APPLE UTILITIES

EXTRA AMPER

Extra Amper is a protocol that allows you to load several of your favorite ampersand routines into auxiliary memory and call each of the routines individually.

If your Apple has 128K, Extra Amper can put the extended 64K of memory to good use. Extra Amper is an assembly language protocol that allows you to convert several of your favorite ampersand (&) routines into *ultrasmart* & routines. A *smart* & routine is one that can be used without interference when another & routine is already in memory. Extra Amper goes three steps further. It allows several routines to be stacked in memory; it uses auxiliary memory rather than main memory; and it has no fixed location in memory, which means it doesn't interfere with programs that do have fixed locations. Since the routines are in auxiliary memory, main memory is freed for other purposes, such as storing a long program or minimizing garbage collection.

The & routines are stored in auxiliary memory, but run from main memory. If the routines were run from auxiliary memory, they would have to be extensively rewritten.

See **Figure 1** for an overview of Extra Amper. When Extra Amper **(Listing 1)** is run, the entire program (including the & routines) is loaded into main memory at $2000. Because the numbering of the listing will change when you add your routines, it has been divided with section headings. During initialization (Section 1 of **Listing 1**), the program is divided into two parts. The command handler (Section 2) that parses an entered command is moved to main memory just above HIMEM, which has been reset to accommodate the command handler.

The program automatically detects if DOS 3.3 or ProDOS is running and adjusts the memory above HIMEM accordingly. The individual routines (Section 3) are dispatched to auxiliary memory beginning at $800 until they are called. When called, each routine is transferred to a routine buffer that resides just above the command handler, and control is passed to the selected routine.

COMMANDS USING EXTRA AMPER

Type the command RUN AMPER.LOADER to install Extra Amper. Then verify that each & routine works properly. To test the error message routine, type:

```
&
```

The NO COMMAND STRING error message will appear. Then type:

```
&A
```

and INCORRECT COMMAND STRING should appear.

Next, check out the hexadecimal-decimal (hex-dec) converter. Enter:

```
&$xxxx
```

This converts a hex number into its decimal equivalent. Enter:

```
&xxxxx
```

This converts a signed decimal value into its hex equivalent. For example, type:

```
&$3ØØ
```

The decimal equivalent, 768, will be returned. This routine is an example of a relocatable routine. Next, test the two nonrelocatable "dummy" routines by entering **&ROUT2** and **&CALL3**, and look for their respective messages.

ENTERING THE PROGRAM

If you have an assembler that can handle multiple ORGs, enter the source code as shown in **Listing 1** and assemble it using EXTRA.AMPER as the object file name.

If you don't have an assembler, or if your assembler can't handle multiple ORGs, enter the Monitor with CALL −151 and key in the hex code from **Listing 1 through line 374**. Then continue entering the hex codes shown in **Listing 2** and save the program with the command:

BSAVE EXTRA.AMPER,A$2000,L$31E

If you are using Key Perfect and an assembler that does not store zeros for the DS (or equivalent) pseudo-op, BLOAD EXTRA.AMPER and perform the following Monitor commands:

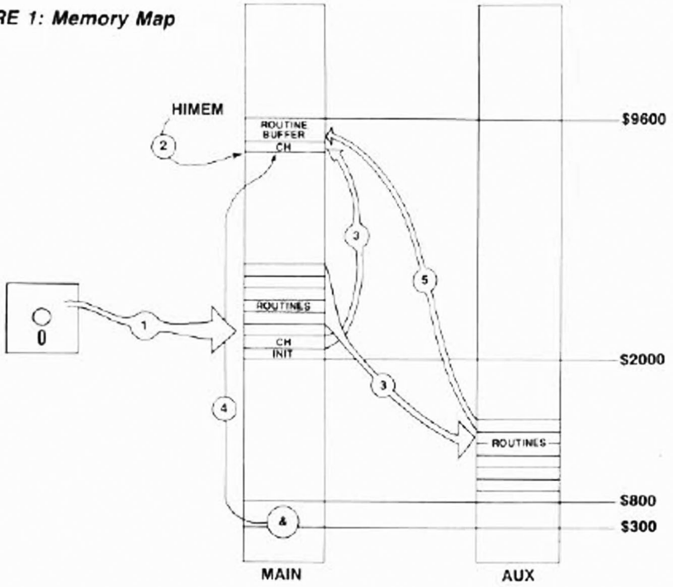
```
2Ø97:Ø N 2Ø98<2Ø97.2ØFEM
218F:Ø N 219Ø<218F.21FEM
```

Then save the program again using the BSAVE command above before running Key Perfect. Enter **Listing 3** and save it with the command: **SAVE AMPER.LOADER** For help with entering *Nibble* listings, see the Typing Tips section.

ADDING & ROUTINES

Now you're ready to add your own & routines to Extra Amper. While it's possible to construct a command list and develop the routine organization without an assembler, it's extremely difficult. (The cost of an assembler is cheap compared to the time you'll waste.) Carefully carry out each of the steps below.

FIGURE 1: Memory Map



LEGEND:

1. *The entire EXTRA.AMPER file is loaded into main memory at $2000.*
2. *HIMEN is set low enough to allow room for the command handier (CH) and routine buffer.*
3. *The command handier is moved above HIMEN and the routines are moved to auxiliary memory at $800.*
4. *The ampersand vector (&) is adjusted to point to the command handier.*
5. *When a particular routine is called, it is moved to the routine buffer.*

1. Remove the two dummy routines **(lines 445-483)**. Remove the hex-dec converter **(lines 379-463)** if you don't want to keep it. Do not remove **Section 3a**, the error message routine.
2. Insert your routines immediately following the error message routine in sequential order, and assign each one a routine number. (If you keep the hex-dec converter, start with routine two at **line 445**.) Label the beginning byte of each routine, BRT*n*a, and the end byte, ERT*n*a, where *n* is the routine number. If only have the object code for the routine, use a disassembler such as Sourceror on the Merlin disk to generate the source code. Be sure that only the main program is used. If the program begins with code that sets the & vector, delete this part of the program. You can recognize this as STA instructions with $3F6 and $3F7 as operands.
3. Enter any new equates at the beginning of the program, such as in **lines 47-57**.
4. Change the origin of *each* routine to ORG ROUTINE ($2200). See **lines 383 and 450** for examples.
5. At the end of each routine calculate the values of the symbols BRT*n* and ERT*n* according to the following formulas:

EXAMPLE 1: Ampersand Routine Before Conversion

```
                     *
                     * EXAMPLE1
                     * BY HAROLD PORTNOY
                     * COPYRIGHT (C) 1987
                     * BY MICROSPARC, INC.
                     * CONCORD, MA  01742
                     *
                     AMPERV   EQU   $3F5
                     COUT     EQU   $FDED

                              ORG   $300

0300: A9 0B                   LDA   #MSGROUT
0302: 8D F6 03                STA   AMPERV+1
0305: A9 03                   LDA   #>MSGROUT
0307: 8D F7 03                STA   AMPERV+2
030A: 60                      RTS

030B: A2 00          MSGROUT  LDX   #$00
030D: BD 19 03       CHREAD   LDA   MESSAGE,X
0310: F0 18                   BEQ   DONE
0312: 20 ED FD                JSR   COUT
0315: E8                      INX
0316: 4C 0D 03                JMP   CHREAD
0319: 8D             MESSAGE  HEX   8D
031A: D3 C1 CD                ASC   "SAMPLE ROUTINE"8D00
031D: D0 CC C5 A0 D2 CF D5 D4
0325: C9 CE C5 8D 00
032A: 60             DONE     RTS
--End assembly, 43 bytes, Errors: 0
```

BRT*n* = ERT*p*+1
ERT*n* = ERT*n*a−BRT*n*+BRT*n*

where *n* represents the number of the current routine and *p* represents the number of the previous routine. These symbols are the actual addresses of the beginning and end of the routine when it is stored in auxiliary memory. All routines will execute at $2200 in main memory. Examples are shown in **lines 442-443 and 462-463**.

6. Print out an assembly listing of the routines and note each three-byte instruction. If the last byte in the instruction refers to an address within the routine, mark the line with a relocation label as shown in **lines 453, 457, 471 and 475**. For the relocation table, the address of the byte to be changed must be the address as it exists when the program is first loaded, not the address assigned by the ORG statement. This is accomplished by adding BRT*n*-BRT0 to each address. Go to Section 1h and list these relocation addresses as defined addresses (DA) in the relocation table, as shown in **lines 174-177**
7. Determine the total number of addresses in the relocation table and equate RELADR with this value **(line 39)**. RELADR cannot exceed 127, which is the effective limit for the number of routines that can be added.
8. Determine the number of bytes in the longest routine. Divide by $100 or 256. If there is a remainder, add one to the quotient. The result is the number of pages that must be reserved above the command handler for the routines. Then add another page for the command handler and equate PAGES with this value **(line 37)**.
9. Construct the command list (Section 2e). The error routine must be the first routine in the list. Note that each routine is added in a specific manner, beginning with 00 as a delimiter.
10. Select a command string for each routine. Carefully scrutinize the command string for a combination of characters that represent an Applesoft BASIC keyword. If a keyword is present in the string, its token must be substituted for the characters; for example, TOKEN would be the byte that represents TO, followed by the ASCII codes for K,E and N.
11. Count the number of characters (or keywords) in the string, and enter this command length after the delimiter **(line 297)**. The ASCII values of the characters or tokens that represent a command string are entered after the command length. For example, ROUT2 is 52 4F 55 54 32 while TOKEN is C1 4B 45 4E, since $C1 is the token value for TO. See the *Applesoft BASIC*

Programmer's Reference Manual, Vol. 2 for a list of the BASIC keywords and their ASCII tokens.

12. Follow the command string in the command list with the defined addresses of the beginning (BRT*n*) and ending (ERT*n*) address of the routine. The entire procedure is repeated for each routine, as demonstrated in Section 2e. The command list must end with 00 FF 00 **(lines 313 - 314)** followed by the end-of-list marker, ENDLIST, in **line 318.**
13. If there is any special subroutine required in the initialization, it should be appended in Section 1i.

Example 1 shows a typical & routine before it was installed in EXTRA.AMPER. The converted version of this program appears in **lines 452-463 Listing 1**. Following the 13-step process just outlined, **Example 1** was converted as follows:

1. This step was skipped, since **Example 1** is one of the dummy routines mentioned.
2. This routine was assigned number two in the sequence and inserted after the hex-dec converter. **Lines 19-27** are the main routine in **Example 1; lines 13-17** serve only to set up the & vector. The label MSGROUT was changed to BRT2a and DONE was changed to ERT2a. In addition, the reference to DONE in **line 21** was changed.
3. An equate for AMPERV is not needed and COUT is already defined **in line 34** of **Listing 1.** No new equates are needed.
4. The origin in **line 11** was changed from $300 to ROUTINE in **line 450** of Listing 1.
5. **Lines 462-463** define the symbols BRT2 and ERT2.
6. The two three-byte instructions that refer to locations within the routine are in **lines 20 and 24** of **Example** 1. The two referenced locations are CHREAD and MESSAGE, which were changed to R2a and R2b respectively. The names were also changed in the source code for **lines 20 and 24.**
7. The two relocation addresses added to the table when this routine was installed are shown in **lines 174-175** of **Listing** 1. This required an adjustment to the value of RELADR in **line 39.**
8. The hex-dec converter is considerably longer than **Example 1,** so no change had to be made the value of PAGES **(line 37).**

9– ROUT2 was selected as the command string for the new rou-
12. tine, so it was entered in the command table in **lines 296-300.** No Applesoft keywords are included in this string and the length is five **(line 297).** The actual auxiliary memory locations for the beginning and end of the program are stored immediately afterward **(lines 299-300).**

13. No special subroutines are required.
14. This routine should assemble with no conflicts.

Installation Considerations

Extra Amper **(Listing 3)** should be installed before any substantial BASIC program because it temporarily occupies memory at $2000; thus, a long BASIC program, its variables, or the Hi-Res screen may be overwritten. Rather than install EXTRA.AMPER directly, it's better to use a short BASIC loader program, such as in **Listing 3.**

HOW IT WORKS

The loader program **Listing 1** starts by checking for ProDOS. **Lines 90-200** handle memory checking under ProDOS, while **lines 210-240** handle memory checking under DOS 3.3.

Under ProDOS, the machine ID byte is read in **line 90.** A value of less than 128 indicates the machine is not a IIe, IIc, or IIGS. In addition, bits 4 and 5 must both be set or else the machine has less than 128K. In **line 130** EXTRA.AMPER is BLOADed and its length is checked in **line 140.** If it's greater than 6K, then the /RAM volume is checked and a warning message is displayed before EXTRA.AMPER is executed.

Things are more complicated under DOS 3.3. In **line 200** the program is terminated if the machine is not a IIe, IIc or IIGS. In **line 210** the IIc and IIGS are accepted without further testing.

In **line 230** the subroutine at **line 370** READs in a short machinelanguage program (a modified version of Apple's ID program in the *Extended 80-Column Text Card Supplement*) to check for auxiliary memory. This is then executed with a CALL 724. The subroutine at **line 180** is used to check if there is really an 80-column card present. The documented soft switch location $C017 is not reliable, since it cannot distinguish between the absence of a card and an extended 80-column card. The machine language routine returns a value of 64 if there is only 64K and a value of 128 if there is 128K. Extra Amper is then BLOADed if the machine has passed all the tests.

Finally, a CALL 8192 executes Extra Amper and a confirmation message is printed.

Refer to **Figure 1** for a graphic portrayal of what happens. On running **Listing 1,** Extra Amper is loaded into main memory beginning at $2000 (See **Figure 1**, arrow 1. The command handler begins at $2100 and the & routines begin at $2200. With initialization, HIMEM is lowered by enough pages of RAM for the command handler and a routine buffer that can accommodate the longest & routine (see arrow 2). The command handler is then installed above the new HIMEM and the & routines moved to auxiliary memory at $800 (arrow 3). The & vector is pointed at the command handler (arrow 4). When a routine is called, it is moved from auxiliary memory to the routine buffer, and program control is transferred to the routine (arrow 5).

In Section 1 of **Listing 1**, pages of memory are allocated for the command handler and for the longest routine. Unless an extensive number of routines is being used (over 12) and the command list is very long, Section 2 will not exceed one page of RAM. HIMEM is lowered by the allocated number of pages. If ProDOS is running, the four pages of buffer used by ProDOS above HIMEM are preserved. In Section 1b, a pointer is saved for use in Section 1g to move the command handler to its final location.

An offset is then determined (section 1c) that will be used in conjunction with the relocation subroutine in Section 1e and the relocation table in Section 1h to make each & routine relocatable. If the routines were not relocatable, then they would have to have a fixed location in memory, and could possibly interfere with other machine language programs (such as a line editor). Relocation is accomplished by adjusting the last byte of all three-byte instructions to internal references in a routine. Note that the offset is between the original location ($2200) of a routine and its final location above the command handler. Although the routines are first directed to auxiliary memory and then to the routine buffer above the command handler, this is immaterial in determining the offset.

The & vector is then pointed at the relocated command handler (Section 1d), and the initialization section concludes by moving the command handler to the space allocated by moving the command handler to the space allocated above HIMEM and the & routines into auxiliary memory (Section 1g). The move to auxiliary memory is accomplished using the AUXMOVE subroutine ($C3111), which is similar to the Monitor MOVE subroutine ($FE2C). The beginning, end and destination addresses are placed in the A-registers ($3C-$43). To move the bytes from main memory to auxiliary memory, the Carry is set prior to calling AUXMOVE. If the move is from auxiliary memory to main memory, the Carry is cleared first.

The command handler is the heart of the Extra Amper program. Each & routine is assigned to a specific command string. The command string immediately follows the &. The command handler compares the entered command string (Section 2a) with the command strings in the command list (Section 2e). If a match occurs, the command handler finds the beginning and ending addresses of the routine in auxiliary memory. It then moves the routine to the routine buffer space above the command handler in main memory using the AUXMOVE subroutine (Section 2c). Control is then passed to the selected routine (Secton 2d).

Section 3 contains all the & routines. As noted, this section along with the command list must be set up in a specific manner so that the selection & routine will be transferred to main memory and run.

LISTING 1: EXTRA.AMPER

```
               ********************************
               * EXTRA.AMPER                  *
               * by Harold Portnoy            *
               * Copyright (c) 1987           *
               * by MicroSPARC, Inc.          *          Merlin
               * Concord, MA  01742           *
               ********************************

               * NOTE: THIS PROGRAM DOES NOT CHECK TO DETERMINE IF
               * EXTENDED MEMORY IS AVAILABLE. DO NOT USE UNLESS YOUR
               * COMPUTER HAS 128K.

               *EQUATES
               *.....Zero page
               A1L      =     $3C          ;A registers
               A2L      =     $3E
               A4L      =     $42
               MEMSIZ   =     $73          ;HIMEM
               CHRGET   =     $B1          ;Get char., advance TXTPTR
               CHRGOT   =     $B7          ;Get char.
               TXTPTR   =     $B8          ;Text pointer
               BEGCH    =     $FC          ;Beginning of com. hand. pointer
               OFFSET   =     $FD          ;Temp for offset
               CMDLEN   =     $FD          ;Command string length
               RPTR     =     $FE          ;Relocation table pointer
               TXTTEMP  =     $FE          ;Temp for TXTPTR
               *.....Page 3
               AMPERV   =     $3F5         ;Ampersand vector
               *.....ProDOS
               ENTRY    =     $BF00        ;ProDOS JMP to MLI
               *.....General
               AUXMOVE  =     $C311        ;Main <=> aux move
               SYNERR   =     $DEC9        ;Syntax error msg
               COUT     =     $FDED        ;Print character
               MOVE     =     $FE2C        ;Monitor MOVE
               *.....Special
               PAGES    =     2            ;Pages above HIMEM for
                                           ; command handler and routine
               RELADR   =     17           ;Number of addresses to be
                                           ; relocated. (min. = 10)
               AUXROUT  =     $800         ;Start of S/R in aux memory
               INIT     =     $2000        ;Start of initialization
               COMHAND  =     $2100        ;Start of command handler
               ROUTINE  =     $2200        ;Start of routine buffer in
                                           ; main memory

               *EQUATES for routine 1 (hex-dec Converter)
               *.....Zero page
               LINNUM   =     $50          ;Line number register
               *..... General
               FRMNUM   =     $DD67        ;Evaluate as number
               ILLQUAN  =     $E199        ;Illegal quantity error
               GETADR   =     $E752        ;Convert to hex
                                           ; and leave in LINNUN
               LINPRT   =     $ED24        ;Print decimal from hex
               PRNTAX   =     $F941        ;Print A,X
               CROUT    =     $FD8E        ;Output CR

                        ORG   INIT

               * SECTION 1: INITIALIZATION
               *
               * Section 1a: Determine if in DOS 3.3 or ProDOS,
               * then get buffer space for command handler (CH)
               * and longest routine.

               *.. Allocate pages for CH and one routine
               *.. above HIMEM. (DOS 3.3 and ProDOS).

2000: A5 74             LDA   MEMSIZ+1     ;Determine HIMEM
2002: 38                SEC                ; and
2003: E9 02             SBC   #PAGES       ;Allocate pages for longest
                                           ; routine + 1 pg for CH
2005: 85 74             STA   MEMSIZ+1     ; reset HIMEM

               *.. Maintain the 4 pages of ProDOS buffer

2007: AE 00 BF          LDX   ENTRY        ;Is it ProDOS?
200A: E0 4C             CPX   #$4C
200C: D0 03             BNE   :1           ;No, then continue
200E: 18                CLC
200F: 69 04             ADC   #$04         ;Yes, 4 pages of buffer

               * Section 1b: Save high byte to beginning address of CH.

2011: 85 FC    :1       STA   BEGCH        ;Pointer to start of CH
2013: 48                PHA                ;Save for ampersand vector

               * Section 1c: Determine offset between high order byte of
               * initial CH location ($2100) and high order byte of
               * CH location above HIMEM.

2014: 38                SEC
2015: E9 21             SBC   #>COMHAND    ;Subtract high byte of initial
2017: 85 FD             STA   OFFSET       ; location to obtain offset

               * Section 1d: Point ampersand vector at relocated
               * CH.

2019: A2 00             LDX   #$00         ;Point & to CH
201B: 8E F6 03          STX   AMPERV+1
201E: 68                PLA
201F: 8D F7 03          STA   AMPERV+2

               * Section 1e: Relocate CH code, as necessary.

2022: A0 00             LDY   #$00
2024: BD 7A 20 RLOOP    LDA   RTBL,X       ;Point to table
2027: 85 FE             STA   RPTR
2029: E8                INX
202A: BD 7A 20          LDA   RTBL,X
202D: 85 FF             STA   RPTR+1
202F: 18                CLC                ;Add offset
2030: B1 FE             LDA   (RPTR),Y
2032: 65 FD             ADC   OFFSET
2034: 91 FE             STA   (RPTR),Y
2036: E8                INX
2037: E0 22             CPX   #RELADR*2    ;Addresses x 2 bytes/address
2039: 90 E9             BCC   RLOOP

               * Section 1f: Jump to special subroutine.
               * If no subroutine then SPECIAL should be an RTS.

203B: 20 96 20          JSR   SPECIAL      ;Omit if no special S/R

               * Section 1g: Move CH code to final location
               * and exit.

203E: A9 00             LDA   #COMHAND     ;Move CH from
2040: 85 3C             STA   A1L          ; initial location
2042: A9 21             LDA   #>COMHAND    ; to CH location
2044: 85 3D             STA   A1L+1        ; above HIMEM
2046: A9 8F             LDA   #ENDLIST
2048: 85 3E             STA   A2L
204A: A9 21             LDA   #>ENDLIST
204C: 85 3F             STA   A2L+1
204E: A9 00             LDA   #0
2050: 85 42             STA   A4L
2052: A5 FC             LDA   BEGCH
2054: 85 43             STA   A4L+1
2056: 20 2C FE          JSR   MOVE

2059: A9 00             LDA   #ROUTINE     ;Move routines from
205B: 85 3C             STA   A1L          ; initial location
205D: A9 22             LDA   #>ROUTINE    ; to aux memory
205F: 85 3D             STA   A1L+1
2061: A9 1E             LDA   #ENDROUT
2063: 85 3E             STA   A2L
2065: A9 2B             LDA   #>ENDROUT
2067: 38                SEC
2068: E9 22             SBC   #>BRT0a
206A: 69 22             ADC   #>ROUTINE
206C: 85 3F             STA   A2L+1
206E: A9 00             LDA   #0
2070: 85 42             STA   A4L
2072: A9 08             LDA   #>AUXROUT
2074: 85 43             STA   A4L+1
2076: 38                SEC
2077: 4C 11 C3          JMP   AUXMOVE

               * Section 1h: Relocation table--high byte addresses
               * of internal references.

207A: 0E 21    RTBL     DA    R1+2         ;R1 to R8 represent the
207C: 1E 21             DA    R2+2         ; addresses in the CH
207E: 2E 21             DA    R3+2
2080: 44 21             DA    R4+2
2082: 49 21             DA    R5+2
2084: 4E 21             DA    R6+2
2086: 53 21             DA    R7+2
2088: 63 21             DA    R8+2
208A: 09 22             DA    R0a+2        ;Relocation addresses after R8
208C: 15 22             DA    R0b+2        ; represent routine addresses
208E: B2 22             DA    R2a+2+BRT2-BRT0 ;Note format for
2090: BB 22             DA    R2b+2+BRT2-BRT0 ; determining relocation
2092: D2 22             DA    R3a+2+BRT3-BRT0 ; addresses
2094: DB 22             DA    R3b+2+BRT3-BRT0

               * Section 1i: Append special subroutines at end of
               * relocation table.

2096: 60       SPECIAL  RTS                ;Return--no special S/R


               *Start new page.

2097: 00 00 00          DS    COMHAND-*    ;Fill to $2100 with 00
209A: 00 00 00 00 00 00 00 00
20A2: 00 00 00 00 00 00 00 00
20AA: 00 00 00 00 00 00 00 00
20B2: 00 00 00 00 00 00 00 00
20BA: 00 00 00 00 00 00 00 00
20C2: 00 00 00 00 00 00 00 00
20CA: 00 00 00 00 00 00 00 00
20D2: 00 00 00 00 00 00 00 00
20DA: 00 00 00 00 00 00 00 00
20E2: 00 00 00 00 00 00 00 00
20EA: 00 00 00 00 00 00 00 00
20F2: 00 00 00 00 00 00 00 00
20FA: 00 00 00 00 00 00
```

LISTING 1: EXTRA.AMPER *(continued)*

```

                 *
                 * SECTION 2: COMMAND HANDLER
                 *

2100: D8                                 194
                 * Section 2a: Check for valid command.

                 * . First save text pointer position.

2101: A5 B8               LDA   TXTPTR
2103: 85 FE               STA   TXTTEMP
2105: A5 B9               LDA   TXTPTR+1
2107: 85 FF               STA   TXTTEMP+1

                 * . Get command length. End of command list denoted
                 * . by $FF.

2109: A0 00               LDY   #$00         ;Zero character counter
210B: C8         FINDCMD  INY
210C: B9 6B 21   R1       LDA   CMDLIST,Y    ;Get length from command list.
210F: 85 FD               STA   CMDLEN       ;Save command length
2111: C9 FF               CMP   #$FF         ;End of command list marker?
2113: F0 27               BEQ   NOCMD        ;Yes, go to error message

                 * . Compare input with command string.

2115: A2 00               LDX   #$00
2117: 20 B7 00            JSR   CHRGOT       ;Get input character
211A: E8         CMDLOOP  INX
211B: C8                  INY
211C: D9 6B 21   R2       CMP   CMDLIST,Y    ;Does char. match cmd. string
211F: D0 0A               BNE   CMDADV       ;No, advance to next string
2121: E4 FD               CPX   CMDLEN       ;Yes. Is length correct?
2123: F0 19               BEQ   ACTCMD       ;Yes, then command active
2125: 20 B1 00            JSR   CHRGET       ;No, get next char.
2128: 18                  CLC
2129: 90 EF               BCC   CMDLOOP      ;Always

                 * . Incorrect command string. Advance to next command
                 * . string and restore text pointer.

212B: C8         CMDADV   INY
212C: B9 6B 21   R3       LDA   CMDLIST,Y    ;Advance to next delimiter
212F: D0 FA               BNE   CMDADV
2131: A5 FE               LDA   TXTTEMP      ;Restore text pointer
2133: 85 B8               STA   TXTPTR
2135: A5 FF               LDA   TXTTEMP+1
2137: 85 B9               STA   TXTPTR+1
2139: 18                  CLC
213A: 90 CF               BCC   FINDCMD      ;Check next command string

                 * Section 2b: No command found. Send error message in
                 * first routine.

213C: A0 02      NOCMD    LDY   #$02         ;Reset counter to first routine

                 * Section 2c: Input matches command string. Find first
                 * and last addresses of routine in command list. Move
                 * routine from aux memory to routine buffer above HIMEM.
                 * Note: Text pointer advanced because most ampersand
                 * routines expect this on entry.

213E: 20 B1 00   ACTCMD   JSR   CHRGET       ;Advance text pointer
2141: C8                  INY
2142: B9 6B 21   R4       LDA   CMDLIST,Y    ;Get beginning address
2145: 85 3C               STA   A1L
2147: B9 6C 21   R5       LDA   CMDLIST+1,Y
214A: 85 3D               STA   A1L+1
214C: B9 6D 21   R6       LDA   CMDLIST+2,Y  ;Get end address
214F: 85 3E               STA   A2L
2151: B9 6E 21   R7       LDA   CMDLIST+3,Y
2154: 85 3F               STA   A2L+1
2156: A9 00               LDA   #$00         ;Destination address
2158: 85 42               STA   A4L
215A: A5 FC               LDA   BEGCH        ;Program pointer
215C: 18                  CLC                ; plus one page
215D: 69 01               ADC   #$01
215F: 85 43               STA   A4L+1
2161: 8D 6A 21   R8       STA   GOROUT+2     ;And point to routine
2164: 18                  CLC                ;Move routine A->M
2165: 20 11 C3            JSR   AUXMOVE

                 * Section 2d: Jump to beginning of routine buffer
                 * above HIMEM.

2168: 4C 00 00   GOROUT   HEX   4C0000       ;JMP to routine in buffer space

                 * Section 2e: Construct the command list.

                 * . Error routine.

216B: 00         CMDLIST  HEX   00           ;Delimiter
216C: 01 00               HEX   0100         ;Cmd length, command string
216E: 00 08               DA    BRT0         ;Begin address of error routine
2170: 51 08               DA    ERT0         ;End address of error routine

                 * . First routine. hex - dec Converter

2172: 00                  HEX   00           ;Delimiter
2173: 01                  HEX   01           ;Command length
2174: 23                  ASC   '#'          ;Command string
2175: 52 08               DA    BRT1         ;Beginning address of routine
2177: AD 08               DA    ERT1         ;End address of routine

                 * . Sample routine

2179: 00                  HEX   00
217A: 05                  HEX   05
217B: 52 4F 55            ASC   'ROUT2'
217E: 54 32
2180: AE 08               DA    BRT2
2182: CD 08               DA    ERT2

                 * . Routine with keyword in command string

2184: 00                  HEX   00
2185: 02                  HEX   02
2186: 8C 33               HEX   8C33         ;CALL3
2188: CE 08               DA    BRT3
218A: 1D 09               DA    ERT3

                 * . Final routine. This must always complete the
                 *   command list.

218C: 00                  HEX   00
218D: FF 00               HEX   FF00         ;End of command list

                 * . End of list marker.

218F: 00         ENDLIST  DS    1

2190: 00 00 00            DS    ROUTINE-*    ;Fill to $2200 with 00
2193: 00 00 00 00 00 00 00 00
219B: 00 00 00 00 00 00 00 00
21A3: 00 00 00 00 00 00 00 00
21AB: 00 00 00 00 00 00 00 00
21B3: 00 00 00 00 00 00 00 00
21BB: 00 00 00 00 00 00 00 00
21C3: 00 00 00 00 00 00 00 00
21CB: 00 00 00 00 00 00 00 00
21D3: 00 00 00 00 00 00 00 00
21DB: 00 00 00 00 00 00 00 00
21E3: 00 00 00 00 00 00 00 00
21EB: 00 00 00 00 00 00 00 00
21F3: 00 00 00 00 00 00 00 00
21FB: 00 00 00 00 00

                 * SECTION 3: ROUTINES

                 * NOTE: The origin of each routine starts at $2200, so
                 * that when eventually transferred above HIMEM, the first
                 * byte will be in the first memory location in buffer.
                 * Transfer is from auxiliary memory, so the command list
                 * must have the true address of the routine in auxiliary
                 * memory. To determine the beginning address of each
                 * routine in aux memory, be sure to use the following
                 * protocol. BRTn and ERTn are the beginning and end
                 * addresses of each routine when in auxiliary memory.
                 * n is the number of the routine.
                 * BRTna and ERTNa are the corresponding addresses
                 * in main memory on first running the program.
                 * 1. BRT0 = AUXROUT ($800)
                 * 2. BRTn = ERT(n-1) + 1
                 * 3. ERTn = ERTna-BRTna+BRTn

                 * Section 3a: First routine is for 'no routine'
                 * error message.


2200: A0 FF      BRT0a    LDY   #$FF         ;End of command list marker
2202: C4 FD               CPY   CMDLEN       ;Same as command length?
2204: F0 0C               BEQ   BADCMD       ;Yes, incorrect command string

                 * . No command string message.

2206: C8         NOCDSTR  INY                ;No, then no command string
2207: B9 1E 22   R0a      LDA   NOCMDSTR,Y   ;Print 'NO COMMAND STRING'
220A: F0 43               BEQ   ERT0a        ;Exit
220C: 20 ED FD            JSR   COUT
220F: 18                  CLC
2210: 90 F4               BCC   NOCDSTR      ;Always taken

                 * . Incorrect command string message.

2212: C8         BADCMD   INY
2213: B9 33 22   R0b      LDA   BADCMDST,Y   ;Print 'INCORRECT COMMAND
2216: F0 37               BEQ   ERT0a        ; STRING'
2218: 20 ED FD            JSR   COUT
221B: 18                  CLC
221C: 90 F4               BCC   BADCMD

                 * . Messages.

221E: 8D 87      NOCMDSTR HEX   8D87
2220: CE CF A0            ASC   "NO COMMAND STRING"
2223: C3 CF CD CD C1 CE C4 A0
222B: D3 D4 D2 C9 CE C7
2231: 8D 00               HEX   8D00
2233: 8D 87      BADCMDST HEX   8D87
2235: C9 CE C3            ASC   "INCORRECT COMMAND STRING"
2238: CF D2 D2 C5 C3 D4 A0 C3
2240: CF CD CD C1 CE C4 A0 D3
2248: D4 D2 C9 CE C7
224D: 8D 00               HEX   8D00
224F: 4C D0 03   ERT0a    JMP   $3D0         ;Last byte is ERT0a+2

                 BRT0     =     AUXROUT
                 ERT0     =     ERT0a+2-BRT0a+BRT0

                 * Section 3b: First ampersand routine. This routine
                 * does not contain any absolute internal references.
                 * Therefore it is relocatable without modification.

                          ORG   ROUTINE

                 * EX: hex -> dec or dec -> hex converter
                 * hex -> dec. Enter &#$xxxx
                 * dec -> hex. Enter &#xxxxx. x is valid hex or dec digit.

2200: 20 B7 00   BRT1a    JSR   CHRGOT       ;Beginning of routine
2203: C9 24               CMP   #'$'         ;Is it hex -> dec
```

LISTING 1: EXTRA.AMPER *(continued)*

```
2205: D0 3D              BNE  DtoH       ;No.  dec -> hex
2207: F0 00              BEQ  HtoD       ;Yes. hex -> dec

         • Convert hexadecimal to decimal.
         2209: A2 00     HtoD      LDX  #$00       ;Initialize A2L
220B: 86 3E              STX  A2L
220D: 86 3F              STX  A2L+1
220F: A0 04              LDY  #$04       ;Allow only for digits
2211: 20 B1 00  H2D      JSR  CHRGET     ;Yes. Convert H -> D
2214: F0 25              BEQ  PRT
2216: 49 30              EOR  #$30       ;Modified
2218: C9 0A              CMP  #$0A       ; monitor
221A: 90 09              BCC  DIG        ; GETNUM
221C: 69 88              ADC  #$88       ; routine
221E: C9 FA              CMP  #$FA
2220: B0 03              BCS  DIG
2222: 4C C9 DE           JMP  SYNERR
2225: A2 03     DIG      LDX  #$03
2227: 0A                 ASL
2228: 0A                 ASL
2229: 0A                 ASL
222A: 0A                 ASL
222B: 0A        NXTBIT   ASL
222C: 26 3E              ROL  A2L
222E: 26 3F              ROL  A2L+1
2230: CA                 DEX
2231: 10 F8              BPL  NXTBIT
2233: 88                 DEY             ;Trap too many digits
2234: 30 02              BMI  ILLNUM
2236: 10 D9              BPL  H2D
2238: 4C 99 E1  ILLNUM   JMP  ILLQUAN

223B: A6 3E     PRT      LDX  A2L
223D: A5 3F              LDA  A2L+1
223F: 20 24 ED           JSR  LINPRT     ;Print dec number
2242: 10 14              BPL  RTN

         • Convert decimal to hexadecimal.

2244: A0 00     DtoH     LDY  #$00
2246: 20 67 DD           JSR  FRMNUM     ;Is it a number?
2249: 20 52 E7           JSR  GETADR     ;hex into LINNUM
224C: A9 A4              LDA  #"$"       ;Print a $
224E: 20 ED FD           JSR  COUT
2251: A6 50              LDX  LINNUM     ;Load hex address
2253: A5 51              LDA  LINNUM+1   ; left in LINNUM
2255: 20 41 F9           JSR  PRNTAX     ;Print hex
2258: 20 8E FD  RTN      JSR  CROUT
225B: 60        ERT1a    RTS             ;End of CONVERT S/R

                BRT1     =    ERT0+1
                ERT1     =    ERT1a-BRT1a+BRT1

         • Section 3c: Example of a routine requiring relocation.
         •  Note that BRT2 represents true address of beginning
         •  of routine and that routine origin is reset to
         •  $2200.

                         ORG  ROUTINE

2200: A2 00     BRT2a    LDX  #$00
2202: BD 0E 22  R2a      LDA  R2MSG,X
2205: F0 18              BEQ  ERT2a
2207: 20 ED FD           JSR  COUT
220A: E8                 INX
220B: 4C 02 22  R2b      JMP  R2a
220E: 8D        R2MSG    HEX  8D
220F: D3 C1 CD           ASC  "SAMPLE ROUTINE"8D00
2212: D0 CC C5 A0 D2 CF D5 D4
221A: C9 CE C5 8D 00
221F: 60        ERT2a    RTS

                BRT2     =    ERT1+1
                ERT2     =    ERT2a-BRT2a+BRT2

         - Section 3d: Example of a routine with BASIC keyword
         - in the command string. See command list CALL3.

                         ORG  ROUTINE

2200: A2 00     BRT3a    LDX  #$00
2202: BD 0E 22  R3a      LDA  R3MSG,X
2205: F0 48              BEQ  ERT3a
2207: 20 ED FD           JSR  COUT
220A: E8                 INX
220B: 4C 02 22  R3b      JMP  R3a
220E: 8D        R3MSG    HEX  8D
220F: D3 C1 CD           ASC  "SAMPLE OF A ROUTINE"8D
2212: D0 CC C5 A0 CF C6 A0 C1
221A: A0 D2 CF D5 D4 C9 CE C5
2222: 8D
2223: D7 C9 D4           ASC  "WITH A BASIC KEYWORD"8D
2226: C8 A0 C1 A0 C2 C1 D3 C9
222E: C3 A0 CB C5 D9 D7 CF D2
2236: C4 8D
2238: C9 CE A0           ASC  "IN THE COMMAND STRING"8D00
223B: D4 C8 C5 A0 C3 CF CD CD
2243: C1 CE C4 A0 D3 D4 D2 C9
224B: CE C7 8D 00
224F: 60        ERT3a    RTS

                BRT3     =    ERT2+1
                ERT3     =    ERT3a-BRT3a+BRT3

         • Section 3e: Determine the address of the last byte

                ENDROUT  =    ERT3+1+ROUTINE  ;End of all routines address

--End assembly, 798 bytes, Errors: 0
```

END OF LISTING 1

```
                  KEY PERFECT 5.0
                       RUN ON
                    EXTRA.AMPER
 ==========================================
    CODE-5.0    ADDR# - ADDR#    CODE-4.0
    --------    -------------    --------
    357847C3     2000 - 204F       2968
    D912161E     2050 - 209F       2850
    5678BE35     20A0 - 20EF         00
    241B263A     20F0 - 213F       1E82
    93D4C9C1     2140 - 218F       25BD
    5678BE35     2190 - 21DF         00
    77B114CD     21E0 - 222F       1D31
    9782D97B     2230 - 227F       24D6
    51BFE1DD     2280 - 22CF       25DB
    6E6BC9BE     22D0 - 231D       25F3
    734614B4 = PROGRAM TOTAL =     031E
```

LISTING 2: Partial Hex Code for Extra Amper *(see instructions)*

```
2252- 20 B7 00 C9 24 D0
2258- 3D F0 00 A2 00 86 3E 86
2260- 3F A0 04 20 B1 00 F0 25
2268- 49 30 C9 0A 90 09 69 88
2270- C9 FA B0 03 4C C9 DE A2
2278- 03 0A 0A 0A 0A 0A 26 3E
2280- 26 3F CA 10 F8 88 30 02
2288- 10 D9 4C 99 E1 A6 3E A5
2290- 3F 20 24 ED 10 14 A0 00
2298- 20 67 DD 20 52 E7 A9 A4
22A0- 20 ED FD A6 50 A5 51 20
22A8- 41 F9 20 8E FD 60 A2 00
22B0- BD 0E 22 F0 18 20 ED FD
22B8- E8 4C 02 22 8D D3 C1 CD
22C0- D0 CC C5 A0 D2 CF D5 D4
22C8- C9 CE C5 8D 00 60 A2 00
22D0- BD 0E 22 F0 48 20 ED FD
22D8- E8 4C 02 22 8D D3 C1 CD
22E0- D0 CC C5 A0 CF C6 A0 C1
22E8- A0 D2 CF D5 D4 C9 CE C5
22F0- 8D D7 C9 D4 C8 A0 C1 A0
22F8= C2 C1 D3 C9 C3 A0 CB C5
2300- D9 D7 CF D2 C4 8D C9 CE
2308- A0 D4 C8 C5 A0 C3 CF CD
2310- CD C1 CE C4 A0 D3 D4 D2
2318- C9 CE C7 8D 00 60
```

END OF LISTING 2

LISTING 3: AMPER.LOADER

```
10  REM  *********************
20  REM  *    AMPER.LOADER    *
30  REM  * BY HAROLD PORTNOY  *
40  REM  * COPYRIGHT (C) 1987 *
50  REM  * BY MICROSPARC, INC *
60  REM  * CONCORD, MA  01742 *
70  REM  *********************
80 D$ =  CHR$ (4):PD =  PEEK (48896) = 76: IF
     NOT PD GOTO 210
90 MI =  PEEK (49048): REM  MACHINE ID BYTE
100  IF MI < 128 THEN  HOME : PRINT "NOT AN A
    PPLE IIE, IIC OR IIGS": END
110  IF MI - 128 < 48 THEN  HOME : PRINT "128
    K REQUIRED": END
```

LISTING 3: AMPER.LOADER *(continued)*

```
120 EF = 1: ONERR  GOTO 300
130  PRINT D$"BLOAD EXTRA.AMPER"
140  IF ( PEEK (48858) * 256 +  PEEK (48857))
      < 6 * 1024 GOTO 270
150 EF = 2: ONERR  GOTO 300
160  PRINT D$"VERIFY /RAM"
170  HOME : PRINT "PROGRAMS IN EXTRA.AMPER EX
     CEED 6K": PRINT "FILES IN /RAM MAY BE OV
     ERWRITTEN": PRINT : PRINT "ESCAPE TO QUI
     T, RETURN TO CONTINUE";: GET Z$: PRINT :
      ON Z$ <  >  CHR$ (27) GOTO 270: END
180  HOME : POKE 49153,0: POKE 49237,0: POKE
     1024,123:A =  PEEK (1024): POKE 49236,0:
      POKE 49152,0: IF A <  > 123 THEN  PRINT
     "128K REQUIRED": END
190  RETURN
200  REM  DOS 3.3 MEMORY CHECK
210  IF  PEEK (64435) <  > 6 THEN  HOME : PRINT
     "APPLE IIE, IIC OR IIGS REQUIRED": END
220  IF  PEEK (64448) = 0 OR ( PEEK (64448) =
     224 AND  PEEK (65055) <  > 96) GOTO 250:
      REM  IIC OR IIGS
230  GOSUB 370: CALL 724: GOSUB 180
240  IF  PEEK (975) <  > 128 THEN  HOME : PRINT
     "128K REQUIRED": END
250 EF = 3: ONERR  GOTO 300
260  PRINT D$"BLOAD EXTRA.AMPER"
270  CALL 8192: REM  RUN EXTRA.AMPER
280  HOME : PRINT  CHR$ (18): PRINT "EXTRA.AM
     PER INSTALLED"
290  END
300 E =  PEEK (222):EL =  PEEK (218) + 256 *
      PEEK (219): POKE 216,0: CALL  - 3288
310  IF EF = 2 AND E = 6 GOTO 270
320  IF E = 6 THEN A$ = "EXTRA.AMPER NOT ON T
     HIS DISK": GOTO 350
330  IF E = 8 THEN A$ = "I/O ERROR--CHECK DRI
     VE DOOR": GOTO 350
340 A$ = "ERROR " +  STR$ (E) + " IN LINE " +
      STR$ (EL)
350  HOME : VTAB 12: PRINT A$: VTAB 21: PRINT
     "ESCAPE TO QUIT, RETURN TO TRY AGAIN";: GET
     Z$: PRINT : IF Z$ =  CHR$ (27) THEN  END

360  ON EF GOTO 120,150,250: END
370  FOR I = 0 TO 104: READ ML: POKE 724 + I,
     ML: NEXT I: RETURN
380  DATA  8,120,173,23,192,48,48,160,42,190,
     17,3,185,0,0,150,0,153,17,3,136,208,242,
     76,1,0
390  DATA  8,160,42,185,17,3,153,0,0,136,208,
     247,104,176,7,169,128,141,207,3,208,12,1
     69,64,141,207
400  DATA  3,208,5,169,32,141,207,3,40,96,169
     ,238,141,5,192,141,3,192,141,0,8,173,0,1
     2,201,238
410  DATA  208,14,14,0,12,173,0,8,205,0,12,20
     8,3,56,176,1,24,141,4,192,141,2,192,76,2
     38,2
420  DATA  234,0
```

END OF LISTING 3

```
              KEY PERFECT 5.0
                  RUN ON
               AMPER.LOADER
==========================================
   CODE-5.0    LINE# - LINE#    CODE-4.0
   --------    -------------    --------
   68CFBAD8       10 -   100      7F4E
   2A9ACB27      110 -   200      B4DD
   4808CB85      210 -   300      8E6A
   80C7C104      310 -   400      FCA5
   6C21547E      410 -   420      2E00
   25E955B3 = PROGRAM TOTAL =     0635
```