LEDIP
A KIM/6502 Text Editor

BY KIUMI AKINGBEHIN
Department of Mathematics
Wayne State University
Detroit, MI 48202

LEDIP (an acronym for *Line EDItor Program*) is a general purpose line-oriented text editor program for 6502-based systems. LEDIP can be used for such purposes as writing letters, preparing texts, and generating source programs.

LEDIP is designed to be memory-efficient and easy to use. Residing in about 1K bytes of memory, LEDIP uses an efficient data structure to minimize the memory occupied by the user's text. LEDIP performs memory compressions and expansions as needed after each line of text is entered. Not a single byte of memory is wasted. In addition, LEDIP allows the user to select the location in memory where the text is stored. LEDIP's small memory requirements make it ideal for memory conscious users. With LEDIP, a reasonable amount of text can be edited in a system with as small as 2K bytes of memory.

Running LEDIP

LEDIP Version K4 (assembly listing shown), runs on KIM systems with at least 1.1K bytes of RAM starting at location 2000 hex and going upwards. Since LEDIP is a text editor and not a memory editor (compare EDITHA/SWEETS, *DDJ* Vol.3, Issue 5, May 78), and I/O device such as a teletype is also needed. Readers with such a configuration may directly key in the object code and enter LEDIP thru location 2000 hex using the G command. LEDIP should respond with the question, "STARTING ADDRESS?". This is the cold entry point; warm entry point is at location 203C hex. Version K4 with the changes indicated in parenthesis will also run on TIM/DEMON systems. Readers who don't feel like keying in a 1.1K object code can obtain paper tape or KIM cassette of LEDIP from the 6502 Program Exchange, 2920 Moana, Reno, NV 89509. Include a $2.50 duplication/distribution fee. Versions of LEDIP for other 6502-based systems including VIM (Synertek's new 6502-based SBC) are also available from The 6502 Program Exchange. JOLT users should note that the TEXT command supplies the rub-outs required by the JOLT resident assembler.

Using LEDIP

LEDIP starts by requesting a starting address for the text from you. Type a four-digit hexadecimal location. Your text will occupy this location and subsequent memory locations. Be sure to specify usable RAM. LEDIP uses 18 contiguous bytes near the top of page zero to store variables and constants pertaining to the text being edited. In addition, LEDIP resides in about 1K bytes of memory. These locations should be reserved for LEDIP's use and should not be used for any other purpose. The FILE command can be used to find out what these locations are. Once a valid starting address is given, LEDIP does some initialization and responds with the prompt character, a slash. A line number or command can now be typed.

A line number can be any four-digit decimal number between 0000 and 9999. Leading zeroes must be included. If a line number is typed after the prompt character, LEDIP automatically goes into the edit mode, types a space, and waits for a line of text to be entered. A line can be of any length between 1 and 252 characters. Any upper or lower case ASCII character can be entered. Control codes and other special codes can also be entered. All control codes, with the exception of the backspace (control H), are stored as received. A backspace deletes the last character entered. Carriage-returns are not allowed within a line. A carriage-return terminates a line. Text lines are modified, replaced, deleted, or inserted using line numbers in a manner similar to BASIC. Note that this technique makes edit-mode commands like DELETE, REPLACE, INSERT, etc. unnecessary.

To add a line of text, type a new line number and then type in the text. To insert a line of text between two existing lines of text, type a line number between the two current line numbers and then enter the text. For instance, to enter a line of text between lines 0022 and 0029, type 0024 and then type the new text. LEDIP will do the memory shifting and manipulations necessary, and will insert the new line between the two current lines. To delete a line, type the line number and a carriage-return. To replace or modify a line, type the line number and then type the new text. To create a blank line, type the line number, at least one space, and then type a carriage-return.

If a command is typed after the prompt, LEDIP automatically goes into the command mode. LEDIP recognizes the following five commands:

LIST – lists the entire text with line numbers
TEXT – lists the entire text without line numbers
FILE – states the block of memory currently occupied by the text
EXIT – returns control to the system monitor program (if present)
CLEAR – clears current workfile and requests location for new text

The FILE command states three blocks of memory: a block of 18 bytes used by LEDIP on zero page, a block of memory occupied by LEDIP, and a block occupied by the user's text. The LIST and TEXT commands can be terminated at any time by using the hardware interrupt or reset and re-entering LEDIP through the warm start. LEDIP should always be

entered through the warm start if the current text is to be preserved. The EXIT command leaves the monitor program counter pointing to the warm start; hence only a G need be typed in most cases to re-enter LEDIP. An accidental CLEAR initiation can be corrected by an interrupt and a jump to the warm start.

LEDIP texts can be saved on tape in two formats for future use: an ASCII format and a hexadecimal format. To save a text in ASCII format, type TEXT, start the paper tape punch or cassette recorder, and then type a carriage return. ASCII formatted type cannot be reloaded into LEDIP for future editing. If future editing is desired, the text should be saved in hexadecimal format. To save a text in hexadecimal format, type FILE. LEDIP will define three memory blocks (e.g. 00D1-00E2, 2000-249D, 0100-01C4). Now type EXIT to return to your system monitor program. The monitor can now be used to save and reload the data contained in the first and third memory blocks. When loading your text thus, LEDIP should be entered through the warm start.

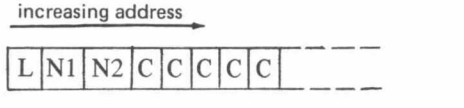
LEDIP checks the validity of commands, line numbers, line lengths, and continually performs read-after-write verifications. An error will result in one of the following error messages:

M! nonexistent memory or memory overflow
C! invalid command or line number
H! improper hex number
L! line too long

In the case of invalid four-letter commands, LEDIP defaults and executes the command whose first letter matches that of the invalid command.

A Brief Look Inside LEDIP

In keeping with the objective of a memory-efficient text editor program, LEDIP uses a sequential linear list (contiguous memory block) of variable length records to store the text. While a linked list or "table of line pointers" approach would have resulted in less code, it was decided that memory usage should be given priority over code reduction in the kind of environment in which LEDIP is likely to be used. The decision to use variable rather than fixed length records is based on the same consideration. Zero page locations STAD (starting address) always points to the top of the list and LOCC (location counter) always points to the bottom of the list. HEXBU (hexadecimal buffer) is invariably used to walk through the list. Each record (line of text) consists of three fields as shown in figure 1. LEDIP makes conservative use of the stack (page one) and only uses 18 bytes on page zero. These two pages are therefore largely available to the user.



L – length of line (one byte)
N1 – line number low order byte
N2 – line number high order byte
CCCC . . . – ASCII characters (variable length)

Figure 1: LEDIP data format.

Since the text list contains no absolute addresses or links, LEDIP is essentially text-relocatable. In fact, the block memory move subroutines in LEDIP can be used to move the text around in memory. Only STAD and LOCC need be changed whenever the text is relocated.

The other main consideration in writing LEDIP was to write an easy-to-use text editor. To achieve this goal, three decisions were made; viz. LEDIP shall be line oriented and not string oriented, line numbers shall be used for all edit-mode operations, non edit-mode commands and error messages shall be kept to one easily remembered minimum. The apparent simplicity with which line numbers are used to edit text lines obscures the actual processes which go on inside LEDIP during edit operations. The flowchart (figure 2) gives a clearer picture of these operations and the routines which are invoked by each. This flowchchart is roughly the second level in a four level top-down flowchart development of LEDIP.

LEDIP readily lends itself to modifications and extensions. Readers who wish to implement additional commands will find that the routines necessary for most additional commands (edit and non-edit) are already in the program. It should be noted that LEDIP does not use any command tables. Three NOP's have been included in the command handler (CMHD) to facilitate this. These NOP's will have to be replaced by an appropriate jump to the code extension. For instance, implementing a single line or line number range LIST only requires changing the contents of STAD and LOCC and then invoking the already existing LIST routines. LEDIP features several useful subroutines which are callable by other programs. These subroutines include block memory moves, ASCII conversion, hexadecimal and decimal character validation, save and restore register, and other routines. Zero page locations defined at the beginning of the program are used to pass parameters to and from these subroutines.

Since the CRLF, SPACE, and type-a-byte subroutines are as easily accessible as the standard read-a-character and type-a-character subroutines in most resident operating (monitor) systems, LEDIP directly calls all five I/O subroutines. All I/O calls flow thru a series of jumps near the end of LEDIP. Hence only ten locations need be changed to implement LEDIP on systems with different I/O configurations. LEDIP saves and restores all registers during I/O calls. Readers writing their I/O subroutines should remember to include proper delay for the CRLF as may be required by the console device. Readers who wish to add pagination to LEDIP listings should note that one inch top and bottom margins on the standard teletype requires 12 blank lines after every 54 text lines.

LEDIP does not feature a software BREAK test since the hardware interrupt or reset can be used to terminate LEDIP listings at any point. KIM users who wish to add a break text would have to poll the 6530 PIA data register at location 1740 hex. TIM users should poll location 6E02 hex. Since all I/O operations flow thru the restore register (RESR) routine, a good place to insert the break test is at the end of the RESR routine. Three NOP's have been included to facilitate this. In implementing a break test, care should be taken to restore the stack and to restore registers destroyed by the break routine. Since LEDIP preserves the syntax of the input text lines, readers who are interested in language translation will find LEDIP a useful basis for the development of an interactive compiling or interpreting language translator.

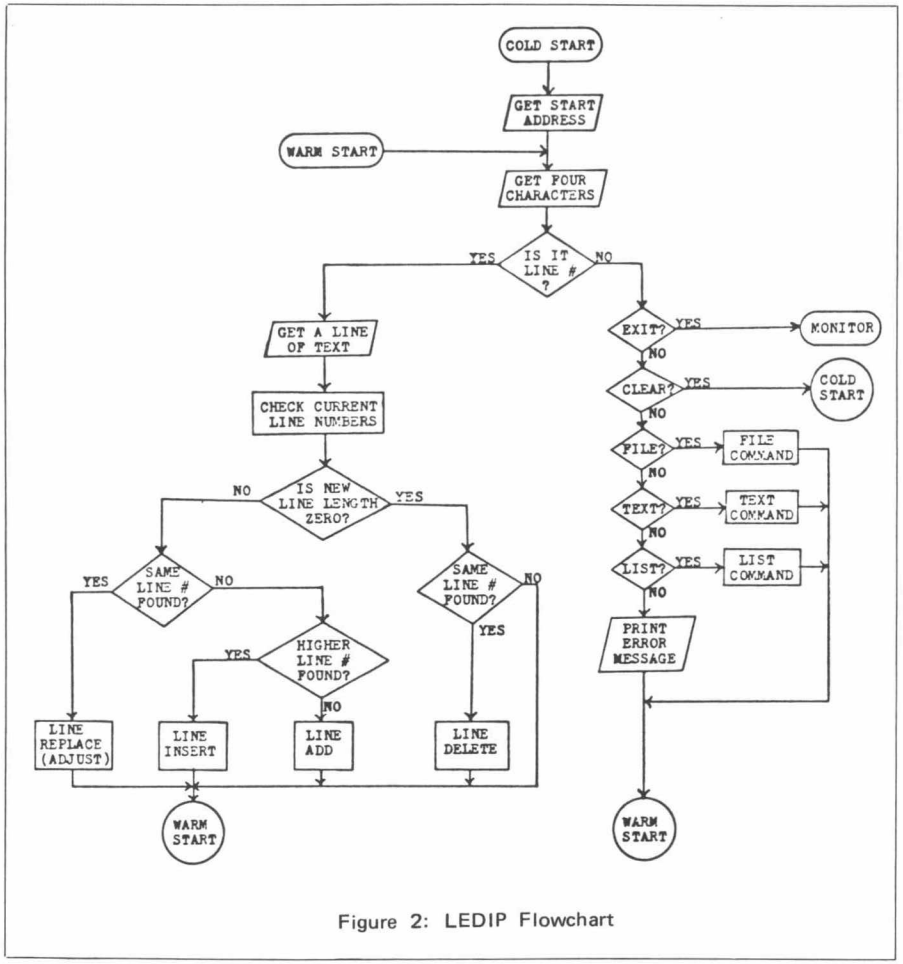


Figure 2: LEDIP Flowchart

```
CARD # LOC   CODE      CARD
    1                ;
    2                ;
    3                ;***********************************************;
    4                ;***********************************************;
    5                ;**                                           **;
    6                ;**     LEDIP (LINE EDITOR PROGAM)            **;
    7                ;**     VERSION K4                            **;
    8                ;**     COPYRIGHT 1978 BY KIUMI AKINGBEHIN    **;
    9                ;**                                           **;
   10                ;***********************************************;
   11                ;***********************************************;
   12                ;
   13                ;
   14                ; ZERO PAGE SCRATCHPAD MEMORY
   15                ;
   16                ;
   17                ASCBUM  = $E2            ASCII BUFFER
   18                ASCBU3  = $E1
   19                ASCBU2  = $E0
   20                ASCBUL  = $DF
   21                HEXBUH  = $DE            HEX BUFFER
   22                HEXBUL  = $DD
   23                STADH   = $DC            STARTING ADDRESS
   24                STADL   = $DB
   25                LOCCH   = $DA            LOCATION COUNTER
   26                LOCCL   = $D9
   27                TEMPR   = $D8            TEMPORARY REGISTER
   28                CHCC    = $D7            CHARACTER COUNTER
   29                MBEGH   = $D6            MEMORY BEGIN
   30                MBEGL   = $D5
   31                MENDH   = $D4            MEMORY END
   32                MENDL   = $D3
   33                MDESH   = $D2            MEMORY DESTINATION
   34                MDESL   = $D1
   35                ;
   36                        .OPT GENERATE    PRINT ASCII STRINGS
   37                        .OPT NOMEM
   38  0000                  *=$2000          ORG AT 2000 HEX
   39                ;
   40                ;
   41                ;***********************************************;
   42                ;***********************************************;
   43                ;**                                           **;
   44                ;**          MAIN ROUTINE FOLLOWS             **;
   45                ;**                                           **;
   46                ;***********************************************;
   47                ;***********************************************;
   48                ;
   49                ;
   50                ; LEDIP COLD ENTRY POINT (CSTAT)
   51                ;
   52                ; THIS ROUTINE TYPES A CRLF TWICE AND THE QUESTION
   53                ; "STARTING ADDRESS?" ON THE CONSOLE DEVICE.
   54                ;
   55  2000  D8      CSTAT   CLD              NO BCD ARITHMETIC
   56  2001  20 42 24        JSR CRLF         TYPE CR, LF
   57  2004  20 42 24        JSR CRLF
   58  2007  A0 EE           LDY #$EE         INITIALIZE FOR INDEXING
   59  2009  B9 88 23 CSTAT1 LDA STADO-$EE,Y
   60  200C  20 54 24        JSR OUTCH        TYPE A CHARACTER
   61  200F  C8              INY
   62  2010  D0 F7           BNE CSTAT1       NEXT CHARACTER
   63                ;
   64                ; READ STARTING ADDRESS (RSTAD)
   65                ;
   66                ; THIS ROUTINE RECEIVES FOUR ASCII CHARACTERS FROM
   67                ; THE CONSOLE DEVICE AND STORES THEM AS RECEIVED
   68                ; IN THE ASCII BUFFER. IT THEN CHECKS TO SEE IF
   69                ; ALL ARE VALID HEX CHARACTERS. ERROR MESSAGE "H"
   70                ; IS TYPED AND LEDIP RETURNED TO THE COLD START
   71                ; IF AN INVALID CHARACTER IS FOUND. CVAH IS CALLED
   72                ; AND STAD, LOCC INITIALIZED OTHERWISE.
   73                ;
   74  2012  20 30 24 RSTAD   JSR RDASC        READ FOUR CHARACTERS
   75  2015  20 07 23         JSR HCHK4        CHECK IF ALL ARE HEX
   76  2018  B0 10            BCS RSTADL
   77  201A  20 42 24         JSR CRLF
   78  201D  A9 48            LDA #'H
   79  201F  20 54 24         JSR OUTCH        TYPE H
   80  2022  A9 21            LDA #$21
   81  2024  20 54 24         JSR OUTCH        TYPE EXCLAMATION
   82  2027  4C 00 20         JMP CSTAT
   83  202A  20 C8 22 RSTADL  JSR CVAH         CONVERT ASCII TO HEX
   84  202D  20 42 24         JSR CRLF
   85  2030  A5 DD            LDA HEXBUL       INITIALIZE STAD AND LOCC
   86  2032  85 DB            STA STADL
   87  2034  85 D9            STA LOCCL
   88  2036  A5 DE            LDA HEXBUH
   89  2038  85 DC            STA STADH
   90  203A  85 DA            STA LOCCH
```

```
                         ;
                         ; LEDIP WARM ENTRY POINT (WSTAT)
                         ;
                         ; THIS ROUTINE ASSUMES THAT LOCC AND STAD ARE
                         ; ALREADY SET, TYPES A PROMPT CHARACTER, RECEIVES
                         ; FOUR CHARACTERS FROM THE CONSOLE DEVICE, AND IF
                         ; ALL ARE NUMERIC, CALLS CVAH (CONVERT ASCII TO
                         ; HEX). CONTROL IS OTHERWISE TRANSFERRED TO CMHD
                         ; (COMMAND HANDLER).
                         ;
203C  D8                 WSTAT   CLD                   NO BCD ARITHMETIC
203D  A9 02                      LDA #2
203F  85 D7                      STA CHCC              INITIALIZE CHCC
2041  20 42 24                   JSR CRLF              TYPE CR, LF
2044  A9 2F                      LDA #$2F
2046  20 54 24                   JSR OUTCH             TYPE PROMPT CHARACTER
2049  20 30 24                   JSR RDASC             READ FOUR CHARACTERS
204C  20 0C 23                   JSR DCHK4             CHECK IF LINE NUMBER
204F  B0 03                      BCS AL8
2051  4C CE 21                   JMP CMHD              ELSE CHECK IF COMMAND
2054  20 C8 22           AL8     JSR CVAH
2057  A5 DD                      LDA HEXBUL            LINE # LOW ORDER BYTE
2059  A0 01                      LDY #1
205B  91 D9                      STA (LOCCL),Y
205D  D1 D9                      CMP (LOCCL),Y         READ-AFTER-WRITE CHECK
205F  F0 03                      BEQ AL3
2061  4C 74 22           AL4     JMP INVM2
2064  A5 DE              AL3     LDA HEXBUH            LINE # HIGH ORDER BYTE
2066  C8                         INY
2067  91 D9                      STA (LOCCL),Y
2069  D1 D9                      CMP (LOCCL),Y         READ-AFTER-WRITE CHECK
206B  D0 F4                      BNE AL4
206D  20 5D 24                   JSR SPACE             TYPE SPACE IF OK
                         ;
                         ; RECEIVE ASCII (RVASC)
                         ;
                         ; THIS ROUTINE RECEIVES A LINE OF ASCII TEXT.
                         ; ERROR MESSAGE "L" IS TYPED AND CONTROL RETURNED
                         ; TO THE WARM START IF LENGTH OF LINE EXCEEDS
                         ; 252 CHARACTERS. CONTROL IS TRANSFERRED TO THE
                         ; APPROPRIATE ROUTINE OTHERWISE.
                         ;
2070  20 4B 24           RVASC   JSR GETCH             READ A CHARACTER
2073  C9 08                      CMP #$08              IS IT BACKSPACE
2075  D0 0A                      BNE RVASC1
2077  A5 D7                      LDA CHCC
2079  C9 02                      CMP #2
207B  F0 F3                      BEQ RVASC
207D  C6 D7                      DEC CHCC              DELETE LAST CHARACTER
207F  D0 EF                      BNE RVASC
2081  E6 D7              RVASC1  INC CHCC              ACCEPT CHARACTER
2083  D0 10                      BNE RVASC2
2085  20 42 24           INVL    JSR CRLF              LINE IS TOO LONG
2088  A9 4C                      LDA #'L
208A  20 54 24                   JSR OUTCH             TYPE L
208D  A9 21                      LDA #$21
208F  20 54 24                   JSR OUTCH             TYPE EXCLAMATION
2092  4C 3C 20                   JMP WSTAT
2095  C9 0D              RVASC2  CMP #$0D              IS IT CARRIAGE-RETURN
2097  F0 0A                      BEQ RVASC4
2099  A4 D7                      LDY CHCC
209B  91 D9                      STA (LOCCL),Y         STORE IF NOT
209D  D1 D9                      CMP (LOCCL),Y
209F  D0 C0              AL2     BNE AL4
20A1  F0 CD                      BEQ RVASC             GET NEXT CHARACTER
20A3  A5 D7              RVASC4  LDA CHCC              CHECK IF ZERO LENGTH
20A5  C9 03                      CMP #3
20A7  D0 32                      BNE LADJ              LINE ADJUST IF NOT
                         ;
                         ; LINE DELETE (LDEL)
                         ;
                         ; CONTROL IS TRANSFERRED TO THIS ROUTINE FOR A
                         ; LINE DELETE OPERATION.
                         ;
20A9  20 A4 22           LDEL    JSR HEXST             SET HEXBU TO STAD
20AC  20 AD 22           LDEL1   JSR CMPHL             COMPARE HEXBU WITH LOCC
20AF  F0 C5                      BEQ AL10
20B1  20 4F 23                   JSR LNCHK             CHECK LINE NUMBER
20B4  B0 03                      BCS AL7
20B6  4C 3C 20           AL10    JMP WSTAT             RETURN TO WARM START
20B9  F0 C6              AL7     BEQ LDEL2
20BB  20 2C 23                   JSR INCHB             SET HEXBU TO NEXT LINE
20BE  4C AC 20                   JMP LDEL1
20C1  A0 C0              LDEL2   LDY #0                GET LENGTH OF LINE
20C3  B1 DD                      LDA (HEXBUL),Y
20C5  85 D8                      STA TEMPR
20C7  20 ED 23                   JSR MOV3              SET UP MOVMB PARAMETERS
20CA  20 60 23                   JSR LSTLC             CHECK IF LAST LINE
20CD  F0 06                      BEQ LDEL3             NO NEED FOR MOVE IF SO
20CF  20 10 24                   JSR MOV4
20D2  20 77 23                   JSR MOVMB             MOVE MEMORY BLOCK
20D5  20 E8 22           LDEL3   JSR DCLC
20D8  4C 3C 20                   JMP WSTAT             RETURN TO WARM START
                         ;
                         ; LINE ADJUST (LADJ)
                         ;
                         ; CONTROL IS TRANSFERRED TO THIS ROUTINE FOR A LINE
                         ; REPLACE OR MODIFY OPERATION.
                         ;
20DB  A5 D7              LADJ    LDA CHCC              GET CHARACTER COUNTER
20DD  A0 00                      LDY #0
20DF  91 D9                      STA (LOCCL),Y         STORE IT
20E1  D1 D9                      CMP (LOCCL),Y         READ-AFTER-WRITE CHECK
20E3  D0 BA                      BNE AL2
20E5  20 A4 22                   JSR HEXST             SET HEXBU TO STAD
20E8  20 AD 22           LADJ1   JSR CMPHL             COMPARE HEXBU WITH LOCC
20EB  D0 06                      BNE LADJ2
20ED  20 3C 23                   JSR INCLC             ACCEPT LINE
20F0  4C 3C 20                   JMP WSTAT
20F3  20 4F 23           LADJ2   JSR LNCHK             CHECK LINE NUMBER
20F6  B0 03                      BCS AL9
20F8  4C A0 21                   JMP LINS              LINE INSERT
20FB  F0 06              AL9     BEQ CMPLL             LINE REPLACE
20FD  20 2C 23                   JSR INCHB
2100  4C E8 20                   JMP LADJ1
2103  A2 00              CMPLL   LDX #0                COMPARE LINE LENGTHS
2105  A1 D9                      LDA (LOCCL,X)
2107  C1 DD                      CMP (HEXBUL,X)
2109  F0 04                      BEQ LADJ3             GO TO CORRECT ROUTINE
210B  B0 0B                      BCS LADJ4
210D  90 5E                      BCC LADJ5
210F  20 C2 23           LADJ3   JSR MOV1              SAME LINE LENGTH
2112  20 77 23                   JSR MOVMB
2115  4C 3C 20                   JMP WSTAT             RETURN TO WARM START
                         ;
                         ; LINE ADJUST 4 (LADJ4)
                         ;
                         ; CONTROL IS TRANSFERRED TO THIS ROUTINE IF
                         ; LENGTH OF CURRENT LINE (HEXBU) IS SHORTER THAN
                         ; LENGTH OF NEW LINE (LOCC).
                         ;
2118  A2 00              LADJ4   LDX #0                FIND LENGTH DIFFERENCE
211A  A1 D9                      LDA (LOCCL,X)
211C  E1 DD                      SBC (HEXBUL,X)
211E  85 D8                      STA TEMPR             SAVE IT
2120  20 19 24                   JSR MOV5
2123  20 77 23                   JSR MOVMB             MOVE MEMORY BLOCK
2126  20 ED 23                   JSR MOV3
2129  20 60 23                   JSR LSTLC             CHECK IF LAST LINE
212C  D0 06                      BNE LADJ46
```

```
231  212E  20 3C 23          JSR INCLC          UPDATE LOCC
232  2131  4C 48 21          JMP LADJ47
233  2134  20 3C 23  LADJ46   JSR INCLC
234  2137  18                CLC
235  2138  A5 D4             LDA MENDH          SET UP MOVMB PARAMETERS
236  213A  85 D2             STA MDESH
237  213C  A5 D3             LDA MENDL
238  213E  65 D8             ADC TEMPR
239  2140  90 04             BCC LADJ43
240  2142  E6 D2             INC MDESH
241  2144  F0 24             BEQ AL6
242  2146  85 D1    LADJ43   STA MDESL
243  2148  20 9A 23          JSR MOVMR          MOVE MEMORY BLOCK
244  214B  20 C2 23  LADJ47   JSR MOV1
245  214E  20 77 23          JSR MOVMB
246  2151  A0 00             LDY #0
247  2153  38                SEC
248  2154  A5 D9             LDA LOCCL
249  2156  F1 D5             SBC (LCCCL),Y
250  2158  B0 02             BCS LADJ44
251  215A  C6 DA             DEC LOCCH
252  215C  18        LADJ44  CLC
253  215D  65 D8             ADC TEMPR
254  215F  85 D9             STA LOCCL
255  2161  90 04             BCC LADJ45
256  2163  E6 DA             INC LOCCH
257  2165  F0 03             BEQ AL6
258  2167  4C 3C 20  LADJ45  JMP WSTAT
259  216A  4C 74 22  AL6     JMP INVM2
260                          ;
261                          ; LINE ADJUST 5 (LADJ5)
262                          ;
263                          ; CONTROL IS TRANSFERRED TO THIS ROUTINE IF LENGTH
264                          ; OF CURRENT LINE (HEXBU) IS LONGER THAN LENGTH OF
265                          ; NEW LINE (LOCC).
266                          ;
267  216D  38        LADJ5   SEC                FIND LENGTH DIFFERENCE
268  216E  A0 00             LDY #0
269  2170  B1 DD             LDA (HEXBUL),Y
270  2172  F1 D9             SBC (LCCCL),Y
271  2174  85 D8             STA TEMPR
272  2176  20 ED 23          JSR MOV3           SET PARAMETERS FOR MOVMB
273  2179  A5 DE             LDA HEXBUH
274  217B  85 D2             STA MDESH
275  217D  A5 DD             LDA HEXBUL
276  217F  18                CLC
277  2180  A2 00             LDX #0
278  2182  61 D9             ADC (LCCCL,X)
279  2184  90 04             BCC LADJ51
280  2186  E6 D2             INC MDESH
281  2188  F0 E0             BEQ AL6
282  218A  85 D1    LADJ51   STA MDESL
283  218C  20 60 23          JSR LSTLC          IS HEXBU AT LAST LINE
284  218F  F0 03             BEQ LADJ52
285  2191  20 77 23          JSR MOVMB
286  2194  20 C2 23  LADJ52  JSR MOV1
287  2197  20 77 23          JSR MOVMB
288  219A  20 E8 22          JSR DCLC           UPDATE LOCATION COUNTER
289  219D  4C 3C 20          JMP WSTAT
290                          ;
291                          ; LINE INSERT (LINS)
292                          ;
293                          ; CONTROL IS TRANSFERRED TO THIS ROUTINE FOR A
294                          ; LINE INSERTION.
295                          ;
296  21A0  20 19 24  LINS    JSR MOV5           INITIALIZE FOR MOVMB
297  21A3  20 77 23          JSR MOVMB
298  21A6  A5 DD             LDA HEXBUL         INITIALIZE AGAIN
299  21A8  85 D5             STA MBEGL
300  21AA  A5 DE             LDA HEXBUH
301  21AC  85 D6             STA MBEGH
302  21AE  20 00 24          JSR MOV33
303  21B1  20 1C 24          JSR MOV52
304  21B4  A5 D1             LDA MDESL
305  21B6  38                SEC
306  21B7  E9 01             SBC #1
307  21B9  B0 C2             BCS LINS1
308  21BB  C6 D2             DEC MDESH
309  21BD  85 D1    LINS1    STA MDESL
310  21BF  20 3C 23          JSR INCLC
311  21C2  20 9A 23          JSR MOVMR          MOVE MEMORY BLOCK
312  21C5  20 C2 23          JSR MOV1
313  21C8  20 77 23          JSR MOVMB          MOVE MEMORY BLOCK
314  21CB  4C 3C 20          JMP WSTAT
315                          ;
316                          ; COMMAND HANDLER (CMHD)
317                          ;
318                          ; CONTROL IS TRANSFERRED TO THIS ROUTINE IF A
319                          ; COMMAND IS TYPED ON THE CONSOLE DEVICE. A CHECK
320                          ; IS MADE TO SEE IF ASCBUM MATCHES THE FIRST
321                          ; LETTER OF ANY OF THE VALID LEDIP COMMANDS: VIZ
322                          ; LIST, TEXT, FILE, AND EXIT. IF A MATCH IS FOUND,
323                          ; LEDIP WAITS FOR A CARRIAGE-RETURN AND THEN
324                          ; TRANSFERS CONTROL TO THE APPROPRIATE ROUTINE.
325                          ; ERROR MESSAGE "C" IS TYPED AND CONTROL
326                          ; RETURNED TO THE WARM START OTHERWISE.
327                          ;
328  21CE  A5 E2     CMHD    LDA ASCBUM         FIRST LETTER OF COMMAND
329  21D0  29 DF             AND #%11011111     MASK, IF LOWER CASE
330  21D2  C9 45             CMP #'E            IS IT E
331  21D4  F0 16             BEQ EXIT           IF SO, EXIT
332  21D6  C9 46             CMP #'F            IS IT F
333  21D8  F0 28             BEQ FILE           IF SO, FILE
334  21DA  C9 54             CMP #'T            IS IT T
335  21DC  F0 51             BEQ TEXT           IF SO, TEXT
336  21DE  C9 4C             CMP #'L            IS IT L
337  21E0  F0 6C             BEQ LIST           IF SO, LIST
338  21E2  C9 43             CMP #'C            IS IT C
339  21E4  F0 13             BEQ CLEAR          IF SO, CLEAR
340  21E6  EA                NOP                ** COMMAND EXTENSION **
341  21E7  EA                NOP                ** COMMAND EXTENSION **
342  21E8  EA                NOP                ** COMMAND EXTENSION **
343  21E9  4C 84 22          JMP INVC
344  21EC  20 B8 22  EXIT    JSR CRSEN          WAIT FOR CR
345  21EF  A9 20             LDA #$20           WSTAT HIGH ORDER BYTE
346  21F1  48                PHA
347  21F2  A9 3C             LDA #$3C           LOW BYTE (TIM, $3D)
348  21F4  48                PHA
349  21F5  08                PHP
350  21F6  4C CC 1C          JMP $1CCC          (USE $7052 FOR TIM)
351  21F9  20 4B 24  CLEAR   JSR GETCH
352  21FC  20 B8 22          JSR CRSEN          WAIT FOR CR
353  21FF  4C 00 20          JMP CSTAT
354                          ;
355                          ; FILE COMMAND (FILE)
356                          ;
357                          ; THIS ROUTINE STATES THE BLOCKS OF MEMORY
358                          ; CURRENTLY BEING USED.
359                          ;
360  2202  20 B8 22  FILE    JSR CRSEN          WAIT FOR CR
361  2205  A0 EA             LDY #$EA           INIT X-REG FOR INDEXING
362  2207  B9 9E 23  FILE1   LDA FTAB-$EA,Y
363  220A  20 54 24          JSR OUTCH          TYPE A CHARACTER
364  220D  C8                INY
365  220E  D0 F7             BNE FILE1          NEXT CHARACTER
366  2210  20 42 24          JSR CRLF
367  2213  A5 DC             LDA STADH          START ADDRESS HIGH BYTE
368  2215  20 66 24          JSR OUTBYT
369  2218  A5 DB             LDA STADL          LOW BYTE
370  221A  20 66 24          JSR OUTBYT
```

```
371  221D  A9 2D        LDA #'-
372  221F  20 54 24     JSR OUTCH
373  2222  A5 DA        LDA LOCCH          LOCATION COUNTER
374  2224  20 66 24     JSR OUTBYT
375  2227  A5 D9        LDA LOCCL
376  2229  20 66 24     JSR OUTBYT
377  222C  4C 3C 20  ECMD  JMP WSTAT        RETURN TO WARM START
378                     ;
379                     ; TEXT COMMAND (TEXT)
380                     ;
381                     ; THIS ROUTINE LISTS THE CURRENT TEXT
382                     ; WITHOUT LINE NUMBERS.
383                     ;
384  222F  20 B8 22  TEXT  JSR CRSEN        WAIT FOR CR
385  2232  20 9E 22     JSR CLHS           TYPE CR, SET HEXBU
386  2235  20 AD 22  TEXT1 JSR CMPHL        CHECK IF LAST LINE
387  2238  D0 06        BNE TEXT2
388  223A  20 42 24     JSR CRLF
389  223D  4C 3C 20     JMP WSTAT
390  2240  20 F4 22  TEXT2 JSR DPASC        TYPE A LINE OF TEXT
391  2243  A9 7F        LDA #$7F           TYPE RUB-OUT
392  2245  20 54 24     JSR OUTCH
393  2248  20 2C 23     JSR INCHB          SET FOR NEXT LINE
394  224B  4C 35 22     JMP TEXT1
395                     ;
396                     ; LIST COMMAND (LIST)
397                     ;
398                     ; THIS ROUTINE LISTS THE CURRENT TEXT
399                     ; WITH LINE NUMBERS.
400                     ;
401  224E  20 B8 22  LIST  JSR CRSEN        WAIT FOR CR
402  2251  20 A4 22     JSR HEXST          SET HEXBU TO STAD
403  2254  20 AD 22  LIST1 JSR CMPHL
404  2257  F0 D3        BEQ ECMD
405  2259  A0 02        LDY #2
406  225B  B1 DD        LDA (HEXBUL),Y     TYPE LINE NUMBER
407  225D  20 66 24     JSR OUTBYT
408  2260  88          DEY
409  2261  B1 DD        LDA (HEXBUL),Y
410  2263  20 66 24     JSR OUTBYT
411  2266  20 5D 24     JSR SPACE          TYPE A SPACE
412  2269  20 F4 22     JSR DPASC          TYPE A LINE OF TEXT
413  226C  20 2C 23     JSR INCHB
414  226F  4C 54 22     JMP LIST1          NEXT LINE
415                     ;
416                     ; INVALID MEMORY (INVM1,INVM2)
417                     ;
418                     ; THIS MULTIPLE ENTRY ROUTINE PRINTS ERROR
419                     ; MESSAGE "M" ON THE CONSOLE DEVICE AND RETURNS
420                     ; CONTROL TO THE WARM START. INVM1 RESTORES THE
421                     ; STACK WHILE INVM2 DOES NOT.
422                     ;
423  2272  68        INVM1 PLA                RESTORE STACK
424  2273  68           PLA
425  2274  20 42 24  INVM2 JSR CRLF
426  2277  A9 4D        LDA #'M
427  2279  20 54 24     JSR OUTCH          TYPE M
428  227C  A9 21        LDA #$21
429  227E  20 54 24     JSR OUTCH          TYPE EXCLAMATION
430  2281  4C 3C 20     JMP WSTAT
431                     ;
432                     ; INVALID COMMAND (INVC)
433                     ;
434                     ; THIS ROUTINE TYPES ERROR MESSAGE "C" ON THE
435                     ; CONSOLE DEVICE AND TRANSFERS CONTROL TO THE WARM
436                     ; START.
437                     ;
438  2284  20 42 24  INVC  JSR CRLF
439  2287  A9 43        LDA #'C
440  2289  20 54 24     JSR OUTCH          TYPE C
441  228C  A9 21        LDA #$21
442  228E  20 54 24     JSR OUTCH          TYPE EXCLAMATION
443  2291  4C 3C 20     JMP WSTAT
444                     ;
445                     ;
446                     ;*************************************************;
447                     ;*************************************************;
448                     ;**                                             **;
449                     ;**  SUBROUTINES FOLLOW IN ALPHABETICAL ORDER   **;
450                     ;**                                             **;
451                     ;*************************************************;
452                     ;*************************************************;
453                     ;
454                     ;
455                     ; ASCII BIAS (ABIAS)
456                     ;
457                     ; THIS MULTIPLE EXIT SUBROUTINE ADDS THE
458                     ; APPROPRIATE BIAS TO THE ASCII CONTENT OF THE
459                     ; ACCUMULATOR TO MAKE IT A HEX CHARACTER.
460                     ; A DESTROYED, X AND Y PRESERVED.
461                     ;
462  2294  C9 3A     ABIAS  CMP #$3A          COMPARE WITH 9
463  2296  B0 03        BCS ABIAS1         MUST BE A-F IF GREATER
464  2298  E9 2F        SBC #$2F           BIAS FOR 0 THRU 9
465  229A  60           RTS
466  229B  E9 37     ABIAS1 SBC #$37          BIAS FOR A THRU F
467  229D  60           RTS
468                     ;
469                     ; CRLF, HEXBU SET TO STAD (CLHS)
470                     ; HEXBU SET TO STAD (HEXST)
471                     ;
472                     ; THIS MULTIPLE ENTRY SUBROUTINE TYPES A CRLF
473                     ; TWICE AND/OR SETS HEXBU EQUAL TO STAD. ENTRY
474                     ; POINT TO JUST SET HEXBU TO STAD IS HEXST.
475                     ; ENTRY POINT FOR BOTH IS CLHS.
476                     ; A DESTROYED, X AND Y PRESERVED.
477                     ;
478  229E  20 42 24  CLHS  JSR CRLF          TYPE CR, LF
479  22A1  20 42 24     JSR CRLF
480  22A4  A5 DB     HEXST  LDA STADL         SET LOW ORDER BYTE
481  22A6  85 DD        STA HEXBUL
482  22A8  A5 DC        LDA STADH          HIGH ORDER BYTE TOO
483  22AA  85 DE        STA HEXBUH
484  22AC  60           RTS
485                     ;
486                     ; COMPARE HEXBU AND LOCC (CMPHL)
487                     ;
488                     ; THIS SUBROUTINE COMPARES THE CONTENTS OF
489                     ; HEXBU AND LOCC. ZERO FLAG IS SET IF THEY
490                     ; ARE EQUAL. ZERO FLAG IS OTHERWISE CLEARED.
491                     ; A DESTROYED, X AND Y PRESERVED.
492                     ;
493  22AD  A5 DD     CMPHL  LDA HEXBUL         COMPARE LOW ORDER BYTES
494  22AF  C5 D9        CMP LOCCL
495  22B1  D0 04        BNE CMPHL1
496  22B3  A5 DE        LDA HEXBUH         HIGH ORDER BYTES TOO
497  22B5  C5 DA        CMP LOCCH
498  22B7  60        CMPHL1 RTS
499                     ;
500                     ; CARRIAGE-RETURN SENSE (CRSEN)
501                     ;
502                     ; THIS SUBROUTINE RECEIVES A CHARACTER FROM THE
503                     ; CONSOLE DEVICE AND CHECKS TO SEE IF IT IS A CR.
504                     ; IF SO, A CRLF IS ECHOED. IF NOT, ERROR MESSAGE
505                     ; C IS TYPED AND CONTROL RETURNED TO THE WARM START.
506                     ; A DESTROYED, X AND Y PRESERVED.
507                     ;
508  22B8  20 48 24  CRSEN  JSR GETCH         RECEIVE CHARACTER
509  22BB  C9 0D        CMP #$0D           IS IT CARRIAGE-RETURN
510  22BD  F0 C5        BEQ CRSEN1
```

```
0511 228F 68          PLA             RESTORE STACK
0512 2290 68          PLA
0513 2291 4C 84 22    JMP INVC        TYPE ERROR IF NOT
0514 2294 20 42 24 CRSEN1 JSR CRLF    ELSE ECHO WITH CRLF
0515 2297 60          RTS
0516                  ;
0517                  ; CONVERT ASCII TO HEX (CVAH)
0518                  ;
0519                  ; THIS SUBROUTINE CONVERTS THE ASCII CONTENTS OF
0520                  ; ASCBU (ASCII BUFFER) INTO THE HEX EQUIVALENT
0521                  ; AND STORES THE ANSWER IN HEXBU (HEX BUFFER).
0522                  ; ABIAS CALLED, A, X, TEMPR DESTROYED, Y CLEARED.
0523                  ;
0524 2298 A0 02    CVAH    LDY #2          SET INDEX
0525 229A A2 04            LDX #4          SET INDEX
0526 229C B5 DE    CVAH1   LDA ASCBUM-4,X  GET CHARACTER
0527 229E 20 94 22         JSR ABIAS       APPLY BIAS AND SHIFT
0528 22D1 0A               ASL A
0529 22D2 0A               ASL A
0530 22D3 0A               ASL A
0531 22D4 0A               ASL A
0532 22D5 85 D8            STA TEMPR
0533 22D7 CA               DEX
0534 22D8 B5 DE            LDA ASCBUM-4,X  NEXT CHARACTER
0535 22DA 20 94 22         JSR ABIAS
0536 22DD 18               CLC
0537 22DE 65 D8            ADC TEMPR
0538 22E0 99 DC 00         STA HEXBUH-2,Y  STORE IN HEXBU
0539 22E3 CA               DEX
0540 22E4 88               DEY
0541 22E5 D0 E5            BNE CVAH1       NEXT CHARACTER
0542 22E7 60               RTS
0543                  ;
0544                  ; DECREMENT LOCATION COUNTER (DCLC)
0545                  ;
0546                  ; THIS SUBROUTINE SUBTRACTS TEMPR (TEMPORARY
0547                  ; REGISTER) FROM LOCC (LOCATION COUNTER) AND STORES
0548                  ; THE ANSWER IN LOCC.
0549                  ; A DESTROYED, X AND Y PRESERVED.
0550                  ;
0551 22E8 38       DCLC    SEC
0552 22E9 A5 D9            LDA LOCCL       GET LOCC LOW ORDER BYTE
0553 22EB E5 D8            SBC TEMPR       SUBTRACT TEMPR
0554 22ED 85 D9            STA LOCCL
0555 22EF B0 02            BCS DCLC1
0556 22F1 C6 DA            DEC LOCCH       PROPAGATE BORROW
0557 22F3 60       DCLC1   RTS
0558                  ;
0559                  ; DUMP ASCII (DPASC)
0560                  ;
0561                  ; THIS SUBROUTINE TYPES THE STRING OF  ASCII
0562                  ; CHARACTERS CONTAINED IN THE DATA BLOCK POINTED
0563                  ; TO BY HEXBU (HEX BUFFER).
0564                  ; A AND Y DESTROYED, X CLEARED.
0565                  ;
0566 22F4 A0 03    DPASC   LDY #3          SET INDEX
0567 22F6 B1 D0    DPASC1  LDA (HEXBUL),Y  GET CHARACTER
0568 22F8 20 54 24         JSR OUTCH       TYPE IT
0569 22FB C8               INY             SET FOR NEXT CHARACTER
0570 22FC 98               TYA
0571 22FD A2 00            LDX #0          CHECK IF LAST CHARACTER
0572 22FF C1 DD            CMP (HEXBUL,X)
0573 2301 D0 F3            BNE DPASC1      NEXT CHARACTER
0574 2303 20 42 24         JSR CRLF
0575 2306 60               RTS
0576                  ;
0577                  ; HEXADECIMAL CHECK (HCHK4)
0578                  ; DECIMAL CHECK (DCHK4)
0579                  ;
0580                  ; THIS MULTIPLE ENTRY SUBROUTINE CHECKS TO SEE IF
0581                  ; ALL THE CONTENTS OF ASCBU (ASCII BUFFER) ARE
0582                  ; VALID HEXADECIMAL OR DECIMAL CHARACTERS. ENTRY
0583                  ; POINT FOR HEX CHECK IS HCHK4. ENTRY POINT FOR
0584                  ; DECIMAL CHECK IS DCHK4. C-FLAG IS CLEARED IF AN
0585                  ; INVALID CHARACTER IS FOUND. C-FLAG IS OTHERWISE
0586                  ; SET.
0587                  ; A, X, AND Y DESTROYED.
0588                  ;
0589 2307 AC FF    HCHK4   LDY #$FF        SET FOR HEX CHECK
0590 2309 4C 0E 23         JMP CHEK1
0591 230C A0 00    DCHK4   LDY #0          SET FOR DECIMAL CHECK
0592 230E A2 04    CHEK1   LDX #4
0593 2310 B5 DE    CHEK2   LDA ASCBUM-4,X  GET A BYTE
0594 2312 C0 00            CPY #0
0595 2314 F0 08            BEQ DCHK
0596 2316 C9 47    HCHK    CMP #$47        CHECK IF HEX
0597 2318 B0 0C            BCS ECHK1
0598 231A C9 41            CMP #$41
0599 231C B0 0A            BCS ECHK2
0600 231E C9 3A    DCHK    CMP #$3A        CHECK IF DECIMAL
0601 2320 B0 04            BCS ECHK1
0602 2322 C9 30            CMP #$30
0603 2324 B0 02            BCS ECHK2
0604 2326 18       ECHK1   CLC             INVALID CHAR, SET C=0
0605 2327 60               RTS
0606 2328 CA       ECHK2   DEX
0607 2329 D0 E5            BNE CHEK2       NEXT CHARACTER
0608 232B 60       ECHK    RTS
0609                  ;
0610                  ; INCREMENT HEXBU (INCHB)
0611                  ;
0612                  ; THIS SUBROUTINE ADDS THE CONTENTS OF THE
0613                  ; MEMORY LOCATION POINTED TO BY HEXBU TO
0614                  ; HEXBU AND STORES THE ANSWER IN HEXBU.
0615                  ; A DESTROYED, X CLEARED, Y PRESERVED.
0616                  ;
0617 232C A2 00    INCHB   LDX #0          CLEAR INDEX
0618 232E A1 DD            LDA (HEXBUL,X)  GET (HEXBU)
0619 2330 18               CLC
0620 2331 65 DD            ADC HEXBUL      ADD TO LOW ORDER BYTE
0621 2333 85 DD            STA HEXBUL
0622 2335 90 04            BCC INCHB1
0623 2337 E6 DE            INC HEXBUH      PROPAGATE CARRY
0624 2339 F0 10            BEQ INCLC?      ERROR IF OVER 65K
0625 233B 60       INCHB1  RTS
0626                  ;
0627                  ; INCREMENT LOCATION COUNTER (INCLC)
0628                  ;
0629                  ; THIS SUBROUTINE ADDS THE CONTENTS OF THE
0630                  ; MEMORY LOCATION POINTED TO BY LOCC TO THE
0631                  ; THE VALUE OF LOCC AND STORES THE ANSWER IN LOCC.
0632                  ; A DESTROYED, X CLEARED, Y PRESERVED.
0633                  ;
0634 233C A2 00    INCLC   LDX #0          CLEAR INDEX
0635 233E A1 D9            LDA (LOCCL,X)   GET (LOCC)
0636 2340 18               CLC
0637 2341 65 D9            ADC LOCCL       ADD TO LOW ORDER BYTE
0638 2343 85 D9            STA LOCCL
0639 2345 90 07            BCC INCLC1
0640 2347 E6 DA            INC LOCCH       PROPAGATE CARRY
0641 2349 D0 03            BNE INCLC1
0642 234B 4C 72 22 INCLC2  JMP INVM1       ERROR IF OVER 65K
0643 234E 60       INCLC1  RTS
0644                  ;
0645                  ; LINE NUMBER CHECK (LNCHK)
0646                  ;
```

```
                        ; THIS SUBROUTINE COMPARES THE LINE NUMBER IN THE
                        ; CURRENT LOCC (LOCATION COUNTER) TO THE LINE NUMBER
                        ; IN HEXBU (HEX BUFFER). ZERO FLAG IS SET IF AN
                        ; IDENTICAL LINE NUMBER IS FOUND IN HEXBU. CARRY
                        ; FLAG IS CLEARED IF A HIGHER LINE NUMBER IS FOUND
                        ; IN HEXBU. HEXBU IS LEFT AS IT IS.
                        ; A AND Y DESTROYED, X PRESERVED.
                        ;
234F  A0 02     LNCHK   LDY #2              SET INDEX
2351  B1 D9             LDA (LOCCL),Y
2353  D1 DD             CMP (HEXBUL),Y      COMPARE LOW ORDER BYTES
2355  90 02             BCC LNCHK1
2357  F0 01             BEQ LNCHK2
2359  60        LNCHK1  RTS
235A  88        LNCHK2  DEY
235B  B1 D9             LDA (LOCCL),Y
235D  D1 DD             CMP (HEXBUL),Y      HIGH ORDER BYTES TOO
235F  60                RTS
                        ;
                        ; LAST LINE CHECK (LSTLC)
                        ;
                        ; THIS SUBROUTINE CHECKS FOR INVALID MEMORY
                        ; BLOCK MOVES WHICH MAY RESULT FROM AN ATTEMPT
                        ; TO ADJUST THE LAST LINE OF TEXT.
                        ; A DESTROYED, X AND Y PRESERVED.
                        ;
2360  A5 D5     LSTLC   LDA MBEGL           CHECK LOW ORDER BYTES
2362  C5 D9             CMP LOCCL
2364  F0 01             BEQ LSTLC1
2366  60                RTS
2367  A5 D6     LSTLC1  LDA MBEGH           HIGH ORDER BYTES TOO
2369  C5 DA             CMP LOCCH
236B  60                RTS
                        ;
                        ; MEMORY MOVE CHECK (MCHEK)
                        ;
                        ; THIS SUBROUTINE CHECKS TO SEE IF A BLOCK MEMORY
                        ; MOVE IS COMPLETE. ZERO FLAG IS SET IF SO.
                        ; A DESTROYED, X AND Y PRESERVED.
                        ;
236C  A5 D3     MCHEK   LDA MENDL           COMPARE LOW ORDER BYTES
236E  C5 D5             CMP MBEGL
2370  D0 04             BNE ENDMC
2372  A5 D4             LDA MENDH           HIGH ORDER BYTES TOO
2374  C5 D6             CMP MBEGH
2376  60        ENDMC   RTS
                        ;
                        ; MOVE MEMORY BLOCK (MOVMB)
                        ;
                        ; THIS SUBROUTINE MOVES THE CONTENTS OF MEMORY BLOCK
                        ; MBEG (MEMORY BEGIN) TO MEND (MEMORY END) INCLUSIVE
                        ; TO THE MEMORY BLOCK BEGINNING AT LOCATION MDES
                        ; (MEMORY DESTINATION). BYTES ARE MOVED IN LOW
                        ; ADDRESS TO HIGH ADDRESS SEQUENCE.
                        ; MBEG AND MDES DESTROYED, Y CLEARED, A DESTROYED,
                        ; X PRESERVED.
                        ;
2377  A0 00     MOVMB   LDY #0              CLEAR INDEX
2379  B1 D5     MOVM1   LDA (MBEGL),Y       MOVE A BYTE
237B  91 D1             STA (MDESL),Y
237D  D1 D1             CMP (MDESL),Y       READ-AFTER-WRITE CHECK
237F  F0 03             BEQ AL1
2381  4C 72 22  AL5     JMP INVM1
2384  20 6C 23  AL1     JSR MCHEK           IS BLOCK MOVE COMPLETE
2387  F0 10             BEQ ENDM
2389  E6 D5             INC MBEGL           SET FOR NEXT BYTE IF NOT
238B  D0 02             BNE MOVM3
238D  E6 D6             INC MBEGH
238F  E6 D1     MOVM3   INC MDESL
2391  D0 E6             BNE MOVM1
2393  E6 D2             INC MDESH           PROPAGATE CARRY
2395  D0 E2             BNE MOVM1           NEXT BYTE
2397  F0 E8             BEQ AL5
2399  60        ENDM    RTS
                        ;
                        ; MOVE MEMORY BLOCK REVERSE (MOVMR)
                        ;
                        ; THIS SUBROUTINE IS IDENTICAL TO MOVMB (MOVE MEMORY
                        ; BLOCK) WITH TWO EXCEPTIONS: VIZ. (1) BYTES ARE
                        ; MOVED IN HIGH ADDRESS TO LOW ADDRESS SEQUENCE,
                        ; (2) MDES INDICATES END (AND NOT BEGINNING) OF
                        ; DESTINATION BLOCK.
                        ; MEND AND MDES DESTROYED. Y CLEARED, A DESTROYED,
                        ; X PRESERVED.
                        ;
239A  A0 00     MOVMR   LDY #0              CLEAR INDEX
239C  B1 D3     MOVR1   LDA (MENDL),Y       MOVE A BYTE
239E  91 D1             STA (MDESL),Y
23A0  D1 D1             CMP (MDESL),Y       READ-AFTER-WRITE CHECK
23A2  D0 DD             BNE AL5
23A4  20 6C 23          JSR MCHEK           IS MOVE COMPLETE
23A7  F0 18             BEQ ENDMR
23A9  38                SEC                 SET FOR NEXT BYTE IF NOT
23AA  A5 D3             LDA MENDL
23AC  E9 01             SBC #1
23AE  85 D3             STA MENDL
23B0  B0 02             BCS MOVR2
23B2  C6 D4             DEC MENDH
23B4  38        MOVR2   SEC
23B5  A5 D1             LDA MDESL
23B7  E9 01             SBC #1
23B9  85 D1             STA MDESL
23BB  B0 DF             BCS MOVR1
23BD  C6 D2             DEC MDESH
23BF  90 D8             BCC MOVR1           NEXT BYTE
23C1  60        ENDMR   RTS
                        ;
                        ; MEMORY MOVE INITIALIZE (MOV1)
                        ;
                        ; MBEG = LOCC
                        ; MEND = LOCC + (LOCC) - 1
                        ; MDES = HEXBU
                        ;
23C2  20 C9 23  MOV1    JSR MOV2
23C5  20 10 24          JSR MOV4
23C8  60                RTS
                        ;
                        ; MEMORY MOVE INITIALIZE (MOV2)
                        ;
                        ; MBEG = LOCC
                        ; MEND = LOCC + (LOCC) - 1
                        ;
23C9  A5 DA     MOV2    LDA LOCCH           SET HIGH ORDER BYTES
23CB  85 D6             STA MBEGH
23CD  85 D4             STA MENDH
23CF  A5 D9             LDA LOCCL           LOW ORDER BYTES TOO
23D1  85 D5             STA MBEGL
23D3  18                CLC
23D4  A2 00             LDX #0
23D6  61 D9             ADC (LOCCL,X)       ADD CONTENTS OF LOCC
23D8  90 09             BCC MOV21
23DA  E6 D4             INC MENDH
23DC  D0 05             BNE MOV21
23DE  68                PLA                 RESTORE STACK
23DF  68                PLA
23E0  4C 72 22  AL11    JMP INVM1           TYPE ERROR
23E3  38        MOV21   SEC
23E4  E9 01             SBC #1
23E6  B0 02             BCS MOV22
23E8  C6 D4             DEC MENDH
```

```
0787  23EA  85 D3      MOV22   STA MENDL
0788  23EC  60                 RTS
0789                    ;
0790                    ; MEMORY MOVE INITIALIZE (MOV3, MOV33)
0791                    ;
0792                    ; MBEG = HEXBU + (HEXBU)
0793                    ; MEND = LOCC - 1
0794                    ;
0795  23ED  A5 DE      MOV3    LDA HEXBUH          SET TO HEXBU
0796  23EF  85 D6             STA MBEGH
0797  23F1  A5 DD             LDA HEXBUL
0798  23F3  18                CLC
0799  23F4  A2 00              LDX #0
0800  23F6  61 DD             ADC (HEXBUL,X)      ADD CONTENTS OF HEXBU
0801  23F8  90 04              BCC MOV31
0802  23FA  E6 D6             INC MBEGH
0803  23FC  F0 E2              BEQ ALL1
0804  23FE  85 D5      MOV31   STA MBEGL
0805  2400  A5 DA      MOV33   LDA LOCCH           MEND = LOCC - 1
0806  2402  85 D4             STA MENDH
0807  2404  A5 D9             LDA LOCCL
0808  2406  38                SEC
0809  2407  E9 01              SBC #1
0810  2409  B0 02              BCS MOV32
0811  240B  C6 D4             DEC MENDH
0812  240D  85 D3      MOV32   STA MENDL
0813  240F  60                 RTS
0814                    ;
0815                    ; MEMORY MOVE INITIALIZE (MOV4)
0816                    ;
0817                    ; MDES = HEXBU
0818                    ;
0819  2410  A5 DD      MOV4    LDA HEXBUL
0820  2412  85 D1             STA MDESL
0821  2414  A5 DE             LDA HEXBUH
0822  2416  85 D2             STA MDESH
0823  2418  60                 RTS
0824                    ;
0825                    ; MEMORY MOVE INITIALIZE (MOV5, MOV52)
0826                    ;
0827                    ; MBEG = LOCC
0828                    ; MEND = LOCC + (LOCC) - 1
0829                    ; MDES = LOCC + (LOCC)
0830                    ;
0831  2419  20 C9 23  MOV5    JSR MOV2
0832  241C  A5 DA      MOV52   LDA LOCCH           MDES = LOCC + (LOCC)
0833  241E  85 D2             STA MDESH
0834  2420  A2 00              LDX #0
0835  2422  18                CLC
0836  2423  A5 D9             LDA LOCCL
0837  2425  61 D9             ADC (LOCCL,X)
0838  2427  90 04              BCC MOV51
0839  2429  E6 D2             INC MDESH
0840  242B  F0 B3              BEQ ALL1
0841  242D  85 D1      MOV51   STA MDESL
0842  242F  60                 RTS
0843                    ;
0844                    ; READ ASCII (RDASC)
0845                    ;
0846                    ; THIS SUBROUTINE READS FOUR ASCII CHARACTERS FROM
0847                    ; THE CONSOLE DEVICE AND STORES THEM AS RECEIVED
0848                    ; IN ASCBU (ASCII BUFFER). FIRST CHARACTER
0849                    ; RECEIVED IS STORED IN HIGHEST LOCATION (ASCBUM).
0850                    ; X CLEARED, A DESTROYED, Y PRESERVED.
0851                    ;
0852  243C  A2 04      RDASC   LDX #4              SET INDEX
0853  243E  20 4B 24   RDASC1  JSR GETCH          GET A CHARACTER
0854  2441  95 DE              STA ASCBUM-4,X      STORE IT
0855  2443  CA                 DEX
0856  2444  D0 F8              BNE RDASC1          NEXT CHARACTER
0857  2446  60                 RTS
0858                    ;
0859                    ; SAVE REGISTERS (SAVR)
0860                    ;
0861  2447  85 D5      SAVR    STA MBEGL
0862  2449  86 D3             STX MENDL
0863  244B  84 D1             STY MDESL
0864  244D  60                 RTS
0865                    ;
0866                    ; I/O JUMPS
0867                    ;
0868  244E  20 47 24   CRLF    JSR SAVR            CARRIAGE-RET LINE-FEED
0869  2451  20 2F 1E           JSR $1E2F           (USE $723A FOR TIM)
0870  2454  4C 6C 24           JMP RESR
0871  2457  20 47 24   GETCH   JSR SAVR            READ A CHARACTER
0872  245A  20 5A 1E           JSR $1E5A           (USE $72E9 FOR TIM)
0873  245D  4C 6E 24           JMP RESR1
0874  2460  20 47 24   CUTCH   JSR SAVR            TYPE A CHARACTER
0875  2463  20 AC 1E           JSR $1EAC           (USE $72C6 FOR TIM)
0876  2466  4C 6C 24           JMP RESR
0877  2469  20 47 24   SPACE   JSR SAVR            TYPE A SPACE
0878  246C  20 9E 1E           JSR $1E9E           (USE $7377 FOR TIM)
0879  246F  4C 6C 24           JMP RESR
0880  2472  20 47 24   CUTBYT  JSR SAVR            TYPE A BYTE
0881  2475  20 3B 1E           JSR $1E3B           (USE $7281 FOR TIM)
0882  2478  A5 D5      RESR    LDA MBEGL           RESTORE REGISTERS
0883  247A  A6 D3      RESR1   LDX MENDL
0884  247C  A4 D1             LDY MDESL
0885  247E  EA                NOP                 ** FOR BREAK TEST **
0886  247F  EA                NOP                 ** FOR BREAK TEST **
0887  2480  EA                NOP                 ** FOR BREAK TEST **
0888  2481  60                RTS
0889                    ;
0890                    ; ASCII TABLES
0891                    ;
0892  2482  53 54      STADO   .BYTE 'STARTING ADDRESS? '
0892  2484  41 52
0892  2486  54 49
0892  2488  4E 47
0892  248A  20 41
0892  248C  44 44
0892  248E  52 45
0892  2490  53 53
0892  2492  3F 20
0893  2494  30 30      FTAB    .BYTE '0001-00E2'
0893  2496  44 31
0893  2498  2D 30
0893  249A  30 45 32
0894  249D  00                 .BYTE $00, $00, $0A, $00
0894  249E  00
0894  249F  0A
0894  24A0  00
0895  24A1  32 30              .BYTE '2000-2490'
0895  24A3  30 30
0895  24A5  2D 32
0895  24A7  34 39 44
0896                    .END

END OF MOS/TECHNOLOGY 650X ASSEMBLY VERSION 4
NUMBER OF ERRORS =    0,  NUMBER OF WARNINGS =    0
```