LA-3792

MASTER

295
12-18-67

LOS ALAMOS SCIENTIFIC LABORATORY
of the
University of California
LOS ALAMOS • NEW MEXICO

**MADMAC, A MADCAP Version of
DEC Code MACRO-8 for Assembling
PDP-8 Codes on MANIAC**

UNITED STATES
ATOMIC ENERGY COMMISSION
CONTRACT W-7405-ENG. 36

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency Thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

LEGAL NOTICE

This report was prepared as an account of Government sponsored work. Neither the United States, nor the Commission, nor any person acting on behalf of the Commission:

A. Makes any warranty or representation, expressed or implied, with respect to the accuracy, completeness, or usefulness of the information contained in this report, or that the use of any information, apparatus, method, or process disclosed in this report may not infringe privately owned rights; or

B. Assumes any liabilities with respect to the use of, or for damages resulting from the use of any information, apparatus, method, or process disclosed in this report.

As used in the above, "person acting on behalf of the Commission" includes any employee or contractor of the Commission, or employee of such contractor, to the extent that such employee or contractor of the Commission, or employee of such contractor prepares, disseminates, or provides access to, any information pursuant to his employment or contract with the Commission, or his employment with such contractor.

This report expresses the opinions of the author or authors and does not necessarily reflect the opinions or views of the Los Alamos Scientific Laboratory.

Printed in the United States of America. Available from
Clearinghouse for Federal Scientific and Technical Information
National Bureau of Standards, U. S. Department of Commerce
Springfield, Virginia 22151

Price: Printed Copy \$3.00; Microfiche \$0.65

LOS ALAMOS SCIENTIFIC LABORATORY
of the
University of California
LOS ALAMOS • NEW MEXICO

Report written: September 22, 1967

Report distributed: November 28, 1967

MADMAC, A MADCAP Version of
DEC Code MACRO-8 for Assembling
PDP-8 Codes on MANIAC

by

Allen C. Larson

LEGAL NOTICE

This report was prepared as an account of Government sponsored work. Neither the United States, nor the Commission, nor any person acting on behalf of the Commission:

A. Makes any warranty or representation, expressed or implied, with respect to the accuracy, completeness, or usefulness of the information contained in this report, or that the use of any information, apparatus, method, or process disclosed in this report may not infringe privately owned rights; or

B. Assumes any liabilities with respect to the use of, or for damages resulting from the use of any information, apparatus, method, or process disclosed in this report.

As used in the above, "person acting on behalf of the Commission" includes any employee or contractor of the Commission, or employee of such contractor, to the extent that such employee or contractor of the Commission, or employee of such contractor prepares, disseminates, or provides access to, any information pursuant to his employment or contract with the Commission, or his employment with such contractor.

**THIS PAGE
WAS INTENTIONALLY
LEFT BLANK**

MADMAC, A MADCAP VERSION OF DEC CODE MACRO-8
FOR ASSEMBLING PDP-8 CODES ON MANIAC

by

Allen C. Larson

ABSTRACT

An assembler for PDP-8 codes written in a subset of the DEC-MACRO-8 language has been entirely written in the Maniac II compiler language MAUCAP. This assembler will accept input from cards or paper tape. The paper tape can either be punched in the Maniac character code or in the ASCII character code. The major difference between MACRO-8 and MADMAC is that MADMAC does not have the macro pseudo-op.

INTRODUCTION

A Digital Equipment Corporation PDP-8 computer has been purchased to control an X-ray diffraction unit at Los Alamos Scientific Laboratory. The assemblers provided by the manufacturer require that the symbolic code being assembled be read in three times to provide both the binary code and a listing. This is a time-consuming process because the only input to this PDP-8, as it exists now, is an ASR-33 teletype unit that has a character read rate of only 10 characters/second. Dr. W. Busing at ORNL has written a FORTRAN and machine language assembler (PALFTN) for use on the CDC-1604. A copy of this code was obtained and used as the starting point for writing MADMAC, a MADCAP version of MACRO-8, the most sophisticated assembler available for the PDP-8.

In this report it is assumed that the reader is familiar with the PAL-III and MACRO-8 languages as described in the manuals published by the Digital Equipment Corporation and that he has a working knowledge of the MADCAP language.

DESCRIPTION

The MADCAP language is ideal for writing purely

logical codes such as an assembler. The language provides for manipulation and examination of bits, which makes a character-by-character scan of a character string a straightforward process.

The basic premise of MADMAC is that it will duplicate all features of PAL-III and MACRO-8 except the use of macro definitions. (PALFTN will accept only a subset of the PAL-III language.) A few additions, such as the pseudo-ops FLTG and DUBL in literal definitions, have been made to the MADMAC language beyond those described as available in MACRO-8.

MADMAC reads the symbolic code one line at a time, stores the line for a complete scan and listing on the second pass, and then performs an initial scan of the line to determine whether there are any symbols defined by the statements on the line. If so, it defines them and adds them to the symbol table. After the complete symbolic code has been read in, all symbols defined, and no errors detected, MADMAC rescans each statement, creates the desired code, and prints a listing. When the second pass is completed the assembler lists the symbols that were defined by the code and gives their values. The literals are then listed with their locations, and, finally, the

new code is punched in a format suitable for the PDP-8 binary loader.

The initial scan of a statement is halted when a symbol not terminated by a comma or equal sign is detected. Termination of the scan in the second pass depends on the type (see below) of the first symbol in the statement. The scan continues to the end of the statement if the symbol is type 1 or 2, is terminated immediately if type 4, is terminated by the first space encountered after a symbol defining the address has been read if type 8, and is terminated by the first space encountered if type 0. The end of the statement is defined as a semicolon, a slash, or the end of the line. On cards, "\$" indicates the end of a statement. Thus more than one statement can be on a card. In addition to the above conditions for terminating a scan, the scan of a type 8 statement will be terminated after the first literal in the statement has been evaluated. Comments may follow a slash. The end of the code is indicated by the symbol \$ appearing as the first non-blank character in the statement.

Symbols can be constructed from any characters in the chosen character set (keypunch, ASCII, etc.). They must not begin with a number smaller than 8, and will be terminated by the first control character encountered. The control characters and their interpretations are:

,	defines a location
+	adds two symbols
-	subtracts two symbols, -N is treated as 0-N
=	defines a symbol
(	defines a current page literal
[or ((	defines a zero page literal
space	causes the preceeding and following symbols to be combined by the inclusive OR
/	defines the start of a comment
;	defines the end of a statement

Operations on symbols are determined by the character terminating the symbol. Spaces and tabs between symbols other than those terminating a symbol are ignored.

The code has 15 error symbols. These are defined in Table I. Only one error indication, the last one detected, is printed for any statement. In some unusual cases the scan is terminated when the error is detected.

Symbols in the symbol table are defined as belonging to one of 5 classes determined by the modifications to be allowed in constructing the code. These classes are described in Table II. The rules given in Table II do not apply to symbols defined with an equal sign.

TABLE I. ERROR TAGS, THEIR MEANINGS AND RESULTS

Error Tag	Meaning	Result
uds	undefined symbol	It was given a value of 0 (except ignored if after *).
rds	redefined symbol	The new value was stored.
nop	an operator was followed only by spaces or comment	If after * or = the whole line is ignored; if after + or - a 0 was probably used.
iop	illegal operation	?
itr	address missing on an order requiring an address	The address was set to page 0, cell 0.
ich	illegal character in an octal number on a literal	The value of the symbol was computed incorrectly.
nuc	illegal combination of permanent symbols requested	Scan of statement terminated; first symbol used for the code.
iae	indirect address error (an I or z was followed by nothing)	An address of 0 was uncd.
gl2	group 1 and group 2 operate instructions should not be combined	They were combined.
pge	illegal page reference	Indirect page reference to symbols on a page other than current or zero page.
cwc	code word changed	A previously defined word of code was overwritten.
2.s	2 periods were encountered in an FLTG number	The second period terminated the number.
+a-	a minus sign was followed by a plus sign	The plus sign was ignored.

TABLE II. CLASSIFICATION OF SYMBOLS

Symbol Type	Description and allowed use
0	All programmer defined symbols are of this type. They can be used in any manner the programmer wishes.
1	Group 1 operate instructions. They can only be combined with other group 1 operate instructions.
2	Group 2 operate instructions. These can only be combined with other group 2 operate instructions.
4	Memory reference instructions. All should have addresses, otherwise the address will be 0.
8	Instructions that cannot be modified or combined. The I/O orders fall in this group.

A listing of the MADCAP code for MADMAC is given in the Appendix. MADMAC is different from MACRO-3 in several important respects. These are:

MADMAC does not recognize macros.

MADMAC does not allow nested literals.

MADMAC assumes a literal is decimal unless told otherwise by an O following the parentheses (or following a minus sign if negative).

MADMAC literals must be numbers, not symbols.

MADMAC literals may be floating point numbers if defined by FLTG or double precision if defined by DUBL. The radix control in MADMAC is always octal.

MADMAC does not need EXPUNGE.

MADMAC does not recognize PAUSE.

MADMAC, at present, only recognizes +, - and space as operators in forming expressions.

OPERATING INSTRUCTIONS

Sense lights 0, 1, 2, 3, 4, 5, and 6 are used to indicate the location and type of input, to control code punching, and to control code processing and listing. The sense lights will be set before reading of the code begins and may be set for the next code as soon as reading has started. The indicated control for the sense lights is:

- 0 on for card decks.
- 1 on for "Japanese" flexo paper tape.
- 2 on for ASCII paper tape.
- 3 off to read next code after punch out without stopping.
- 4 on to punch out only after ~~99~~ The computer will stop at of-100 on this option even if 3 is off.

5 on to perform pass 2 even though errors were detected in pass 1.

6 on to include the permanent symbols in the symbol table listing.

The location of secondary codes to be assembled into one program can be further controlled by a character following \$ in the last card in the deck. The character 0 indicates that the next code is on cards; a 1, on paper tape in "Japanese" flexo code; a 2, on paper tape in ASCII code; and a blank indicates that the sense lights will be examined to locate the next code. The secondary codes must be complete in themselves, although they may have the page and location counter preset.

The FDP-3 memory load and all location counters are reset after a code has been punched out. The complete permanent symbol table can be changed only when a request for its location is printed on the MK-III. If the track is typed as 0 0 0 0, a new symbol table will be read in from cards; if the track is typed as a number less than 7500, track 7598 will be used. Track numbers larger than 7599 are illegal. Individual symbols in the permanent symbol table can be redefined with the equal sign. An error "rds" will then be printed in the margin, and sense 5 must be on to assemble the code (sense 5 deletes the test on errors detected in the first pass, which would provide only a listing of the symbolic deck and the first pass errors).

The MADMAC code, as a binary tape including a "standard" permanent symbol table, is available from the author.

APPENDIX

Listing of the MADCAP code

MADMAC

All quotation marks except those enclosing comments were in red in the original listing. All other symbols that were red in the original listing are underlined here. For explanation of the red symbols, see the MADCAP manual.

"Madcap Code to assemble PDP-8 Codes"

```

(word-set)Symb0 to 500
(word-set)Octe0 to 500
(80 characters)Stm0 to 2000
(word-set)decimal()
(word-set)symbol()
(real)search()
(string)rdasci()
(word-set)find()
(real)equatr()
(real)equato()
(word-set)octal()
(word-set)litr0 to 4095
(word-set)A, B, Cw, Smb, S, S1
(word-set)flt0 to 4095
(word-set)dblp()
Err0 to 2000
(4 characters)Erms0 to 15 is [ " ", "uds", "rds", "nop", "lep", "itr"
cont.      , "ich", "nc", "iae", "gl2", "pge", "cwc", "2.s", " ", "+a-", " "]
t = 0
(word-set)Cd0 to 4095
Lt, Lc, Ld0 to 45
new page
print: "set sense 0 on for cards, 1 on for JAP flexo tape,"
cont.      " 2 on for ASCII code"
print: "set sense 3 on to punch after $$ only"
print: "if sense 3 is off, turn sense 4 on to"
cont.      " stop after code is punched"
print: "all above sense lights may be reset for"
cont.      " the next code after reading starts"
print: "sense 5 on to ignore pass 1 errors, can be set at anytime"
print: "sense 6 on to list the permanent symbols in symbol table "
```

```

stop #1
#3  Flag = 0
    read console by "Symbol table is on track dddd": x
    if x > 0, go to #10
    i = 0
#1  new card
    read card by "c6(c6)2":Symb1, A, B
    Octeq1=(Ax12)+(Bx4)+(16)
    Symb1 = Symb1x3f3f3f3f3f3f
    if Symb1 = 101010101010, go to #2
    i = i+1
    go to #1
#2  type:"if symbol table is to be written on disk"
    read console by "type initial track no x": x
    if x > 7500
        write 2 tracks x, #088
        i
        for j = 0 to 1
            Symbj, Octeqj
        go to #11
#10  if x < 7500, then x = 7598
    read 2 tracks x, #088
    i
    for j = 0 to 1
        Symbj, Octeqj
#11  Pertab = 1                                "Permanent table length"
#13  for i = 0 to 45: Lt1, Lc1 = 0
    Page = 1
    for i = 0 to 4096: Cd1 = (20)
#12  Itab = Pertab
    Terr, Ng = 0
    for i = 0 to 45: Ld1 = Lc1
    Dpage = Page
    End = 0

```

```

Lite = bank
if Flag = 0: Type = ordinal(Litex3)-1
n = 0                                     "# of statements read"
#20  if Type = 0
      new card
      read card: Stmtn
      go to #25
    if Type = 1
      read: Stmtn
      go to #25
    Stmtn = rdasci()
#25  (672 elements)Strng = Stmtn
      Errn = 0
      Pass = 1
      j,t = 0
#26  execute iscan()
      if Errn > 0: Ierr = Ierr+1
      if End > 0, go to #27
      if j > 0
        n = n+1
        go to #20
      if Smbx3f3f3f3f3f3f3f - 0ff32d201010
        Lcpage = Lcpage+2
      if Smbx3f3f3f3f3f3f3f = 0d2e0b251010
        Lcpage = Lcpage+1
      execute wrapup()
      if j > 0, go to #26
      n = n+1
      go to #20
#27  if End > 1, go to #700
      if sense 5 is on, go to #100
      if Ierr > 0
        Ipg = 0
        for j = 0 to n-1

```

```

        if (j(mod 60)) = 0
            new page
            print: date
            print by "Error tag(,)30Statement(,)30Page x": Ipg
            Ipg = Ipg+1
            print by "(,)3m3(,)6m80":ErmsErrj, Stmtj
            print by "/x4,errors were detected in pass 1": Ierr
            go to #13

#100  n = -1                                "Second pass"
      End = 0
      Pass = 2
      Ipg = 1
      for i = 0 to 32: Lci = Ldi
      Page = Dpage
      Line = 0
      Fpt, Dbl, Cdc = 0

#110  n = n+1
      Strng = Stmti1
      Sc = 0
      Lf = 1
      j = 0

#115  Ow = 0
      t = 0
      if Fpt > 0
          execute spaces()
          A = Strngx8
          if ordinal(A) > 9
              Fpt = 0
              go to #116
          Cdc = 36
          Ow = fltgpt()
          go to #400
      if Dbl > 0
          execute spaces()

```

```

    A = Strngx6
    if ordinal(A) > 9
        Dbl = 0
        go to #116
    Ow = dblp()
    Cdc = 24
    go to #400
#116  t = 0
    execute iscan()
    if j > 0, go to #500
    if End > 0, go to #600
    if Smb = 101010101010
        if t < 4
            Errn = 4
            go to #500
        r = t
        execute spaces
        Ow = find(symbol())
        if r = 5: Ow = subset(ordinal(12_Ow)+1)
        if Ng > 0
            Errn = 1
            go to #400
    if Smbx3f3f3f3f3f3f=0f252d201010 "fltg"
        Fpt = 1
        Cdc = 36
        Ow = fltgpt()
        go to #400
    if Smbx3f3f3f3f3f3f=0d2e0b251010 "dubl"
        Dbl = 1
        Cdc = 24
        Ow = dblp()
        go to #400
    Ow = find(Smb)
    if Ng > 0

```

```

Errn = 1                                "undefined symbol"
go to #400                                "store and print"
if Ow > (19), go to #400                  "no modification allowed"
if Ow > (18)
    execute spaces()
    if j > 0
        Errn = 5
        go to #400
    if Strngx16 = 4949
        Strng = Strng-(8)
        go to #1160
    if Strngx7 = 49                      "a current page literal"
        Ow = Ow+80
        Lpg = Page
        go to #117
    if Strngx7 = 44                      "a zero page literal"
#1160    Ow = Ow+ffffffffffff7f
        Lpg = 0
#117    A = literal()
#1171    for j = 1 to LtLpg
        if A = Cd128(Lpg+1)-j
            if A-(12) = 1, go to #118
            if A-(24) = 1
                j = j+1
                go to #118
            if A-(36) = 1
                j = j+2
                go to #118
#1170    LtLpg = LtLpg+1
        j = LtLpg
        if Cd128(Lpg+1)-j ≠ (20)
            Errn = 11
        Cd128(Lpg+1)-j = A

```

```

        A = A-(12)
        if A1, go to #1170
#118      Ow = Ow+subset(128-j)
          go to #400
#120      S = symbol()
          if Sx3f3f3f3f3f3f = 221010101010
            Ow = Ow+100          "set indirect bit"
#130      execute spaces()
          if j = 0, go to #120
          Errn = 8          "indirect address error"
          go to #400
          if Sx3f3f3f3f3f3f = 331010101010
            Ow = Owx fffffffffff7f
            go to #130
          A = find(S)
          if Ng > 0
            Errn = 1          "undefined symbol"
            go to #400
#135      if t > 3
          execute spaces()
          if j = 0
            r = t
            B = find(symbol())
            if Ng > 0: Errn = 1
            if r = 5: B = subset(ordinal(12-B)+1)
            A = subset(ordinal(A)+ordinal(B))
            go to #135
          Errn = 3
          Wpage = ordinal((A-(7))x5)
          Rp = Page
          if Lcpage = 0: Rp = Rp-1
          if Rp = Wpage
            Ow = Ow+(Ax7)+(7)
            go to #400

```



```

if Wpage = 0
    Ow = Ow+(Ax7)
    go to #400
if Ow 1 (8)
    A = (Ax12)+(12)
    Lpg = Page
    Ow = Ow+(8,7)
    go to #1171

Errn = 10
go to #400
if Ow x(16,17) ≠ 0
#150    if t > 3, go to #135
        execute spaces()
        if j > 0, go to #400
        A = find(symbol())
        if Ng > 0
            Errn = 1
            go to #400
        if (OwΔA)x(16,17)≠0
            Errn = 9      "Group 1 and Group 2 operate"
        Ow = Ow+A
        go to #150
#160    if t > 3
        execute spaces()
        if j > 0
            Errn = 3
            go to #400
        r = t
        A = find(symbol())
        if Ng > 0: Errn = 1
        if r = 5: A = subset(ordinal(12-A)+1)
        Ow = subset(ordinal(A)+ordinal(Ow))
        go to #160

```

"presentley codes without bits 16,17 or 18=1"

"will not be modified"

```
#400 1 = 128xPage+LcPage-1
      if Cd1 ≠ (20)
          Errn = 11          "code word changed"
      if Cdc > 0
          Cdc = Cdc-12
          Cd1 = (word-set Ow-(Cdc))x12
          if Cdc > 0: LcPage = LcPage+1
          go to #500
      Cd1 = Ow
#500  if (Line(mod 60)) = 0
      new page
      print by "x": date
      print by "Error,tag,,,Loc1,Code(, )15Statement(, )30Page,x":Tpg
      Ipg = Ipg + 1
      Line = Line+1
      A = subset(1)
      if Lf > 0
          print by ",,,m3(,)18m80":ErmsErrn, Stmtn
          go to #110
      if Cd1 ≥ (18)
          S1 = Cd1-(7)
          B = S1-(1)
          S = B-(1)
          Ow = Cd1x7
      if Sc > 0
          if Cd1 ≥ (18)
              print by ",,,m3(,)504,,o(b)203"
              ErmsErrn,A,S,B,S1,Ow
              go to #590
          print by ",,,m3(,)504,,04": ErmsErrn,A,Cd1
          go to #590
```

```

if Cd1 ≥ 18)
    print by ",,,m3(,)5c4,,o(b)2o3,m80"
        ErmsErrn,A,S,B,S1,Ow,Stmtn
    go to #590
print by ",,,m3(,)5c4,,o4,,,m80":ErmsErrn,A,Cd1,Stmtn
#590 if Cdc > 0
    Errn = 0
    Sc = 1
    go to #400
execute wrapup()
if j > 0
    Sc = 1
    Errn = 0
    go to #115
go to #110
#600 Line = 0                                "print reference table"
    i = Pertab+1
    if sense 6 is on: i = 0
    for j = 1 to Itab
        if (Line(mod 60)) = 0
            new page
            print by "x": date
            print by "Symbol(,)5Value(,)15Reference(,)30Page,x": Ipg
            Ipg = Ipg+1
            (6 character)Stg = xxxxxxxxxxxx
            S = Symbj
            for i = 0 to 5
                (1 character)Strg = (Sx8)1(48-8)
                S = S-(8)
                Stg = Strg+Stg
            print by "m6(,)6c4(,)9(,,,o4)12": Stg, Octeqj
            Line = Line+1

```

```

    if Lite  $\geq$  (3), go to #12
#700 Line = 0
    for j = 0 to 31
        if Ltj > 0
            for i = Ltj, Ltj-1, ..., 1
                A = subset(128(j+1)-1)
                if (Line(mod 60)) = 0
                    new page
                    print by "x": date
                    print by "Loc,Value(,)30x": Ipg
                    Ipg = Ipg+1
                Line = Line+1
                print by "04,04": A, Cd128(j+1)-1
    for j = 0 to 99
        | 93 (00)
        Cksum = 0
        Orig = 0
    for j = 0 to 4095
        if Cdj = (20)
            Orig = 0
            loop back
        if Orig = 0
            A = subset(j)
            B = [(A-(6))x6]+(6)
            A = Ax6
            Cksum = Cksum+ordinal(A)+ordinal(B)
            | 93 (B)
            | 93 (A)
            Orig = 1
        A = Cdjx12
        B = A-(6)
        A = Ax6
        Cksum = Cksum+ordinal(A)+ordinal(B)

```

```

      | 93 (B)
      | 93 (A)
A = subset(Cksum)
B = (A-(6))x6
A = Ax6
      | 93 (B)
      | 93 (A)
for j = 0 to 100
      | 93 (80)
if Litex(3,4) = 0 , go to #13
step #100
go to #3
"iscan, procedure to scan initial portion of stmt"
(... iscan(; all)
(word-set)Pge is 290a200c1010
(word-set)Clcs is 3f3f3f3f3f3f
(word-set)Val
#26 execute spaces()
if j = 1, go to exit "entire string was spaces or tabs"
if Strngx7=(0,6) "an asterisk"
    Strng = Strng - (8)
    execute spaces()
    if j > 0
        Errn = 3
        go to exit
    Smb = symbol()
    go to #50 "Symbolic location"
if Strngx24 = 78566c
    End = 1
    go to exit
if Strngx6=5 "$ or super step code"
    Flag = 0
    if Strngx 3f3f = 1f1f "2 $ or super step codes"
        End = 2

```

```

        go to exit                "end of run"
End = 1
A = (Strng-(8))x8
if ordinal(A) < 3
    Type = ordinal(A)
    Flag = 1
    go to exit                    "end of pass"
Smb = symbol()
if SmbxCls=Pge                    "Smb devoid of color"
    execute spaces()              "or case = page"
    if j = 1
        Page = Page+1            "nothing after page on the line"
        go to exit
    Smb = symbol()
    Val = find(Smb)
    if Ng > 0
        Errn = 1
    Page = ordinal(Val)
    if t > 3
        r = 6
        execute spaces()
        Smb = symbol()
        A = find(Smb)
        if Ng > 0
            Errn = 1
        if r = 4: Page = Page+ordinal(A)
        if r = 5: Page = Page-ordinal(A)
    j = 1
    go to exit
if t = 1, go to #45                "Smb terminated by sp., etc"
if t = 2                "Smb was terminated with a,"
    if Pass = 2, go to #26
    Ng = equatr(Smb, 128xPage+LcPage)

```

```

    if Ng > 0
        Errn = 2
        go to exit
    go to #26
if t = 3                                "Smb was terminated with an ="
    if Pass = 2
        j = 1
        go to exit
    execute spaces()
    if j = 1                            "error"
        Errn = 3
        go to exit
    S = symbol()
    if S-(40) = 39
        A = subset(Pagex128+LcPage)
        go to #33
    A = find(S)
    if Ng > 0                            "error"
        Errn = 1
        go to exit
#33    if t-2 ≠ 0
        Ow = (7)
#34    r = t
        execute spaces()
        if j > 0, go to #40
        S1 = symbol()
        if S1x3f3f3f3f3f3f = 221010101010
            A = A+(8)
            go to #34
        if S1x3f3f3f3f3f3f = 331010101010
            Ow = 0
            go to #34
        B = find(S1)
        if Ng > 1                        "error"

```

```

        Errn = 1
        go to exit
    if r = 4
        A = subset(ordinal(A)+ordinal(B))
        go to #34
    if r = 5
        A = subset(ordinal(A)+ordinal(12-B)+1)×12
        go to #34
#35    if A > (18)
        A = A+(B×7)+0w
        go to #34
        A = A+B
        go to #34
#40    Ng = equato(Smb,A)
        if Ng > 0
            Errn = 2
            go to exit
        Errn = 4
        go to exit
#45    if LcPage ≥ 127
        Page = Page+1
        LcPage = -1
        LcPage = LcPage+1
        Lf = 0
        go to exit
#50    A = find(Smb)
        if Ng > 0
            Errn = 1
            go to exit
        x = ordinal(A)
        if t ≥ 4
            r = t
            execute spaces()
            S = symbol()

```

"error"

"define a symbolic loc."

"error"


```

        A = find(S)
        if r = 4: x = x+ordinal(A)
        if r = 5: x = x-ordinal(A)
        if x < 0: x = x+4096
    Page = ordinal(subset(x)-(7)×5)
    LcPage = x(mod 128)
    j = 1
...
"Procedure to read ASCII tapes and convert to string"
    "max 80 character / string"
(... rdasci()
    (string)St
    (word-set)Blk is hex ffffffff
    (1 character)Char0 to 63 is ["Δ","a","b","c","d","e","f"
cont.      ,"g","h","i","j","k","l","m","n","o","p","q","r","s"
cont.      ,"t","u","v","w","x","y","z",["|","|","|"]," ",
cont.      ," ", "±", " ", "#", "$", "π", "×", " ", "(", ")", "*", "+"
cont.      ," ", "- ", ". ", "/ ", "0", "1", "2", "3", "4", "5", "6", "7", "8"
cont.      ,"9", ":", ";", "<", "=", ">", "√"]
    k = 0
    (word-set) I
    St = Blk
#1      |      91      (I)
    if I=(0,2,3,7), go to #2
    if I = 8, go to #1
    if ordinal(I) < 160, go to #1      "illegal character"
    k = k+1
    if k > 80, go to #1
    I = I×6
    j = ordinal(I)
    St = St+Charj
    go to #1
#2      rdasci() = St
...

```

```

"equate octal, procedure to assign octal equivalent to a symbol"

    "octal to octal"
(... equato(S,V; Octeq,Symb,Itab)
    (word-set)S,V
    (real)m
    Ng = 0
    m = search(S)
    if m ≥ 0
        Ng = 1
        go to #1
    Itab = Itab+1
    OcteqItab = V
    SymbItab = S
#1   equato() = Ng
...)

"octal, procedure convert an octal symbol to an octal #"
(... octal(Smb; Ng)
    (word-set)Smb
    Ng = 0
    i = 0
    (word-set)O = 0
    j = 0
    A = Smb
#1   if Ax6 = (4)
        A = A - (8)
        go to #1
#2   if A = 0, go to #3
    if Ax(8-3) ≠ 0
        Ng = 1
        go to #3
    O = O + (word-set Ax3 + (j))
    j = j+3
    A = A - (8)
    go to #2

```

```

#3      octal() = 0
...
      "equate real, procedure to assign octal equivalent to a symbol"
      "real to octal"
(...   equatr(S,V; Octeq, Symb, Itab)
      (word-set)S
      (real)m
      Ng = 0
      m = search(S)
      if m ≥ 0
          Ng = 1
          go to #1
      Itab = Itab+1
      OcteqItab = subset(V)
      SymbItab = S
#1      equatr() = Ng
...
      "Procedure to search for end of statement"
(...   wrapup(; 1,j,Strng)
      j = 0
#1      (word-set)A = Strngx24
      if Ax8 = ff, go to exit
      if Ax8 = 36, go to exit
      if A = 755643, go to #2
      if A = 435675, go to #2
      Strng = Strng-(8)
      if Ax8 = 5
          j = 1
          go to exit
      go to #1
#2      j = 1
      Strng = Strng-(24)
...
      "Procedure to skip spaces and tabs"
(...   spaces(;1,j,Strng)

```

```

      j = 0
#1   (word-set)A = Strngx8
      if Ax6 = (4), go to #2           "Spaces"
      if A = 43, go to #3             "semicolon"
      if Ax6 = (4,2,1,0), go to #2    "tab"
      if A = 36, go to #3             "/"
      if A = 8, go to #3
      go to exit
#2   Strng = Strng - (8)
      go to #1
#3   j = 1
    ...)

    "find, procedure to determine the octal equivalent"
      "of a symbol"
    (... find(S;Ng,Page,Lc,Octeq,Err,n)
      (word-set)S, (word-set)V
      (real)m
      if Sx(48-43)=0
        V = octal(S)
        if Ng > 0
          Errn = 6
          Ng = 0
          find() = V
          go to exit
      Ng = 0
      if S-(40) = 39
        find() = subset(128xPage+LcPage-1)
        go to exit
      m = search(S)
      if m < 0
        Ng = 1
        go to exit
      find() = Octeqm
    (...)
```

```

"Procedure to read a literal"
(...  literal(;all)
      Strng = Strng-(8)
      execute spaces()
      if Strngx3f3f3f3f = 202d250f          "fltq"
          Strng = Strng-(32)
          execute spaces()
          if j > 0
              Errn = 6
              go to #1
          A = fltgpt()+(36)
          go to #1
      if Strngx3f3f3f3f = 250b2e0d          "dubl"
          Strng = Strng-(32)
          execute spaces()
          if j > 0
              Errn = 6
              go to #1
          A = dhlp()+(24)
          go to #1
      m = 0
      if Strngx8 = 35
          m = 1
          Strng = Strng-(8)
      if Strngx8 = 34
          if m = 1: Errn = 14
          Strng = Strng-(8)
      Smb = symbol()
      A = Smb-(40)
      if Ax6 = 28
          A = octal(Smb+(8)+(4))
          if Ng > 0
              Errn = 6
          go to #2

```

```

        A = decimal(Smb)
#2      if m = 1: A = subset(ordinal(12-A)+1)
        A = (A×12)+(12)
#1      literal() = A
    ...)
        "decimal to octal conversion"
    (... decimal(A; Err, n)
        (word-set)A
        C = 0
        E = 1
#1      if A×6 = (4)
            A = A-(8)
            go to #1
#2      if A×6 = 0, go to #3
        B = A×8
        D = ordinal(B)
        if D > 9
            Errn = 6
            go to #4
        C = C+D×E
        E = 10E
#4      A = A-(8)
        go to #2
#3      decimal(A;Err,n) = subset(C)
    ...)
        "symbol, procedure to extract symbols from the string"
    (... symbol(;i,t,Strng)
        u = 0
        t = 1
        (word-set)Sm, Ch
#1      Ch = Strng×8
        if Ch ≥ 5, go to #100          "ff or 1f"
        if Ch = 43, go to #100        ":@"
        if Ch = 44, go to #100        "["

```

```

    if Ch = 49, go to #100      "("
    if Ch = 36, go to #100      "/"
    if Ch = 39                  "."
        if u = 0
            u = 1
            Sm = Ch
            Strng = Strng-(8)
            go to #1
        go to #100
    Strng = Strng-(8)
    if Ch = (4), go to #100      "Space"
    if Ch = 75, go to #2        "comma"
    if Ch = 38, go to #3        "equal sign"
    if Ch = 34, go to #4        "plus sign"
    if Ch = 35, go to #5        "minus sign"
    if Ch = 17, go to #100      "tab"
    if Ch = 40, go to #100      ")"
    if Ch = 45, go to #100      "]"
    u = u+1
    if u < 7
        Sm = (Sm+(8))+Ch
        go to #1
#2    t = 2                      "Symbol is a location"
        go to #100
#3    t = 3                      "Symbol is to be defined"
        go to #100
#4    t = 4
        go to #100
#5    t = 5
#100  for i = u to 5
        Sm = (Sm+(8))+(4)
        symbol() = Sm
    ...)

"fltgp, procedure to extract a floating point #"

```

```

(... fltgpt(; Strng, Err, n, j)
execute spaces()
m = 0
if Strngx8=35 "minus"
    Strng = Strng-(8)
    m = 1
p, o = 0
e = 0
q = 0
(word-set)A, B
#1 B = Strngx8
C = ordinal(B)
if C > 9
    if C = 57
        if p > 0
            Err_n = 12 "2 period"
            go to #10
        p = 10
        go to #3
    if C = 58, go to #10 "spaces"
    if C = 64, go to #9 ")"
    if C = 69, go to #9 "]"
    if C = 73, go to #10 ":"
    if C = 54, go to #10 "/"
    if C = 255, go to #10 "ff"
    if (C(mod 64)) = 14, go to #5 "E"
    Err_n = 6 "illegal characters"
    go to #10
#2 if p = 0
    o = ox10+C
    go to #3
o = o+C/p
p = px10
#3 Strng = Strng-(8)

```



```

go to #1
#5 Strng = Strng-(8)
execute spaces()
if j > 0
    Errn = 3
    go to #10
if Strngx8=35
    e = 1
    Strng = Strng-(8)
if Strngx8=34
    if e > 0
        Errn = 14
        Strng = Strng-(8)
        "+follows-"
#6 B = Strngx8
C = ordinal(B)
if C > 9, go to #10
q = qx10+C
Strng = Strng-(8)
go to #6
#9 Strng = Strng-(8)
#10 if e > 0: q = -q
o = ox(10)q
A = word(o)x43
(real)x=[word(o)x(47-44)-(17)]+[word(o)x(47)-(4)]+(44)
x = 16(x-1)
#12 if (43-27)xA ≠ 0
    A = A-(1)
    x = x+1
    go to #12
A = A-(4)
if x < 0: x = x+4096
if m > 0: A = subset(ordinal(24-A)+1)
fltgppt() = (A+(subset(x)+(24)))x36
...)
```

```

"dblp, procedure to extract a double precision integer"
(... dblp(; Strng, Err, n, j)
  execute spaces()
  if j > 0
    Errn = 3
    go to exit
  p = 0
  m = 0
  if Strngx8=35
    m = 1
    Strng = Strng-(8)
  if Strngx8=34
    if m > 1: Errn = 14
    Strng = Strng-(8)
#1 (word-set)A = Strngx8
  b = ordinal(A)
  if b > 9
    if b = 64, go to #9
    if b = 69, go to #9
  p = p×10+b
  Strng = Strng-(8)
  go to #1
#9 Strng = Strng-(8)
#10 A = subset(b)
  if m > 0: A = subset(ordinal(24-A)+1)
  dblp() = A×24
...)

"search, procedure to locate a predefined symbol"
(... search(S; Symb, Itab)
  (word-set)S
  for m = 0 to Itab
    if (S△Symbm)x3f3f3f3f3f3f=0
      go to #1
  m = -1
#1 search() = m
...)

```