DATE May 24, 1962

SUBJECT MACRO ASSEMBLY PROGRAM
TO PDP Distribution List

ABSTRACT This memo is a description of the PDP-1 version of the Macro Assembly Program, which was written at MIT.
A detailed writeup (now being done at MIT) will be available shortly. This memo is intended as an outline for people wishing to use Macro in the meantime, and does not attempt to describe the features in complete detail.
Macro is a two pass assembly system. Symbols are assigned and storage allocated during pass 1. The statements of the Macro language tape are translated into binary on pass 2, and punched out on tape. This tape is the binary version of the program, and need only be read in to be run.
Section I of this memo describes the language available to the user of Macro.
Section II contains operating instructions, and an explanation of the English tape format.
Section III contains three sample programs.

FROM Harrison Morse

APPROVED BY

## SECTION I - Description of the Macro Language

### A. Character Set

digits 0-9
lower case letters a-z

| Punctuation Characters | | Meaning |
|---|---|---|
| + | plus | add values |
| - | minus | subtract values |
| ␣ | space | add values |
| ( | left parenthesis | enclose constant word |
| ) | right parenthesis | enclose constant word |
| | over bar | define variable symbol |
| . | period | has value of current address |
| , | comma | assign address tag; separate agreements of psuedo-instructions |
| = | equal sign | assign symbol on left of = |
| / | slash | begin comment; set current address |
| ↵ | carriage return | termination character |
| →\| | tab | termination character |

The symbols ␣ , ↵ , →| are used in the memo to represent characters which are invisible on a printout.

### B. Symbols

Symbols are delimited by any of the punctuation characters mentioned in A.

1. Number Symbol

   Any sequence of digits delimited on the left and right by punctuation characters. The value of a number symbol is the number itself.

2. Value Symbols

   1, 2, or 3 digits or letters at least one of which is a letter. Each value symbol has associated with it a number which will fit in one PDP-1 word, (called the value). In particular, address tags have as their values the addresses at which they were assigned, and instruction mnemonics have as their values the instruction codes.

   Address tags are assigned with a comma (i.e., abc,) while other symbols may be assigned with an equal sign (i.e., lac=200000 , c=-6).

   Symbol values are combined by Macro without regarding what they represent.

3. Psuedo-instruction Symbols

Psuedo-instruction symbols are 4 or more letters or digits, at least one of which is a letter. Although differentiated from each other by the first six characters, psuedo-instruction names may be abbreviated to four characters, as long as the first three are not identical with the first three of any other psuedo-instruction. Macro has 13 permanent psuedo-instructions and in addition, the names of all Macro instructions defined by the programmer become psuedo-instructions. (See Appendix)

Psuedo-instructions in Macro's permanent vocabulary:

| | |
|---|---|
| start | indicates end of an English tape |
| decimal | all following numbers will be converted as decimal |
| octal | all following numbers will be converted as octal |
| constants | allocates constants storage |
| variable | allocates variable storage |
| dimension | allocates a block for the indicated variables in the variable storage |
| flexo [illegible] | input 3 concise characters |
| character [illegible] | input 1 concise character |
| text | input a string of concise characters |
| repeat N,A | repeat the assembly of A, N times |
| expunge | erase all symbols from Macro's symbol table |
| define | initiate a Macro instruction definition |
| terminate | terminate a Macro instruction definition |

Throughout this memo, number symbols will be called numbers, value symbols (i.e., 1, 2, or 3 character symbols) will be referred to as symbols, and psuedo-instruction symbols will be called psuedo-instructions.

Examples:

| | |
|---|---|
| numbers | 1 |
| | 243 |
| | 200000 |
| symbols | a |
| | 1a |
| | abc |
| | 12c |
| | z21 |
| psuedo-instructions | start |
| | 123abc |
| | name |
| | name1 |

C. Syllables

A syllable is a symbol, a number, the punctuation character period (.), the flexo input psuedo-instruction flexo or char, or a constant (a word enclosed in parenthesis).

Examples of syllables:
a1
100
1z2
flex abc
flexo now
(add a+1)

D. Words

A word is a string of syllables connected by the arithmetic operators plus, minus, or space, delimited on the left by tab, carriage return, left parenthesis, or equal sign; and on the right by a tab or carriage return. A word may be a single number of symbol so delimited, or a string of symbols connected by the arithmetic operators. If the word is delimited on the left by an equal sign then the symbol to the left of the equal sign is assigned a value equal to that of the word. Otherwise, the word is a storage word and will become part of the binary version of the program being assembled. The arithmetic operators plus and space are equivalent and mean add, while the operator minus means subtract.

Examples of words:
saz ↵
lac a ↵
1000-20 ↵
add b+2 ↵
jmp .-2 ↵
a+b-c-2 ↵
lac (add a+1) ↵

E. The character slash /

Solidus has two meanings. If immediately preceded by a tab or carriage return then slash initiates a comment, which is terminated by the next tab or carriage return. If slash is preceded by a word, then the address part of the word indicates to Macro the address into which the next instruction or data word will go. Normally, Macro translates the first instruction or data word into register 4, and succeeding instructions or data words into succeeding registers. If the programmer wishes to break this sequence or wishes to start translating into some register other than 4, then a slash may be used to set the new address.

F. Indirect Addressing

Indirect addressing is indicated by the symbol i which has the value 10,000.

Example: lac i abc

G. The character .

The character . has as its value the current address.

Example: law . is equivalent to a, law a

H. Psuedo-instructions

1. Flexo input psuedo-instructions

flexo Δ

The psuedo-instruction flexo causes the concise codes for the three characters following the space to be read into one word and this word taken as the value of the syllable. The concise code for the character ... will go into bits 12-17 of the word, ... into bits 6-11 and ... into bits 0-5.

Example: flexo Δ boy
flexo Δ ...

character Δ

The psuedo-instruction character causes the concise character ... to be read into the right, middle or left six bits of the word, the position determined by z being r, m or l.

Example: character Δ r0
char Δ ma
law char r0 is equivalent to law 20

text Δ A ... A

The psuedo-instruction text will read the flexo code for the characters starting with the second character following the space and terminating with the next occurence of the first character following the space. The characters are stored three to a word as would be done by repeated uses of the psuedo-instruction flexo. The delimiter A may be any character and neither occurence of A is included in the string.

Example: text Δ. The codes for the characters in this string will be stored in memory, three to a register.

2. Constants

The Macro Assembly System has available a facility by which the program constants may be automatically stored. A constant must follow the rules for a word and is delimited on the left by a left parenthesis. The right delimiter may be a right parenthesis, carriage return, or tab. The value of the syllable ( ) is the address of the register containing . The constant will be stored in a constants block, and the address of will replace ( ). The occurence of the Psuedo-instruction constants in the program indicates to Macro where these constants are to be stored.

Examples of the use of constants:

```
add (1)
lac (add z 1)
lio (flexo abc)
lac (-760000
one = (1)
```

3. Variables

Macro has a facility similar to the constants facility for the automatic assignment of variable storage. If the first occurence of a symbol contains an over barred character, then this symbol is defined as a variable. The first occurence of a variable must not be within a Macro instruction definition. To avoid this, preceed the first use of the variable within a Macro definition with = . All variables will be assigned locations in a block beginning at the occurence of the psuedo-instruction variables.

4. Dimension

dimension (L ), (L ), . . ., (L )

The psuedo-instruction dimension allows the programmer to assign a block of storage to each variable , , . . ., . The blocks will have L , L , . . ., L registers respectively, and will be assigned in the variable storage. As the use of a symbol in a dimension state defines the variable, the first occurence in the program need not be over barred.

Examples of the use of variables and dimension:

```
dimension abc (20), zef (20 + 1)
lac az
add 1bc
```

5. Repeat

The psuedo-instruction repeat N, A will cause the assembly of the statement A to be repeated N times. A is the statement (s) delimited on the left by the comma and

on the right by the first carriage return following the comma. N must follow the rules for a word and must be separated from repeat by a space. If A is a statement which causes words to be stored in the program, then N must have the same value on pass 2 as it did on pass 1. If N is negative then the repeat instruction is illegal.

6. Start

The psuedo-instruction start indicates the end of the English tape. start A must be followed by a carriage return and a stop code. A is the address at which execution of the program is to begin, and causes a jmp A instruction punched at the end of the binary tape on pass 2.

7. Expunge

expunge will cause all symbols to be erased from Macro's symbol table on pass 1. On pass 2, expunge has no effect.

8. Decimal

The psuedo-instruction decimal tells Macro to take all numbers as decimal.

9. Octal

The psuedo-instruction octal tells Macro to take all numbers as octal.

I. Macro Instructions

A Macro instruction is a sequence of instructions which has been given a name by the programmer. He may then cause this group of instructions to be assembled into his program by calling the Macro instruction by name.

1. Defining a Macro instruction

A Macro instruction definition is initiated by the psuedo-instruction define and terminated by the psuedo-instruction terminate. Example:

define

NAME $\alpha, \beta, \ldots, \chi$

(Body of definition)

terminate

The name of the Macro instruction must obey the rules for psuedo-instruction symbols and must not conflict with any other psuedo-instruction in Macro's vocabulary (i.e., its first six letters must not be the same as the first six of any other psuedo-instruction). The body of the Macro definition consists of the instructions the programmer wishes to be assembled at the point the Macro instruction is called. The use of the punctuation character slash to set the current address within a Macro definition is illegal. Assigning a symbol (other than a dummy symbol) with an equal sign within a Macro definition is also illegal.

The dummy symbols [illegible], [illegible], . . ., [illegible] following the name indicates to Macro that the actual values for these symbols will be assigned at call time. A dummy symbol must obey the rules for value symbols (I.B.2) and in addition, must have at least one character in upper case. The dummy symbols may be used within the Macro definition in the same manner that ordinary symbols are, though again a character must be in upper case. Dummy symbols may be manipulated and additional dummy symbols assigned within a Macro definition by use of the equal sign.

When the Macro instruction is called in the program, the value of the word used in place of the dummy symbol [illegible] (the first argument) will replace each use of the dummy symbol [illegible] within the definition. Similarly, the value of the word which replaces the dummy symbol [illegible] (the second argument) will replace each occurence of the dummy symbol [illegible] within the definition, etc. The arguments are separated by commas. Other Macro instructions may be used freely within the Macro definition if they have been previously defined. Address tags may be assigned within a Macro definition, but the value so assigned is relative to the first instruction of the Macro. For this reason, the symbol R used within the Macro definition has the value (while the Macro is being assembled) of the address of the first instruction of the Macro, and may be used within the Macro instruction definition.

2. Calling the Macro instruction

Following is an example of a Macro instruction to feed N lines of tape:

```
define   feed N
         law i N
         cli
         ppa
         add (1)
         spa
         jmp .-3
         terminate
```

Whenever the name of the Macro instruction feed is used in the program, the six instructions contained in definition will be assembled at that point, with N replaced by the number in the call.

To assemble the instructions to feed 100 lines of tape, one could now write:

```
feed 100
```

## SECTION II - Use of the Macro Assembly System

### A. English Tape Format

Each English 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 preceeded by any character other than carriage return. On the English 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 binary version of the program, they may be placed anywhere in the program. The psuedo-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 psuedo-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 binary 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

1. Read in Macro

   a. 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.

2. Place English tape in the reader

   The test word and test address toggle switches must all be zero.

3. To begin pass 1 on the English tape, depress continue

   Macro will stop shortly after encountering the stop code which follows the start at the end of the English tape.

4. 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 English program to be on more than one tape. For example, routines need be written only once, and the tapes processed with each program which uses the routine. Tapes processed in this fashion produce a single binary tape from the multiple English tapes.

5. To process the English 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 binary 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.

6. 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.

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

8. 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.

| SENSE SWITCH | MEANING |
|---|---|
| 1 | Symbol punch |
| 2 | Symbol print alphabetic |
| 3 | Symbol print numeric |
| 4 | Restore Macro |

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 English Program.

The symbol punch may be read by DDT 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) by →| s , which causes only the symbol table to be punched, or (3) by →| 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

Occassionally, 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 have the following meaning:

| BIT | MEANING |
|---|---|
| 0 | examine the remainder of the test word |
| 1 | Down - pass 1<br>Up - pass 2 |
| 2 | Down - Reset initial address to zero<br>Up - Continuation, do not reset address |
| Bits 3 - 5 have meaning only on pass 2 | |
| 3 | Up - punch binary tape<br>Down - don't punch |
| 4 | Up - punch input routine<br>Down - punch jmp 7751 |
| 5 | Up - punch title<br>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 Macro will not check the parity bit.

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 occured. ccc is the symbolic address at which the error occured. dddd is the name of the last psuedo-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 \x | An undefined symbol has been encountered by Macro. \x will indicate how the undefined symbol was being used. The value of the symbol will be taken as zero. |

| \x | Meaning |
|---|---|
| a | In a psuedo-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 psuedo-instruction |

| | |
|---|---|
| mdt | Multiple definition of a tag. The definition of this address tag does 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 psuedo-instruction or some other Macro instruction name. The Macro instruction will be redefined. |

| | |
|---|---|
| ilp | Parity error. The character is ignored. |
| ipi | An illegal psuedo-instruction. Ignore all characters preceeding 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. |
| tmm | Too many Macro instructions defined. Assembly cannot proceed. |
| ilf | Illegal format. |
| tmc | Too many constants. Assembly cannot proceed. |

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.

## SECTION III - Examples

Example 1: Octal Interger print Subroutine and test

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

            poc=jda opt                 /also call by poc

100/opt,    0           /subroutine starts in 100
            dap opx
            law i 6
            dac occ     /occ is a variable, will contain th count
opc,        lac opt
            ral 7
            dac opt
            and (7      /get first digit - 7 is a constant
            sza i       /test for digit 0
            law char r0                 /get concise code for 0
            rcr 777
            rcr 777     /put code in IO
            tyo
            isp occ     /more characters?
            jmp opc     /yes
opx,        jmp .       /no, exit

variables               /assign variables defined in opt here

tes,        lat         /test octal print
            poc         /call octal print
            lio (char r
                        /the char c.r. into the AC
            tyo
            hlt
            jmp tes     /print another number

constants               /all constants will be stored starting here

start tes   /this indicates the end of the
                        /program, and will cause a jmp tes
                        /to be punched at the end of the
                        /binary tape.
                        /start must be followed by a c.r. or
                        /tab, and a stop code
```

Example 2: punch 100 lines of tape feed, punch
all thats 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      /1 is a constant
            spa
            jmp go 2    /punch 100 lines of tape feed
            lio (char r
                        /character is c.r.
            tyo
go1,        clf 1       /listen for character to be typed
            szf i 1     /skip on not flag 1
            jmp .-1     /listen loop
            tyi         /get typed character
            dio tem     /a variable
            lac tem
            add (ral    /set up to calculate parity bit
            dac . 2
            law 5252
            hlt         /this will be modified
            and (200    /mask parity bit
            ior tem     /combine with character
            rcr 777
            rcr 777     /rotate into IO for punching
            ppa
            law char r
                        /character is c.r.
            sas tem     /test for carriage return
            jmp go1
go2,        law i 200
            cli
            ppa
            add (1
            spa
            jmp .-3     /feed 200 lines, this could be a subroutine
go3,        lio (flexo ok
                        /the three characters o,k,c.r. into the IO
            tyo
            rir 77
            tyo
            rir 77
            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 i N      /definition of a macro to
            cli          /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 777
            rcr 777
            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
            print A      /macros may be used within macros
            rir 77
            tyo
            rir 77
            tyo
            term

/actual program is on the next page
```

```
/this program will generate exactly the same code
/as did example 2

/program begins here

go,         feed 100
            print char r
                        /again char is c.r.
go1,        clf 1
            szf i 1
            jmp .-1
            tyi
            dio tem
            lac tem
            add (ral
            dac . 2
            law 5252
            hlt
            and (200
            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 sample program

            n=6

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

/the above instructs Macro to save 20,14,16
/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

/the above repeat generates a series of instructions
/sad (1
/jsp g
/sad (2
/jsp g
/sad (4
/jsp g
/etc through the powers of 2 up to 400000

            octal

/upon entry AC contains 2xP+c+1, where P is the power of 2

g,          sub (c+1    /AC now contains 2xP
            sar 1
            dac pow     /deposit power of 2 in pow
            hlt

constants
start
```

## APPENDIX I - Symbols in Macro's Permanent Vocabulary

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