MACRO ASSEMBLY PROGRAM
for
PROGRAMMED DATA PROCESSOR -1
(PDP-1)
TABLE OF CONTENTS

PART I
INTRODUCTION ................................................................. 1

PART II
THE MACRO SOURCE LANGUAGE
A. Processing of the Source Language by MACRO ...................... 5
B. Notation ........................................................................... 6
C. Syllables and Expressions: Format of a Source Program .......... 6
D. Use of Expressions .......................................................... 8
E. Pseudo-Instructions ......................................................... 10
F. Automatic Storage Assignment ......................................... 13
G. Macro-Instructions .......................................................... 17

PART III
USE OF MACRO ASSEMBLY
A. English Tape Format ........................................................ 23
B. Operation of the MACRO Assembly Program ...................... 23
C. Test Word Control of MACRO ........................................... 25
D. Error Stops During a MACRO Assembly ............................... 26

PART IV
EXAMPLES ........................................................................... 29

APPENDIX 1
SYMBOLS IN MACRO'S PERMANENT VOCABULARY .................. 35
PART I

INTRODUCTION

Coding for a digital computer is the process of setting down the exact sequence of instructions, constants and tables which must be placed in the memory of a specific computer to perform some desired computation. The art of coding is far advanced from the early days of automatic computation when all instructions of a program were written in numerical form using the number system of the machine being used. This was coding in machine language. Now, assembly programs are available for nearly all digital computers which allow the user a more convenient language for writing the instructions forming his program. MACRO is such an assembly program for the DEC PDP-1 computer. It translates a source program written in a language, which we shall call the MACRO source language, into an object program, which can be read directly into the PDP-1 computer.

A MACRO source program takes the form of a punched paper tape prepared using the standard FIO-DEC Flexowriter with the Concise III typeface as given in Appendix A, or using an on-line editing program such as EXPENSIVE TYPEWRITER. An example of a source program is given in Figure 1, exactly as the Flexowriter would print it. The first line of a source program is a title line, and the program concludes with a start line, as in the example. A MACRO object program is a binary tape punched by the PDP-1 under control of the MACRO assembly program. The format of the binary tape is detailed in Appendix B. It contains the title line of the source program in readable form, an input routine which is read in the read-in mode by the PDP-1 and placed in the last registers of memory, blocks of instructions and constants making up the object program, and, finally a start block indicating the address of the first instruction to be executed.
SUM
n=100
100/
a,   law tab
dap b
dzm s
b,   lac .
add s
da c s
idx b
sas c
jmp b
hlt

Figure 1 A MACRO source program

An assembly program such as MACRO permits the user to prepare source
language programs using symbolic names or, simply, symbols to represent the numerical
values of instruction codes, machine addresses, and parameters. Not only does this make
the program easier to read and understand, but it also allows a degree of flexibility not
otherwise possible. For example, a program properly written in the MACRO source
language can be placed at any location in the PDP-1 memory by changing a single line
in the source program.

Normally an assembly program produces one machine language instruction for
each instruction of the source program. However, some lines in the source program,
known as pseudo-instructions, are directions to the assembly program and do not directly
produce instructions in the object program. An example is the pseudo-instruction start
used to denote the end of a MACRO source program. Also, MACRO provides a means of
associating symbolic names called macro-instructions with sequences of instructions
according to the user's desires.

Part II of this memorandum describes the MACRO source language in terms of how
it is converted into machine language by the MACRO assembly program. Part III presents
operating instructions for assembling source program tapes written in the MACRO language.
PART II

The MACRO Source Language

A. Processing of the source language by MACRO

A MACRO source program may be thought of as a long linear string of characters in which the alphabet includes the various typewriter functions—tabulation, carriage return, backspace and case shifts—as well as the visible characters. In assembling a program, MACRO scans the string of characters making up the source program starting from the title line and continuing through the stop code following the first start line. Two passes of this nature are required for a complete assembly. On the first pass, the values of symbolic addresses are determined and storage areas are reserved for variables, tables, and constants. During the second pass the symbol values are used in evaluating the instructions and constants making up the user’s program, and the object program is punched in the form of a binary tape.

Certain lists are maintained by MACRO for the purpose of carrying out the assembly process. The symbol table contains an entry for each symbolic address, instruction code or parameter used by the programmer in his source language program. The entry consists of the character group forming the symbol, and its value represented as an eighteen bit number.

The macro-instruction list contains the sequence of instructions making up each macro instruction defined by the user’s source program. The symbolic names of pseudo and macro-instructions are filed in the pseudo-instruction list which contains a reference to the macro-instruction list for macro names, or to the appropriate routine in MACRO in the case of pseudo-instructions.

A current location counter in MACRO indicates the object program location in which the next instruction translated will be placed. A current radix indicator controls whether a string of digits in the source program is treated as octal (base 8) or decimal (base 10).

At the beginning of an assembly, the macro-instruction list is empty, and the pseudo-instruction list contains only the names of system pseudo-instructions described in later
The symbol table contains the list of permanent symbols and values given in Appendix C. At the beginning of each pass, the current location is set to 4 and the current radix is set to octal.

B. Notation

In the following description of the MACRO source language, its structure will be illustrated by character strings which could appear as part of a source program. These strings will be enclosed in brackets < >. For clarity, the signs <→> and <→> will be used to represent tabulation and carriage return, and <_> will be used for space when needed for emphasis. The abbreviations tab and cr will be used for tabulation and carriage return in format descriptions.

C. Syllables and Expressions: Format of a Source Program

As mentioned in the Introduction, a MACRO source program consists of a title line, a body, and a start line. A title line is an arbitrary string of characters terminated by a cr. The complete title is punched by MACRO in readable form at the beginning of the object program tape. A middle dot in the title will cause only the characters preceding it to be punched.

The start line must have the form illustrated in the source program example on page 2. It must be terminated by a cr and followed by a stop code, which will be the last character read on the source program tape.

The body is a series of expressions, which are the basic units of a MACRO source program. An expression is a string of characters representing an instruction of a program, a constant used by a program, table entries, or other data. Some examples are:

<add 100>
<dio t+2>
<i+1234>
<m+n>
<123-a->

Expressions are usually delimited by tab, cr, slash, comma, or equals. The significance
of an expression depends on the context in which it appears.

More precisely, an expression is one or more syllables separated by the characters plus, minus, or space. Examples of syllables are:
<add>
<100>
<1224>
<i>

MACRO computes the value of an expression by summing the values of its component syllables. If a syllable is preceded by a plus sign or space, the syllable value is added in forming the sum, if preceded by a minus sign, the complement of the syllable value is added. The plus sign may be omitted before the first syllable of an expression. The addition is performed in the 18-bit one's complement arithmetic of the PDP-1, except that if the sum is zero, it will be evaluated as minus zero 777777 if any syllable was other than plus zero. Since this addition is associative - the order in which the syllables of an expression are written does not affect its value. In the following examples, the current radix is assumed set for octal. The two expressions of each pair will represent the same value to MACRO.

<add> = <400000>
<-234> = <776543>
<-0> = <777777>
<+x> = <x>
<dac 123> = <240123>
<lac i a+2> = <a+210002>
<-1+1> = <777777>

Syllables can take a number of forms, two of which will be mentioned here.

1) Symbols - A symbol is a string of one, two, or three letters and digits in which at least one letter appears. A symbol may represent an instruction as in dac, or a symbolic address such as x in lio x. A symbol is defined if there is a corresponding entry in the symbol table, otherwise it is undefined. The value of a symbol is the 18-bit number associated with it in the symbol table if it is defined, and minus zero if it is undefined.
2) Integers - An integer is a string of the digits 0, 1, ... , 9 and is evaluated as an octal or decimal integer according to the current radix. The value of an integer is the 18-bit representation of the integer. Thus the largest integer taken as its face value is <777777> in octal or <262143> in decimal.

The value of an integer above these limits is taken modulo \(2^{18} - 1\).

D. Use of Expressions

The meaning of an expression to MACRO is determined by the context in which it appears in the source program, and usually by the character immediately following it.

1) Storage word - An expression followed immediately by a tab or cr is a storage word.

\(<\text{mp jes}\>\)

\(<+103\text{-opr}\>

The 18-bit number representing the value of the word is entered in the object program at the address given by the current location counter in MACRO. The location counter is then advanced by one. A storage word may be an instruction forming part of a program, a constant used by the program, or data.

2) Location assignment - An expression immediately followed by a slash is a location assignment:

\(<100/>\)

\(<\text{tab}+120/>\)

The current location is set equal to the address portion of the value of the expression. Thus

\(<100/\text{sza} \text{ jmp 100}\>\)

or

\(<100/\text{sza} \text{ jmp 100}>\)

in the source program will place the instruction sza in the object program at register 100, and the instruction jmp 100 in register 101.

If, on Pass 1, a location assignment contains any undefined symbols, the current location becomes indefinite.
3) Symbolic address tag - An expression immediately followed by a comma is an address tag:

\[ \text{<beg,>} \]
\[ \text{<100,>} \]
\[ \text{<a+12,>} \]

If the expression contains one syllable, and this syllable is an undefined symbol not preceded by a minus sign, and the current location is not indefinite, the symbol is entered in MACRO's symbol table and assigned a value equal to the current location. Otherwise, the value of the expression compared with the current location, and a disagreement will cause an error printout. The current location is not changed by a symbolic address tag.

Using a symbolic address, the preceding example could be written as

\[ \text{<100/a, sza \rightarrow imp \ a2>} \]

The programmer should note that location assignments and symbolic address tags, in themselves, have no effect on the object program, but rather direct the process of assembly. Also, he should observe their inverse character. The location assignment sets the current location counter to the value of an expression, while the address tag sets the value of a symbol equal to the current location. Hence the sequences

\[ \text{<100/a,b,>} \]

or

\[ \text{<100/} \quad \rightarrow \quad a, b \quad \text{<a2>}} \]

assign 100 as the value of both symbols a and b. A sequence such as

\[ \text{<1000/tab,2}} \]

\[ \text{tab+n/} \]

is frequently used to reserve a block of registers for a table of data or computed results. In the example the block starts at register 1000, is named by the symbol tab, and contains a number of registers given by the value of the symbol n.

4) Symbolic parameter assignment - A symbol immediately followed by an equal sign, an expression, and a tab or cr is a parameter assignment. It assigns the symbol to the left of the equal sign a value given by the expression to the right, if the expression is defined. If the expression is undefined, no action is taken. For example:
The parameter assignment facility is useful for setting table lengths and other properties of the object program. It provides a means of defining new operation codes to simplify the writing of programs and for preparation of instruction sets for interpretive programs.

Comments - A string of characters which commences with a slash is a comment. The string is ended by a tab or cr. Comments are ignored by MACRO and may be used to label selections of a program, annotate important instructions, and give the reader of the typescript information about the program. Example:

\(<\text{get}, \quad \rightarrow \text{lac} \quad \rightarrow / \text{This is a comment}\>\)

Current location syllable - the character \(<.>\) is a special syllable whose value is equal to the current location.

Hence,

\(<\text{sza} \rightarrow \text{jmp} \rightarrow 12>\)

is an alternate way of writing

\(<a, \text{sza} \rightarrow \text{jmp} \rightarrow q2>\)

The use of the features of the MACRO assembly program that have been discussed so far are illustrated by the program example on page 2. The reader should be sure he thoroughly understands this example before proceeding further in this memo.

E. Pseudo-instructions

Pseudo-instructions in the MACRO source language are directions to the MACRO assembly program which govern the way in which subsequent information in the source program is processed. The pseudo-instruction \(\text{start}\), which informs MACRO of the end of the source language program, has already been introduced.

Typographically, a pseudo-instruction is a string of at least four letters and digits, in which at least one of the first four is a letter. The string is followed by a terminating character which may be space, plus, minus, tab, or cr. For convenience, the character \(\leftarrow\) will be used to represent any of these terminating characters. A pseudo-
instruction may always be abbreviated to four characters.

The pseudo-instructions of MACRO are described below:

1) End of source program

The pseudo-instruction start denotes the end of the source language program. The expression following start gives the address of the instruction in the object program which is to be executed first, and MACRO will include the appropriate start block in the binary program tape. The line

\(<\text{start beg}\underline{+2}>\)

will terminate scanning of the source program and cause the word \(\text{jmp beg}\underline{+2}\) to be punched as the start block of the binary tape. When the binary tape is subsequently read into the PDP-1, control will go to register 'beg\underline{+2}' after the start block is read.

2) Radix control

The pseudo-instructions octal and decimal control the current radix for evaluation of integer syllables. The string \(<\text{octal}>\) anywhere in the source program sets the current radix to eight, and the string \(<\text{decimal}>\) sets the current radix to ten. These pseudo-instructions may appear as syllables of value zero in an expression, for example:

\(<\text{octal} 44+\text{decimal} 27 \rightarrow \)

is equivalent to

\(<\text{octal} \rightarrow 77 \rightarrow \text{decimal} \rightarrow \)

3) Suppression of input routine

The string \(<\text{noinput}>\) anywhere in the source program will suppress punching of the input routine on the binary program tape.

4) Storage of character codes

The pseudo-instruction character, flexo, and text are provided to allow the programmer a convenient means of storing character codes for printout by his program, or for comparison against alphanumerical data accepted by his program. For reference, the six-bit codes for the Concise III character set used with the PDP-1 are included as Appendix A of this memorandum.

The pseudo-instruction character is used to place a character code in the left, middle or right six-bit portion of an eighteen bit word. The string \(<\text{character}>\) is followed
by \( r, m, \) or \( 1 \) according to the position desired, and then the character whose code is desired. Thus

\[
\text{<char ra> is the same as <000061>}
\]
\[
\text{<char mb> is the same as <006200>}
\]
\[
\text{<char lc> is the same as <630000>}
\]

The above strings are pseudo-instruction syllables, and may be used in the same manner as symbols or integers in forming expressions. For example

\[
\text{<=char rx> is equivalent to <777750>}
\]

The pseudo-instruction flexo is used to compile three character codes into one eighteen bit word. Thus

\[
\text{<flexo dec> is the same as <646563>}
\]

and may also be written as

\[
\text{<char rc+char me+char ld>}
\]

The pseudo-instruction text is used to assemble a long string of characters by groups of three into successive words in the object program. The string to be assembled is enclosed between two appearances of the same character, which immediately follows the string <text>. It is suggested that the character period be used to enclose the string, although any legal character may be so used. Of course, the character selected cannot appear within the string itself. For example, the string

\[
\text{<text .Error.>}
\]

is equivalent to

\[
\text{<flex E \rightarrow flex erro \rightarrow char lr \rightarrow>}
\]

5) The repeat pseudo-instruction

The repeat pseudo-instruction provides a convenient way of placing a sequence of similar expressions in a block of the object program. The string

\[
\text{<repeat=n,>}
\]

causes MACRO to scan and assemble the following characters a number of times equal to the value of the expression \( n \). The string of characters scanned and assembled is the range of the repeat; it starts immediately after the comma, and continues up to and including the next carriage return. The expression giving the order of the repeat
(n in the above example) must be nonnegative and definite when the repeat is encountered during the first pass. If the value of the expression is zero, the range of the repeat is ignored.

As an example of the use of repeat, the following sequence forms a table of squares of length n.

\[
\begin{align*}
  \text{\textcolor{red}{U}} &= 0 \quad \text{\textcolor{red}{V}} = 1 \\
  \text{\textcolor{red}{U}} &= \text{\textcolor{red}{U}} + \text{\textcolor{red}{V}} \\
  \text{\textcolor{red}{V}} &= \text{\textcolor{red}{V}} + 2
\end{align*}
\]

6) Emptying of symbol table

Occasionally it is desirable to delete all symbols from the symbol table, when using MACRO to assemble certain symbolic data tapes. The string \textit{<expunge>} appearing in the source program deletes all symbols, including the initial list given in Appendix C, from the MACRO symbol table at the time it is encountered during the first pass. It has no effect during the second pass.

7) Other pseudo-instructions

The remaining pseudo-instructions - \textit{constants}, \textit{dimension}, \textit{variables}, \textit{define}, and \textit{terminate} - will be discussed in connection with automatic storage assignment and macro-instructions in the following sections.

F. Automatic storage assignment

Several features have been provided in the MACRO assembly program which automatically assign storage locations for the constants used by a program and the variables and tables manipulated by the program. These features reduce the amount of typing required to prepare a complete source language program, simplify editing, and make the source program typescript more readable.

\textbf{Constants} - An expression enclosed in parentheses is a \textit{constant syllable} and may appear as a syllable in storage words, and parameter assignments. MACRO will compute the value of the expression enclosed and place it in a constants area of the object program as explained below. The value of a constant syllable is the address where the enclosed word is placed by MACRO. The location at which constant words are placed is determined by the next appearance of the pseudo-instruction \textit{constants} following the constant syllable.
For example, in the program illustration on page 2, the line
\[ <c, \text{tab+n}> \]
could be omitted if the line
\[ <\text{sad c}> \]
were replaced by
\[ <\text{sad (tab+n)}> \]
and the string
\[ <\text{constants}> \]
inserted before the start line.

When the pseudo-instruction \textit{constants} is scanned by MACRO, the constants expressions assembled since the last use of the pseudo-instruction \textit{constants}, or since the beginning of the program, are placed in the object program starting at the current location. Constant words having the same numerical value are entered only once. The current location is advanced to an address somewhat beyond the register in which the last constant is placed, leaving a small gap of unused registers between the constants area and any following portion of the program. (This gap arises because MACRO reserves blocks of registers for constants words during the first pass when some of these words may not be defined.)

Some additional points on the use of constant syllables are:
1) The closed parenthesis may be omitted from constant syllables immediately followed by one of the terminating characters comma, tab or cr.
2) Recursive use of constant syllables is permitted, that is, a constant syllable may appear within an expression forming a new constant syllable:
\[ <\text{lac (add (com)}> \]
or simply
\[ <\text{lac (add (com)}> \]
This may be continued to a depth of eight levels.

Variables - A symbol typed with a bar over at least one of its characters at its first appearance in the source program is a \textit{variable}, for instance:
\[ <\text{sym}> \text{ or } <\text{a12}> \]
All symbols identified as variables become defined on the subsequent appearance of the
pseudo-instruction <variables >. The pseudo-instruction variables must follow all defining appearances of variables and may appear only once in any source program. The variables are assigned to sequential locations starting at the location of the pseudo-instruction variables. Their initial contents is undefined. For instance, the sequence

< lac a → add b →/ dac a →/ . . .

. . . b, 0 →/ a, 0 >

is equivalent to

< lac a → add To →/ dac a →/ . . .

. . . variables >

except that the contents of registers a and b of the object program will be zero in the first case, and undefined in the second.

Tables - Blocks of registers may be reserved for tables by means of the dimension pseudo-instruction. The string

<dimension x(n), y(m), z(m+n) >

for example, reserves three blocks of lengths given by the values of the expressions n, m, and m+n. The first address of each block is assigned as the value of the symbols x, y, and z. The reserved blocks are placed at the location in the object program specified by the variables pseudo-instruction. The initial contents of the reserved blocks is undefined in object program. The following rules apply:

1. The expressions given as lengths of blocks in a dimension pseudo-instruction must be definite when scanned on the first pass.
2. The symbols assigned to blocks by a dimension statement must be previously undefined.

The use of dimension, variables, and constants in a complete MACRO source program is illustrated in figure 3. This program will produce exactly the same object program as the introductory example on page 2 except that the initial contents of register s is zero in the earlier version and undefined here.
SUM
n=100
dimension tab (n)
100/
a,    law tab
dap b
dzm s
b,    lac .
add s
idx b
sas (lac tab+n
jmp b
hlt

tvariables
tconstants
start a

FIGURE 3
A MACRO source program with automatic storage allocation
G. Macro-instructions

Very frequently, the same sequence of instructions is required at many places in a computer program. For example the two instruction sequence

spa

spiracy

forms the absolute value of the contents of the PDP-1 accumulator. To simplify writing of the source language program and provide a more meaningful source program typescript, it is convenient to represent such a sequence by a special name such as absolute. The macro-instruction feature of the MACRO assembly program makes this possible. The source program sequence

<define
  absolute
  spa
  cma
  terminate>

defines a macro-instruction with name absolute. When the string

<absolute>

subsequently appears in the source program, the sequence of words, spa, cma, will be copies into the object program.

In the more common instances, the recurring instruction sequence is not identical in each appearance, but the words of a basic sequence are modified by additive parameters. A frequent combination is

lac x

dac y

which moves the quantity in register x to register y. The MACRO-instruction facility allows the user to represent such a sequence by a name with a group of parameters such as

<move x, y>

This representation is established by the definition
<define  move A, B>
<define  lacc A
<define  dac B
<define  terminate>

which must appear in the source program prior to use of the macro instruction. In this example A and B are dummy symbols which are symbols in which at least one letter is in upper case. When MACRO scans the macro-instruction

<move x, y>

it copies the word sequence from the definition into the object program substituting the values of the arguments x and y for the dummy symbols A and B respectively.

Defining a Macro-instruction

A macro-instruction definition consists of four parts; the pseudo-instruction define, the macro instruction name and dummy symbol list, the body, and the pseudo-instruction terminate. Each part is followed by at least one tabulation or carriage return.

The macro instruction name has the same form as a pseudo instruction -- a string of at least four letters and digits of which at least one of the first four characters is a letter. The name is terminated by a space or by a tab or cr if there is no dummy symbol list. The first six characters of a macro instruction name must distinguish that name from all other macro names and all pseudo-instructions. The dummy symbol list consists of as many distinct dummy symbols as desired, separated from each other by commas, and from the macro name by a space. Since dummy symbols have no meaning outside of a macro definition, the same dummy symbols may be used in many definitions without harm.

The body of a macro definition is an arbitrary sequence of storage words in which any dummy symbol from the dummy symbol list may appear as a syllable. The pseudo-instructions character, flexo, text, octal, decimal, and nainput may be used within the body of a macro definition. Constant syllables may appear in any expressions and dummy symbols may be used as syllables in constant expressions.

Using a Macro-instruction

A macro instruction consists of a macro instruction name followed by an
argument list, and a tabulation or carriage return. The argument list consists of expressions separated by commas, in correspondence with the dummy symbols listed in the definition of the macro-instruction. The first expression of the argument list must start with space, plus, or minus to separate it from the macro name. The expressions in the argument list may contain constant syllables and the pseudo instructions character, flexo, octal or decimal. When the current location syllable <.> appears in an argument list its value is taken as the current location at the time the macro-instruction name is scanned.

MACRO assembles a macro-instruction by evaluating the expressions in the argument list, substituting these values for corresponding dummy symbols in the definition and copying the resulting sequence of storage words into the object program. The current location is advanced for each word copied. If an argument expression is omitted, its value is taken as zero.

An Example

To illustrate the definition and use of a macro instruction is illustrated by a program to store zeros in a block of registers. This program can be assigned the name clear by the definition

```
<define clear A, N
law A
dap .+1
dzm
idx .-1
sas (dzm A+N
jmp .-3
terminate>
```

When the line

```
<clear tab, 1002>
```

appears later in the source program, the instruction sequence

```
law tab
dap .+1
dzm
idx .-1
```
sas (dzm tab+100
jmp -.3

is inserted in the object program. The resulting sequence will clear a hundred registers
starting with register tab.

Address tags within a macro definition

Before MACRO scans the body of a macro instruction definition, the current
location is set to zero: it is then advanced by one for each storage word included in the
definition. Therefore, an address tag in the body of a macro definition will be assigned
a value equal to the number of storage words in the body up to that point. Symbols
defined in this manner are entered in the symbol table as usual and may be referred to at
any point in the source program. Note that a given symbol should not be used as an
address tag in several definitions as this would attempt to define the same symbol twice.

Addressing within a macro definition

In longer macros it is frequently necessary for some instructions in the sequence
making up the macro to address other instructions in the sequence. The address parts of
such instructions must be given different values each time the macro-instruction is used
at a different object program location. To provide a convenient means of handling this
problem, a special dummy symbol <R> is provided. This dummy symbol may always be used
in the body of a macro definition and should not appear in the dummy symbol list. When
a macro-instruction is used, the current location at the time its name is scanned is
substituted for <R> in each appearance. In illustration, the preceding definition of
<clear A, N> may also be written as

<define
clear A, N
law A
dap a+R
a,
dzm
idx a+R
sas (dzm A+N
jmp a+R
terminate>

20
When the current location syllable <.> is used in the body of a macro definition as in the earlier example, symbol <R> is automatically included.

**Cascading macro definition**

Once defined, a macro instruction may be used in the body of another macro definition. In this case the expressions in the argument list of the macro instruction may also include any dummy symbols from the new definition. An example is given in Figure 4, which is a third way of writing the summation program introduced on page 2.

**Dummy symbol assignments**

Provision has been made for creating new dummy symbols and reassigning the meaning of existing dummy symbols within the body of a macro-instruction definition. This is accomplished by a dummy symbol assignment. The format is the same as for a parameter assignment except the left side must be a dummy symbol, and the expression on the right may contain dummy symbols. When a macro-instruction is subsequently used in the source program, the value substituted for the new dummy symbol is computed from the values of arguments substituted for dummy symbols according to the right hand side of the dummy symbol assignment. For instance the string

\[ <C=A+B-100> \]

will create a new dummy symbol <C> whose value is the sum of the arguments substituted for A and B, plus the number -100. Dummy symbols may be reassigned in terms of themselves:

Thus

\[ <X=X+X+X+X> \]

will cause four times the value of the argument corresponding to dummy symbol X to be substituted for X in subsequent appearances.
SUM

define initialize A, B, law B, dap A
   terminate

define accumulate T, add T, dac T
   terminate

define index A, B, C, idx A, sas (B
   jmp C, terminate

100/

define total T, N, S
   initialize b & R, T
   dzm S
   b,
   lac
   accumulate S
   index b+R, lac T+N, b+R
   terminate

n=100
dimension tab (n)

a,
total tab, n, s

hlt
variables
constants
start a

FIGURE 4

Cascading macro-instruction
PART III

USE OF THE MACRO ASSEMBLY

A. Source Program Tape Format

Each source program tape must begin with the title of the program. This title may consist of any legal flexo characters and is terminated by the first carriage return preceded by any character other than carriage return. On the source program tape the characters plus and space are equivalent as are the characters tab and carriage return. The MACRO language is format free; that is, the positioning of any character or syllable does not effect its meaning, and redundant characters are ignored (i.e., 3 spaces are equivalent to 1 space, successive tabs and/or carriage returns are equivalent to one tab or carriage return).

As MACRO instructions must be defined before they are used, the usual practice is to put these definitions at the beginning of the program. However, since MACRO definitions do not take up any space in the object program, they may be placed anywhere in the program. The pseudo-instruction constants and variables cause all previously mentioned constants and variables to be stored, so these must follow the last definition of a constant or a variable. On pass 1, a block of constants storage equal to the number of the left parenthesis in the program is saved. On pass 2, only the unique constants are stored in the constants block. Since the number of unique constants is usually less than the number of actual constants, especially in large programs, the pseudo-instruction constants is usually placed just before start at the end of the program.

The binary loader which MACRO punches at the beginning of the object program tape executes the jmp instruction at the end of the tape upon encounter. For this reason, it is usually wise to have the program begin at a hlt instruction so that depressing the continue switch on the console will start execution of the program after it is read-in.

B. Operation Of The MACRO Assembly Program

g. Read in MACRO - If a symbol punch containing symbols or MACRO definition needed by the program about to be assembled must be used, place the symbol punch in the reader and depress read-in.
b Place source program tape in the reader - The test word and test address toggle switches must all be zero.

c To begin pass 1 on the source program tape, depress continue - MACRO will stop shortly after encountering the stop code which follows the start at the end of the tape.

d If there are more tapes to be processed by pass 1, place the next tape in the reader and depress start. Continue this until all tapes to be processed on pass 1 have been processed. Multiple processing allows the source program to be on more than one tape. For example, routines need to be written only once, and the tapes processed with each program which uses the routine. Tapes processed in this fashion produce a single object tape from the multiple source tapes.

e To process the source tape(s) by pass 2, place the first tape in the reader and depress continue. MACRO will punch some blank tape, read some tape, punch the title at the beginning of the object tape in readable form followed by the binary input routine in read-in mode, and then begin punching the binary version of the program in blocks of 100 words (or less).

f MACRO will stop shortly after encountering the stop code following the start block as it did on pass 1. To process more tapes by pass 2, follow the same procedure as in 4.

g When all tapes to be processed by pass 2 have been processed, depress continue to punch the jump block at the end of the object tape. The assembly is now complete.

h To print and/or punch the symbols and/or MACRO definitions, and/or restore MACRO for processing another program, place the MACRO symbol package in the reader, set the desired sense switches, and depress the read-in switch. The sense switches have the following meaning to the MACRO Symbol Package.

<table>
<thead>
<tr>
<th>SENSE SWITCH</th>
<th>MEANING</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Symbol punch</td>
</tr>
<tr>
<td>2</td>
<td>Symbol print alphabetic</td>
</tr>
<tr>
<td>3</td>
<td>Symbol print numeric</td>
</tr>
<tr>
<td>4</td>
<td>Restore MACRO</td>
</tr>
</tbody>
</table>

24
The symbol punch routine will punch out a binary version of MACRO’s symbol and/or MACRO definition table. This symbol punch may be read into MACRO at a later date so that a program which refers to symbols defined in the just assembled program may be assembled, or symbolic corrections may be assembled for the present program. Also, by punching out the MACRO definitions, a person can collect a system tape of commonly used MACRO instructions without defining these in every source program.

The symbol punch may be read by DDT (Digital Debugging Tape) to permit symbolic debugging of a program.

If a symbol punch is requested, MACRO will punch a small amount of tape feed and wait for the title to be typed on the Flexowriter. The title may consist of any characters with the exception of tab and carriage return. The title may be terminated in three ways: (1) by a carriage return, which causes both the symbol table and MACRO definition table to be punched, (2) \rightarrow s, which causes only the symbol table to be punched, or (3) by \rightarrow m, which causes only the MACRO definition table to be punched.

The symbol print routines will print all symbols defined during the assembly of the program and will also print the limits of the constants storage area(s) used. If any symbols in MACRO’s permanent vocabulary were redefined, then MACRO will also print the original definition of these symbols.

Restore will restore MACRO, allowing the user to begin at step 2 to assemble another program.

C. Test Word Control Of MACRO

Occasionally, it becomes necessary to break the normal sequence of operation allowed by the use of the continue and start switches on the PDP-1 console. This may be done by using the test word to command MACRO. Whenever the start switch is depressed with the test address equal to zero, MACRO will examine the test word. The bits in the test word. The bits in the test word have the following meaning:

<table>
<thead>
<tr>
<th>BIT</th>
<th>MEANING</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>examine the remainder of the test word</td>
</tr>
<tr>
<td>1</td>
<td>Down - pass 1</td>
</tr>
<tr>
<td></td>
<td>Up - pass 2</td>
</tr>
<tr>
<td>2</td>
<td>Down - Reset initial address to zero</td>
</tr>
<tr>
<td></td>
<td>Up - Continuation, do not reset address</td>
</tr>
</tbody>
</table>
Bits 3 - 5 have meaning only on pass 2

3
    Up - punch object tape
    Down - don't punch

4
    Up - punch input routine
    Down - don't punch input routine

5
    Up - punch title
    Down - don't punch title

for example: to repeat pass 2 on a tape which has just been assembled set TW to 670000 and depress start.

MACRO normally checks the parity bit while reading and will indicate any parity errors detected. If sense switch 6 is up then the parity check is suppressed.

D. Error Stops During A MACRO Assembly

Upon detecting an error, MACRO will print out the following:

    aaa  bbbb  ccc  dddd  eee

aaa is the three letter code indicating the error. bbbb is the octal address at which the error occurred. ccc is the symbolic address at which the error occurred. dddd is the name the last pseudo-instruction encountered. In the case of an error caused by a symbol, eee will be that symbol. Following is a list of the error indications in MACRO:

ERROR          MEANING
    us  α

    An undefined symbol has been encountered by MACRO. α will indicate how the undefined symbol was being used. The value of the symbol will be taken as zero.

 α     Meaning
    a    In a pseudo-instruction argument
    w    In a word
    c    In a constant
    p    In a parameter assignment (assignment with an equal sign)
    m    In a MACRO instruction definition
l  In a location assignment
r  In a repeat instruction (the count)
s  In a start pseudo-instruction
d  In a dummy symbol assignment

mdt  Multiple definition of a tag. The definition of this address tag does not correspond to a previous definition. The symbol is not redefined.

mdd  Multiple definition in a dimension statement.

mdm  Multiple definition of a MACRO instruction. The name of this MACRO instruction conflicts in its first 6 characters with a pseudo-instruction or some other MACRO instruction name. The MACRO instruction will be redefined.

zpa  Illegal parameter assignment.

mdv  Multiple definition of a variable.

ilp  Parity error. The character is ignored.

ipi  An illegal pseudo-instruction. Ignore all characters preceding the next tab or carriage return.

ilr  Illegal repeat instruction, N is minus. Ignore.

ids  Illegal dummy symbol in the title of a MACRO instruction definition.

sce  Storage capacity exceeded. Assembly cannot proceed.
ilf  Illegal format.

tmc  Too many constants. Assembly cannot proceed.

tmv  Too many variables.

tmp  Too many parameters (dummy symbols).

Upon coming to a halt after an error printout on pass 1, start and continue have the same effect and will continue processing pass 1. Upon coming to a halt after an error printout on pass 2, start will continue processing pass 2 as before. If continue is depressed then pass 2 processing is continued with punching suppressed.

Setting test word bit 17 to a 1 has the same effect as pressing continue after each error printout.
PART IV

EXAMPLES

EXAMPLE 1: Octal \textbf{Integer} print Subroutine and \textbf{test}

/octal print subroutine
/call by jda opt with number in AC

\begin{verbatim}
poc=jda opt \hspace{1cm} /also call by poc
100/opt, 0 \hspace{1cm} /subroutine starts in 100
dap opx
law i 6
dac occ \hspace{1cm} /occ is a variable, will contain count
opc,
lac opt
ral 3s
dac opt
and (7 \hspace{1cm} /get first digit - 7 is a constant
sza 1 \hspace{1cm} /test for digit 0
law char r0 \hspace{1cm} /get concise code for 0
tcr 9s
tcr 9s \hspace{1cm} /put code in IO
tyo
isp occ \hspace{1cm} /more characters?
jmp opc \hspace{1cm} /yes
opx,
jmp . \hspace{1cm} /no, exit
\end{verbatim}

variables \hspace{1cm} /assign variables defined in opt here
tes,
lat \hspace{1cm} /test octal print
poc \hspace{1cm} /call octal print
lio (char r \hspace{1cm} /the char c.r. into the AC
tyo
hlt
jmpl tes \hspace{1cm} /print another number

constants \hspace{1cm} /all constants will be stored starting here

start tes \hspace{1cm} /this indicates the end of the
\hspace{1cm} /program, and will cause a jmp tes
\hspace{1cm} /to be punched at the end of the
\hspace{1cm} /binary tape.
\hspace{1cm} /start must be followed by a c.r. or
\hspace{1cm} /tab, and a stop code
EXAMPLE 2: punch 100 lines of tape feed, punch all that is printed on the typewriter until a carriage return, then punch 200 lines of tape feed and type OK on the typewriter.

example 2 - make a Friden writer

200/go,
  law i 100
  cli
  ppa
  add (1
  spa )
  jmp go 2
  lio (char r
  ) /1 is a constant

  tyo /character is c.r.
  go1,
  clf 1
  szf i 1
  jmp .-1
  tyi /listen for character to be typed
  dio tem
  lac tem
  add (r al
  ) /set up to calculate parity bit
  dac . 2
  law 5252
  xx
  and (200
  ) /mask parity bit
  ior tem
  rcr 9s
  rcr 9s
  /combine with character
  ppa
  law char r
  /rotate into IO for punching
  sas tem
  jmp go1

  go2,
  law i 200
  cli
  ppa
  add (1
  spa )
  jmp .-3
  go3,
  lio (flexo ok
  ) /feed 200 lines, this could be a subroutine
  ril 6s
  tyo /the three characters o,k,c.r. into the IO
  ril 6s
  tyo
  ril 6s
  tyo
  ril 6s
  tyo
  hlt
  jmp go

variables
constants
start go
EXAMPLE 3: the same as example 2, but with macros

/Macro definitions for big Friden writer

define
feed N
law 1 N          //definition of a macro to
cl1              //feed N lines of tape
ppa
add (1           //constants may be used within
spa              //a macro definition
jmp .-3
term

define
exchange         //only the first 6 characters count
rcr 9s
rcr 9s
term

define
print A          //A is the dummy parameter
lio (A           //whenever print is used
tyo              //the argument will be stored in the
term             //constants table

define
print3 A
lio (A
ril 6s
tyo
tyo
ril 6s
tyo
ril 6s
tyo
term
/program begins here

go,  feed 100
   print char r
    
/go1,
   clf 1
   szf i 1
   jmp .-1
   tyi _
   dio tem
   lac tem
   add (ral
   dac . 2
   law 5252
   hlt
   and (100
   ior tem
   exchange
   ppa
   law char r
   sas tem
   jmp go1

/go2,
   feed 200
   print3 flexo ok
   hlt
   jmp go

variables
constants
start go
Example 4: use of the dimension statement and the repeat instruction.

/a do nothing program

n=6
dimension abc(20),tb1(n+n),tb2(n+n+2)

/the above instructs Macro to save 20,14,16 octal
registers respectively for the variables abc,tb1,tb2

/generate 22 ppa instructions

a,    repeat 22, ppa

/generate a table of the integers 0,1,...,n-1

b,    repeat n,..-b

/generate a routine which will test for
/successive powers of two

tp2,    decimal
      z=1
      repeat 18, sad (z     jsp q     z=z+z
      hlt

octal

/the above repeat generates a series of instructions like
/sad (1
/jsp q
/sad (2
/jsp q
/sad (4
/jsp q
/etc., through the powers of 2 up to 400000 base 8
/notice that sad (z and jsp q are terminated by tabs
/and therefore are in the range of the repeat

/upon entry AC contains 2×P+tp2+1, where P is the power of 2

q,    sub (tp2+1       /AC now contains 2×P
sar 1s
dac pow    /deposit power of 2 in pow
hlt

variables
constants
start
## APPENDIX 1

**SYMBOLS IN MACRO’S PERMANENT VOCABULARY**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Code</th>
<th>Symbol</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>hlt</td>
<td>760400</td>
<td>lac</td>
<td>200000</td>
</tr>
<tr>
<td>xor</td>
<td>60000</td>
<td>jsp</td>
<td>620000</td>
</tr>
<tr>
<td>xct</td>
<td>100000</td>
<td>jmp</td>
<td>600000</td>
</tr>
<tr>
<td>tyo</td>
<td>730003</td>
<td>jfd</td>
<td>120000</td>
</tr>
<tr>
<td>tyi</td>
<td>720004</td>
<td>jda</td>
<td>170000</td>
</tr>
<tr>
<td>skp</td>
<td>640000</td>
<td>jsp</td>
<td>460000</td>
</tr>
<tr>
<td>szo</td>
<td>641000</td>
<td>iot</td>
<td>720000</td>
</tr>
<tr>
<td>szm</td>
<td>640500</td>
<td>ior</td>
<td>40000</td>
</tr>
<tr>
<td>szf</td>
<td>640000</td>
<td>idx</td>
<td>440000</td>
</tr>
<tr>
<td>sza</td>
<td>640100</td>
<td>i</td>
<td>10000</td>
</tr>
<tr>
<td>sub</td>
<td>420000</td>
<td>xx</td>
<td>760400</td>
</tr>
<tr>
<td>stf</td>
<td>760010</td>
<td>esm</td>
<td>720055</td>
</tr>
<tr>
<td>spq</td>
<td>650500</td>
<td>dzm</td>
<td>340000</td>
</tr>
<tr>
<td>spi</td>
<td>642000</td>
<td>dpy</td>
<td>730007</td>
</tr>
<tr>
<td>spa</td>
<td>640200</td>
<td>dis</td>
<td>560000</td>
</tr>
<tr>
<td>sma</td>
<td>640400</td>
<td>dip</td>
<td>300000</td>
</tr>
<tr>
<td>ksz</td>
<td>640000</td>
<td>dio</td>
<td>320000</td>
</tr>
<tr>
<td>sir</td>
<td>676000</td>
<td>dap</td>
<td>260000</td>
</tr>
<tr>
<td>sil</td>
<td>666000</td>
<td>dac</td>
<td>240000</td>
</tr>
<tr>
<td>scr</td>
<td>677000</td>
<td>cma</td>
<td>761000</td>
</tr>
<tr>
<td>sas</td>
<td>520000</td>
<td>clo</td>
<td>651600</td>
</tr>
<tr>
<td>sar</td>
<td>675000</td>
<td>cli</td>
<td>764000</td>
</tr>
<tr>
<td>sal</td>
<td>665000</td>
<td>clf</td>
<td>760000</td>
</tr>
<tr>
<td>sad</td>
<td>500000</td>
<td>clc</td>
<td>761200</td>
</tr>
<tr>
<td>rrb</td>
<td>720030</td>
<td>cla</td>
<td>760200</td>
</tr>
<tr>
<td>rpb</td>
<td>730002</td>
<td>cks</td>
<td>720033</td>
</tr>
<tr>
<td>rpa</td>
<td>730001</td>
<td>cfd</td>
<td>720074</td>
</tr>
<tr>
<td>rir</td>
<td>672000</td>
<td>cdf</td>
<td>720074</td>
</tr>
<tr>
<td>ril</td>
<td>662000</td>
<td>cal</td>
<td>160000</td>
</tr>
<tr>
<td>rcr</td>
<td>673000</td>
<td>and</td>
<td>20000</td>
</tr>
<tr>
<td>rcl</td>
<td>663000</td>
<td>add</td>
<td>400000</td>
</tr>
<tr>
<td>rar</td>
<td>671000</td>
<td>1s</td>
<td>1</td>
</tr>
<tr>
<td>ral</td>
<td>661000</td>
<td>2s</td>
<td>3</td>
</tr>
<tr>
<td>ppb</td>
<td>730006</td>
<td>3s</td>
<td>7</td>
</tr>
<tr>
<td>ppa</td>
<td>730005</td>
<td>4s</td>
<td>17</td>
</tr>
<tr>
<td>opr</td>
<td>760000</td>
<td>5s</td>
<td>37</td>
</tr>
<tr>
<td>nop</td>
<td>760000</td>
<td>6s</td>
<td>77</td>
</tr>
<tr>
<td>mus</td>
<td>540000</td>
<td>7s</td>
<td>177</td>
</tr>
<tr>
<td>lsm</td>
<td>720054</td>
<td>8s</td>
<td>377</td>
</tr>
<tr>
<td>lio</td>
<td>220000</td>
<td>9s</td>
<td>777</td>
</tr>
<tr>
<td>law</td>
<td>700000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>lat</td>
<td>762200</td>
<td></td>
<td></td>
</tr>
<tr>
<td>lap</td>
<td>760300</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>