Line Processor Protocol

< LP, MCS4.NLS;150, >, 21-MAR-75 19:04 RLB2 ;;;;

(protocol) Protocol for TENEX <-> Line Processor interactions

Introduction

This document is a detailed description of the Line Processor protocol. It is intended to serve as a guide to anyone wishing to implement the Line Processor protocol, as well as, a piece of documentation for the Line Processor.

It should be pointed out here that the Line Processor contains a very small, slow microcomputer with little read/write memory. For this reason the protocol is terse and error reports and/or recovery almost non-existant. The Line Processor terminal is treated more as a hardware device than an intelligent terminal.

There are two types of line processors - alpha and graphic. Alpha line processors are used in configurations consisting of the line processor alpha/numeric display, mouse, keyset, and possibly a hard copy printer or a cassette drive. Graphics line processors are used in the the minimum graphics configuration consisting of a/n display, mouse, keyset, and either a Tektronix 4012 or 4014 storage tube display.

Conventions

Coordinates

Alpha

Coordinates designate character positions. For example (1,1) is the second character on the second line up from the bottom.

The origin is at the lower left corner of the screen.

As components of the protocol, coordinates are passed as one byte of X and one of Y and always have 40B added to them to get them in the printing character range. This limits the max coordinate value to 137B which is 95 decimal.

Graphics

The mouse is used to track the cursor on either the a/n display or the Storage tube. A switch acts as a toggle to select which screen is to be tracked. Coordinate values are identical to the alpha line processor when they originate from the a/n display, although they are sent as two bytes each of x and y. Graphics coordinates from the storage tube are sent as 10 bit values in the range 1024 to 2047, with 1024 at the lower left of the screen.

TTY Simulation

In TTY simulation, scrolling always takes place on a line feed (LF) not a carriage return (CR). Carriage return does the obvious thing and no more.

Special and Control Characters

Protocol strings begin with 33B and are followed with an operation type character in the range 40B to 120B.

When outside a protocol string, all control characters (0 thru 37B) are ignored by the Line Processor, except:

- When the cursor is being tracked:
  - ^G which rings a bell if possible
  - CR and LF which do the right thing
- Notice that backspace character (^H) is not implemented in TTY simulation (i.e. when the cursor is being tracked).
- When the cursor has been positioned:
  - ^G which rings a bell if possible

^H which does a backspace cursor

When inside a protocol string, RUBOUT is NOT ignored. When outside, it is ignored.

Conventions for this document

In this document, octal numbers are followed by "B".

"Unescorted" means that characters are sent as is without wrapping them in an protocol sequence.

Line Processor to Main Computer Protocol

Communication in this direction will adhear generally to the IMLAC protocol as outlined in (IJOURNAL,14345,).

In particular:

Keyboard characters 40B thru 177B are unescorted.

Keyboard characters 0 thru 37B are sent as:

34B, 43B, char+140B, coordinates

NOTE: An alternate (and preferred) way is to send these control characters as is (unescorted) except for 2B, 4B and 30B. Those are sent as above.

Mouse button changes are send as:

34B, 43B, buttons+100B, coordinates

where buttons is the binary image of button positons (000 thru 111 binary).

Keyset strokes 1 thru 32B are send as:

stroke+140B (e.g. 1 -> a)

keyset strokes 33B thru 37B are sent as:

33B -> 54B (,)
34B -> 56B (.)
35B -> 73B (;)
36B -> 77B (?)
37B -> 40B (space)

For alpha line processors coordinates are X + 40B, Y + 40B.

For graphics line processors coordinates are X(bits 10 - 6 (MSB's)) + 40B, X(bits 5 - 0 (LSB')) + 40B, Y(bits 10 - 6) + 40B, Y(bits 5 - 0).

When not in coordinate mode the mouse buttons are ignored and keyboard control characters (0 thru 37B) are sent in unescorted fashion.

At power-up and after the "system-reset" button is pushed, the Line Processor signals the Main computer by sending:

(176B, 177B)

The purpose of this is to indicate to the applications program that the Line Procesor is now in a "power-up" state (see below).

When the Line Processor detects an error that it cannot live with, it sends a string to the applications prorgam and dies with an error code flashing in the lights. The user is then forced to hit "system-reset". The string is as follows:

(176B, 41B, Ccount', Chars)

Where Ccount' is 40B more than the number of characters that follow. Currently 8 characters are sent, and the string looks like:

(176B, 41B, 50B, err', ctl', trk', rpt', sw', obuf', b1', b2')

Where the ' indicates that 40B has been added.

err: The error code, one of

10B = output buffer to display overun (impropper padding).

11B = some other buffer overun (e.g. printer buffer)
12B = strange error relating to display output buffer.
13B = protocol sequence error (e.g. bad comand)
14B = protocol value error (e.g. bad coordinate).

ctl: control state parameter (0 = not in a command)

trk: mouse tracking code:
0 = positioned
1 = tracking
2 = cursor in small TTY window
5: cursor at unknown position
12B = Cursor in full screen window

rpt: repeat code, normally zero

sw: sense switch immage in order 0-1-2-3 (sw3=LSB)

obuf: display output buffer character count

b1: possibly low order 4 bits of last input char

b2: possibly high order 4 bits of last input char

From Main Computer to the Line Processor

The following functions are sent by the applications program and performed by the Line Processor. All codes, except the escape (33B) should be printing characters. Padding characters should be RUBOUTs (177B). The baud rate factor (f) and and display type are obtained by the applications program by sending an interogate command.

Note:

The cursor is generally used to track the mouse. Some commands stop the tracking and allow the cursor to be used for display manipulation. "Tracking mode" refers to whether the mouse is being tracked by the cursor or not.

Display-terminal dependent parameters:

The following table yields the timing and other factors required by the protocol that depend on the type of terminal connected to the Line Procesor. That type, DItype, is obtained from the interrogate command (see below).

param		Ditype=		
	1	2	3	4
Del	80	7	1	17
Ins	0	7	30	17
Clr	5	6	3	17
Xmark	No	Yes	Yes	Yes

Del is the time to delete a line.
Ins is the time taken to insert a new line.
Clr is the time taken to clear the screen.
Xmark indicates if a marked character needs to be re-written after the mark is removed.
See the interrogate response command for other display parameters.

Position cursor on alpha display and stop tracking mouse.

Send(33B, 40B, X', Y')
X' = X coord (0 thru Xmax) + 40B
Y' = Y coord (0 thru Ymax) + 40B

result:

Positions cursor to specified location. Tracking stops until a "resume tracking" or a reset is received. Any unescorted characters will be written on the screen and the cursor will be advanced once after each character.

```
      writing beyond the end of the line is not advised as the
      result depends on the terminal manufacturer and model.
Specify (small) TTY simulation window on alpha display
   Send( 33B, 41B, top, bottom)
      top = Y' for top line of window
      bottom = Y' for bottom line of window
   result:
      Invokes a small TTY simulation window of specified size
      and location. This window will be used until a new one
      is specified or a reset is received. This does not
      change the tracking mode.
Reset
   Send( 33B, 51B )
   result:
      screen cleared
      TTY simulation window set to full screen
      bug selection stack reset
      resume tracking (see)
   padding:
      Send pads as for clear screen.
Resume tracking mouse
   Send( 33B, 42B )
   result:
      The cursor is used to track the mouse. Any unescorted
      characters will go into the TTY simulation window
      currently in use.
Write string of blanks
   Send( 33B, 43B, N' )
      N' = number of blanks to be written.
   result:
      The specified number of blanks are written starting at
      the current cursor position.  The cursor is left at the
      character position following the last blank. Assumes the
      cursor has been positioned appropriately beforehand.
      This command is a no-op if N' is not >= 41B AND <= 177B.
   padding:
      This command must have N/f padding characters following
      it.
Push bug selection
   Send( 33B, 46B, X', Y' )
   result:
      The coordinates are pushed on a stack and the character
      at that location is somehow brought to the user's
      attention. The stack will hold a maximum of 8
      selections.  This command includes a resume tracking.
   padding:
      This command must have 8/f padding characters following
      it.
Pop bug selection
   Send( 33B, 47B )
   result:
      The top entry on the bug selection stack is popped.  The
      corresponding character on the screen is no longer
      marked in a special way.  If the stack is empty, this
      command is a no-op.  This command includes a resume
      tracking operation.
```

```
          For some DItypes, tne applications program must restore
          the character or the marked position will be replaced by
          a space.
       padding:
          This command must have 8/f padding characters following
          it.
    Delete selected line
       Send( 33B, 44B )
       result:
          The cursor position selects a line to be removed from
          the screen.  All following lines are moved up one line.
          The contents of the last line are undefined.  The X
          coordinate should be zero, otherwise tne results are
          undefined.
       padding:
          This command requires Del/f padding characters (Del is
          obtained from the table).
    Insert selected line
       Send( 33B, 45B )
       result:
          The line which the cursor is on, and all following
          lines, are moved down one line.  The cursor is not
          moved, and hence is on a blank line.  Lines above the
          cursor are not altered. The last line (before the
          execution of this command) should be considered "lost."
          The X coordinate should be zero, otherwise the results
          are undefined.
       padding:
          This command requires Ins/f padding characters (Ins is
          obtained from the table).
    Clear screen
       Send( 33B, 50B )
       result:
          The entire screen is cleared.  The cursor position is
          not generally known.  The TTY simulation window location
          and the bug selection stack are not altered. The
          tracking mode is not changed.
       padding:
          This command requires Clr/f pad characters;
    Interrogate line processor
       Send( 33B,  55B )
       result:
          A response to the interrogate command is sent as a
          protocol string of this form:
             34B, 46B, Xmax+40B, Ymax+40B, LPtype, Dtim Rate
             Where
                Xmax is the maximum x coordinate
                Ymax is the maximum y coordinate
                LPtype is in [40B-177B] and designates type
                   The least significant four bits of LPtype
                   designate display terminal type (call it
                   DItype)
                      Currently defined are:
                         (1) Delta Data 5200
                         (2) Hazeltine H2000
                         (3) Data Media Elite 2500
```

(4) Lear Siegler ADM-2

The most significant three bits designate Line Processor type (call it Type)

Currently defined are:

(0) Complete alpha line processor with copy printer receiver for cassette drive

(2) Line Processor with Mouse, Keyset, Printer

(6) Graphics line processor with Tektronix 4014

(7) Graphics line processor with Tektronix 4012

Dtim is a characteristic delay time. For proper scrolling, a line feed (LF) must be followed by (Dtim+14)/f pad characters.

Rate indicates the Line Processor receive baud rate:

300 buad: 100B, f=32 decimal

600 baud: 60B, f=16

1200 baud: 50B, f=8

2400 baud: 44B, f=4

4800 baud: 42B, f=2

9600 baud: 41B, f=1

The baud rate factor, f _ Rate-40B;

Note: Any additions to LPtype should be assigned by ARC personel for best results. See DIA or CHI @SRI-ARC.

This command does not change the tracking mode.

Turn off coordinate mode

Send(33B, 60B)

result:

Turns off the coordinate mode in the Line Processor.

This does not change the tracking mode.

Mouse buttons become inactive, keyboard control characters sent to main computer without protocol formating.

Turn on coordinate mode

Send(33B, 61B)

result:

Turns on the coordinate mode in the Line Processor.

This does not change the tracking mode.

Mouse buttons become active, keyboard control characters are sent in input protocol format.

Begin standout mode

Send(33B, 56B)

result:

All following text written on the screen will be altered is some way from "normal" text. This unfortunately includes characters which go into the TTY simulation window also, so don't leave the line processor in this state indefinitely. Does not change the tracking mode.

End standout mode

Send(33B, 57B)

result:

Subsequent text written on the screen will be in "normal" mode. Does not change the tracking mode.

TENEX RESTARTING

The Line Processor will detect a TENEX restart, by looking for the ten 33B's it sends out at startup time. At that time it will place itself in a state as though the hardware reset button had been pushed.

Open printer (alpha line processor only)

Send(33B, 53B)

Result:

Opens the printer for output. Protocol to the printer must be observed: (1) open it. (2) wait for protocol string "request" (below). (3) send strings in response to requests. (4) close it.

"Request" string, sent back to the main computer:

0B NULL

Each request enables the application program to send an additional 16 characters via the printer string protocol below.

Note: The count indicates the Line Processor storage allocated for the next printer string. Sending a longer string will result in a "receive error" (error light on panel).

Close printer (alpha line processor only)

Send(33B, 54B)

Result:

Closes the printer. Actual close will not take place until all characters in the output buffer are printed. That is, the close may follow the last string of characters immediately. It is possible (but very unlikely) that a "request" protocol string may be sent to the main computer after the close is sent to the Line Processor.

Printer string (alpha line processor only)

Send(33B, 52B, Dev, Count+40B, <characters>)

Result:

The Dev is normally 40B and is ignored by Line Processors with one printer. The Count must not be greater than the sum of the counts in all "request" protocol string not already fulfilled. It may be less. The actual character string may contain any characters. They will be sent to the printer without translation or special handling.

Note:

Strings may be sent to the printer without opening it if timing constraints are observed carefully. In this case the applications program must know the baud rate of the printing device a well as the Line Processor - Main computer line. The program just issues printer strings and no requests are sent back to the Main computer by the Line processor. This was a deliberate implementation to allow higher speed printing over networks without waiting for the response. Observe that if strings are sent too fast the printer buffer in the Line Processor will overflow: data will be lost and the Line Processor will die. The printer buffer normally holds 47 characters..

Open graphics display (graphics line processor only)

```
      Send( 33B, 53B )
      Result:
         Disables mouse tracking on the graphics display.
   Close graphics display (graphics line processor only)
      Send( 33B, 54B )
      Result:
         Ensables mouse tracking on the graphics display.
   Write graphics display (graphics line processor only)
      Send( 33B, 52B, Dev, Count+40B, <characters> )
      Result:
         The Dev is normally 40B and is ignored by Line
         Processors.  Characters from the application program are
         written directly on the graphics display.  Since the
         characters are not buffered, the graphics display must
         be connected at a higher baud rate than the external
         processor.
   Set graphics cursor resolution (graphics line processor only)
      Send( 33B, 62B, N' )
      Result:
         N controls the mask applied to the cursor coordinates
         before they are used to position the cursor on the
         graphics display:
            N = 0  Mask = 0
              = 1       = 1 LSB is cleared (etc)
              = 2       = 3
              = 3       = 7
              = 4       = 17B
              = 5       = 37B
Application notes:
   Avoid writing text (or "string of blanks") beyond the end of a
   line: the display may insert an unwanted line or drop the
   extra characters.
   Avoid positioning the cursor to any x>Xmax or y>Ymax.
   Avoid doing an insert line on the last line: the display may
   scroll the entire screen.
   Delta Data (DItype=1) must be treated as a special case in the
   following respect:
      When writing text at (x,y) on a line which does not already
      have text on it up to position x (e.g. after a clear screen
      or insert line), the applications program must send x/f pad
      characters after the first character written at position
      (x,y).  The display takes that long to move a CR symbol
      into the proper display memory location.  (Our thanks to
      Delta Data).
      We expect to stop supporting Delta Datas soon.
NOTE:
   The Line Processor has a reset button on it (which will be
   used only on rare occations).  After power up or a hardware
   reset, the following state prevails:
      The screen is clear, the mouse tracking in operation.
      The bug selection stack is empty.
      The full screen TTY simulation is in effect.
      Coordinate mode is NOT in effect.
      Printer is closed
   All TTY simulation windows currently work as follows:  Text is
   inserted in the last line and "scrolling" occurs on each line
```

feed (i.e. it does not start on the top line of the window as you may prefer). A CR moves the cursor to left margin, a LF effects a line break. Typing beyond the last character of the line causes a line "wrap" - i.e. new text replaces the old line, starting from the left margin. The only way to clear a small TTY window is to send N line feeds into it, where N is the number of lines in the window.

The usual sequence from the applications program will be to position the cursor and perform some function, or write text, or both. It must end such a sequence with a "resume tracking" command. Any broadcast messages, links, etc. that come down the line between the cursor position and the "resume tracking" will go wherever the cursor happens to be.

Normally, broadcast messages and the like will go into the TTY simulation window. The difference being that they are not preceeded by a position cursor command.

REENTER code in NLS will clear and repaint the entire screen

Mouse tracking will be done by the Line Processor under the following conditions:

IF the terminal has received a "resume tracking" command since the last position cursor command, AND

IF there is no input from the TEN, AND

the mouse coords have changed since the last mouse tracking operation, or the cursor has been moved since the last mouse tracking operation.

Tracking stops under the following conditions:

A position cursor command comes from the TEN.

Summaries

Line processor to Exernal processor

CHAR SEQUENCE	MEANING
(all line processors)	
CHARACTER	Normal Character
(Ascii values 1B to 177B except 0 (String request), 2 (^B), 4 (^D), 34B (BCESC), and 176B (Reset))	
BCESC 46 MX MY TP DT BD	Interrogate Response
176 177	System Reset
176 41 CCNT CCHRS	Error report
(alpha line processors)	
BCESC 43 CC X Y	Optional Sequence For Control Chars
BCESC 43 CC X Y	Sequence For ^D (CA), ^B (CDOT), ^X (CD)
BCESC 43 MB X Y	Sequence For Mouse Buttons
0 (NULL)	String request
(graphics line processors)	
BCESC 45 CC X1 X2 Y1 Y2	Optional Sequence For Control Chars
BCESC 45 CC X1 X2 Y1 Y2	Sequence For ^D (CA), ^B (CDOT), ^X (CD)
BCESC 45 MB X1 X2 Y1 Y2	Sequence For Mouse Buttons

Where:

All numbers are in octal

CCNT = number of CCHRS + 40
CCHRS = CCNT-40 data bytes; each byte is offset by 40
CC = control character + 140
MB = current mo]se button state + 100
X = current x corrdinate + 40
Y = current y corrdinate + 40
X1 = top 6 significant bits of x coordinate + 40
X2 = least significant 6 bits of x coordinate + 40
Y1 = top 6 significant bits of y coordinate + 40
Y2 = least significant 6 bits of y coordinate + 40
MX = maximum x coordinate + 40
MY = maximum y coordinate + 40
TP = line processor type and version + 40
DT = terminal delay time characteristic + 40
BD = line processor receive baud rate + 40

Exernal processor to Line processor

COMMAND	CODE	PADDING
position	33B, 40B, X', Y'	none
TTY window	33B, 41B, Y TOP', Y BOTTOM'	none
resume tracking	33B, 42B	none
write blanks	33B, 43B, N'	N/F
delete line	33B, 44B	DEL/F
insert line	33B, 45B	INS/F
push bug	33B, 46B, X', Y'	8/F
pop bug	33B, 47B	8/F
clear screen	33B, 50B	CLR/F
reset	33B, 51B	CLR/F
printer string	33B, 52B, DEV, CNT', String	see text
open printer port	33B, 53B	see text
close printer port	33B, 54B	none
interrogate	33B, 55B	none
standout mode on	33B, 56B	none
standout mode off	33B, 57B	none
coordinate mode off	33B, 60B	none
coordinate mode on	33B, 61B	none
cursor resolution	33B, 62B, N'	none
remote resart	10 - 33B's	none

(mcs4) MCS-4 Assembler in TREE META

```
   FILE msc4 CHECK
      META file
      ERROR: -> '; $st :end[]*;
      SIZE: S=1000 M=100 K=50 N=1000 L=10 G=10;
      DUMMY: add mt lh neg;
      FIELDS: OP=[4:8] OPA=[4:4] OP8=[8:4] TYPE=[4:18]  P=[4:8]
         AD1=[4:8] AD2=[4:4] AD3=[4:0] AD8=[8:0];
      ATTRIBUTES: reg pair;
   % declarations  parsing %
       file = ("FILE" / -> "FILE" ) .ID <"-MCS-4 ASSMBLER 12/11/73">
         <"-FILE "*1> @S defned &DISCARD
         [>^mcs4]
         $declare $st :end[]*;
          end =>
           >^mcsend $SYMS( ?@ defned *$ / <"undefined symbol: " *$ > )
           &TABLES;
```

```
    declare =
      "SET" #<',> ( ( .ID / .UID ) '= .NUM :dec[2]*) '; /
      "REGISTER" #<',> ( .ID '= @S reg .NUM :dec[2]*) '; /
      "PAIR" #<',> ( .ID '= @S pair .NUM :regpair[2]*) '; ;
    dec [-,-] => >*1_*N2;
    regpair [-,-] => >*1 _ LSH(*N2)1;
% statements %
    st = ["END" &FAIL ]
      .$( .ID ': &LABEL ) :label[$] * instr '/ -> '; * ;
      label [$] => $( >*$ );
    instr = op1 / op2 / op3 / .UID (sym4 :simp[2] / :simp[1]);
       simp % simple: OP and optional address %
         [-] => *V1^OP8 \0;
         [-,-] => *V1^OP stopa[*2] \0;
       sym =
         ( .ID
            ( ?@ defned / <*1 " undefined" LOC> )
            / .NUM :con[1] ) [".LH" :lh[1] / ".RH" ] /
         '- sym :neg[1] ;
       sym4 = sym $( '+ sym :add[2]/ '- sym :neg[1] :add[2]);
       stopa
         [-] => + val4[*1]^OPA;
       val4
         [add] := 0 + val4[*1:1] + val4[*1:2];
            % above is ugly but can't start exp with construct that
            appears to be a node test %
         [con] := *N1:1;
         [0] := 0;
         [neg] := -val4[*1:1] ;
         [lh[con]] := *N1:1:1 ;/ 16;
         [lh] := *V1:1 ;/ 16;
         [-] := *V1;
       val
         [-] => +val4[*1];
    op1 =
      "JCN" sym4 [',] adr :two[=1, "JCN", 2] /
      "ISZ" sym4 [',] adr :two[=1, "ISZ", 2] /
      "FIM" regpr [',] data :two[=1, "FIM", 2] ;
    op2 =
      "JUN" adr :two[=2, "JUN", =0, 1] /
      "JMS" adr :two[=2, "JMS", =0, 1] ;
    op3 =
      "FIN" regpr :one[=3, "FIN", =0, 1] /
      "SRC" regpr :one[=3, "SRC", =1, 1] /
      "JIN" regpr :one[=3, "JIN", =1, 1] /
      "DATA" data :gendata[1] /
      "ADR" adr :genadr[1] /
      "PAGE" :page[] /
      "ZERO" .NUM :zro[1];
    regpr = .ID ?@ pair;
    data = adr / '(
      sym4 ( ', sym4 :double[2] / :val[1] ) ') ;
       con
         [-] => *N1;
       double
         [-,-] => + val4[*1]^AD2 + val4[*2]^AD3;
      adr = .ID / .NUM :con[1];
      page => &BSS MASK(lc+255)7400B-lc, ;
   % instruction generation %
      gendata
        [.ID] => 4^TYPE *1\1; % 8 bit reloc address %
        [double] => 4^TYPE *1\1;
        [con] => *N1:1^OP8 \0; % 8 bit data word %
        [val] => +val4[*1:1]^OP8 \0;  % data word ( 8 bits ) %
      genadr
        [-] => 4^TYPE *1\1; % address - 8 bits %
      one [-,-,-,-] => % one 8 bit instruction %
        *N1^TYPE % instruction type %
        *V2^OP % opcode %
        stopa[*4] % OPA field %
        [?*N3#0 20B] \0; % special opcode bit %
      two [-,-,-,-] => % two words, OP OPA adr %
        *N1^TYPE % opcode type %
        *V2^OP % opcode %
        stopa[*3] \0 % OPA field ends first byte %
        *4\1 ; % address %
      zro [-] => &BSS *N1,;
   END of MCS-4
(pprog) Program to punch tapes for programmer board
   (punch) FILE % to punch tape for MCS-4 programmer (l10,)
   (punch.rel,) %
   % declarations %
      (oprec) RECORD ugn1[4], opa[4], op[4];
      (adrec) RECORD  ad3[4], ad2[4], ad1[4], q[6], type[4];
      EXTERNAL sysovr;
      DECLARE intel=1001, prolog=1002, lprolog=1003; % codes for
      programmer type %
      DECLARE progend=1010, progcr=1011; % codes for Pro-log %
      DECLARE
         l10stk[50],
         ugly=777777000001B, % add to L10 string to make TENEX string %
         lc, % location counter %
         cell, % address of last cell sent to programmer %
         pdevice, % punch device %
         pjfn, % jfn for paper tape punch %
         ojfn, % jfn for printer listing %
         ijfn, % jfn for listing input %
         adr1, % first address %
         adr2, % last address to program plus 1 %
         string[20], % line buffer %
         leadch=377B, % rubout for leader character %
         one='N, % INTEL one character %
         zero='P, % INTEL zero character %
         direct=1, %0=paper tape, #0 = directly to programmer %
         monitor, % =1 means echo programmer stuff on TTY %
         progtype, % programmer type (intel or prolog) %
         adrerr=0, % address errors count %
         comflg=0, % comment flag, true=inside comment in ijfn text %
         lastf=0, % flag, TRUE means we have buffered one char %
         lastchar, % this is the buffered char %
         laste, % this is the end code for confirm %
         lastcell, % this is the location for the char %
      tabs=34; % number of chars to tab if no binary stuff %
   REGISTER
      stack=9, mark=10, r1=1, r2=2;
   SET l10sz=50;
   SET loader=761265B, loadexit=761321B;
      % symbols for TENLDR are at 777332,,764332 %
% procedures %
   (main) PROCEDURE; % main entry points in here %
      (sysovr):
         stack.LH _ -$l10sz; stack.RH _ $l10stk;
         error($"stack overflow");
      (jump): GOTO loadereturn;
      (envect): GOTO start; GOTO rstart;
      (init): % set entry vector %
         !sevec(4B5, 2B6+$envect);
         !haltf;
      (start): % starting location %
         !reset; !clzff(4B5);
         stack.LH _ -$l10sz; stack.RH _ $l10stk;
         adrerr _ 0;
         [$loadexit] _ jump;
         !psout($"specify REL file - end with ALT - "+ugly);
         GOTO loader;
         % NOTICE:
            loader is the reenter location of TENLDR and loadexit is
            the location of the JSYS HALTF in TENLDR (just before
            sysovr).  They must be fixed up each time TENLDR is
            changed !!! %
         (loadereturn): % return point from loader %
         LOOP BEGIN
            !psout($"punch file: "+ugly);
            IF NOT SKIP !gtjfn(060003B6, 100000101B) THEN
               BEGIN
               jerror(r1);
               REPEAT LOOP;
               END;
            pjfn _ r1;
            pdevice _ !dvchr(pjfn); % device designator %
            direct _ 0;
            CASE pdevice.LH OF
               =600012B, =0: % TTY: %
                  BEGIN % directly to TTY port, hence to programmer
                  %
                  direct _ 1;
                  progtype _ 1;
                  IF NOT SKIP !asnd(pdevice) THEN jerror(r1);
                  END
               =600005B: % PTP: %
                  BEGIN
                  IF NOT SKIP !asnd(pdevice) THEN jerror(r1);
                  progtype _ 1;
                  END;
               =600015B: % NIL: %
                  progtype _ 0;
               ENDCASE % file %
                  progtype _ 1;
```

```
IF progtype THEN CASE !pbin(!psout($"programmer type is
(L, I, or P) "+ugly)) OF
   ='L, ='l: % Lineprocessor and 1200 baud prolog %
      BEGIN
      !psout($"ineprocessor and 1200 baud prolog"+ugly);
      CASE !pbin() OF
         =CR, =EOL, =CA: NULL;
         ENDCASE
            BEGIN
            !psout($"? "+ugly);
            REPEAT CASE 2;
            END;
      progtype _ lprolog;
      END;
   ='I, ='i: % Intel %
      BEGIN
      !psout($"ntel"+ugly);
      CASE !pbin() OF
         =CR, =EOL, =CA: NULL;
         ENDCASE
            BEGIN
            !psout($"? "+ugly);
            REPEAT CASE 2;
            END;
      progtype _ intel;
      END;
   ='P, ='p: % Pro-log %
      BEGIN
      !psout($"ro-log"+ugly);
      CASE !pbin() OF
         =CR, =EOL, =CA: NULL;
         ENDCASE
            BEGIN
            !psout($"? "+ugly);
            REPEAT CASE 2;
            END;
      progtype _ prolog;
      END;
   ENDCASE
      BEGIN
      !psout($"? "+ugly);
      REPEAT CASE ;
      END;
IF progtype THEN
   CASE !pbin(!psout($"want to see echo from
   programmer?"+ugly)) OF
      =CR, ='Y, ='y, =EOL, =CA: monitor _ 1;
      ENDCASE
         BEGIN
         !bout(101B, EOL);
         monitor _ 0;
         END
   ELSE monitor _ 0;
IF NOT SKIP !openf(pjfn, 10B10+3B5) THEN
   BEGIN
   jerror(r1);
```

```
      REPEAT LOOP;
      END;
   EXIT LOOP;
   END;
!psout($"entire file to be programmed?"+ugly);
CASE !pbin() OF
   =CA, ='Y, ='y, =EOL:
      BEGIN
      ijfn _
         open($"sequential listing input: ",
            160003B6, 7B10+2B5);
      IF !dvchr(ijfn).LH = 600015B THEN
         % i.e. ijfn is NIL: %
         BEGIN
         ijfn _ 0;
         IF NOT SKIP !gtjfn(400001B6, $"NIL:"+ugly) THEN
            BEGIN
            jerror(r1);
            error($"cannot proceed");
            END;
         ojfn _ r1; % NIL: also %
         IF NOT SKIP !openf(ojfn,7B10+1B5) THEN
            BEGIN
            jerror(r1);
            error($"cannot proceed");
            END;
         END
      ELSE
         ojfn _ open($"listing output: ", 660003B6,
         7B10+1B5);
      adr1 _ $mcs4;
      adr2 _ $mcsend;
      END;
   ENDCASE
      (sstart): BEGIN % restart entry point %
      adr1 _ input1($"from: ")+$mcs4;
      adr2 _ MIN(input1($"thru ")+$mcs4+1, $mcsend);
      (rstart): % restart entry point adr1,2 setup %
      lastf _ 0;
      !bout(101B, EOL);
      IF NOT SKIP !gtjfn(400001B6, $"NIL:"+ugly) THEN
         BEGIN
         jerror(r1);
         error($"cannot proceed");
         END;
      ojfn _ r1;
      IF NOT SKIP !openf(ojfn,7B10+1B5) THEN
         BEGIN
         jerror(r1);
         error($"cannot proceed");
         END;
      ijfn _ 0;
      END;
IF direct THEN
   BEGIN
   !cfibf(pjfn);
```

```
         !cfobf(pjfn);
         END;
      stack.LH _ -$l10sz; stack.RH _ $l10stk;
      output();
      findline();
      !sout(ojfn, $string .V 18M6, 0);
      IF NOT SKIP !nout(ojfn, adrerr, 10) THEN jerror(r3);
      !sout(ojfn, $" address errors"+ugly, 0);
      IF NOT SKIP !closf(pjfn) THEN jerror(r1);
      IF NOT SKIP !closf(ojfn) THEN jerror(r1);
      IF NOT SKIP !closf(ijfn) THEN jerror(r1);
      IF NOT SKIP !nout(101B, adrerr, 10) THEN jerror(r3);
      !psout($"  address errors"+ugly);
      !pbout(EOL);
      !psout($"successful completion"+ugly);
      IF direct THEN
         BEGIN
         !psout($"deassign device? "+ugly);
         CASE !pbin() OF
            =EOL, =CR, ='Y, ='y:
               IF NOT SKIP !reld(pdevice) THEN jerror(r1);
            ENDCASE NULL;
         END;
      !haltf;
   END.
(input1) PROCEDURE % input a number from the user %
   % arguments %
      (s); % an optional string to be typed %
   LOOP
      BEGIN
      IF s THEN
         BEGIN
         !pbout(EOL); !psout(s+ugly);
         END;
      IF NOT SKIP !nin(100B, 0, 10) THEN jerror(r3)
      ELSE EXIT END;
   RETURN(r2) END.
(open) PROCEDURE % open a file %
   % formals %
      (s, % string %
      getw, % gtjfn word %
      opnw); % openf word %
   LOCAL jfn;
   LOOP BEGIN
      !psout(s+ugly);
      IF NOT SKIP !gtjfn(getw, 100000101B) THEN
         BEGIN
         jerror(r1);
         REPEAT LOOP;
         END;
      jfn _ r1;
      IF NOT SKIP !openf(jfn, opnw) THEN
         BEGIN
         jerror(r1);
         REPEAT LOOP;
         END;
      RETURN(jfn);
   END;
   END.
(output) PROCEDURE; % main output procedure %
   LOCAL w;
   lc _ cell _ adr1; % first symbol in program %
   leader();
   IF lc#smcs4 THEN % not at start of prog - check for split
   instr %
      CASE [lc-1].type OF
         =1, =2:
            BEGIN % special case - start of 2nd half of 2 byte
            instr %
            w _ [lc]-$mcs4; % relocate addr %
            punchst();
            punchbyte(w.ad2,0);
            punchbyte(w.ad3,1);
            punchend();
            BUMP lc, cell;
            checklc();
            END;
         ENDCASE NULL;
   WHILE lc<adr2 DO
      BEGIN
      findline();
      punchlc();
      w _ [lc];
      punch1();
      !sout(ojfn, $string .V 18M6, 0);
      IF w=0 THEN % string of zeros case : keep listing aligned %
         WHILE [(lc_cell_lc+1)]=0 AND lc<adr2 DO
            BEGIN
            checklc();
            punchlc();
            punch1();
            !bout(ojfn,EOL);
            END
      ELSE BUMP lc,cell;
      checklc();
      END;
   leader();
   RETURN END.
(checklc) PROCEDURE; % check for edge of ROM page %
   IF (lc-$mcs4) .A 8M = 0 THEN leader();
   RETURN END.
(leader) PROCEDURE; % punch leader or setup programmer%
   LOCAL i;
   IF direct THEN
      BEGIN
      IF lastf THEN
         confirm(lastcell, lastchar, laste);
      lastf _ 0;
      IF lc>=adr2 THEN RETURN;
      !psout($"type CR when PROM is ready"+ugly);
      CASE bincnr() OF
         =CR, =EOL, =CA: NULL;
         ENDCASE REPEAT CASE;
      CASE progtype OF
         =intel:
            BEGIN
            !bout(pjfn,'P);
            !disms(750);
            IF NOT SKIP !nout(pjfn,(lc-$mcs4) .A 8M,140003B6+10)
            THEN jerror(r3);
            !disms(750);
            IF NOT SKIP !nout(pjfn,MIN((adr2-$mcs4 -1),
            (lc-$mcs4) .V 255) .A 8M, 140003B6+10) THEN
            jerror(r3);
            !disms(750);
            END;
         =prolog:
            BEGIN
            !bout(pjfn,'*); confirm($mcs4,'*, progcr);
            !bout(pjfn,'P); confirm($mcs4,'P, progcr);
            i _ lc-$mcs4;
            !bout(pjfn,hex(i.ad2)); confirm($mcs4,hex(i.ad2), 0);
            !bout(pjfn,hex(i.ad3)); confirm($mcs4,hex(i.ad3), 0);
            i _ MIN( (adr2-$mcs4 -1), (lc-$mcs4) .V 8M) .A 8M;
            !bout(pjfn,hex(i.ad2)); confirm($mcs4,hex(i.ad2), 0);
            !bout(pjfn,hex(i.ad3));
            confirm($mcs4,' ,0); % ?????????? %
            END;
         =lprolog:
            BEGIN
            lpout(pjfn,'*); confirm($mcs4,'*, progcr);
            lpout(pjfn,'P); confirm($mcs4,'P, progcr);
            i _ lc-$mcs4;
            lpout(pjfn,hex(i.ad2)); confirm($mcs4,hex(i.ad2), 0);
            lpout(pjfn,hex(i.ad3)); confirm($mcs4,hex(i.ad3), 0);
            i _ MIN( (adr2-$mcs4 -1), (lc-$mcs4) .V 8M) .A 8M;
            lpout(pjfn,hex(i.ad2)); confirm($mcs4,hex(i.ad2), 0);
            lpout(pjfn,hex(i.ad3));
            confirm($mcs4,' ,0); % ?????????? %
            END;
         ENDCASE NULL;
      END
   ELSE FOR i_0 UP 1 UNTIL = 75 DO !bout(pjfn,leadch);
   RETURN END.
(binchr) PROCEDURE; % do a pbin %
   !pbin(); RETURN(r1) END.
(punchlc) PROCEDURE; % put location on listing%
   LOCAL i; % location %
   i _ lc-$mcs4;
   CASE progtype OF
      =intel:
         BEGIN
         IF i .A 2M = 0 THEN
            BEGIN
            !bout(pjfn, CR); !bout(pjfn,LF);
            IF NOT SKIP !nout(pjfn,i,140004B6+10) THEN
            jerror(r3);
            !bout(pjfn,' );
            END;
         END;
      =prolog:
         NULL;
      =lprolog:
         NULL;
      ENDCASE NULL;
   IF NOT SKIP !nout(ojfn, i,140004B6+10) THEN jerror(r3);
   % put hex address on listing %
   !bout(ojfn, ' ); !bout(ojfn,'P);
   IF NOT SKIP !nout(ojfn,(i.ad1),140001B6+10) THEN jerror(r3);
   !bout(ojfn,':);
   !bout(ojfn,hex(i.ad2));
   !bout(ojfn,hex(i.ad3));
   RETURN END.
(punch1) PROCEDURE; % punch instr. (maybe two bytes) %
   LOCAL
      w, % the instruction word %
      f; % flag for opcode FIM or not %
   w _ [lc];
   CASE w.type OF
      =0, =3: % 8 bit instr (OP OPA) %
         BEGIN
         punchst();
         punchbyte(w.op,0);
         punchbyte(w.opa,1);
         punchend();
         !sout(ojfn, $"            "+1 .V 18M6, 0);
         END;
      =1: % 16 bit instr OP OPA + 8 bit adr %
         BEGIN
         punchst();
         punchbyte(w.op,0);
         punchbyte(w.opa,1);
         punchend();
         f _ (IF w.op=2 %FIM% THEN 1 ELSE 0);
         BUMP lc,cell;
         checklc();
         IF lc>=adr2 THEN RETURN;
         w _ [lc]-$mcs4;
         punchst();
         punchbyte(w.ad2,0);
         punchbyte(w.ad3,1);
         IF w.ad1 #
            (CASE (lc-$mcs4) .A 8M OF
               =255: (lc-$mcs4+1)/400B;
               ENDCASE (lc-$mcs4)/400B )
            AND f=0 THEN punchrr()
            ELSE punchend();
            % adr err if address not within next PROM if at 255
            or this PROM, but not on FIM instr in any case %
         END;
      =2:  % 16 bit instr  OP + 12 bit adr %
         BEGIN
         punchst();
         punchbyte(w.op,0);
         BUMP lc;
         w _ [lc]-$mcs4;
         punchbyte(w.ad1,1);
         punchend();
         BUMP cell; % only place lc and cell diverge %
         checklc();
         IF lc>=adr2 THEN RETURN;
         punchst();
         punchbyte(w.ad2,0);
         punchbyte(w.ad3,1);
         punchend();
         END;
      =4:  % relocatable address - 8 bits %
         BEGIN
         w _ w-$mcs4;
         punchst();
         punchbyte(w.ad2,0);
         punchbyte(w.ad3,1);
         punchend();
         !sout(ojfn, $"            "+1 .V 18M6, 0);
         END;
      ENDCASE
         error($"illegal instr type");
   puneol();
   RETURN END.
(hex) PROCEDURE(x); % convert x to HEX character %
   CASE x OF
      IN [0,9]: RETURN(x+'0);
      IN [10,15]: RETURN(x-10+'A);
      ENDCASE error($"illegal hex value");
   END.
(punchst) PROCEDURE; % punch starting char, if any %
   CASE progtype OF
      =intel:
         BEGIN
         !bout(pjfn,'B);
         END;
      =prolog: NULL;
      =lprolog: NULL;
      ENDCASE NULL;
   !bout(ojfn, ' );
   RETURN END.
(punchend) PROCEDURE; % punch ending char if any %
   CASE progtype OF
      =intel:
         BEGIN
         !bout(pjfn,'F);
         confirm(cell,0,0);
         END;
      =prolog: NULL;
      =lprolog: NULL;
      ENDCASE NULL;
   !bout(ojfn,' );
   RETURN END.
(punchrr) PROCEDURE; % like punchend, but address error displayed
%
```

```
   CASE progtype OF
      =intel:
         BEGIN
         !bout(pjfn,'F);
         confirm(cell,0,0);
         END;
      =prolog: NULL;
      =lprolog: NULL;
      ENDCASE NULL;
   !bout(ojfn,'A);
   BUMP adrerr;
   RETURN END.
(confirm) PROCEDURE %confirm response from programmer %
   (addr, % the address being programmed %
   c,     % the return character if progtype=prolog,lprolog %
   x);    % echo confirmation code %
   LOCAL t, f, waitime;
   IF direct AND !dvchr(pjfn).LH # 600015B THEN
      % i.e. pjfn is not NIL: %
      BEGIN
      f _ 0;
      CASE progtype OF
         =intel:
            BEGIN
            !disms(1000); % at least 10 chars to send %
            waitime _ 600;
            END;
         =prolog:
            BEGIN
            waitime _ 0;
            !disms(
               (CASE x OF
                  =progend: 25;
                  =progcr: 20
                  ENDCASE 0)
               );
            END;
         =lprolog:
            BEGIN
            waitime _ 0;
            !disms(
               (CASE x OF
                  =progend: 25;
                  =progcr: 20
                  ENDCASE 0)
               );
            END;
         ENDCASE NULL;
      LOOP BEGIN
         t _ !time();
         WHILE SKIP !sibe(pjfn) DO
            IF (!time()-t) > waitime AND f=2 OR (!time()-t) >
            5000 THEN
               BEGIN
               IF f=2 THEN RETURN;
               IF NOT SKIP !sibe(pjfn) THEN EXIT;
         % give 'm one more chance %
      !psout($"  no confirmation for word "+ugly);
      IF NOT SKIP !nout(101B, addr-$mcs4,140004B6+10)
      THEN jerror(r3);
      !pbout(EOL);
      !psout($" type CR, S, R, P or ? for help"+ugly);
      CASE binchr() OF
         =CA, =EOL, =CR: NULL;
         ='S, ='s:
            BEGIN
            !pbout(EOL);
            !psout($"hit reset on the programmer box
            before proceeding "+ugly);
            !pbout(EOL);
            GOTO sstart;
            END;
         ='R, ='r:
            BEGIN
            !pbout(EOL);
            !psout($"hit reset on the programmer box
            before proceeding "+ugly);
            !pbout(EOL);
            adr1 _ addr;
            GOTO rstart;
            END;
         ='P, ='p:
            BEGIN
            !pbout(EOL);
            !psout($"hit reset on the programmer box
            before proceeding "+ugly);
            !pbout(EOL);
            adr1 _ (addr-$mcs4) .A 777400B + $mcs4;
            GOTO rstart;
            END;
         ='?:
            BEGIN
            !pbout(EOL);
            !psout($"type CR to continue"+ugly);
            !pbout(EOL);
            !psout($"S to start over (respecify start
            and finish)"+ugly); !pbout(EOL);
            !psout($"R to restart from this word"+ugly);
            !pbout(EOL);
            !psout($"P to restart at first word of this
            prom"+ugly); !pbout(EOL);
            REPEAT CASE;
            END;
         ENDCASE
            BEGIN
            !psout($"type ? for help, fella"+ugly);
            !pbout(EOL);
            REPEAT CASE;
            END;
      RETURN;
      END;
WHILE NOT SKIP !sibe(pjfn) DO
```

```
            BEGIN
            !bin(pjfn);
            r2 _ r2 .A 7M;
            IF monitor THEN
               IF r2 < 40B THEN
                  BEGIN
                  %!pbout('^');% %to get your control characters
                  printed%
                  %!pbout(r2+40B);%
                  !bout(101B);
                  END
               ELSE !bout(101B);
            CASE progtype OF
               =intel:
                  IF r2 IN (40B,'z] THEN
                     CASE f OF
                        =0: IF r2='B THEN f _ 1;
                        =1: IF r2='F THEN f _ 2;
                        =2: IF r2='F AND done(addr)
                           THEN f _ 2 ELSE f _ 3;
                        ENDCASE NULL;
               =prolog, =lprolog:
                  BEGIN
                  CASE x OF
                     =progend: % demand CR, string space %
                        CASE f OF
                           =0: IF r2=c THEN f_1;
                           =1: IF r2=LF THEN f_4
                              ELSE IF r2=CR OR r2=EOL THEN f_3;
                           =3: IF r2='  THEN f_2
                              ELSE IF r2='/ AND done(addr) THEN
                              f_2;
                           ENDCASE NULL;
                     =progcr: % demand CR, LF %
                        CASE f OF
                           =0: IF r2=c THEN f_1;
                           =1: IF r2=EOL THEN f_2;
                           ENDCASE NULL;
                     ENDCASE % demand the char %
                        CASE f OF
                           =0: IF r2=c THEN f_2;
                           =2: f_3;
                           ENDCASE NULL;
                  END;
               ENDCASE NULL;
            END;
         END;
      END;
   RETURN END.
(done) PROCEDURE(addr); % return true if end of PROM%
   RETURN(
      IF (addr+1-$mcs4) .A 8M = 0
         OR addr=adr2-1 THEN 1
      ELSE 0);
   END.
(puncnbyte) PROCEDURE(bits,e); % punch one 4 bit byte and list it
```

```
%
   % bits=byte to punch, e=true if 2nd 4-bit byte %
   LOCAL w,i, x;
   IF progtype=intel OR ijfn#0 THEN
      BEGIN
      r1 _ bits; !LSH r1,32; x _ r1;
      FOR i_0 UP 1 UNTIL = 4 DO
         BEGIN
         r2 _ x; r1 _ 0; !LSHC r1,1;
         w _ r1; x _ r2;
         IF w THEN
            BEGIN
            IF progtype=intel THEN !bout(pjfn,one);
            !bout(ojfn, '1);
            END
         ELSE
            BEGIN
            IF progtype=intel THEN !bout(pjfn,zero);
            !bout(ojfn, '0);
            END;
         END;
      !bout(ojfn, ' );
      END;
   CASE progtype OF
      =intel: NULL;
      =prolog:
         BEGIN
         IF lastf THEN confirm(lastcell, lastchar, laste);
         lastchar_hex(4M-bits);
         !bout(pjfn,hex( 4M-bits));
         lastf _ 1;
         laste _ IF e THEN progend ELSE 0;
         lastcell _ cell;
         END;
      =lprolog:
         BEGIN
         IF lastf THEN confirm(lastcell, lastchar, laste);
         lastchar_hex(4M-bits);
         lpout(pjfn,hex( 4M-bits));
         lastf _ 1;
         laste _ IF e THEN progend ELSE 0;
         lastcell _ cell;
         END;
      ENDCASE NULL;
   RETURN END.
(puneol) PROCEDURE; % punch end of instruction stuff, if any %
   CASE progtype OF
      =intel:
         % IF (lc-smcs4) .A 2M = 0 THEN BEGIN
            !bout(pjfn, CR);
            !bout(pjfn, LF);
            END;%
         NULL;
      =prolog: NULL;
      =lprolog: NULL;
      ENDCASE NULL;
```

```
         RETURN END.
      (lpout) PROCEDURE (jfn,char);
         % output a character to the copy printer port of a
         lineprocessor %
         !bout(jfn,33B);
         !bout(jfn,52B);
         !bout(jfn,40B);
         !bout(jfn,41B);
         !bout(jfn,char);
         RETURN;
         END.
      (findline) PROCEDURE; % scan ijfn text for next instr %
         LOCAL
            x, % character %
            slashflg, % true if line had '/ on it %
            i; % index into string %
         IF ijfn=0 THEN RETURN(string_174B9);
         slashflg _ 0;
         LOOP
            BEGIN
            !gtsts(ijfn);
            IF r2 .A 1B9 THEN  % end of file %
               BEGIN
               string _ 174B9; % EOL,0 %
               RETURN;
               END;
            !sin(ijfn, $string .V 18M6, 100, LF);
            ^r2 _ 0; % ensure null %
            i _ $string .V 4407B8;
            LOOP CASE (x_^i) OF
               ='/: IF comflg=0 THEN slashflg_1;
               =LF, =0: BEGIN
                  IF slashflg THEN RETURN;
                  FOR i_0 UP 1 UNTIL >= tabs DO !bout(ojfn,' );
                  !sout(ojfn, $string .V 18M6, 0);
                  REPEAT LOOP 2;
                  END;
               ='%: comflg _ IF comflg THEN 0 ELSE 1;
               ENDCASE NULL;
            END
         END.
      (error) PROCEDURE % general error routine %
         % argument %
            (s); % a atring %
         !pbout(EOL); !psout($"error: "+ugly);
         !psout(s+ugly); !pbout(EOL);
         !haltf;
         RETURN END.
      (jerror) PROCEDURE % jsys error writing procedure %
         % argument %
            (errorn); % error number %
         !erstr(101B, 4B11+errorn, 0);
         !JFCL; !JFCL; !pbout(EOL);
         RETURN END.
      FINISH
(directions) How to program a PROM
```

```
To compile the program (and obtain a REL file)
   Go into NLS.
   Load the desired NLS file containing the program.
   From the programs subsystem COMPILE FILE using (MCS4,) to the rel
   file of your choice.
   Quit.
   You are now at TENEX EXEC ( @ ).
Write a PROM set and/or obtain an assembly listing
   The routine <LP>MCSLDR>SAV drives PROM programmers and creates
   assembly listings.
   To obtain an assembly listing before programming PROMS.
      Get a rel file as above.
      Get a sequential file coresponding to the source of the rel
      file.  (For example, OUTPUT SEQUENTIAL FILE)
      From EXEC run <LP>MCSLDR.SAV
         Answer questions namely:
            Give your rel file followed by <ESC>
            Punch file is NIL: <CR>
            Provide the name of he text file <CR>
            Provide the name of a file to save the listing <CR>
         When MCSLDR  finishes copy the listing file tto a printer
   (Paper tape and the Intel programmer are essentially obsolete)
   To obtain a prom set from a PROLOG  programmer connected as a
   terminal to the host machine or connected to a line processor
   with a copy printer receiver.
      Get a rel file as above.
      From EXEC run <LP>MCSLDR.SAV
         Answer questions namely:
            Give your rel file followed by <ESC>
            Punch file is TTY: <CR> (for a line processor)
                          TTYnn: <CR> (for a local terminal)
            Programmer type is L for a line processor <CR>
                               P for a local terminal <CR>
            Either echo mode is OK
            If less than the full file is to be programmed provide
            the inclusive bounds in DECIMAL!
               The PROM boundaries are:
                  prom 0    0 -  255
                       1  256 -  511
                       2  512 -  767
                       3  768 - 1023
                       4 1024 - 1279
                       5 1280 - 1535
                       6 1536 - 1791
                       7 1792 - 2047
            Provide the name of he text file <CR> or NIL: <CR>
            Provide the name of a file to save the listing <CR> or
            NIL: <CR>
            With the PROLOG power off insert a erased PROM into the
            COPY socket, turn he power, and press RESET.  Enter a
            <CR> to the terminal.  The MCSLDR will continue to drive
            the PROLOG until complete by requesting a <CR> for each
            new PROM as above.
Create a new MCSLDR
   MCSLDR is a stand alone tenex routine.  The source is stored in
   (LP,MCS4,PPROG).  Obtain a rel file named punch.rel (for example)
   then Goto Tenex.
      <arcsubsys>TENLDR <CR>
      /S <cr>
      punch <cr>
      <andrews>110run <cr>
      <arcsubsys>stenex <cr>
      <altmode>
      (there will be two undefined referances)
      DDT <CR>
      init <ESC> g  (initializes MCSLDR and exits ddt)
      SSAVE <ESC> <ESC> <ESC> <LP>MCSLDR.SAV <CR>
OLD INTEL DOCUMENTATION
   Setup to Program the PROM
      Setup the INTEL programming board
         Connect the INTEL board to TEN tty port xx (currently using
         26 octal).
         Connect a terminal to the grey box (if desired) and set the
         grey box switches for INTEL <-> TEN connection.
      On the TENEX terminal, say
         ASSIGN <altmode> TTYxx: <cr>  (e.g. TTY26: <cr> )
   Run the punch program
      On the TENEX terminal, type
         DDT <cr>
         start<altmode>G
         [punch file: ] TTYxx: <cr>
         [entire file to be punched? ] <cr>
         [sequential text input: ] <prog>.TXT <cr>
            or, if no listing is desired, type NIL:
         [listing output: ] LPT: <cr>
            or, if no listing is desired, type NIL:
         the program will say "type CR when PROM is ready"
      Double check the setup, and type CR when you are ready.
   Did it work correctly?
      Expect to see the following on the TENEX terminal (and on the
      terminal connected to the grey box, if connected)
         P
         000
         yyy (in decimal)
            where yyy is the last cell of the PROM to be programmed
         then a bunch of things like BNPPNNPNPF
            Where N=1, P=0, and the whole thing represents a PROM
            word.
            There may be either one or two of these per line.
            Locations appear in the left margin.  They are program
            locations, not PROM locations:  These are the same for
            the first PROM, but program location 256 is PROM
            location zero for the second PROM, etc.
         A final "F" on a line by itself means the PROM is done.
      Look for the following on the TENEX terminal:
         "Type CR when PROM is ready" when a PROM is finished means
         that the program wants to do another PROM.  Remove the
         finished one, put in a new one, and type CR when ready.
         The message "successful completion" means you are done.
         The message "file not closable" is standard when using the
         TTYxx: port.
      If you are unable to program a PROM word, you will see $$$?
```

after the BNPP...F thing for the word that failed. The TENEX terminal should say "can't program that cell" and quit.

Programming will fail if:

1) the PROM is not erased
2) the programming switch is set to disabled (on INTEL board)
3) INTEL board is not setup right
4) the PROM is not seated in the socket

If TENEX crashed or the programming is stopped somehow, you may re-program without erasing the PROM - i.e. you may re-write the PROM if you write the same thing again.

Variations:

You may punch a paper tape by giving PTP: as the punch file rather than TTYxx:. In that case, just type CR when the punch program says "type CR when PROM is ready".

You may just obtain a listing of the program by giving NIL: as the punch file, and giving the TXT file as sequential test file and LPT: as the listing file.

You may program only certain locations by saying no when the punch program asks "entire program to be punch?". In that case, you provide two program locations x1 thru x2, and only locations x1 thru x2 will be programmed.