PART XIII: IF MY MEMORY SERVES ME RIGHT

Learn how to use the soft switches and built-in routines necessary to program with auxiliary memory.

My first home computer had 4 kilobytes of memory. At the time, 4K was plenty. After all, I could write and run some nifty BASIC programs and play Microchess. Who could ask for anything more?

But as my knowledge and interest in programming grew, I started hungering for more memory. In 1981, I purchased a used 48K Apple II Plus. Forty-eight thousand bytes of memory seemed like a banquet. For dessert, I added a language card to increase the memory to 64K. At that point, I thought I had it all.

I soon realized, however, that much of that 64K is inaccessible:

- The disk operating system (DOS 3.3) uses over 10K.
- The zero page, stack and screen buffer use 2K.
- The High-Res graphics screen uses 8K.
- The 16K (language) card is inaccessible from Applesoft BASIC.

The first 12 installments of Scott Zimmerman's column, Nibbling at Assembly Language are now available as *The Beginner's Guide to Apple II Assembly Language*, a special book and disk package. See the Products Order Card in this issue for details.

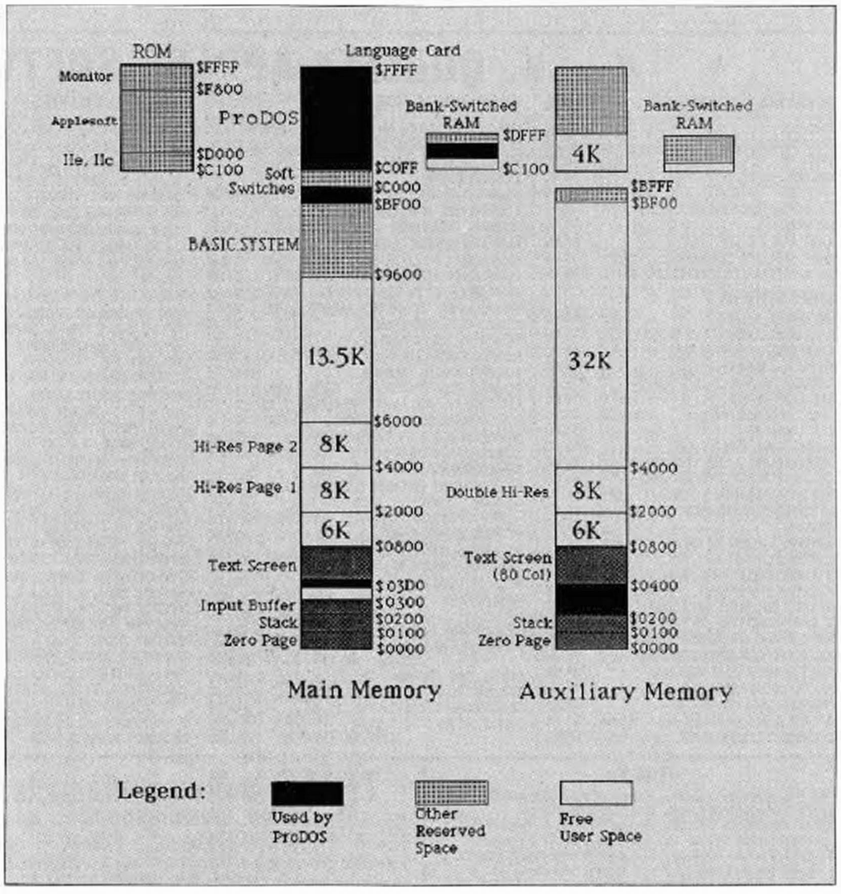

FIGURE 1: Apple 128K Memory Map

TABLE 1:
MEMZAP Keyboard Commands

Command	Fraction
Return	Toggles between main memory and auxiliary memory.
Right-Arrow	Next page (256 bytes) of memory.
Left-Arrow	Previous page (256 bytes) of memory.
Up-Arrow	Higher in memory by 4K ($1000 bytes).
Down-Arrow	Lower in memory by 4K ($1000 bytes).
I, J, K, L	Move cursor on memory page. This selects the particular byte within the memory page.
Period (.)	Increments the memory value at the cursor. *Note.* The period is on the same key as the greater-than sign (>), which suggests that the value in memory will become greater than it is.)
Comma (,)	Decrements the memory value at the cursor. (*Note:* The comma is on the same key as the less-than sign (<), which suggests that the value will decrease.)
Slash (/)	Increases the memory value at the cursor by 16 ($10), i.e., increments the high nibble.
Question mark (?)	Decreases the memory value at the cursor by 16.
Escape	Exits MEMZAP.

For Applesoft programmers, the full-meal deal of 64K is just an appetizer of 36K or, if your program uses High-Res graphics, just a small snack of 28K.

In 1984 I bought a 128K Apple IIc. By that time, of course, I knew what 128K meant for Applesoft BASIC programmers: still only 36K. For assembly language programmers, however, a 128K machine contains a full-course meal of 86K of free user space (see **Figure 1**).

If you haven't learned how to chew and swallow 86K of memory, you can easily get indigestion. This article will help you digest the extra bytes in auxiliary memory. Once you understand the concepts and have written an application or two, you'll be eating big bytes of memory with ease.

128K APPLE MEMORY MAP

The memory map in **Figure 1** shows the main and auxiliary memory available for your assembly language programs, but also reveals the difficulty in using auxiliary memory: it has the same range of addresses as main memory.

You can access (read from or write to) main memory with operations like LDA $0900 or STA ($19),Y, but how do you load a byte from $0900 in auxiliary memory or store a byte to the auxiliary memory address contained in $19? How does the 6502/65C02 know which bank of memory to access? The answer is the I/O soft switches.

These soft switches are located on page $C0 ($C000 to $C0FF) of memory (see **Figure 1**). You are already acquainted with at least three soft switches:

1. $C000 is the keyboard switch; its high bit is clear (0) if no key has been pressed, and set (1) if a key has been pressed.
2. $C010 is the keyboard strobe (accessing it clears the high bit of the keyboard switch at $C000).
3. $C030 is the speaker switch; accessing it toggles the diaphragm of Apple's built-in speaker.

If you are an experienced Applesoft BASIC programmer, you may recognize several graphics switches. For example, POKE −16304,0 (equivalent to STA $C050) switches the Apple from text mode to graphics mode, while POKE −16302,0 (STA $C052) switches from mixed graphics and text to all graphics. POKE −16299,0 (STA $C055) switches from display screen 1 to display screen 2.

Certain soft switches (see the next section) let you read and write data between main memory and auxiliary memory. Furthermore, the ROM in a 128K Apple provides two helpful memory management routines: MOVEAUX, for transferring data from main memory to auxiliary (or vice versa), and XFER, for transferring program control from a main memory routine to an auxiliary memory routine (and vice versa).

Soft Switches

The following is an explanation of the major soft switches for using auxiliary memory. I use the more descriptive soft switch labels given by Glen Bredon in his article, "Using Auxiliary Memory in the Apple IIe and IIc" (*Nibble* Vol. 6/No. 5) and, in some cases, my own labels. Apple's labels, however, are given in parentheses in the following discussion.

AUXREAD (RAMRD, on) at $C003 switches from reading main to reading auxiliary memory. After you access AUXREAD (with an STA $C003 operation), any data fetched by the 6502/65C02 from within the memory range $0200 to $BFFF comes from auxiliary memory rather than main memory. Unfortunately, this applies not only to variables, arrays and other program data, but also to machine language instruction code. In other words, accessing AUXREAD switches program control from main memory (if the program counter is in the range $0200 to $BFFF of main memory) to auxiliary memory. This means that a program running in main memory can't directly read (fetch) data from the auxiliary bank of memory. Various approaches exist for overcoming this problem, one of which you will see later.

MAINREAD (RAMRD, off) at $C002 is the opposite of AUXREAD. After an STA $C002 command, any data accessed by the microprocessor in the range $0200 to $BFFF comes from main memory. And concomitantly, the program counter proceeds to load instruction code from main memory.

READFLG (RAMRD, flag) at $C013 provides a means of determining which of these two switches is on. If MAINREAD is on (which means that AUXREAD is off), the high bit of READFLG is clear; if AUXREAD is on (and MAINREAD is off), the high bit of READFLG is set. Since your program controls the memory space directly with the soft switches, you don't need to use this flag. If you need to check the flag, however, just do an LDA READFLG followed by BPL THERE (to branch to label THERE if the flag is clear) or followed by BMI THERE (to branch to THERE if the flag is set).

AUXWRT (RAMWRT, on) at $C005 sets the switch for writing to auxiliary memory. The write switches (AUXWRT and its opposite, MAINWRT) are easier to use than the read switches, since changing the memory bank for a write does not change the memory bank from which the 6502/65C02 accesses program code. Therefore, you can easily write to auxiliary memory from main, or write to main memory from auxiliary, just by accessing the appropriate switch.

MAINWRT (RAMWRT, off) at $C004 sets the switch for writing to main memory.

WRITEFLG (RAMWRT, flag) at $C014 indicates which memory bank is selected for writing. The high bit is clear if main memory (MAINWRT) is selected and is set if auxiliary memory (AUXWRT) is selected.

STORE8ON (80STORE, on) at $C001 allows access to the memory bank of *either* text screen memory space (in the address range $0400 to $0800); see **Figure 1**), as specified by the read/write soft switches. Normally, however, STORE8 is off, and you can only access the current display page. When STORE8ON is written to (with a STA $C001 operation), the microprocessor then accesses the text screen memory space specified by the read/write soft switches.

STORE8ON (80STORE, off) at $C000 turns off the above switch, causing the

microprocessor to again access only the current display-page memory space. You may be disturbed that this switch has the same address as the KEYBD switch for accessing keyboard input. But in practice, everything works fine: use LDA $C000 to check the keyboard input, and use STA $C000 to turn off the STORE8 switch. Executing STA $C000 does not actually store data at the location, since the soft switch page is part of ROM (read only memory).

STORE8FLG (80STORE, flag) at $C018 gives the current setting of the STORE8ON and STORE8OF switches. The high bit of STORE8FLG is set when STORE8 is on and clear when STORE8 is off.

AUXZP (ALTZP, on) at $C009 turns on the zero page and stack (memory address range $0000 to $01FF) of auxiliary memory for reading and writing. The advantage of keeping this memory range separate from the range $0200 to $BFFF (selected with the other read and write switches) is that routines in main memory and in auxiliary memory can share the same zero page and stack — allowing communication between the routines and at the same time, allowing them to access big blocks of data in different memory banks.

MAINZP (ALTZP, off) at $C008 turns on the zero page and stack of main memory. **ZPFLG** (ALTZP, flag) at $C0016 gives the current setting of the AUXZP and MAINZP switches. If the high bit of ZPFLG is set, AUXZP is active; if the high bit is clear, MAINZP is active.

Memory Management Routines

In addition to the soft switches, Apple ROM contains two valuable routines for managing auxiliary memory: MOVEAUX and XFER.

MOVEAUX at $C311 moves data (including program code) from main to auxiliary or from auxiliary to main memory. To use AUXMOVE, follow these steps:

1. Put the beginning address of the block of memory you want moved in $3C,$3D (using the standard low byte, high byte order).
2. Put the end address of the block in $3E,$3F.
3. Put the destination address in $42,$43.
4. Set the Carry (C) flag (with SEC) to move data from main to auxiliary memory, or clear the Carry (with CLC) to move data from auxiliary to main memory.
5. Do a JSR AUXMOVE.

AUXMOVE can be used to move program data between memory banks and, more importantly, to move part of your assembly language program into auxiliary memory.

XFER at $C314 transfers program control (JMP) from the current bank of memory (main or auxiliary) to the other bank (auxiliary or main). To use XFER, follow these steps:

1. Store the new routine's starting address at $3ED,$3EE (using the standard low byte, high byte order).
2. Set the Carry (SEC) if the transfer is from main to auxiliary memory, or clear the Carry (CLC) if the transfer is from auxiliary to main.
3. Clear the Overflow (V) flag (with CLV) to use the zero page and stack of main memory. Set the Overflow (V) flag to use the zero page and stack of auxiliary memory. Since the 6502/65C02 lacks an SEV (Set Overflow) opcode, you must use a trick: the command BIT $FF58. The ROM location $FF58 contains the value $60 (the opcode for RTS), which equals %01100000. Since the BIT command transfers bits 6 and 7 of memory into bits 6 (the N-flag) and 7 (the V-flag) of the Processor Status Register, BIT $FF58 sets the Overflow flag, as desired.
4. Do a JMP XFER. This is the same as JMP ADDRESS (when ADDRESS is stored at $3ED,$3EE), except that program control goes to the opposite memory bank, as specified by the program, rather than to the current memory bank.

Programming With Auxiliary Memory

Now that you know something about the major soft switches and ROM routines for managing auxiliary memory, you are ready to apply your knowledge to assembly language programming. Follow these general program development steps:

1. Write and debug your routines in main memory. Even though some (or all) of your program will eventually reside in auxiliary memory, you can avoid many difficulties by first writing the subroutines for main memory. In this way, you can use standard debuggers and programming aids to help you during program development. Even the Monitor routines accessed by **CALL -151** are difficult to use with auxiliary memory. You may need to write some specialized drivers to test your subroutines before assembling them together in your final program. (A driver is a short program whose only function is to test a subroutine by initializing variables and preparing the computer system for the subroutine call, and by then calling the subroutine.)
2. Add the following to your main program:
   a. Code to relay calls between main and auxiliary memory.
   b. A routine to move program parts to their final addresses in main and auxiliary memory, using the ROM routines MOVE ($FE2C) and AUXMOVE ($C311). You may, for example, have some subroutines in main memory, some in auxiliary memory, and a relay routine to make calls from one bank of memory to the other.
   c. A routine to write data from one bank of memory to another.
   d. A routine to read data from one bank of memory to another.
3. Add lines to your assembly language source code for calculating the starting and ending addresses for MOVE and AUXMOVE.

The best way to understand all of this is to see an actual example.

MEMZAP

MEMZAP, shown in **Listing 1**, is a handy utility for examining and changing memory. Most of the program resides in auxiliary memory, freeing main memory for your Applesoft BASIC programs or other application. MEMZAP demonstrates the use of most of the soft switches and the ROM routines discussed in previous sections.

MEMZAP requires a 128K Apple IIe, IIc or IIGS.

Entering the Program

Type the assembly language source code of MEMZAP into your assembler/editor, assemble the program, and save the source and object codes to disk with the base name MEMZAP. If you are entering the program from the Monitor, when you get to **line 422** continue entering the bytes as if they continued at address $0BEB, rather than at $300. Save the program with the command:

```
BSAVE MEMZAP,A$900,L$335
```

For additional help, see the Typing Tips section.

Using the Program

To use MEMZAP, type **BRUN MEMZAP** if it's not in memory; or type **CALL 768** from BASIC or **300G** from the Monitor, if the program currently resides in memory. You'll immediately see the MEMZAP screen, with a full page of memory represented in both hex and ASCII form. The keyboard commands that control MEMZAP are shown in **Table 1**.

When you first **BRUN MEMZAP**, the main memory is active. Press Return to switch to auxiliary memory. This changes only the major block of memory, from $0200 to $BFFF, and does not change the zero page or stack.

How It Works

I won't take the time to explain all the logic in MEMZAP, since most of it would just be a review of information covered in earlier articles. I will focus instead only on aspects relating to the use of auxiliary memory, and hope you take the time to study the rest of the code.

The actual program starts at line 61 of **Listing 1**, with a jump to MOVEPGM, the routine for moving program parts into their final running locations. I could have kept the MOVEPGM routine at the beginning of the program, but for convenience chose to add it to the end.

The MOVEPGM routine is given in **lines 386-413**. The first segment moves the relay and main memory routines into page 3 ($0300). The second segment moves most of the rest of MEMZAP into auxiliary memory using the ROM routine AUXMOVE, as explained earlier. Notice that the beginning address of the block to be moved is the same as the destinaton (target) address in auxiliary memory. Although this isn't necessary, it is convenient in this program.

The segment of MEMZAP that stays in main memory is given in **lines 422-460**. Notice that even though this is part of the same listing, the ORG has been reset to $300 (see **line 420**). Some assemblers will not support multiple ORG statements; if you have a question about it, check your user manual. (The MicroSPARC Assembler 3.0 handles multiple ORGs, but miscalculates the program length after assembly.)

The page 3 code contains three short routines: **RELAY** initializes the 80-column card by doing the equivalent of PR#3 (**lines 422-423**). For some reason, the 80-column card gobbles up but does not respond to the first byte sent to it, so MEMZAP does a JSR CROUT to sent a carriage return. **Lines 425-431** follow the steps necessary (see Memory Management Routines) to use XFER for transferring control from main memory to a routine in auxiliary memory.

MAINCALL is the entry point in main memory for subroutine calls from auxiliary memory. MEMZAP has only two routines located in main memory, LOADBYT and WRMSTRT (see **lines 455-460**). Even though these two routines are short, with several long subroutines the same principles would apply:

1. Each subroutine residing in main memory and called by auxiliary memory is assigned an even number, e.g. 0, 2, 4, 6 (see **lines 54-55 of Listing 1**).
2. To call a routine (see **lines 257-258**), load the X-Register with the routine number and do a JSR CALLMAIN.
3. In main memory, each subroutine address (minus one) is a member of an address table (see ADRTBL, **lines 452-453**).
4. CALLMAIN, located in auxiliary memory (see **lines 350-358**), saves the Accumulator in the zero page (which main and auxiliary memory share), and then uses XFER to jump to MAINCALL in main memory.
5. MAINCALL immediately calls DOCALL (see **lines 433** and **444-450**), which pushes the address of the designated main memory routine onto the stack, and then jumps to the address of the routine via an RTS. This works because an RTS is just like a JMP, except that the effective destination is *one byte past the memory address on top of the stack*. That is why the address table (see ADRTBL in **lines 452-453**) contains the routine address minus one.
6. The RTS at the end of the subroutine in main memory (e.g., see **line 460**) causes a return to the address immediately after MAINCALL (see **line 434**).
7. MAINCALL uses XFER to transfer control back to auxiliary memory at the address RETURN (**line 358**), which contains a simple RTS.

In the above example, the program in auxiliary memory calls subroutines in main memory, but making calls in the opposite direction would work in an equivalent way.

LOADBYT has the simple task of loading the Accumulator with a byte of memory specified by the zero-page pointer (BYTE, PAGE), which points to the byte address of the MEMZAP cursor. LOADBYT is necessary because a routine in auxiliary memory can't directly read a byte in main memory. If you tried a simple STA MAINREAD (the soft switch to read main memory), program control would also immediately revert to main memory, and your Apple would go off into never-never land. The program must instead read memory of an opposite bank in an indirect fashion: by jumping memory banks with XFER, loading the desired byte, and jumping back to the original memory bank.

WRMSTRT is not really a subroutine, since control never returns to the caller. This is just the way MEMZAP exits through the DOS vector $3D0, which is, of course, in main memory.

MEMZAP has two valuable subroutines, PEEKMEM and POKEMEM, whose function you should understand.

PEEKMEM (lines 254-263 of Listing 1) reads a byte of memory, just like the PEEK command in Applesoft BASIC. The address of the byte it PEEKS is stored in $19,$1A (designated BYTE and PAGE, respectively, in MEMZAP). The routine starts (**line 254**) by setting the offset index to zero, but in your applicaton program, you may want to use the offset as a variable or an array index to access a block of memory. **Line 255** checks AUXFLG, which is zero when we want to access main memory, or $FF when we want to access auxiliary memory. The BIT opcode sets the Negative (N) flag of the Processor Status Register if AUXMEM is $FF (and the BMI branch in **line 256** is taken) or clears the N-flag if AUXMEM is zero (and BMI in **line 256** is not taken). If the branch is not taken, the PEEK is in main memory, and the program uses CALLMAIN to get the byte value from main memory (with the routine LOADBYT). If the branch is taken, the PEEK is in the auxiliary memory, and a simple LDA (BYTE),Y gets the desired byte value into the Accumulator.

The PEEKMEM routine executes an STA STORE80F to make sure that a read to the auxiliary text page ($0400 to $0800) accesses the desired memory bank, rather than defaulting to main memory. After getting the byte, the routine executes an STA STORE80N to make sure that any screen printing (through COUT) goes to the proper memory locations.

POKEMEM (lines 265-273) also uses AUXFLG to determine which memory bank to access. Writing to an opposite bank of memory is simple compared to reading, since the program can just use soft switches. If AUXFLG is clear (for accessing main memory), POKEMEM turns on the main memory write soft switch with STA MAINWRT (**line 267**), and then POKEs (writes) the value in the Accumulator into memory. After storing a byte, POKEMEM accesses AUXWRT to make sure auxiliary memory is reactivated. The routine also uses STORE80F and STORE80N to make sure the POKE goes to the proper memory bank of the display page.

Take a moment now to go back to the section Programming With Auxiliary Memory and see if you can follow each step through MEMZAP. Here is a summary of each step:

1. I wrote and debugged MEMZAP in main memory before using it in auxiliary memory.
2. The following line numbers contain the designated code:
   a. **Lines 350-358, 422-431, and 433-453** relay calls between main and auxiliary memory.
   b. **Lines 386-413** move program parts to their final addresses.
   c. **Lines 265-273** write data from one bank of memory to another.
   d. **Lines 254-263 and 459-460** read data from one bank of memory to another.
3. **Line 414** (RELSTRT, for relay start) gives the starting address of the routine, which eventually resides at $300 in main memory, and **line 463** (RELEND, for relay end) gives the end address. **Line 62** (MEMZAP) gives the beginning address of the data moved into auxiliary memory, and **line 380** (ENDATA) gives the end address.

PROGRAMMING TIPS

Here are a few additional tips to help you program with auxiliary memory:

1. During program development, you can use a different ORG than the final one. For example, the ORG of MEMZAP is $900, but I used $9000 during most of its development since some of my programming utilities use the lower range of memory. Then you can change it to its final location after debugging.
2. If possible, use the same zero page and stack for routines in main and in auxiliary memory. As you can see, MEMZAP keeps the same zero page and stack, which facilitates transfer of data between the two memory banks, and greatly simplifies programming.
3. Use page 3 as an interface between main

and auxiliary if possible. This way, your relay routines won't be overwritten by Applesoft BASIC, which normally starts at $0800 of main memory.

4. For simplicity, use only the auxiliary memory range from $0800 to $BFFF. Other banks of memory have free space but they're harder to manage.
5. If you want to use all regions of memory, carefully read your Apple IIe or IIc reference manual to understand the required soft switches. Although more complicated, the same basic principles apply.
6. Most of these same techniques (soft switches, move routines, and transfer routines) can also be applied to programming big memory cards, such as Applied Engineering's RamWorks. Check the user's manual provided by the manufacturer for details.
7. To get the most from MEMZAP, you may want to make some enhancements or modifications:
   a. Allow direct user input of bytes, rather than just incrementing or decrementing the nibbles.

MEMZAP is a handy utility for examining and changing memory.

   b. Give a menu of commands at the bottom of the screen.
   c. Let the user set the step size for stepping through pages of memory.
   d. Modify the choice of input keys or the layout of the screen display to suit your personal tastes.
8. To use the 80-column card in your applications, carefully look over MEMZAP **lines 422-424** (initialization of the 80-column card) and **lines 336-338** (the GOTOXY routine). Avoid moving the cursor (with GOTOXY) to column zero. Strange things happen unless you stay in the range `X >= 1 and X <=79`.
9. Using auxiliary memory exacts a price: longer program development time and slightly slower program execution. But the additonal memory available for your programs is usually worth the price.

REFERENCES

1. Bredon, G. "Using Auxiliary Memory in the Apple IIe and IIc," *Nibble*, Vol. 6/No. 5, pp. 76-89.
2. Apple Computer, Inc., *Apple II Reference Manual and Reference Manual Addendum: Monitor ROM Listings.* See the section entitled "Auxiliary Memory and Firmware." (To do serious assembly language programming on the Apple, these reference manuals are essential.)

LISTING 1: MEMZAP

```
The Assembler 3.0

Source File - MEMZAP

  0
  1                     ********************************************
  2                     *                                          *
  3                     *                 MEMZAP                   *
  4                     *                                          *
  5                     *           by S. Scott Zimmerman          *
  6                     *             Copyright (c) 1987           *
  7                     *             by MicroSPARC, Inc.          *
  8                     *             Concord, MA  01742           *
  9                     *                                          *
 10                     *          MicroSPARC Assembler 3.0        *
 11                     *                                          *
 12                     ********************************************
 13
 14                             ORG $900          ;Low in memory
 15
 16                     ********************************************
 17                     *  EQUates                                 *
 18                     ********************************************
 19
 20                     GENPTR  EQU $00           ;General pointer
 21                     VERT    EQU $02           ;Vertical tab save
 22                     HORIZ   EQU $03           ;Horizontal tab save
 23                     BYTE    EQU $19           ;Current byte of page
 24                     PAGE    EQU $1A           ;Current page of memory
 25                     ADDRS   EQU $1B           ;Current cursor address
 26                     AUXFLG  EQU $1D           ;0=Main, FF=Aux memory
 27                     VAL     EQU $1E           ;Byte value save
 28                     CH      EQU $24           ;Cursor horizontal
 29                     BEGMOVE EQU $3C           ;Monitor A1L
 30                     ENDMOVE EQU $3E           ;Monitor A2L
 31                     TARGET  EQU $42           ;Monitor A4L
 32                     ACCUM   EQU $FE           ;Save accumulator
 33                     XREG    EQU $FF           ;Save X register
 34                     XFERADR EQU $3ED          ;Memory transfer adr
 35                     KEYBD   EQU $C000         ;Keyboard input char
 36                     STOREBOF EQU $C000        ;Enable 80c mem access
 37                     STOREBON EQU $C001        ;Disable 80c mem access
 38                     MAINWRT EQU $C004         ;SSwitch to write main
 39                     AUXWRT  EQU $C005         ;SSwitch to write aux
 40                     STROBE  EQU $C010         ;Clear keyboard strobe
 41                     AUXMOVE EQU $C311         ;Move data main->aux
 42                     XFER    EQU $C314         ;Routine transfer
 43                     SETNORM EQU $FE73         ;Set normal video
 44                     SETINV  EQU $FE77         ;Set inverse video
 45                     PRNTAX  EQU $F941         ;Print (A,X)
 46                     TABV    EQU $FB5B         ;Cursor vertical tab
 47                     HOME    EQU $FC58         ;Clear screen
 48                     CROUT   EQU $FDBE         ;Carriage return output
 49                     PRBYTE  EQU $FDDA         ;Print byte in hex
 50                     COUT    EQU $FDED         ;Character output
 51                     MOVE    EQU $FE2C         ;Monitor move routine
 52                     OUTPORT EQU $FE95         ;Output port (PR#)
 53                     ********************************************
 54                     NLOADBYT EQU 0            ;Main memory rtn numbers
 55                     NWRMSTRT EQU 2
 56
 57                     ********************************************
 58                     *  MEMZAP Program (moved into auxiliary memory)  *
 59                     ********************************************
 60
 61 0900  4C B1 0B              JMP MOVEPGM       Go move program parts
 62 0903  20 D1 09  MEMZAP      JSR INIT          Initialize system
 63 0906  AD 00 C0  KEYLOOP     LDA KEYBD         Key pressed?
 64 0909  10 FB                 BPL KEYLOOP       No, loop more
 65 090B  2C 10 C0              BIT STROBE        Clear keyboard strobe
 66 090E  C9 E0                 CMP #$E0          ;Is it lower case?
 67 0910  90 02                 BCC OKAY          ;No, okay as is
 68 0912  29 DF                 AND #%11011111    ;Convert to upper case
 69 0914  C9 9B         OKAY    CMP #$9B          ;ESC (Quit)?
 70 0916  D0 05                 BNE MM00          ;No, check next
 71 0918  A2 02                 LDX #NWRMSTRT     ;Call BASIC warmstart
 72 091A  4C 3B 08              JMP CALLMAIN      ; thru main memory
 73 091D  C9 95         MM00    CMP #$95          ;Right arrow (CTRL-U)?
 74 091F  D0 08                 BNE MM01          No, check next
 75 0921  A9 01                 LDA #1            Yes, increment one
 76 0923  20 99 09      CHNGP   JSR CHNGPAGE      Go change page
 77 0926  4C 06 09              JMP KEYLOOP
 78 0929  C9 88         MM01    CMP #$88          Left arrow (CTRL-H)?
 79 092B  D0 04                 BNE MM02          No, check next
 80 092D  A9 FF                 LDA #-1           Yes, decrement page
 81 092F  30 F2                 BMI CHNGP
 82 0931  C9 8A         MM02    CMP #$8A          ;Down arrow (CTRL-J)
 83 0933  D0 04                 BNE MM03          ;No, check next
 84 0935  A9 F0                 LDA #-16          ;Yes, go to prev 4K
 85 0937  30 EA                 BMI CHNGP
 86 0939  C9 8B         MM03    CMP #$8B          ;Up arrow (CTRL-K)?
 87 093B  D0 04                 BNE MM04          ;No, check next
 88 093D  A9 10                 LDA #16           ;Yes, go to next 4K
 89 093F  D0 E2                 BNE CHNGP
 90 0941  C9 CB         MM04    CMP #"K           ;K pressed?
 91 0943  D0 08                 BNE MM05          ;No, proceed
 92 0945  A9 01                 LDA #1            ;Yes, increment byte
 93 0947  20 B2 09      CHNGB   JSR CHNGBYTE      ;Go to change byte
 94 094A  4C 06 09              JMP KEYLOOP
 95 094D  C9 CA         MM05    CMP #"J           ;J pressed?
 96 094F  D0 04                 BNE MM06          ;No, proceed
 97 0951  A9 FF                 LDA #-1           ;Yes, go to prev byte
 98 0953  30 F2                 BMI CHNGB
 99 0955  C9 CD         MM06    CMP #"M           M pressed?
100 0957  D0 04                 BNE MM07          No, proceed
101 0959  A9 10                 LDA #16           ;Yes, next paragraph
102 095B  D0 EA                 BNE CHNGB
103 095D  C9 C9         MM07    CMP #"I           ;I pressed?
104 095F  D0 04                 BNE MM08          ;No, proceed
105 0961  A9 F0                 LDA #-16          ;Yes, prev paragraph
106 0963  30 E2                 BMI CHNGB
107 0965  C9 AE         MM08    CMP #"            ;Period pressed?
108 0967  D0 08                 BNE MM09          ;No, check next
109 0969  A9 01                 LDA #1            Yes, increment val
110 096B  20 C2 09      CHNGV   JSR CHNGVAL       ;Go change value
111 096E  4C 06 09              JMP KEYLOOP
112 0971  C9 AC         MM09    CMP #","          ;Comma pressed?
113 0973  D0 04                 BNE MM10          ;No, check next
114 0975  A9 FF                 LDA #-1           ;Yes, decrement
115 0977  D0 F2                 BNE CHNGV
116 0979  C9 AF         MM10    CMP #"/           ;Slash pressed?
117 097B  D0 04                 BNE MM11          No, check next
118 097D  A9 10                 LDA #16           ;Yes, incr by $10
119 097F  D0 EA                 BNE CHNGV
120 0981  C9 BF         MM11    CMP #"?           ;Question pressed?
121 0983  D0 04                 BNE MM12          No, check next
122 0985  A9 F0                 LDA #-16          ;Yes, decr by $10
123 0987  D0 E2                 BNE CHNGV
124 0989  C9 8D         MM12    CMP #$8D          ;RETURN pressed?
125 098B  D0 09                 BNE NOKEY         ;No, no key selected
126 098D  A5 1D                 LDA AUXFLG        ;Get current flag val
127 098F  49 FF                 EOR #$FF          ;Toggle the flag
128 0991  85 1D                 STA AUXFLG        ;Save new value
129 0993  20 E3 09              JSR DOPAGE2       ;Go do new page
130 0996  4C 06 09      NOKEY   JMP KEYLOOP       Get next key
131
132                     ********************************************
133                     *  MEMZAP subroutines (also in auxiliary mem)  *
134                     ********************************************
135
136 0999  85 1E         CHNGPAGE STA VAL          ;Save change value
137 099B  18                    CLC               ;Prepare to add
138 099C  65 1A                 ADC PAGE          ;Add value to page
139 099E  AA                    TAX               ;Put here for test
140 099F  E0 C0                 CPX #$C0          ;Softswitch page?
141 09A1  D0 08                 BNE CP            ;No, change page
142 09A3  24 1E                 BIT VAL           ;Is value negative?
143 09A5  30 03                 BMI LOW           ;Yes, make page low
144 09A7  E8                    INX               ;No, make page high
145 09A8  D0 01                 BNE CP            Always
146 09AA  CA            LOW     DEX               Decrement page
147 09AB  86 1A         CP      STX PAGE          Save new page
148 09AD  86 1C                 STX ADDRS+1       Save new address
149 09AF  4C E0 09              JMP DOPAGE        Go handle new page
150                     ********************************************
151 09B2  48            CHNGBYTE PHA              ;Save change value
152 09B3  20 AF 0A              JSR PCURS         ;Clear cursor
153 09B6  68                    PLA               ;Restore change value
154 09B7  18                    CLC               ;Prepare to add
155 09B8  65 1B                 ADC ADDRS         ;Add value and address
156 09BA  85 1B                 STA ADDRS         ;Save new value
157 09BC  20 F6 0A              JSR PRNTADR       ;Print new address
158 09BF  4C AC 0A              JMP PRNTCURS      ;Print new cursor
159                     ********************************************
160 09C2  85 1E         CHNGVAL  STA VAL          ;Save add value
161 09C4  20 82 0A              JSR PEEKMEM       Get current byte
162 09C7  18                    CLC               ;Prepare to add
163 09C8  65 1E                 ADC VAL           ;Add change value
164 09CA  20 97 0A              JSR POKEMEM       ;Poke new byte value
165 09CD  20 AC 0A              JSR PRNTCURS      ;Print cursor there
166 09D0  4C 49 0A              JMP PRASCII       Print ASCII value
167                     ********************************************
168 09D3  A9 00         INIT    LDA #0            ;Init vars to zero
169 09D5  85 1A                 STA PAGE          ;Zero the memory page
170 09D7  85 1B                 STA ADDRS         ;Zero cursor address
171 09D9  85 1C                 STA ADDRS+1
172 09DB  85 1D                 STA AUXFLG        ;0 = Main memory
173 09DD  20 ED 09              JSR PRNTITLE      ;Print title
174 09E0  20 10 0A      DOPAGE  JSR PRPGADR       ;Print page addresses
175 09E3  20 BC 0A      DOPAGE2 JSR PRSTATUS      Print status
176 09E6  20 36 0A              JSR PRNTPAGE      Print memory page
177 09E9  20 AC 0A              JSR PRNTCURS      Print memory cursor
178 09EC  60                    RTS               From INIT
179                     ********************************************
180 09ED  A2 24         PRNTITLE LDX #36          Set horizontal
181 09EF  A0 00                 LDY #0            Set vertical
182 09F1  20 24 0B              JSR GOTOXY        Move cursor there
183 09F4  20 77 F2              JSR SETINV        ;Set inverse video
184 09F7  A2 4D                 LDX #TITLEMSG     ;Print title message
185 09F9  A0 0B                 LDY #TITLEMSG/
186 09FB  20 2A 0B              JSR MESSAGE
187 09FE  20 73 FE              JSR SETNORM       ;Set normal mode
188 0A01  A2 1E                 LDX #30           ;Set horizontal
189 0A03  A0 02                 LDY #2            ;Set vertical
190 0A05  20 24 0B              JSR GOTOXY        Move cursor there
191 0A08  A2 56                 LDX #BYLINE
192 0A0A  A0 0B                 LDY #BYLINE/
193 0A0C  20 2A 0B              JSR MESSAGE       Print by-line
194 0A0F  60                    RTS
195                     ********************************************
196 0A10  A5 1A         PRPGADR  LDA PAGE         ;Get address
197 0A12  85 01                 STA GENPTR+1      ;Set HOB
198 0A14  A9 00                 LDA #0            ;Start page boundary
199 0A16  85 00                 STA GENPTR        ;Set LOB
200 0A18  A0 04                 LDY #4            ;Start vertical pos
201 0A1A  A2 01         ADRLOOP  LDX #1           ;Set horiz value
202 0A1C  20 24 0B              JSR GOTOXY        ;Move cursor there
203 0A1F  C8                    INY               ;Ready for next
204 0A20  A5 01                 LDA GENPTR+1      ;Get HOB of address
```

LISTING 1: MAMZAP *(continued)*

```
0A22  A6 00     GENPTR  LDX GENPTR      ;Get LOB of address
0A24  20 41 F9          JSR PRNTAX      ;Print A,X in hex
0A27  A9 BA             LDA #":         ;Get a colon
0A29  20 ED FD          JSR COUT
0A2C  18                CLC             ;Prepare to add
0A2D  A5 00             LDA GENPTR      ;Add 16 ($10) to adr
0A2F  69 10             ADC #$10
0A31  85 00             STA GENPTR      ;Save result
0A33  D0 E5             BNE ADRLOOP     ;No, proceed
0A35  60                RTS             ;End PRNTADR
                        *---------------------------------------
0A36  A0 00     PRNTPAGE LDY #0         ;Init index
0A38  84 19     PAGELOOP STY BYTE       ;Save as byte
0A3A  20 04 0B          JSR GETBYTAB    ;Get the byte & tab
0A3D  20 DA FD          JSR PRBYTE      ;Print a byte in hex
0A40  20 49 0A          JSR PRASCII     ;Go print ASCII char
0A43  A4 19             LDY BYTE        ;Restore byte
0A45  C8                INY             ;More bytes on page?
0A46  D0 F0             BNE PAGELOOP    ;Yes, go to next
0A48  60                RTS             ;No, end of page
                        *---------------------------------------
0A49  A4 02     PRASCII LDY VERT        ;Get vertical pos
0A4B  A6 03             LDX HORIZ       ;Get horiz index
0A4D  BD A3 0B          LDA HASC,X      ;Get horiz value
0A50  AA                TAX             ;Move for GOTOXY
0A51  20 24 0B          JSR GOTOXY      ;Move cursor there
0A54  A5 1E             LDA VAL         ;Restore byte value
0A56  C9 40             CMP #$40        ;In low range?
0A58  B0 0C             BCS CHKMID      ;No, go check mid
0A5A  20 77 F2          JSR SETINV      ;Set inverse video
0A5D  18                CLC             ;Prepare to add
0A5E  A5 1E             LDA VAL         ;Restore value
0A60  69 A0             ADC #$A0        ;Move to normal range
0A62  49 60             EOR #%01100000  ;Adjust bits
0A64  D0 16             BNE ASCOUT      ;Print it
0A66  C9 80     CHKMID  CMP #$80        ;Midrange?
0A68  B0 0C             BCS CHKCTRL     ;No, go check ctrl
0A6A  20 77 F2          JSR SETINV      ;Set inverse video
0A6D  A5 1E             LDA VAL         ;Restore value
0A6F  18                CLC             ;Prepare to add
0A70  69 60             ADC #$A0-$40    ;Adjust to print range
0A72  49 60             EOR #%01100000  ;Adjust bits
0A74  D0 06             BNE ASCOUT      ;Print it
0A76  C9 A0     CHKCTRL CMP #$A0        ;Ctrl char (< space)?
0A78  B0 02             BCS ASCOUT      ;No, go output it
0A7A  A9 AE             LDA #"          ;Make it a period
0A7C  20 ED FD  ASCOUT  JSR COUT
0A7F  20 73 F2          JSR SETNORM     ;Set normal video
                        *---------------------------------------
0A82  A0 00     PEEKMEM LDY #0          ;Set offset to zero
0A84  24 1D             BIT AUXFLG      ;Which memory?
0A86  30 06             BMI AUXPEEK     ;Auxiliary
0A88  A2 00             LDX #NLOADBYT   ;Call LOADBYT
0A8A  20 3B 0B          JSR CALLMAIN    ; thru main memory
0A8D  60                RTS
0A8E  8D 00 C0  AUXPEEK STA STORE80F    ;Turn off display select
0A91  B1 19             LDA (BYTE),Y    ;Get memory
0A93  8D 01 C0          STA STORE80N    ;Turn on display select
0A96  60                RTS             ;End of PEEKMEM
                        *---------------------------------------
0A97  24 1D     POKEMEM BIT AUXFLG      ;Main or auxiliary?
0A99  30 03             BMI AUXPOKE     ;Auxiliary
0A9B  8D 04 C0          STA MAINWRT     ;Set for main write
0A9E  A0 00     AUXPOKE LDY #0          ;Zero offset
0AA0  8D 00 C0          STA STORE80F    ;Turn off display select
0AA3  91 19             STA (BYTE),Y    ;Save result
0AA5  8D 05 C0          STA AUXWRT      ;Set for auxiliary
0AA8  8D 01 C0          STA STORE80N    ;Select display
0AAB  60                RTS             ;End of POKEMEM
                        *---------------------------------------
0AAC  20 77 F2  PRNTCURS JSR SETINV     ;Set inverse video
0AAF  A5 1B     PCURS   LDA ADCRS       ;Set BYTE
0AB1  85 19             STA BYTE
0AB3  20 04 0B          JSR GETBYTAB    ;Get current byte, tab
0AB6  20 DA FD          JSR PRBYTE      ;Print a byte in hex
0AB9  20 73 F2          JSR SETNORM     ;Set normal video
                        *---------------------------------------
0ABC  A2 01     PRSTATUS LDX #1         ;Set cursor position
0ABE  A0 16             LDY #22
0AC0  20 24 0B          JSR GOTOXY
0AC3  A2 6C             LDX #STATMSG    ;Print message
0AC5  A0 0B             LDY #STATMSG/
0AC7  20 2A 0B          JSR MESSAGE
0ACA  20 DE 0A          JSR PRMEMST     ;Print memory status
0ACD  A2 28             LDX #40         ;Set cursor
0ACF  A0 16             LDY #22
0AD1  20 24 0B          JSR GOTOXY
0AD4  A2 88             LDX #ADRMSG     ;Print message
0AD6  A0 0B             LDY #ADRMSG/
0AD8  20 2A 0B          JSR MESSAGE
0ADB  4C F6 0A          JMP PRNTADR     ;Print current address
                        *---------------------------------------
0ADE  A2 09     PRMEMST LDX #9          ;Set cursor
0AE0  A0 16             LDY #22
0AE2  20 24 0B          JSR GOTOXY
0AE5  24 1D             BIT AUXFLG      ;Is it main or aux?
0AE7  30 06             BMI PM          ;Auxiliary
0AE9  A2 7E             LDX #MAINMSG    ;Main
0AEB  A0 0B             LDY #MAINMSG/
0AED  D0 04             BNE PMM         ;Always
0AEF  A2 74     PM      LDX #AUXMSG
0AF1  A0 0B             LDY #AUXMSG/
0AF3  4C 2A 0B  PMM     JMP MESSAGE
                        *---------------------------------------

0AF6  A2 32     PRNTADR LDX #50         ;Set cursor
0AF8  A0 16             LDY #22
0AFA  20 24 0B          JSR GOTOXY      ;Move cursor there
0AFD  A6 1B             LDX ADDRS       ;Get current address
0AFF  A5 1C             LDA ADDRS+1
0B01  20 41 F9          JSR PRNTAX      ;Print A,X in hex
                        *---------------------------------------
0B04  A5 19     GETBYTAB LDA BYTE       ;Get current byte
0B06  4A                LSR             ;Divide by 16
0B07  4A                LSR
0B08  4A                LSR
0B09  4A                LSR
0B0A  18                CLC             ;Prepare to add
0B0B  69 04             ADC #4          ;Start at top
0B0D  A8                TAY             ;Save as vertical
0B0E  84 02             STY VERT        ;Save result
0B10  A5 19             LDA BYTE        ;Restore byte
0B12  29 0F             AND #%00001111  ;Toss out HOB
0B14  AA                TAX             ;Make it an index
0B15  86 03             STX HORIZ       ;Save result
0B17  BD 93 0B          LDA HTAB,X      ;Get horiz pos
0B1A  AA                TAX             ;Move for GOTOXY
0B1B  20 24 0B          JSR GOTOXY      ;Move cursor there
0B1E  20 82 0A          JSR PEEKMEM     ;Get memory
0B21  85 1E             STA VAL         ;Save value
0B23  60                RTS             ;Exit from GETBYTAB
                        *---------------------------------------
0B24  86 24     GOTOXY  STX CH          ;Save horizontal tab
0B26  98                TYA             ;Set vertical tab
0B27  4C 5B FB          JMP TABV        ;Go move cursor
                        *---------------------------------------
0B2A  86 00     MESSAGE STX GENPTR      ;Set message pointer
0B2C  84 01             STY GENPTR+1    ;Do HOB
0B2E  A0 00             LDY #0          ;Init the index
0B30  B1 00     MSGLOOP LDA (GENPTR),Y  ;Get character
0B32  F0 06             BEQ MSGEND      ;Quit if zero
0B34  20 ED FD          JSR COUT
0B37  C8                INY             ;Go to next char
0B38  D0 F6             BNE MSGLOOP     ;Branch always
0B3A  60        MSGEND  RTS             ;End of MESSAGGE
                        *---------------------------------------
0B3B  85 FE     CALLMAIN STA ACCUM      ;Save accumulator
0B3D  18                CLC             ;Set for main memory
0B3E  B8                CLV             ;Clear V for main memory
0B3F  A9 17             LDA #MAINCALL   ;Set address in main mem
0B41  8D ED 03          STA XFERADR
0B44  A9 03             LDA #MAINCALL/
0B46  8D EE 03          STA XFERADR+1
0B49  4C 14 C3          JMP XFER        ;Go to rtn in main mem
0B4C  60        RETURN  RTS             ;Return to caller

                        ****************************************
                        *  Data:                               *
                        ****************************************

0B4D  A0 CD C5  TITLEMSG ASC " MEMZAP "
      CD DA C1 D0 A0
0B55  00                BRK
0B56  E2 F9 A0  BYLINE  ASC "by S. Scott Zimmerman"
      D3 AE A0 D3 E3 EF
      F4 F4 A0 DA E9 ED
      ED E5 F2 ED E1 EE
0B6B  00                BRK
0B6C  CD E5 ED  STATMSG ASC "Memory:"
      EF F2 F9 BA
0B73  00                BRK
0B74  C1 F5 F8  AUXMSG  ASC "Auxiliary"
      E9 EC E9 E1 F2 F9
0B7D  00                BRK
0B7E  CD E1 E9  MAINMSG ASC "Main      "
      EE A0 A0 A0 A0 A0
0B87  00                BRK
0B88  C1 E4 E4  ADRMSG  ASC "Address: $"
      F2 E5 F3 F3 BA A0
      A4
0B92  00                BRK
0B93  07 0A 0D  HTAB    DFC 7,10,13,16,19,22,25,28
      10 13 16 19 1C
0B9B  20 23 26          DFC 32,35,38,41,44,47,50,53
      29 2C 2F 32 35
0BA3  39 3A 3B  HASC    DFC 57,58,59,60,61,62,63,64
      3C 3D 3E 3F 40
0BAB  42 43 44          DFC 66,67,68,69,70,71,72,73
      45 46 47 48 49
              ENDDAT    EQU *-1

                        ****************************************
                        *  Routines to ove program parts:      *
                        ****************************************

0BB3  A9 00     MOVEPGM LDA #RELAY      ;Move short main
0BB5  85 42             STA TARGET      ;Save target
0BB7  A9 03             LDA #RELAY/
0BB9  85 43             STA TARGET+1
0BBB  A9 EB             LDA #RELSTRT    ;Set starting
0BBD  85 3C             STA BEGMOVE
0BBF  A9 0B             LDA #RELSTRT/
0BC1  85 3D             STA BEGMOVE+1
0BC3  A9 35             LDA #RELEND     ;Set ending
0BC5  85 3E             STA ENDMOVE
0BC7  A9 0C             LDA #RELEND/
0BC9  85 3F             STA ENDMOVE+1
0BCB  A0 00             LDY #0          ;Prepare for MOVE
0BCD  20 2C FE          JSR MOVE        ;Move it
                        *---------------------------------------
0BD0  A9 03             LDA #MEMZAP     ;Move pgm to auxiliary
0BD2  85 3C             STA BEGMOVE
0BD4  85 42             STA TARGET      ;Make target same adrs
0BD6  A9 09             LDA #MEMZAP/
```

LISTING 1: MEMZAP *(continued)*

```
0BD8  85 3D        STA BEGMOVE+1
0BDA  85 43        STA TARGET+1
0BDC  A9 B2        LDA #ENDDAT
0BDE  85 3E        STA ENDMOVE
0BE0  A9 0B        LDA #ENDDAT/
0BE2  85 3F        STA ENDMOVE+1
0BE4  38           SEC                 ;Set for MAIN->AUX
0BE5  20 11 C3     JSR AUXMOVE
0BE8  4C 00 03     JMP RELAY           ;Start program
                   RELSTRT  EQU *

                   ;-----------------------------------------
                   ; Routines for main memory (at $300):
                   ;-----------------------------------------

                            ORG $300           ;Put in user memory

0300  A9 03        RELAY    LDA #3             ;Set to 80-column card
0302  20 95 FE             JSR OUTPORT        ;Do PR#3
0305  20 8E FD             JSR CROUT          ;To init 80-col card
0308  38                   SEC                ;Set for auxiliary
0309  B8                   CLV                ;Clear for main ZP
030A  A9 03                LDA #MEMZAP        ;Store address of rtn
030C  8D ED 03             STA XFERADR
030F  A9 09                LDA #MEMZAP/
0311  8D EE 03             STA XFERADR+1
0314  4C 14 C3             JMP XFER           ;Go to the routine
                   ;-----------------------------------------
0317  20 2D 03     MAINCALL JSR DOCALL         ;Do the call
031A  38                    SEC                ;Set for auxiliary
031B  B8                    CLV                ;Use main zero page
031C  85 FE                 STA ACCUM          ;Save accumulator
031E  A9 4C                 LDA #RETURN        ;Set for return to aux
0320  8D ED 03              STA XFERADR
0323  A9 0B                 LDA #RETURN/
0325  8D EE 03              STA XFERADR+1
0328  A5 FE                 LDA ACCUM          ;Restore accumulator
032A  4C 14 C3              JMP XFER           ;Go to auxiliary memory
                   ;-----------------------------------------
032D  BD 3B 03     DOCALL   LDA ADRTBL+1,X     ;Get address of routine
0330  48                    PHA                ;Push on stack for call
0331  BD 3A 03              LDA ADRTBL,X       ; through RTS
0334  48                    PHA
0335  A5 FE                 LDA ACCUM          ;Restore accumulator
0337  A6 FF                 LDX XREG           ;Restore X register
0339  60                    RTS                ;Jump to routine
                   ;-----------------------------------------
033A  46 03        ADRTBL   DFC LOADBYT-1,LOADBYT-1/
033C  3D 03                 DFC WRMSTRT-1,WRMSTRT-1/

                   ;-----------------------------------------
033E  20 58 FC     WRMSTRT  JSR HOME           ;Clear screen
0341  8D 01 C0             STA STORE80N       ;Turn on display select
0344  4C D0 03             JMP $3D0           ;Go to BASIC warmstart
                   ;-----------------------------------------
0347  B1 19        LOADBYT  LDA (BYTE),Y      ;Get byte from main
0349  60                    RTS
                   ;-----------------------------------------
                   ENDMAIN  EQU *
                   RELEND   EQU RELSTRT+ENDMAIN-RELAY

000 Errors
```

END OF LISTING 1

KEY PERFECT 5.0
RUN ON
MEMZAP

CODE-5.0	ADDR# - ADDR#	CODE-4.0
4267DB92	0900 - 094F	2919
9F5F4B56	0950 - 099F	255E
D70278F5	09A0 - 09EF	288F
4596A380	09F0 - 0A3F	2543
D456C58B	0A40 - 0A8F	2523
FFAA2DF9	0A90 - 0ADF	270C
144710D7	0AE0 - 0B2F	276F
16E97DB8	0B30 - 0B7F	2A06
1F895142	0B80 - 0BCF	264D
E7377833	0BD0 - 0C1F	2741
A369A993	0C20 - 0C34	0C20
DA9EC48B =	PROGRAM TOTAL =	0335