2ø); the simple block format, if on DECtape, can be read by MACDMP, or HACTRN (the DDT in time sharing) and on paper tape is preceded by a SBLK loader in RIM format; the relocatable format can be read by the linking loader. Virtual quantities, and strorage words with non-zero relocation, may not occur in absolute mode output.

RIM↵	selects read-in mode format output
RIM1↵	selects half-word read-in mode format output
SBLK↵	selects simple block format output
RELOCA↵	selects relocatable format output and sets current location counter to ø with relocation 1.
1PASS↵	like RELOCA, but also causes a 1-pass assembly.

SBLK is initially selected.

.SLDR	causes output of SBLK loader and enters SBLK format
WORD Δ wrd↵	The value of wrd is output as a 36-bit word. In relocatable and simple block formats, which output storage words in blocks, any block currently being accumulated is immediately output so that wrd appears in the proper place and between blocks.

NOSYMS	Suppresses output of symbol table which normally accompanies binary program.
.NSTGW	enables SWD error printout to occur if a storage word is generated.
.YSTGW	disables SWD error printout. .YSTGW is initially selected.

Loader Commands.

In relocatable format, certain pseudoinstructions are available to cause special action by the Linking Loader when encountered.

.LIBRA ␣ namelist ↵	This must occur before any storage words. It tells the loader that the program to follow is a library program. The entries in the namelist, separated by commas, are either names or groups of names separated by space, +, and -. An entry is said to be satisfied by a list of symbols if either (a) the entry is a name and that name appears in the list of symbols; or (b) the entry is a group of names, all of which names preceded by space or + are in the list of symbols, and none of which names preceded by - are. The Linking Loader will omit to load a library program unless one or more entries in its namelist are satisfied by the list of undefined global symbols used in programs already loaded.
.LIFS ␣ namelist ↵	Load If Seen.
.ELDC ↵	End Load Time Conditional. Any storage words appearing between .LIFS and .ELDC are passed on to the loader which will load them only if one or more entries in the namelist are satisfied by the list of defined and undefined global symbols used in programs already loaded. Note: .ELDC should always be followed by a location assignment (LOC pseudoinstruction). Sometimes if there is a series of load-time conditionals with no intervening non-conditional matter the LOC need only be used after the last. Notice that LOC $." may be used to continue loading storage words into subsequent locations where the effect of the load-time conditionals is not necessarily known. Load Time Conditionals may be nested.

.LIFE △ word ↵
.LIFLE △ word ↵
.LIFL △ word ↵
.LIFG △ word ↵
.LIFGE △ word ↵
.LIFN △ word ↵

Load Time Conditionals on Value. Any one of these starts a range which is ended by .ELDC. In each case the word is evaluated by the Loader, and the storage words in the range are loaded only if the value of word is respectively $= \emptyset$, $\leq \emptyset$, $< \emptyset$, $> \emptyset$, $\geq \emptyset$, $\neq \emptyset$.

.LNKOT

if output is relocatable format, causes immediate output of all accumulated linking pointers.

.LOP△f1△f2△f3↵

like .OP, but occurs when encountered by loader in relocatable format. (f1 is an instruction, f2 is taken as the contents of the specified accumulator, and f3 as the contents of the specified memory location). Has no value; loader sets variable .LVAL1 to resultant accumulator content, and .LVAL2 to resultant memory contents. (Initially, .LVAL1 and .LVAL2 have the value $\emptyset$).

.GLOBA△nam1 △nam2...↵

has same effect as appearance of each name followed by a ", but generates no storage words.

.LIBRQ△nam1 △nam2...↵

will output the given names to the relocatable loader such that the loader "sees" them in this program when queried by load-time conditionals. No definition is given to the loader, and a use of .LIBRE does not cause the names to be seen at all by the assembler.

ERROR MESSAGES

An error message typed out by MIDAS is in the following form:

SYM+NN MMM L XXX CCC

SYM+NN is the current location in terms of symbols; MMM is the current location in octal; L is the depth in macro-instructions; XXX is the offending syllable (if any); CCC is a 3-character error code, as listed below. A dagger † in the following listing marks a fatal error, which MIDAS will not proceed past.

[All undefined symbol errors treat the symbol as having value Ø, relocation Ø]

USW	undefined symbol in a storage word
USC	undefined symbol in a constant word
USP	undefined symbol in a parameter assignment (to right of =)
USR	undefined symbol in count field of REPEAT
UCD	undefined symbol in test field of conditional assembly
USE	undefined symbol in END pseudoinstruction
USL	undefined symbol in LOC pseudoinstruction
USB	undefined symbol in BLOCK pseudoinstruction
USS	undefined symbol in numeric field of SQUOZE pseudoinstruction
USM	undefined symbol in field following \ in macroinstruction argument
USO	undefined symbol in OFFSET pseudoinstruction
MDT	multiply defined tag. A variable, defined symbol or macroinstruction appeared to the left of : and was not redefined.
MDV	multiply defined variable. A defined symbol or macroinstruction appeared followed by ' , and was NOT redefined.
RES	reserved. An attempt was made to redefine a pseudoinstruction. It was NOT redefined.

ILT	illegal tag. Something other than a symbol appeared to the left of : and no action was taken.
IPA	illegal parameter assignment. Something other than a symbol appeared to the left of = and no action was taken.
QPA	questionable parameter assignment. A macro name or pseudoinstruction appeared to the left of = and WAS redefined.
IEQ	illegal EQUALS. One of the arguments was not a name, no action was taken.
MGT	multiply defined tag at virtual location.
N6B	not sixbit. A non-printing character appeared in the text argument of SIXBIT or to the right of ' .
EPO	exponent overflow. Floating point number too large.
XSG	lost significance. Fixed point number too large.
UFM	unused format of fields in a word. The word is taken as ∅.
SWD	(turned on by .NSTGW) storage word generated.
NOS	no separator. A(B)C or ASCII /X/3, for instance.
ILC	illegal closing. Attempt to match (with] or [with)
ILA	virtual quantity used in absolute assembly
IRA	non-zero relocation used in absolute assembly
IRL	illegal relocation. A field with relocation 1 was put other than with its right end between bits 17 and 18 or 35 and 36.

IRC	illegal relocation count. A field had a relocation other than +1 or Ø.
IGS	illegal virtual. A virtual quantity was put other than with its right end between bits 12 and 13, 17 and 18, 3Ø and 31, or 35 and 36.
IMY	illegal multiplication of virtual quantities
IDV	division by virtual quantity
PNI	. used when location indefinite, Ø assumed
CLI	: used at indefinite location
LAI	location assignment contains illegal relocation
†TCA	too many constant word areas
†TVA	too many variable areas
†TMC	too many constant words in one area
†CLD	constant word area location differs from previous pass
†VLD	variable area location differs from previous pass
†CRD	relocation of constant word area differs from previous pass
†VRD	relocation of variable area differs from previous pass
CRI	relocation of constant word area illegal
†TMA	too many dummy symbols in macro definition
†TMD	too much arguments (strings too long) at use of macroinstruction
†MCE	macroinstruction capacity exceeded
†SCE	storage capacity exceeded
ILF	illegal format. Programmer's intent unclear.
†IAE	internal assembler error
†ILM	assembler made illegal memory reference

OPERATING INSTRUCTIONS NON TIME-SHARING

With MACDMP, load from the MACDMP SYSTEM tape the dump-mode file MIDAS. (There may exist also a file MIDAS NEW. It may have new features, or fix errors in old. Its use for assemblies instead of the MIDAS file would be appreciated as an *in vivo* check-out of the new version.) When MIDAS is started, it will accept a string of commands from the on-line Teletype, ended by two successive ALTMODE characters. The effect is the same whether all the commands are in one string or they are each in a separate string delimited by ALT MODES. (Ⓢ will be used as a symbol for ALT MODE.) In normal use, one should give commands in the following order: (1) select an input file; (2) select an output medium; (3) direct the assembly; (4) file the output (numbers (1) and (2) may be interchanged).

(1) Input Selection: the command nERname1 ␣ name2 Ⓢ will select for input the file named *name1* *name2* or the DECtape on unit *n*. If the ER command is not given, input is selected from the paper tape reader or on-line teletype. (The reader is checked initially, and after each character is typed in.)

(2) Output Selection: the command mEI will select for output on the DECtape on unit *m*. If the EI command is not given, output will be on paper tape.

(3) Assembly: the command A causes the program to be assembled. This is the command normally used; it is composed of five parts which can be commanded separately. In order, these are

I initialize. Forget all symbols, macros, etc. except those in the Initial Symbol Table; set the Current Radix to 8; set the Current Location Counter to 1ØØ with relocation Ø; select SBLK output format; clear all input and output buffers.

O pass one. Read the input file and assemble it, defining symbols and macros, developing values of storage words, etc. No output is done unless the ↑PASS pseudoinstruction was used in the program.

S punch SBLK loader if SBLK is output format.

T pass two. If ↑PASS was not used, read the input file and assemble it, outputting storage words (and in the relocatable format, various data concerning virtual usages) to the output device previously selected. If PASS↑ was used, do none of this but instead check if any symbols were used which are still undefined.

L symbols. Output a symbol table (unless NOSYMS was used).

(4) Output Filing: the command EF name1 ␣ name2 Ⓢ will file DECtape output in SBLK mode for absolute-mode output or RELOC mode for relocatable mode output. There is no need for this if output was to paper tape, or if the assembled program is not going to be used (for instance, due to a large number of assembly errors).

When one program has been assembled, another may be without reloading MIDAS. To leave MIDAS and return to MACDMP, use the command D.

There are also several characters which may be typed in and take effect immediately, without being part of command strings. These are various letters with the CTRL key hold:

\A		type out all input characters processed by MIDAS. This includes expansions of macros, IRP, etc.
\Z		turn off typeout.
\C	(bell)	Quit current activities and wait for typed-in commands. It is advisable to go to MACDMP and load a fresh MIDAS if you have Quit.
\E		Turn on line printer output.
\B		Turn off line printer output.
\R		Commence output to paper tape.
\I		Cease paper-tape output.

\W	Commence DECtape output.
\V	Cease output to DECtape.
\Q	resume reading from DECtape.
\S	interrupt reading DECtape.

OPERATING INSTRUCTIONS TIME SHARING

To DDT type MIDAS \H (alternatively, any other commands which will load and start the program TS MIDAS on device SYS). Into MIDAS type a command line comprised of (a) the destination file designation; (b) the character ← (shift 0); (c) the source file designation; (d) a carriage return. A file designation consists of: a) (optional) a "system name" ended by ; (semicolon). If no system is specified, the name the user gave as he logged in is assumed. b) a device name ended by : (colon). c) the file name, consisting of two subnames separated by a space. If a system name or device name is not specified in the destination description, the name used in the source description will be taken. If no file name is given in the destination description, one will be assumed whose first subname is the first subname of the source file, and whose second subname is BIN. When the carriage return is typed, MIDAS will read the designated source file, and output the assembled result and file it accordingly to the destination file designation. The title (on each pass), PRINTX output, etc. and error messages will appear on the user's console. When finished, MIDAS will do a Value Return to DDT.

Radix-5Ø SQUOZE Code

This is the form in which symbol names are represented internally and in symbol table output.

Character	Value(octal)	Character	Value(octal)
null	Ø	J	24
Ø	1	K	25
1	2	L	26
2	3	M	27
3	4	N	3Ø
4	5	O	31
5	6	P	32
6	7	Q	33
7	1Ø	R	34
8	11	S	35
9	12	T	36
A	13	U	37
B	14	V	4Ø
C	15	W	41
D	16	X	42
E	17	Y	43
F	2Ø	Z	44
G	21	.	45
H	22	$	46
I	23	°/o	47

Say the name is ABCDEF; the Squoze code is

$$2\emptyset + 5\emptyset * <17 + 5\emptyset * <16 + 5\emptyset * <15 + 5\emptyset * <14 + 5\emptyset * 13>>>>>,$$

i.e. the last character is least significant in the radix-5Ø (octal) representation. If the name has fewer than 6 characters, the last characters are taken as null.

The radix-5Ø scheme leaves 4 free bits at the high-order end of a 36-bit word; these are used as Code Bits to describe the type and relocation of the symbol.

Initial Symbol Table

Symbol	Value (octal)	Symbol	Value (octal)
1PASS	pseudo	AOJ	(340000)
ADD	(270000)	AOJA	(344000)
ADDB	(273000)	AOJE	(342000)
ADDI	(271000)	AOJG	(347000)
ADDM	(272000)	AOJGE	(345000)
AND	(404000)	AOJL	(341000)
ANDB	(407000)	AOS	(350000)
ANDCA	(410000)	AOSA	(354000)
ANDCAB	(413000)	AOSE	(352000)
ANDCAI	(411000)	AOSG	(357000)
ANDCAM	(412000)	AOSGE	(355000)
ANDCB	(440000)	AOSL	(351000)
ANDCBB	(443000)	AOSLE	(353000)
ANDCBI	(441000)	AOSN	(356000)
ANDCBM	(442000)	APR	0
ANDCM	(420000)	ASCII	pseudo
		ASCIZ	pseudo
ANDCMB	(423000)	ASH	(240000)
ANDCMI	(421000)	ASHC	(244000)
ANDCMM	(422000)	BLKI	(700000) pseudo
ANDI	(405000)	AOJLE	(343000)
ANDM	(406000)	AOJN	(346000)
AOBJN	(253000)	BLKO	(700101000) pseudo
		BLOCK	pseudo
AOBJP	(252000)	BLT	(251000)
CAI	(300000)	CONSZ	(700300) pseudo
CAIA	(304000)	CR	114
CAIE	(302000)	DATAI	(700040) pseudo
CAIG	(307000)	DATAO	(700140) pseudo
CAIGE	(305000)	DC	200
CAIL	(301000)	DCSA	300
CAILE	(303000)	DCSB	304
		DECIMA	pseudo
		DEFINE	pseudo
CAIN	(306000)	DIS	130
CAM	(310000)	DIV	(234000)
		DIVB	(237000)
		DIVI	(235000)
CAMA	(314000)	DIVM	(236000)
CAME	(312000)	DPB	(137000)
		DPBI	(136000)

Symbol	Value	Symbol	Value
CAMG	(317ØØØ)	END	pseudo
CAMGE	(315ØØØ)	EQUALS	pseudo
CAML	(311ØØØ)	EQV	(444ØØØ)
CAMLE	(313ØØØ)	EQVB	(447ØØØ)
CAMN	(316ØØØ)	EQVI	(445ØØØ)
CLEAR	(4ØØØØØ)	EQVM	(446ØØØ)
		EXCH	(25ØØØØ)
CLEARB	(4Ø3ØØØ)	EXP	pseudo
CLEARI	(4Ø1ØØØ)	EXPUNG	pseudo
CLEARM	(4Ø2ØØØ)	FAD	(14ØØØØ)
CONI	(7ØØ24Ø)pseudo	FADB	(143ØØØ)
CONO	(7ØØ2ØØ)pseudo	FADL	(141ØØØ)
CONSO	(7ØØ34Ø)pseudo	FADM	(142ØØØ)
CONSTA	pseudo		
FADR	(144ØØØ)	FSBM	(152ØØØ)
FADRB	(147ØØØ)	FSBR	(154ØØØ)
FADRL	(145ØØØ)	FSBRB	(157ØØØ)
FADRM	(146ØØØ)	FSBRL	(155ØØØ)
FDV	(17ØØØØ)	FSBRM	(156ØØØ)
FDVB	(173ØØØ)	FSC	(132ØØØ)
FDVL	(171ØØØ)	HLL	(5ØØØØØ)
FDVM	(172ØØØ)	HLLE	(53ØØØØ)
FDVR	(174ØØØ)	HLLEI	(531ØØØ)
FDVRB	(177ØØØ)	HLLEM	(532ØØØ)
FDVRL	(175ØØØ)	HLLES	(533ØØØ)
FDVRM	(176ØØØ)	HLLI	(5Ø1ØØØ)
FMP	(16ØØØØ)	HLLM	(5Ø2ØØØ)
FMPB	(163ØØØ)	HLLO	(52ØØØØ)
FMPL	(161ØØØ)	HLLOI	(521ØØØ)
FMPM	(162ØØØ)	HLLOM	(522ØØØ)
FMPR	(164ØØØ)	HLLOS	(523ØØØ)
FMPRB	(167ØØØ)	HLLS	(5Ø3ØØØ)
FMPRL	(165ØØØ)	HLLZ	(51ØØØØ)
FMPRM	(166ØØØ)	HLLZI	(511ØØØ)
FSB	(15ØØØØ)	HLLZM	(512ØØØ)
FSBB	(153ØØØ)	HLLZS	(513ØØØ)
FSBL	(151ØØØ)	HLR	(544ØØØ)
HLRE	(574ØØØ)	HRLOI	(525ØØØ)
HLREI	(575ØØØ)	HRLOM	(526ØØØ)
HLREM	(576ØØØ)	HRLOS	(527ØØØ)
HLRES	(577ØØØ)	HRLS	(5Ø7ØØØ)
HLRI	(545ØØØ)	HRLZ	(514ØØØ)
HLRM	(546ØØØ)	HRLZI	(51ØØØØ)
HLRO	(564ØØØ)	HRLZM	(516ØØØ)
HLROI	(565ØØØ)	HRLZS	(517ØØØ)

Symbol	Value	Symbol	Value
HLROM	(566000)	HRR	(540000)
HLROS	(567000)	HRRE	(570000)
HLRS	(547000)	HRREI	(571000)
HLRZ	(554000)	HRREM	(572000)
HLRZI	(555000)	HRRES	(573000)
HLRZM	(556000)	HRRI	(541000)
HLRZS	(557000)	HRRM	(542000)
HRL	(504000)	HRRO	(560000)
HRLE	(534000)	HRROI	(561000)
HRLEI	(535000)	HRROM	(562000)
HRLEM	(536000)	HRROS	(563000)
HRLES	(537000)	HRRS	(543000)
HRLI	(505000)	HRRZ	(550000)
HRLM	(506000)	HRRZI	(551000)
HRLO	(524000)	HRRZM	(552000)
HRRZS	(553000)	IORB	(437000)
IBP	(133000)	IORI	(435000)
IDIV	(230000)	IORM	(436000)
IDIVB	(233000)	IRP	pseudo
IDIVI	(231000)	IRPC	pseudo
IDIVM	(232000)	IRPS	pseudo
IDPB	(136000)	JFCL	(255000)
IF1	pseudo	JRA	(267000)
IF2	pseudo	JRST	(254000)
IFE	pseudo	JSA	(266000)
IFG	pseudo	JSP	(265000)
IFGE	pseudo	JSR	(264000)
IFL	pseudo	JUMP	(320000)
IFLE	pseudo	JUMPA	(324000)
IFN	pseudo	JUMPE	(322000)
IFSE	pseudo	JUMPG	(327000)
IFSN	pseudo	JUMPGE	(325000)
ILDB	(134000)	JUMPL	(321000)
IMUL	(220000)	JUMPLE	(323000)
IMULB	(223000)	JUMPN	(326000)
IMULI	(221000)	LDB	(135000)
IMULM	(222000)	LDBI	(134000)
IOR	(434000)	LPT	124
MOVEM	(202000)	LOC	pseudo
MOVES	(203000)	LSH	(242000)
MOVM	(214000)	LSHC	(246000)
		MOVE	(200000)
		MOVEI	(201000)
		POP	(262000)
		POPJ	(263000)
		PRINTC	pseudo

Symbol	Value
MOVMI	(215000)
MOVMM	(216000)
MOVMS	(217000)
MOVN	(210000)
MOVNI	(211000)
MOVNM	(212000)
MOVNS	(213000)
MOVS	(204000)
MOVSI	(205000)
MOVSM	(206000)
MUL	(224000)
MULB	(227000)
MULI	(225000)
MULM	(226000)
NOSYMS	pseudo
NULL	pseudo
OCTAL	pseudo
OFFSET	pseudo
ORCA	(454000)
ORCAB	(457000)
ORCAI	(455000)
ORCAM	(456000)
ORCB	(470000)
ORCBB	(473000)
ORCBI	(471000)
ORCBM	(472000)
ORCM	(464000)
ORCMB	(467000)
ORCMI	(465000)
ORCMM	(466000)
PI	4
SETCAM	(452000)
SETCM	(460000)
SETCMB	(463000)
SETCMI	(461000)
SETCMM	(462000)
SETM	(414000)
SETMB	(417000)
SETMI	(415000)
SETMM	(416000)
SETO	(474000)
SETOB	(477000)
SETOI	(475000)
SETOM	(476000)
SETZ	(400000)
SETZB	(403000)
SETZI	(401000)
SETZM	(402000)
SIXBIT	pseudo
SKIP	(330000)

Symbol	Value
SKIPA	(334000)
SKIPE	(332000)
SKIPG	(337000)
SKIPGE	(335000)
TDC	(650000)
TDCA	(654000)
TDCE	(652000)
PRINTX	pseudo
PTP	100
PTR	104
PUSH	(261000)
PUSHJ	(260000)
RADIX	pseudo
RELOCA	pseudo
REPEAT	pseudo
RIM	pseudo
RIMI	pseudo
ROT	(241000)
ROTC	(245000)
SBLK	pseudo
SETA	(424000)
SETAB	(427000)
SETAI	(425000)
SETAM	(426000)
SETCA	(450000)
SETCAB	(453000)
SETCAI	(451000)
SKIPL	(331000)
SKIPLE	(333000)
SKIPN	(336000)
SOJ	(360000)
SOJA	(364000)
SOJE	(362000)
SOJG	(367000)
SOJGE	(365000)
SOJL	(361000)
SOJLE	(363000)
SOJN	(366000)
SOS	(370000)
SOSA	(374000)
SOSE	(372000)
SOSG	(377000)
SOSGE	(375000)
SOSL	(371000)
SOSLE	(373000)
SOSN	(376000)
SQUOZE	pseudo
SUB	(274000)
SUBB	(277000)
SUBI	(275000)
SUBM	(276000)
TLNA	(605000)
TLNE	(603000)
TLNN	(607000)

Symbol	Value
TDCN	(656000)
TDN	(610000)
TDNA	(614000)
TDNE	(612000)
TDNN	(616000)
TDO	(670000)
TDOA	(674000)
TDOE	(672000)
TDON	(676000)
TDZ	(630000)
TDZA	(634000)
TDZE	(632000)
TDZN	(636000)
TERMIN	pseudo
TITLE	pseudo
TLC	(641000)
TLCA	(645000)
TLCE	(643000)
TLCN	(647000)
TLN	(601000)
TR2	(620000)
TRZA	(624000)
TRZE	(622000)
TRZN	(626000)
TSC	(651000)
TSCA	(655000)
TSCE	(653000)
TSCN	(657000)
TSN	(611000)
TSNA	(615000)
TSNE	(613000)
TSNN	(617000)
TSO	(671000)
TSOA	(675000)
TSOE	(673000)
TSON	(677000)
TSZ	(631000)
TSZA	(635000)
TSZE	(633000)
TSZN	(637000)
TTY	120
TWX	310
UTC	210

Symbol	Value
TLO	(661000)
TLOA	(665000)
TLOE	(663000)
TLON	(667000)
TLZ	(621000)
TLZA	(625000)
TLZE	(623000)
TLZN	(627000)
TRC	(640000)
TRCA	(644000)
TRCE	(642000)
TRCN	(646000)
TRN	(600000)
TRNA	(604000)
TRNE	(602000)
TRNN	(606000)
TRO	(660000)
TROA	(664000)
TROE	(662000)
TRON	(666000)
UTS	214
VARIAB	pseudo
WORD	pseudo
XCT	(256000)
XOR	(430000)
XORB	(433000)
XORI	(431000)
XORM	(432000)
XWD	pseudo
XWORD	pseudo
.	Current Location
.ASCII	pseudo
.BEGIN	pseudo
.ELDC	pseudo
.END	pseudo
.FNAM1	pseudo(t-s only)
.FNAM2	pseudo(t-s only)
.FORMA	pseudo
.GLOBA	pseudo
.GO	pseudo
.GSSET	pseudo
.LIBRQ	pseudo
.LNKOT	pseudo
.LOP	pseudo
.NSTGW	pseudo
.TAG	pseudo
.TYPE	pseudo
YSTGW	pseudo

NOTE:

In the time-sharing version there are a great many additional initial symbols, for example, system calls, status bits etc. This list changes frequently in accordance with the development of the time-sharing system, so no attempt has been made to include them here. To avoid conflict with these symbols, bear in mind that each such symbol begins with a period.

Binary Formats

RIM (loaded by loader at 2Ø)

```
DATAI PTR, ADDRESS
CONTENTS
DATAI PTR, ADDRESS
    .
    .
    .
CONTENTS
JRST   START
```

RIM1

```
HRRI     17, RIGHTHALF
HRRM     17, ADDRESS
HRRI     17, LEFTHALF
HRLM     17, ADDRESS
HRRI     17, RIGHTHALF
HRRM     17, ADDRESS
    .
    .
    .
HRRI     17, LEFTHALF
HRLM     17, ADDRESS
JRST     START
```

SBLK

(1) SBLK loader in RIM format: Read Left-to-Right

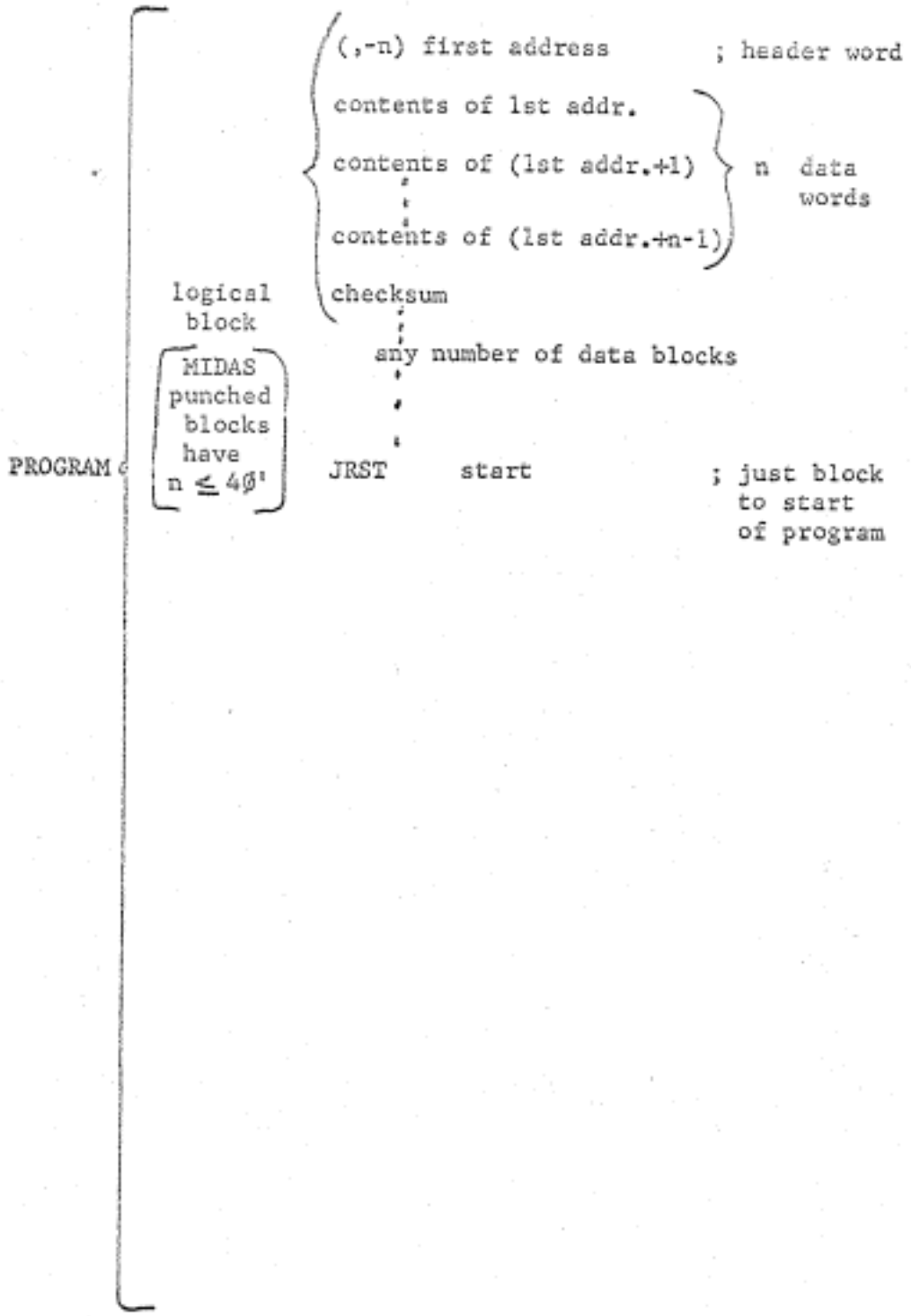
* numbers are octal *

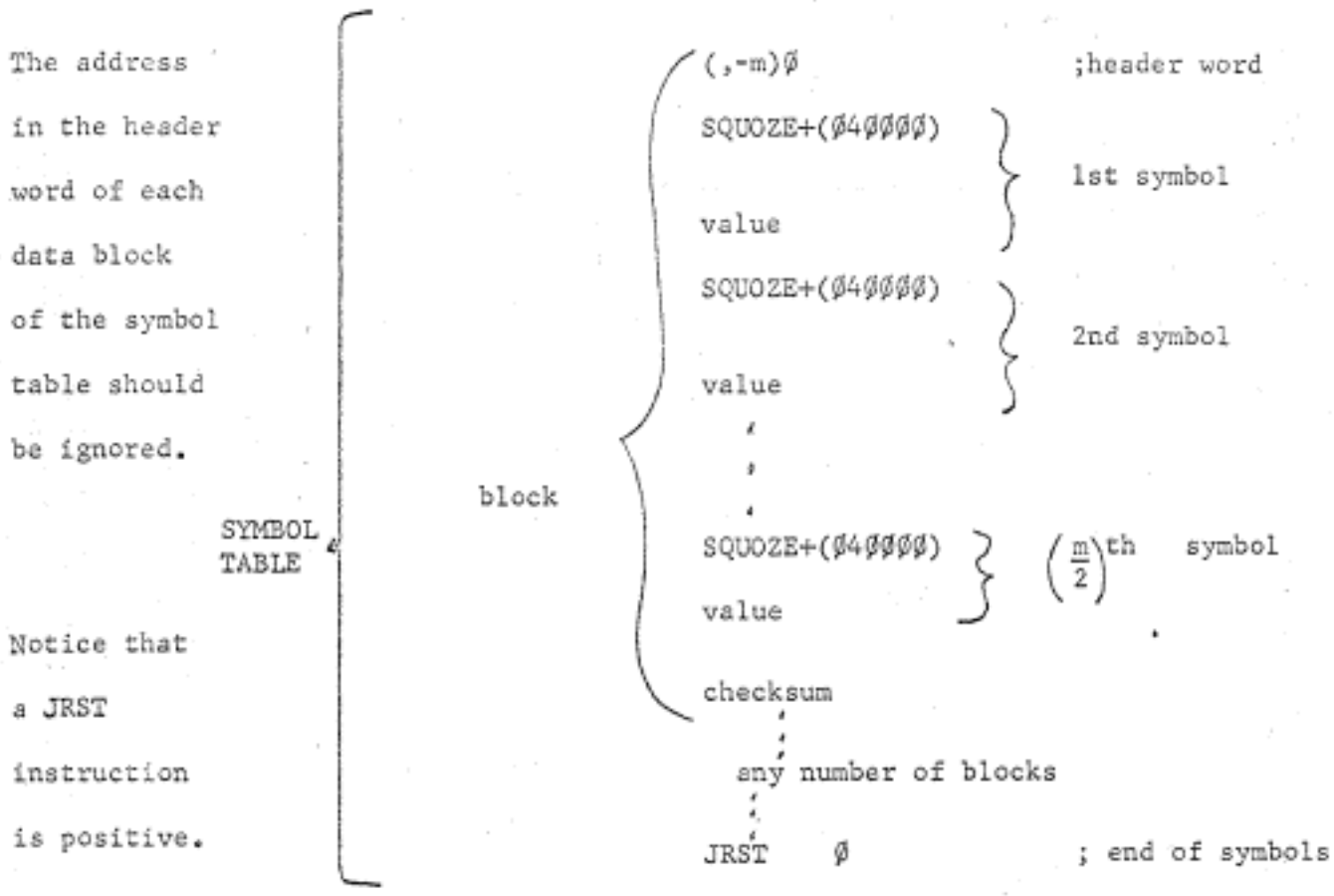
```
DATAI  PTR,Ø
DATAI  PTR,1
DATAI  PTR,2
DATAI  PTR,3
DATAI  PTR,4
DATAI  PTR,5
DATAI  PTR,6
DATAI  PTR,7
DATAI  PTR,1Ø
DATAI  PTR,11
DATAI  PTR,12
DATAI  PTR,13
DATAI  PTR,3Ø
DATAI  PTR,31
DATAI  PTR,32
DATAI  PTR,33
DATAI  PTR,34
DATAI  PTR,35
DATAI  PTR,36
JRST   1
```

```
CONO   PTR,6Ø
JSP    14,3Ø
DATAI  PTR,16
MOVE   15,16
JUMPGE 16,16
JSP    14,3Ø
DATAI  PTR,(16)
ROT    15,1
ADD    15,(16)
AOBJN  16,5
MOVEI  14,33
JRST   3Ø
CONSO  PTR,1Ø
JRSTT  3Ø
JRST   (14)
DATAI  PTR,16
CAMN   15,16
JUMPA  1
JRST   4,Ø
```

(2) Blocks read by SBLK Loader

The checksum for each data block is computed as follows: take the header word; rotate it to the left one bit and add the first data word; rotate the result to the left one bit and add the second data word; etc. After the nth data word the result should equal the checksum.




The address in the header word of each data block of the symbol table should be ignored.
Notice that a JRST instruction is positive.
SYMBOL TABLE
block
(,-m)Ø
;header word
SQUOZE+(Ø4ØØØØ)
value
1st symbol
SQUOZE+(Ø4ØØØØ)
value
2nd symbol
SQUOZE+(Ø4ØØØØ)
value
(m/2)th symbol
checksum
any number of blocks
JRST Ø
; end of symbols

Suggested Symbolic Format

The format-free nature of the MIDAS symbolic language allows a great deal of typographic variety, but some of this variety should generally be given up for the sake of readability of one's program. The suggestions which follow are not hard-and-fast rules, but in most cases will represent a consensus as to good practice.

(1) A storage word should appear on a line indented one tab stop.

(2) An address tag (symbol followed by colon) should not be indented.

(3) All address tags referring to a given location should appear on separate lines, with the contents of that location indented once on the same line as the last (or only) tag.

(4) Blank lines (redundant carriage returns, line feeds, etc.) may occur wherever useful to demarcate parts of a routine.

(5) The entire typescript should be divided into pages (by form-feed characters). The logical divisions of the program should coincide with page divisions. Although a page will hold up to about 6Ø. lines, it is good practice whenever convenient to put only 45. or 5Ø. lines per page.

(6) Within a storage word, use space rather than tab as a field delimiter.

(7) Close all ([and < groups with)] and >.

(8) When a comment is applied to a storage word, put one tab before the ;.

(9) Pseudoinstructions and macro names which generate one or more storage words should generally be treated like storage words.

(10) END and TITLE are usually not indented.

(11) With the possible exception of Ø, accumulators should as a rule be referred to symbolically, with such symbols defined at the beginning of the program.

(12) Generally it is a good idea to put a parameter assignment (use of =) at the earliest point in the program where its right side is defined.

EXAMPLE

The following page is an example of coding in the MIDAS language. (It is the first page of a program which uses the TV camera.) It demonstrates many, though by no means all, of the features of MIDAS. The FIX macro is indeed a sequence of instructions which will convert a floating-point number to fixed-point.

```
TITLE FIND THAT BALL

RELOCATABLE

MSCHN=2
TVHCHN=3
TVVCHN=4
TVCCHN=5
TVA=770
MSC=760
LPDL=100

DEFINE BALLP N,NO
        SETCM Y,X
        JUMPE Y,NO
REPEAT N-1,[        LSH X,1
        ANDCM Y,X
]
        JUMPE Y,NO
        TLNN Y,777000
        JRST .+3
        LSH Y,-11
        TLO Y,011000
        FSC Y,26
        FAD Y,Y
        LSH Y,-27
TERMIN

DEFINE FIX
        MULI X,400
        TSC X,X
        ASH X+1,-243(X)
        TERMIN

P''=1
Y''=4   ;Y
T''=5   ;T
X''=3   ;X
D=2
E=6
DING:   JRST 4,THRU

LOOK:   SETOM LCK1
        SETOM LCK2
        CLEARM TAKP
        MOVE T,TAKN
        MOVEM T,TAKENO
        CLEARM LETAKE
        CLEARM PNTSS
        SETOM TVIOLE
        SETOM CANCEL
        CLEARM MSTIME
        CONO MSC,MSCHN
        JRST LOOKA
```