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

by

Allen C. Larson
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 compiler language MADCAP. 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-3 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 code 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 FDP-5 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 5, 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:

<table>
<thead>
<tr>
<th>Error Tag</th>
<th>Meaning</th>
<th>Result</th>
</tr>
</thead>
</table>
| 0         | undefined symbol         | It was given a value of 0 (except ignored if after * )/.
| 1         | redefined symbol         | The new value was stored.                        |
| 2         | an operator was followed | If after * or = the whole line is ignored;      |
| 3         | by spaces or comment     | if after + or - a 0 was probably used.          |
| 4         | illegal operation        | The address was set to page 0, cell 0.          |
| 5         | address missing on an    | The value of the symbol was computed           |
| 6         | order requiring an       | incorrectly.                                    |
| 7         | address in an octal      | Scan of statement terminated; first symbol      |
| 8         | number on a literal      | used for the code.                              |
| 9         | illegal combination of   | An address of 0 was used.                       |
| 10        | permanent symbols        | They were combined.                             |
| 11        | indirect address error   | Indirect page reference to symbols on a page    |
| 12        | (an I or z was followed | other than current or zero page.                |
| 13        | by nothing)              | A previously defined word of code was          |
| 14        | illegal page reference   | overwritten.                                    |
| 15        | code word changed        | The second period terminated the number.        |
| 16        | 2 periods were encountered | The plus sign was ignored.                    |
| 17        | in an FITG number        |                                                 |
| 18        | a minus sign was followed |                                                 |
| 19        | by a plus sign           |                                                 |
TABLE II. CLASSIFICATION OF SYMBOLS

<table>
<thead>
<tr>
<th>Symbol Type</th>
<th>Description and allowed use</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>All programmer defined symbols are of this type. They can be used in any manner the programmer wishes.</td>
</tr>
<tr>
<td>1</td>
<td>Group 1 operate instructions. They can only be combined with other group 1 operate instructions.</td>
</tr>
<tr>
<td>2</td>
<td>Group 2 operate instructions. These can only be combined with other group 2 operate instructions.</td>
</tr>
<tr>
<td>3</td>
<td>Memory reference instructions. All should have addresses, otherwise the address will be 0.</td>
</tr>
<tr>
<td>4</td>
<td>Instructions that cannot be modified or combined. The I/O orders fall in this group.</td>
</tr>
</tbody>
</table>

A listing of the MADCAP code for MADMAC is given in the Appendix. MADMAC is different from MACRO-9 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 deck.
- 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 "$". 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)Symbol to 500
(word-set)Octet to 500
(80 characters)Stmt to 2000
(word-set)decimal()
(word-set)symbol()
(real)search()
(string)rdasci()
(word-set)find()
(real)equatr()
(real)equate()
(word-set)octal()
(word-set)literal()
(word-set)A, B, Ow, Smb, S, S1
(word-set)fltpt()
(word-set)dblp()
Err to 2000

(4 characters)Erms to 15 is [" ","uds","rds","nop","iop","itr"
cont. ","ich","nco","iae","g12","pge","cwc","2.s","","+a-",""
]
t = 0
(word-set)C0 to 4095

Lt, Lc, Ld to 48

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 "

6
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(06)2": Symb1, A, B
Octeq1 = (A*12) + [(B*4) + (16)]
Symb1 = Symb1 x 3f3f3f3f3f3f
if Symb1 = 101010101010, go to #2
i = i+1
go to #1

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

1
for j = 0 to i
Symbj, Octeqj
go to #11

#10
if x < 7500, then x = 7598
read 2 tracks x, #088

1
for j = 0 to i
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
Ierr, Ng = 0
for i = 0 to 45: Ld1 = Lc1
Dpage = Page
End = 0
Lite = bank
if Flag = 0: Type = ordinal(Lite\times3)-1
n = 0
"# of statements read"

#20 if Type = 0
    new card
    read card: Stmt\_n
    go to #25
if Type = 1
    read: Stmt\_n
    go to #25
Stmt\_n = rdasc1()

#25 (672 elements)String = Stmt\_n
Err\_n = 0
Pass = 1
j,t = 0
execute iscan()
if Err\_n > 0: Ierr = Ierr+1
if End > 0, go to #27
if j > 0
    n = n+1
    go to #20
if Smbx3f3f3f3f3f3f3f3f3f3f3f3f3f3f3f = 0ef5e0b25101111
    Lc\_Page = Lc\_Page+2
if Smbx3f3f3f3f3f3f3f3f3f3f3f3f3f3f3f = 0d2e0b251010
    Lc\_Page = Lc\_Page+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":Errs_Err
  Stmt_j

  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: Lc_i = Ld_i
Page = Dpage
Line = 0
Fpt, Dbl, Cdc = 0

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

#115 Ow = 0
  t = 0
  if Fpt > 0
    execute spaces()
    A = Strng*8
    if ordinal(A) > 9
      Fpt = 0
      go to #116
    Cdc = 36
    Ow = fltgpt()
    go to #400
  if Dbl > 0
    execute spaces()
A = string 6
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
        Err_n = 4
        go to #500
    r = t
execute spaces
Ow = find(symbol())
if r = 5: Ow = subset(ordinal(12-Ow)+1)
if Ng > 0
    Err_n = 1
    go to #400
if Smbx3f3f3f3f3f3f3f3f=0f252d201010
    "fltg"
    Fpt = 1
    Cdc = 36
    Ow = fltgpt()
    go to #400
if Smbx3f3f3f3f3f3f3f3f3f=9d2e0b251010
    "dubl"
    Dbl = 1
    Cdc = 24
    Ow = dblp()
    go to #400
Ow = find(Smb)
if Ng > 0
Err_n = 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
    Err_n = 5
    go to #400
if Strngx16 = 4949
  Strng = Strng-(8)
  go to #160
if Strngx7 = 49  "a current page literal"
  Ow = Ow+80
  Lpg = Page
  go to #17
if Strngx7 = 44  "a zero page literal"
  Ow = Owx..................
  Lpg = 0
#160  A = literal()
#17  for j = 1 to Lt_Lpg
    if A = Cd_{128}(Lpg+1)-J
      if A-(12) = 1, go to #18
      if A-(24) = 1
        J = J+1
        go to #18
      if A-(36) = 1
        J = J+2
        go to #18
#17  Lt_Lpg = Lt_Lpg+1
      j = Lt_Lpg
    if Cd_{128}(Lpg+1)-J \neq (20)
      Err_n = 11
      Cd_{128}(Lpg+1)-J = A
A = A-(12)
if A#1, go to #170
#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 = Ow_ffffff

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+(A+x7)+(7)
go to #400
if Wpage = 0
    Ow = Ow+(Ax7)
    go to #400
if Ow < (A)
    A = (Ax12)+(12)
    Lpg = Page
    Ow = Ow+(8,7)
    go to #1171
Err_n = 10
    go to #400
if Ow %16,17 ≠ 0
    if t > 3, go to #135
    execute spaces()
    if j > 0, go to #400
    A = find(symbol())
    if Ng > 0
        Err_n = 1
        go to #400
    if (Ow\(\times\)(16,17) ≠ 0
        Err_n = 9  "Group 1 and Group 2 operate"
    Ow = Ow+A
    go to #150
#160 if t > 3
    execute spaces()
    if j > 0
        Err_n = 3
        go to #400
    r = t
    A = find(symbol())
    if Ng > 0: Err_n = 1
    if r = 5: A = subset(ordinal(12-A)+1)
    Ow = subset(ordinal(A)+ordinal(Ow))
    go to #160
"presently codes without bits 16, 17 or 18 = 1"

"will not be modified"

#400 1 = 120 x Page + Lc Page - 1

if Cdl > (20)

   Errn = 11    "code word changed"

if Cdc > 0

   Cdc = Cdc - 12
   Cdl = (word-set Ow - (Cdc)) x 12

if Cdc > 0, Lc Page = Lc Page + 1

   go to #500

Cdl = Ow

#500

if (Line(mod 60)) = 0

   new page
   print by "x": date
   print by "Error, tag, Lc, Code, 15 Statement, 30 Page, x": Tpg
   Tpg = Tpg + 1

   Line = Line + 1
   A = subset (1)

if Lf > 0

   print by ",, m3 (x) 1 am a 0", Errn, Stmt n

   go to #110

if Cdl > (18)

   Sl = Cdl - (2)
   B = Sl - (1)
   S = B - (1)
   Ow = Cdl x 7

if Sc > 0

   if Cdl > (18)

      print by ",, m3 (x) 50 4 , a (b) 2 363"

      Errn, A, S, B, Sl, Ow

      go to #590

      print by ",, m3 (x) 50 4 , a 4": Errn, A, Cdl

      go to #590
if Cdc > 0
   Err_n = 0
   Sc = 1
   go to #400
execute wrapup()
if j > 0
   Sc = 1
   Err_n = 0
   go to #115

go to #110

#600 Line = 0
   "print reference table"
i = Pertab+1
if sense 6-1n am: i = 0
for j = i 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 = "s"s"s"s"s"s"s"s"s"s"
      S = Symb_j
      for i = 0 to 5
         (1 character)Strg = (3x8)i(48-8)
         S = S-(8)
         Stg = Strg+Strg,
      print by "m6[,]6s4[,]9[,]64,12": Stg, Octeq_j
   Line = Line+1
if Lite > (3), go to #12.

#700 Line = 0

for j = 0 to 31

  if Lt_j > 0

    for i = Lt_j, Lt_j-1,...,1

      A = subset(128(j+1)-1)

      if (Line(mod 60)) = 0

        new page

        print by "x": date

        print by "Loc,,,Value(j)30x": Ipg

        Ipg = Ipg+1

        Line = Line+1

        print by "n4, ,04": A, Cd_{128(j+1)-1}

    for j = 0 to 99

      "punch out code"

      | 93 (00)

      Cksum = 0

      "check sum = 0"

      Orig = 0

    for j = 0 to 4095

      if Cd_j = (20)

        Orig = 0

        loop back

      if Orig = 0

        A = subset(j)

        B = [(A-(6))×6]+(6)

        A = A×6

        Cksum = Cksum+ordinal(A)+ordinal(B)

        | 93 (B)

        | 93 (A)

        Orig = 1

        A = Cd_j×12

        B = A-(6)

        A = A×6

        Cksum = Cksum+ordinal(A)+ordinal(B)
A = subset(Cksum)
B = (A-(6))x6
A = Ax6
for j = 0 to 100
if Litex(3,4) = 0, go to #13
stop #100
go to #3
"iscan, procedure to scan initial portion of stmt"
(... iscan(); all)
(word-set)Pge is 200a200c1010
(word-set)Cls is 3f3f3f3f3f3f
(word-set)Val
#26 execute spaces()
if j = 1, go to exit "entire string was spaces or tabs"
if Stringx7=(0,6) "an asterisk"
String = String - (8)
execute spaces()
if j > 0
Errn = 3
go to exit
Smb = symbol()
go to #50 "Symbolic location"
if Stringx24 = 78566c
End = 1
go to exit
if Stringx6=5 "$ or super stop code"
Flag = 0
if Stringx 3f3f = 1f1f "2 $ or super stop codes"
End = 2
go to exit

End = 1
A = (Strng-(8))×8
if ordinal(A) < 3
    Type = ordinal(A)
    Flag = 1
    go to exit

Smb = symbol()
if Smb×Cla=Page
    execute spaces()
if j = 1
    Page = Page+1
    "nothing after page on the line"
    go to exit
Smb = symbol()
Val = find(Smb)
if Ng > 0
    Err, = 1
    Page = ordinal(Val)
if t > 3
    r = 6
    execute spaces()
Smb = symbol()
A = find(Smb)
if Ng > 0
    Err, = 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 = equalr(Smb, 128×Page+LePage)
if Ng > 0
    Err_n = 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"
        Err_n = 3
        go to exit
    S = symbol()
    if S-(40) = 39
        A = subset(Page128+Lc_page)
        go to #33
    A = find(S)
    if Ng > 0  "error"
        Err_n = 1
        go to exit
    #33
    if t-2 ≠ 0
        Ow = (7)
    #34
    execute spaces()
    if j > 0, go to #40
    S1 = symbol()
    if S1×3f3f3f3f3f3f = 221010101010
        A = A+(8)
        go to #34
    if S1×3f3f3f3f3f3f = 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)x12
go to #34

#35
if A > (18)
    A = A+(Bx7)+ow
go to #34
    A = A+w
    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
Lc = 0
    go to exit

#50
A = find(Smb)
    "define a symbolic loc." if Ng > 0
    Errn = 1
    go to exit
    "error"

x = ordinal(A)
if t > 4
    r = t
execute spaces()
S = symbol()
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)x5)
LC_page = x(mod 128)
j = 1
...

"Procedure to read ASCII tapes and convert to string"
"max 80 character / string"
(... rdasci() (string)St
(word-set)Blank is hex ffffffff
(1 character)Char0 to 63 is ["\",",""","",","","","",","",","",","",",","",","",","]
cont.
"g","h","i","j","k","l","m","n","o","p","q","r","s"
cont.
"t","u","v","w","x","y","z","[","]","\","
cont.
",",","\",",","\",",","\",",",","\",",",","\",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",",","}k = 0
(word-set) I
St = Blank
#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*8
j = ordinal(I)
St = St+Char_j
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
octeq,Itab = V
Symb,Itab = 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 = 3mb
#1 if A≥(4)
    A = A-(8)
    go to #1
#2 if A = 0, go to #3
if A ×(8-3) ≠ 0
    Ng = 1
    go to #3
O = O+(word-set A×3+(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
    Octeq,Itab = subset(V)
    Symb,Itab = S
#1 equatr() = Ng
...)
"Procedure to search for end of statement"
(... wrapup(; i,j,Strng)
  j = 0
#1 (word-set)A = Strng\textsubscript{24}
  if A\textsubscript{8} = ff, go to exit
  if A\textsubscript{8} = 36, go to exit
  if A = 755643, go to #2
  if A = 435675, go to #2
  Strng = Strng-(8)
  if A\textsubscript{8} = 5
      j = 1
      go to exit
  go to #1
#2 j = 1
  Strng = Strng-(24)
...)
"Procedure to skip spaces and tabs"
(... spaces(;i,j,Strng)
j = 0

#1 (word-set)A = Strng - 8
   if A = 6 = (4), go to #2 "Spaces"
   if A = 43, go to #3 "semicolon"
   if A = 4 = (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 S = (6-43) = 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 + Lc - Page - 1)
     go to exit
  m = search(S)
  if m < 0
     Ng = 1
     go to exit
  find() = Octeq_m
...

"Procedure to read a literal"
(... literal(); all)

Strng = Strng-(8)
execute spaces()

if Strngx3f3f3f3f = 202d250f
    Strng = Strng-(32)
    execute spaces()
if j > 0
    Errn = 6
    go to #1
A = fltgpt()+{36}
    go to #1

if Strngx3f3f3f3f = 250b2e0d
    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 AX6 = {4}
        A = A-{8}
        go to #1
    #2  if AX6 = 0, go to #3
        B = AX6
        D = ordinal(B)
        if D > 9
            Err_n = 6
            go to #4
        C = C+DxE
        F = 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,String)
    u = 0
    t = 1
    (word-set)Sm, Ch
    #1  Ch = Stringx8
        if Ch > 5, go to #100  "if or 1f"
        if Ch = 43, go to #100  ";"
        if Ch = 44, go to #100  "]"
if Ch = 39, go to #100
if Ch = 36, go to #100
if Ch = 38
    if u = 0
        u = 1
        Sm = Ch
        Strng = Strng-(8)
goto #1
go to #100
Strng = Strng-(8)
if Ch = 32, go to #100  "Space"
if Ch = 13, go to #2  "comma"
if Ch = 33, 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 = 41, 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
    goto #100
#5  t = 5
#100 for i = u to 5
    Sm = (Sm+(8))+t(4)
symbol() = Sm
...
"fltgpt, procedure to extract a floating point #"
execute spaces()
m = 0
if Strng×8=35  "minus"
    Strng = Strng-(8)
    m = 1
p, e = 0
e = 0
q = 0
(word-set)A, B
#1
B = Strng×8
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
#2
if C = 30, go to #10  "spaces"
if C = 64, go to #9  ")"
if C = 69, go to #9  "]"
if C = 63, 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
    e = e×10+C
    go to #3
#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
        "follows-"
        Strng = Strng-(8)
#6 B = Strngx8
    C = ordinal(B)
    if C > 9, go to #10
    q = qx10+c
    Strng = Strng-(8)
    go to #5
#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)
fltgpt() = (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)
  #4
  (word-set)A = Strngx8
  b = ordinal(A)
  if b > 9
    if b = 64, go to #9
    if b = 69, go to #9
  p = px10+b
  Strng = Strng(8)
  go to #i
  #9
  Strng = Strng(8)
  #10
  A = subset(b)
  if m > 0: A = subset(ordinal(24-A)+1)
  dblp() = Ax24
  ...
)

"search, procedure to locate a predefined symbol"

(...
  search(S; Symb, Itab)
  (word-set)S
  for m = 0 to Itab
    if (S=Symb_m)x3f3f3f3f3f3f3f3=0
      go to #i
  m = -1
  #1
  search() = m
  ...
)