# 6800 ROM MONITOR
# VERSON 1.0

# USERS GUIDE

Southwest Technical Products Corporation
219 W. Rhapsody San Antonio, Texas 78116

Scanned and edited by Michael Holley   Sept 17, 2000   holleymj@aol.com

## SWTPC SWTBUG® (SWATBUG) MONITOR ROM

One of the features of the SWTPC 6800 Computer System is that the conventional programmer’s console has been replaced with a monitor ROM. The programmer’s console consists of all the pretty switches and lights often found on similar microcomputers that are used to bootstrap the system after power up. The programmer’s console not only raises the cost of the system, but more often than not is confusing and tedious to use for both be-ginning and experienced programmers. The monitor ROM on the other hand is a permanently stored program that gives the computer the intelligence required to communicate with the operator thru an interfaced terminal system immediately after power up without flipping switches for 10 minutes. This technique makes the computer do the work of simplifying communication between itself and the operator.

SWTBUG® is the name of the monitor program used in the SWTPC 6800 Computer System. It might be thought of as kind of a mini-operating system since it gives the operator command control over the computer system.

Features of the SWTBUG® ROM include:

* Memory Examine and Change
* Program loading from cassette or paper tape thru the control interface or thru I/O port # 0.
* Program saving to cassette or paper tape
* Go to user program
* Display contents of registers
* Erase SWTPC CT-1 024 terminal system screen
* SWTPC MF-68 floppy disk boot
* Byte search
* Breakpoint debugging
* Vectored hardware and software interrupts to user defined addresses

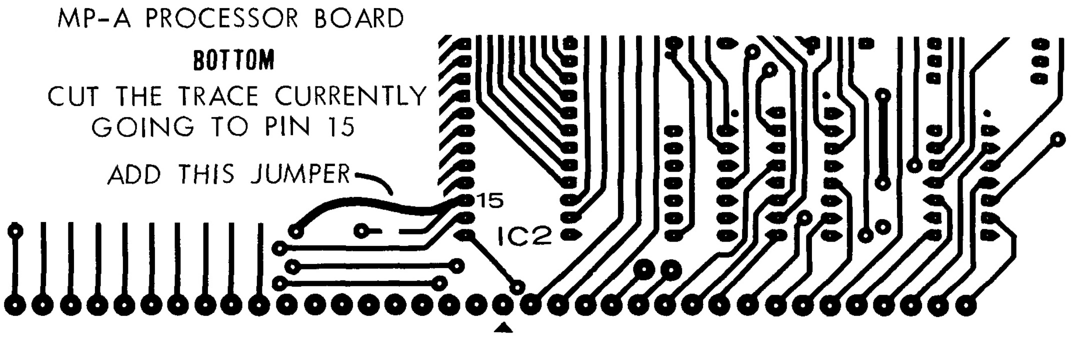
SWTBUG® is a permanently stored program and cannot be erased or lost by either a loss of power or user program error. It is always resident in the computer while power is ON and need never be loaded into the machine. Subroutines within the ROM are documented and available to the user to simplify programming and conserve on the use of user RAM memory. Character input and output, string output and return to monitor are just a few subroutines available to the user.

SWTBUG® is a 1K byte program and is addressed high in memory, far above the amount of RAM memory required for most user programs. Since SWTBUG® does require a small amount of RAM memory for operation, a 128 byte scratchpad RAM has been implemented on the processor board so that no external user RAM memory is required for monitor operation. There is even enough extra room in this RAM so that short programs such as memory diagnostics can be loaded into and run from the scratchpad RAM without requiring any external user RAM memory. Extra care however must be exercised to avoid overstoring any memory locations required for proper monitor operation. Complete de-tails on this are given later in this writeup. The SWTBUG® ROM is located from memory addresses EOOO thru E3FF. The scratchpad RAM is located from memory addresses A000 thru A07F. Both components are physically located on the processor board and are functional any time the system is powered up. Whenever computer control is transferred from SWTBUG® to the user program, there are only four ways to get back into the SWTBUG® command mode. The first is to put a jump to the CONTRL entry point address within SWTBUG® as the last step of your program. The second is to incorporate a command or action within your program which transfers program control to the CONTRL entry point address within SWTBUG’ The third is to depress the front panel RESET button. The fourth is turn the computer OFF and then back ON again. The fourth method is rather drastic and wipes out all RAM memory data, it is only mentioned to let you know that the computer always powers up with the SWTBUG® monitor in the command mode.

**SWTBUG® INSTALLATION**

SWTBUG® is a MOS device and MOS integrated circuits are susceptible to damage by static electricity. Although some degree of protection is provided internally within the integrated circuits, their cost demands the utmost in care. Before opening and/or installing SWTBUG® you should ground your body and all metallic tools coming into contact with the leads, thru a 1M ohm ¼ watt resistor. The ground must be an "earth" ground such as a water pipe, and not the circuit board ground. As for connection to your body, attach a clip lead to your watch or metal ID bracelet. Make absolutely sure that you have the resistor connected between you and the "earth" ground, otherwise you will be creating a dangerous shock hazard. Avoid touching the leads of the integrated circuits any more than necessary when installing it, even if you are grounded. Static electricity should be an important consideration in cold, dry environments. It is less of a problem when it is warm and humid.

When using SWTBUG® with an MP-A processor card, one board change is necessary since SWTBUG® is a full 1 K ROM and MIKBUG® was a 1/2K device, If this ROM is replacing MIKBUG, remove MIKBUG® from its socket. On the back side of the MP-A board you will notice that pin 15 of IC-2 (ROM) is grounded. The land coming from pin 15 should be broken and a wire added as shown below.



SWTBUG® should now be installed in the socket for IC-2. Be sure to orient the ROM correctly when re-installing. The semicircle notch or dot should match with the MP-A board's component layout drawing. When installing SWTBUG® in the MP-A2 processor board no board modifications are necessary. Follow the instructions supplied with the MP-A2 instruction set.

**SWTBUG® OPERATION**

The SWTBUG® firmware enables the computer to communicate with a terminal to perform various programming and debugging functions. SWTBUG® will communicate with a terminal via either a MP-C control interface or MP-S ACIA serial interface on I/O port 1. An optional MP-C interface can be installed on I/O port 0 for punch and load functions. Although SWTBUG® is essentially compatible with MIKBUG, be sure to read the COMPATIBILITY section before running any programs written for MIKBUG. Below is a detailed description of each SWTBUG® command.

**RESET**

Upon receiving a RESET command, as during power up, SWTBUG® will initialize the system to receive commands from a terminal. When the RESET button is pushed, control will transfer to location E0D0 of SWTBUG®. The RESET button should be used for exiting loops or malfunctioning programs. After resetting, the computer should respond with a carriage return, line feed and a $ sign. At this point, SWTBUG® is waiting for commands. If breakpoints are being used, the RESET function will not disable breakpoints. The BREAKPOINT function should be referenced for additional information.

**MEMORY EXAMINE AND CHANGE M (addr)**

The Memory Examine and Change function can be used to enter machine code programs and to display and/or change the contents of memory. The Memory Examine and Change function should be used as follows:

1.) Type **M**. The computer should echo the **M** and output a space.
2.) Type in the four digit hexadecimal address that you wish to examine and/or change. The computer should respond with a carriage return, line feed, $, the address and the data that is stored at this address.
3.) At this point the user has the option of advancing, either forward or backward, to the next memory location, or changing the data stored at the displayed address and advancing to the next location or of exiting the **M** function.
   a.) To display the next sequential address and data, a line feed or any character other than 0123456789ABCDEF:;^=>? or a space or a carriage return may be entered. Any leading spaces that are entered will be ignored by the memory change function.
   b.) To display the next sequential address going backward from the present location, a ^ should be entered.
   c.) To change the data stored at the displayed location, enter the new data, either with or without a leading space. If a non-hex value, such as a 3Q is entered the data will remain unchanged and the memory change function will be exited. If the data is unable to be changed (write protected memory, etc.) a ? will be output and the memory change function will be exited.
   d.) To exit the Memory Examine and Change function, type a carriage return.

Below is an example. The underlined parts are what was entered by the user.

```
$M 0100                 MEMORY LOCATION 0100 OPENED
$0100 00    .           DISPLAY NEXT LOCATION
$0101 BD       .        DISPLAY NEXT LOCATION – SPACE IGNORED
$0102 5D    .           DISPLAY NEXT LOCATION
$0103 C1    01          CHANGE CONTENTS TO 01
$0104 C9         23     CHANGE CONTENTS TO 23. SPACES IGNORED
$0105 15    ^           READ PREVIOUS LOCATION
$0104 23    .           DISPLAY NEXT LOCATION
$0105 15    3Q          ENTER NON-HEX VALUE
$M 0105                 SWTBUG CONTROL RESUMED. OPEN NEW LOCATION
$0105 15                EXIT BY HITTING CARRIAGE RETURN
$M E000                 OPEN ANOTHER LOCATION
$E000 FE    56?         ATTEMPTED TO CHANGE A WRITE PROTECTED MEMORY
$                       SWTBUG CONTROL RESUMED
```

**REGISTER DUMP FUNCTION R**

The **R** command will display the current contents of the MPU's pushdown stack. “Current” means the status of data stored on the stack immediately before SWTBUG® control is resumed. The following is a typical register dump:

```
      $R
      $FF    16   23   17EC   0103   A042
      $

CONDITION   ACC   ACC   INDEX   PGM.   STACK
CODES       B     A     REG     CTR.   PTR
```

The condition codes are as defined below:

| BIT NO. | LABEL | CONDITION CODE |
|---|---|---|
| 0 | C | Carry-borrow |
| 1 | V | Overflow |
| 2 | Z | Zero |
| 3 | N | Negative |
| 4 | I | Interrupt mask |
| 5 | H | Half carry |

In the above example the condition code of "3C" can be interpreted as follows:

| $3C_{16}$ = | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|
| | BIT 7 | BIT 6 | BIT 5 | BIT 4 | BIT 3 | BIT 2 | BIT 1 | BIT 0 |

Below are two examples of how the **R** command works. Assume that this small program was entered to change certain registers.

```
$M 0100
$0100 CE    8E        LOADS STACK POINTER TO 100
$0101 12    10
$0102 34    00
$0103 86    CE        LOAD INDEX REGISTER WITH 1234
$0104 00    12
$0105 C6    34
$0106 FF    86        LOAD ACCUMULATOR A WITH 00
$0107 FD    00
$0108 08    C6        LOAD ACCUMULATOR B WITH FF
$0109 23    FF
$010A 67    7E        JUMP BACK TO SWTBUG CONTROL
$010B F7    E0
$010C 60    E3
$010D DD
AT THIS POINT THE STATUS WILL NOT BE PUSHED ON THE STACK
```

```
$R
$FF DC FC 6EFD 0100 A042   REGISTER DUMP BEFORE RUNNING PROGRAM
$G
$R
$FF DC FC 6EFD 0100 A042   NOTE R DUMP THE SAME AFTER RUNNING
$J 0100
$R
$FF DC FC 6EFD 0100 A042   THE SAME AFTER A JUMP
$M 010A                    AT THIS POINT THE JUMP TO SWTBUG CONTROL
$010A 7E  3F               IS REPLACED BY A SWI. NOTE THE VALUE
$010B 50                   OF THE REGISTERS AFTER THE NEXT DUMP.
$G
$F9 FF 00 1234 010A 0FF9   REGISTER DUMP GIVEN BY SWI
$R
$F9 FF 00 1234 010A 0FF9   DUMP SHOWS PROGRAM CHANGES
$
$FF DC FC 6EFD 0100 A042   R DUMP AFTER RESET
```

In the above program you will notice that the register dump after running the program is the same as before even though the program contained statements that changed the processor's registers. The register dump did not reflect these changes because the new conditions were not pushed on the computer's stack. Note, however, the register dump did reflect the change when the last instruction was a software interrupt -- a SWI instruction will push the processor's status on the stack and then display the contents of the registers.

**CT-1024 CLEAR SCREEN COMMAND C**

The C command outputs a home-up ($10_{16}$) and an erase to end of frame ($16_{16}$) control characters for the clearing of the screen on a SWTPC CT-1024 or equivalent terminal system.

**GO TO USER'S PROGRAM FUNCTION G**

Upon entering a **G** command, SWTBUG® will transfer control to the user's program by executing a RTI (return from interrupt) instruction. This effectively causes the computer to jump to the memory address stored in memory locations A048 and A049 in the SWTBUG® RAM. A048 contains the most significant byte and A049 the least significant byte of the memory address. If, for example, you wish to execute a program starting at 0100, change A048 to a 01 using the **M** function and change A049 to 00. Typing a **G** will then cause the computer to execute the program whose starting address is 0100, Upon entering a program using the **G** function, the stack pointer will be set at A049. The **G** function is also used to restart a program after a breakpoint has been moved. In this case A048 and A049 should not be changed. See the Breakpoint section for further information.

**JUMP TO USER'S PROGRAM J(addr)**

The **J** command will cause the computer to execute the program whose starting address is given in the entered address. The **J** command does not look at locations A048 and A049. The stack pointer will be set at A042 upon entering a program with a jump command. Example: J 0100. If a non-hex character is entered, SWTBUG® control will resume.

**ASCII TAPE PUNCH COMMAND P**

The **P** command provides a means for storing the contents of specified memory on either cassette or paper tape. Normal output from the computer during a punch is thru either an MP-C or MP-S serial interface on I/O port 1, but a MP-C interface on I/O 0 can be selected. (See the description of the 0 command.) To use the **P** command, the upper and lower limits of the range to be punched must first be stored in locations A002-A005 of the SWTBUG@ RAM. If you wanted to punch from addresses 0123 to 4567 (inclusive) you would use the memory examine and change functions to set computer memory as follows.

| | | |
|---|---|---|
| A002 → | 01 | MOST SIGNIFICANT BYTE OF LOWER ADDRESS |
| A003 → | 23 | LEAST SIGNIFICANT BYTE OF LOWER ADDRESS |
| A004 → | 45 | MOST SIGNIFICANT BYTE OF UPPER ADDRESS |
| A005 → | 67 | LEAST SIGNIFICANT BYTE OF LOWER ADDRESS |

Typing **P** would turn the punch on and output the specified memory data. A sample punch output is as follows:

```
$M A002
$A002 02    01        MSB OF LOW ADDRESS
$A003 72    00        LSB OF LOW ADDRESS
$A004 EF    01        MSB OF HIGH ADDRESS
$A005 00    20        LSB OF HIGH ADDRESS
$A006 5F
$P                    TAPE PUNCH COMMAND
S11301008E1000CE12348600C6FF3FE0E3DD005DB2
S11301108090E05160F73A8201F500FFC79771D1F2
S104012000DA
$
```

```
   S1       13      0100    8E1000CE12348600C6FF3FE0E3DD005D    B2
START OF   BYTE     BLOCK                 DATA                 CHECK
  BLOCK    COUNT   STARTING                                     SUM
CHARACTER  (HEX)   ADDRESS
```

The S1 at the start of the block is used to tell the load routine that valid punch data follows. Each punch block must begin with the S1. The $13_{16}$ is the number of bytes that follow in the block. In this case two bytes are required for the starting address of the block (01 and 00), $10_{16}$ bytes are required for the data (8E 10 00 CE 12 34 86 00 C6 FF 3F E0 E3 DD 00 50) and one byte is required for the checksum (B2). The checksum is generated by adding the complement of the start of block address and the data, 8 bits at a time. At load-in time another checksum is generated by the load routine which must match with the one generated at punch time.

As the punch begins a PUNCH ON ($12_{16}$) control character will be output to the punch device. If a MP-C control interface is the selected interface unused lines on the PIA will be strobed to turn on the punch function of a SWTPC AC-30 or equivalent tape interface. (See the PIA Strobing section for more information.) When the punch is completed a PUNCH OFF ($14_{16}$) control character will be output and another PIA line will be strobed. User control is then returned to SWTBUG. To complete the tape making procedure you will have to enter the end of tape command described below.

## END OF TAPE COMMAND E

The **E** command will punch the contents of the program counter (A048-A049) and an S9 to tape. The S9 is decoded by the load routine as an “end of tape” marker. For example, if you wish to save a program that resides from 000 to 000F and whose starting address is 005 you would perform the following sequence:

```
$M A002
$A002 FC 00
$A003 3E 00
$A004 A0 00
$A005 49 0F
$A006 4C
```

```
$M A048
$A048 01 00
$A049 03 05
$A04A F4
$P
$S113OOOOA0501001EF023FF01A01B351B37022443
$E
S105A04800050DS9
$
$
```

A048 and A049 are automatically transferred to A002-A005 and punched to tape by the E command. A short delay follows the S9 to allow clean load-ins on cassette tape. Appropriate punch on/off commands are automatically sent.

## TAPE LOADER FUNCTION L

The **L** function is used to load either a MIKBUG® or SWTBUG® formatted program from either paper or cassette tape. To use the **L** function, first set up the tape in the loading device and type **L**. As with the punch command, a **L** function will load from an MP-S or MP-C on I/O port 1 or if selected a MP-C on I/O port 0. A READER ON ($11_{16}$) control character is output at the beginning of the load and terminal echo is disabled. Again the PIA is strobed for use with a tape interface. When the S9 end of tape marker is read, control will return to SWTBUG®. If a checksum error is detected, a ? will be printed and control will return to SWTBUG® The load routine verifies that the data being loaded is actually stored in the correct memory locations. If, for some reason, a byte is not stored (bad memory, etc.) the computer will respond with a ?. To abort the load function the RESET switch can be pressed. Before attempting to load a SWTPC binary formatted tape, be sure to read the COMPATIBILITY section.

## OPTIONAL PORT COMMAND 0 (not zero)

The **0** (not zero) command enables the user to load from or punch to a MP-C control interface plugged on to port 0. To use the **0** command, type 0 followed by the desired option P, E or L (punch, end of tape or load). The same rules apply for using the P, E and L functions as described earlier. The 0 command will not support an ACIA type serial interface on port 0.

NOTE: When using a MP-C interface on port 0, a RESET will not, automatically turn off the reader and punch as is done when installed on port 1. This makes it impossible to create or load a binary formatted tape from an MP-C interface installed on I/O port 0 using existing binary load and punch programs.

## SOFTWARE BREAKPOINTS B(addr)

The **B** command enables the user to enter breakpoints (software interrupts) in a program for debugging purposes. Breakpoints enable a program to be stopped at any point for register examination, memory changing, etc. To use the breakpoint function, first load in your program and set up A048 and A049 to the starting address of the program. Type **B** and then enter the address where you want to set the first breakpoint. SWTBUG® will store the data that was at this address and replace it with a 3F (software interrupt). Typing **G** for Go to User Program will start program execution. When the program reaches the 3F, execution will stop and the processor's registers will be displayed. The **B** function can be used again to move the breakpoint to a new location -- SWTBUG® automatically replaces the data at the old breakpoint location with the original instruction. A **G** should be used to re-start the program -- **do not** reset A048 and A049 -- SWTBUG® automatically pulls the restart location from the stack. To remove breakpoints, type **B** followed by a carriage return. Breakpoints should always be removed in this way after using the breakpoint function. If, while using breakpoints, the system is reset, the correct location on the stack will be lost. After resetting, the program counter addresses (A048, A049) should be reinitialized to the beginning of the program and execution restarted.

A previously set breakpoint will remain and may be changed or removed as described earlier. If, when using the B command, a non-hex value is entered the previous breakpoint will be removed and SWTBUG® control will resume.

There are several things that one must be aware of when using breakpoints to insure proper operation.

1.) The breakpoint function uses the same locations as do vectored software interrupts; therefore, vectored software interrupts should not be used with break points.
2.) The SWI jump location, A012, will be set to El24 when breakpoints are not in use, as after power up, and will be set to El23 when breakpoints are in use. This location serves as a pointer to tell the computer what to do when a 3F is seen. The RESET button will not reset this location to the non-breakpoint state. The breakpoint-activated state can only be exited by typing **B** followed by a carriage return. If you are using breakpoints in a program that "bombs out" and you hit the RESET switch, you must clear the present breakpoint before going on to another program. If this is not done before a new program is loaded in, the first time the **B** command is used one byte of the new program will be replaced by the stored byte from the last program.
3.) Do not set a breakpoint to an address where a breakpoint is already set. Doing so will cause the computer to lose the original program data.

| DO NOT | THIS IS OK |
|---|---|
| B 1377 | B 1377 |
| G | B 0100 |
| B 1377 | B1377 |
| | B carriage return |

4.) The breakpoint routine uses SWTBUG® RAM locations A014-A016; therefore, programs which use these three bytes should not be used in conjunction with breakpoints.

**DISK BOOT D**

SWTBUG® contains the boot necessary to initialize a SWTPC MF-68 disk system. Typing **D** will transfer control to the disk operating system, (if attached). If **D** is accidentally typed with no disk attached, the RESET button must be pressed. Since the disk boot contains no error detection, it may need to be typed more than once to do a boot.

**JUMP TO PROM PROGRAM Z**

Typing **Z** will transfer control to a program stored in PROM (if applicable) whose starting address is at C000. Typing **Z** is equivalent of typing **J** C000

**BYTE SEARCH F (high address) (low address) (byte)**

The **F** (find] command will search memory from the specified low address to tile high address, inclusive, and will display all memory locations containing the byte specified. For example, to find all memory locations between 0100 and 0200 that contain 8E, the following command should be used: F 020001008E. Note that no spaces may be used between addresses and that the high address goes first.

```
$F 020001008E
```

If a non hex value is entered, SWTBUG® control will resume.

## VECTORED SOFTWARE INTERRUPTS

Normally when encountering a SWI (3F) instruction, the computer will display the processor's registers and SWTBUG® control will be resumed. If desired, the 3F command can be vectored to anywhere in memory, just like the NMI and IRQ interrupts. To use the vectoring capability simply store the service routine address at location A012-A013 in the SWTBUG® RAM. When a 3F is encountered, processor control will be transferred to the memory address stored in A012-A013. Note: each time the system is RESET, A012 will be reset to the location of the register dump routine. This means that any program which uses vectored SWI's should set up this location each time it is executed. If the location you wish to vector to is 10D0, for example, the following statements at the beginning of the program will set up the vector correctly:

```
LDX # $10D0     LOAD VECTOR ADDRESS
STX $A012       STORE VECTOR
```

Vectored software interrupts should not be used in conjunction with breakpoints since the breakpoint routine uses locations A012-A013.

## VECTORED INPUT/OUTPUT

If desired, input and output can be vectored to a MP-S or MP-C interface on ports other than #1. Locations A00A-A00B contains the port address that the subroutines INEEE and OUTEEE use for inputting and outputting characters. To use vectored input/output your program must store the desired I/O address in A00A–A00B before any I/O is done. Below is a list of I/O address assignments for each port:

| PORT | ADDRESS |
|---|---|
| 0 | 8000 |
| 1 | 8004 |
| 2 | 8008 |
| 3 | 800C |
| 4 | 8010 |
| 5 | 8014 |
| 6 | 8018 |
| 7 | 801C |

The program statements that would set up the correct port would be as follows:

```
LDX #$8018      I/O on port 6
STX $A00A       Store
```

SWTBUG® will look at the port and will self-configure for either a MP-C or MP-S type interface.

NOTE: Any time that SWTBUG's control sequence is initiated or when the RESET button is pushed, the I/O address will be reset to port # 1. Therefore complete SWTBUG® monitor control cannot be moved to another port.

## USING NON-MASKABLE INTERRUPTS

Using non-maskable interrupts is very similar to using vectored software interrupts. A non-maskable interrupt will occur whenever the NMI line on the computer's bus is grounded either through hardware or by an ACIA or PIA. When the NMI is seen, processor control will be transferred to the location stored in A006 and A007. For example if an NMI service routine is desired at location 1000 the following statements should be used at the beginning of your program to set up the correct NM1 jump address.

```
LDX #$1000
STX  $A006
```

## USING MASKABLE (IRQ) INTERRUPTS

Using regular maskable interrupts is the same as using non-maskable interrupts except that when the IRQ line is grounded processor control will jump to the address stored in A000 and A001. The computer will only respond to the interrupt if the processor's interrupt mask bit 1 is 0. A CLI instruction at the beginning of your program will insure this condition.

## PIA STROBING

### Use of the Control Interface for Read/Punch-On/Off Decoding

SWTBUG® software contains subroutines to send out pulses to unused pins of the PIA integrated circuit on the MP-C serial control interface that can be used for automatic reader / punch controls. These pulses can be used if you are using a SWTPC AC-30 cassette interface and a terminal in which access to the control command decoding is denied.

If you intend to use the read/punch control logic output on the MP-C control interface board, make the following connections from the indicated pins of ICI on the MP-C control interface board to the specified pins of a twelve pin male connector shell. The connector pinning shown below is correct for a SWTPC AC-30 cassette interface and will need modifications for other units. Be sure to make the wires long enough to reach your AC-30 where the connector will be plugged. If you have access to your terminal's 16X baud rate clock, the terminal's clock bus should be broken and the 16X clock IN and OUT lines brought out to the same connector.

| | |
|---|---|
| MP-C IC1 pin 7 (read on) | 12 pin male shell female pin 1 |
| MP-C IC1 pin 4 (punch on) | 12 pin male shell female pin 2 |
| MP-C IC1 pin 6 (read off) | 12 pin male shell female pin 3 |
| MP-C IC1 pin 5 (punch off) | 12 pin male shell female pin 4 |
| Terminal's 16X clock OUT | 12 pin male shell female pin 5 |
| Terminal's 16X clock IN | 12 pin male shell female pin 6 |
| MP-C ground | 12 pin male shell female pin 12 |

These signals are low going pulses and are about 15 microseconds wide. They are not buffered and should drive a maximum of only one standard TTL load.

PIA stroking will work only on SWTBUG®'s **L**, **P** and **E** functions. Strobing is not supported in BASIC and some other SWTPC software.

## OPERATING THE MP-A2 PROCESSOR BOARD AT BAUD RATES HIGHER THAN 1200 BAUD

The MP-S Serial Interfaces available for the SWTPC 6800 Computer System are capable of operating at baud rates up to 9600 baud. Although baud rate clocks for 110, 150, 300, 600 and 1200 baud are generated, buffered and fed onto the mother board by IC4 of the MP-A2 board, clocks for additional baud rates are also available from IC4 as well. The table below gives the baud rate and respective output pin number of IC4.

| BAUD RATE | MP-A2 IC4 pin |
|---|---|
| 75 | 9 |
| 200 | 6 |
| 1800 | 15 |
| 2400 | 3 |
| 3600 | 16 |
| 4800 | 2 |
| 7200 | 17 |
| 9600 | 1 |

To use the selected clock, run an insulated jumper between the specified pin and pin 13 of IC10 on the MP-A2 board. Run another insulated jumper between pin 12 of IC10 and either the UD1 or UD2 bus connections points at the connector edge of the MP-A2 circuit board. IC10 is a low power TTL buffer which must be inserted between the baud rate clock generator and the mother board bus. Since user defined lines UD1 and UD2 are carried on just the 50-pin main board bus and lines UD3 and UD4 are carried on just the 30-pin interface board, it will be necessary to jumper two of the buses together to provide the selected baud rate clock on the interface card bus. Each serial interface card to be operated with the selected baud rate clock will have to be jumpered so its clock is acquired from the selected user defined line rather than one of the five original baud rate clocks already present.

## OPERATING THE MP-A PROCESSOR BOARD AT BAUD RATES HIGHER THAN 1200 BAUD

When using the MP-S serial interface with an MP-A processor board, baud rate clocks for up to 9600 baud are available from the baud rate generator on the MP-A processor board. The table below shows the baud rates available and from which pin of IC4 on the MP-A board they are derived. These 16X baud rate clocks are best feed back to the interface boards via the user defined lines provided on the mother board. These baud rates of course are in addition to the 110, 150, 300, 600 and 1200 baud rate clocks already provided on the mother board.

| BAUD RATE | MP-A IC4 pin |
|---|---|
| 75 | 9 |
| 200 | 6 |
| 1800 | 15 |
| 2400 | 3 |
| 3600 | 16 |
| 4800 | 2 |
| 7200 | 17 |
| 9600 | 1 |

## COMPATIBILITY

Although SWTBUG® has been written to be as compatible as possible with MIKBUG® and with software supplied by SWTPC, it can never be completely MIKBUG® compatible. All major subroutines of SWTBUG® are address and function compatible with MIKBUG®, but if you have a program that enters into the middle of a MIKBUG® routine for some reason, program modifications will be necessary. The following is a list of the MIKBUG® compatible subroutines and strings along with their entry point addresses.

| | | | |
|---|---|---|---|
| E040 | LOAD19 | E0C8 | OUT4HS |
| E047 | BADDR | E0CA | OUT2HS |
| E055 | BYTE | E0D0 | START |
| E075 | OUTCH | E0E3 | CONTRL |
| E078 | INCH | E19C | MCLOFF |
| E07B | PDATA2 | E19D | MCL |
| E07E | PDATA1 | E1AC | INEEE |
| E0BF | OUT2H | E1D1 | OUTEEE |

If any doubt exists as to the compatibility of a particular program, it should be disassembled and any references to memory locations E000–E1FF be verified. Since SWTBUG® is more complex than MIKBUG, more RAM area must be used in the 6810 SWTBUG® RAM. If vectored software interrupts and breakpoints are not being used, the area from A014 to A033 and from A04A to A07F can be used for small, temporary programs such as memory diagnostics. Note that some programs written for MIKBUG® use locations A034-A036 - these locations are not available for use in SWTBUG®

## LOADING BINARY TAPES THRU SWTBUG®

SWTBUG® was written to accept the binary formatted tapes supplied by SWTPC. These tapes include 4K BASIC, 8K BASIC, CORES and DESEMBLER. When loading these tapes the following rules must be followed:

1.) The tape reading device (AC-30, etc.) must be locked in the read on mode during the binary load.
2.) Binary tapes must be loaded in thru port # 1, the control port. The optional load from port 0 command is not supported in binary. You may load in ASCII however.
3.) When using a PIA type interface to load binary tapes, the unused lines used for reader/punch on/off strobing are not activated.

NOTE: This does not mean that SWTBUG® is equipped with a binary loader-only certain SWTPC binary tapes that contain a special binary loader (in ASCII) will work correctly.

To load the tape simply follow the instructions given for loading an ASCII tape, but keep the reader locked on.

## SPECIAL NOTES ON USING AN ACIA AND PROGRAM MODIFICATIONS

Many available 6800 programs written for MIKBUG® assume that a PIA type MP-C control interface is being used and may address this port directly. When using an ACIA type interface, these references need to be changed. For example, some programs, such as BASIC and CORES, poll the PIA periodically to see if a character has been typed in. This is done in order to kick out of a loop or a print sequence. (BASIC uses CTL..C. and CO-RES uses CTL..X.) The source statements that do this usually take the following form:

```
B6 8004      LDA A PIAD                LOAD A FROM DATA REG.
2B 03        BMI PRINT                 BRANCH IF NOT CHAR. SEEN
7E XX XX     JMP READY or RESTART
                      PRINT REMAINDER OF SEQUENCE
```

To change to a MP-S serial interface, this code can sometimes be replaced as follows;

```
LDA A PIAD →        ASR A              SEE IF CHAR. LOADED
BMI PRINT →         BCS PRINT          BRANCH IF CHAR. INPUT
                    JMP READY
                      PRINT REMAINDER OF SEQUENCE
```

Before modifying any programs on your own, you should have a working knowledge of SWTBUG'S ACIA input routine, ACIA operation, and your particular program. The following is a list of patches to some SWTPC supplied programs.

**BLKJAK - SWTPC 6800 Black Jack Program**

| LOCATION | DATA |
|---|---|
| 0270 | 7E 064A |
| 0647 | 7E 026D |
| 064A | B6 E008 |
| 064D | 27 08 |
| 064F | B6 8004 |
| 0652 | 2B F3 |
| 0654 | 7E 0275 |
| 0657 | B6 8004 |
| 065A | 47 |
| 065B | 24EA |
| 065D | 20 F5 |

With the above modifications BLKJAK will be compatible with either an ACIA or PIA type interface.

**CO-RES Ver. 1.0 and 1.01 ACIA Modifications**

| LOCATION | DATA |
|---|---|
| 1682 | 47 |
| 1683 | 24 02 |
| 1685 | 20 A0 |

BASIC 8K and 4K up to an including Ver. 2.0 cannot be modified for ACIA operation. Later versions should be purchased.

## GENERAL RULES FOR PROGRAM WRITING

Although for a user program to be functional it need only work with the exact system it was written for, following a few simple rules reduces program modifications for 6800 systems using other monitors. Following these rules will make your programs more professional and versatile. Some general guidelines are as follows:

1.) Minimize the number of references made to the ROM.
2.) Do not use strange, in-between SWTBUG® addresses. Generally only the routines BADDR, BYTE, PDATA1, INHEX, OUT4HS, OUT2HS, CONTRL, INEEE and OUTEEE should be used.
3.) For large programs, vector I/O through a jump instruction for ease of change to match other I/O packages. Example:

| DON'T | DO | | |
|---|---|---|---|
| JSR INEEE | JSR INPUT | → | |
| JSR INEEE | JSR INPUT | → | INPUT: JMP INEEE |
| JSR INEEE | JSR INPUT | → | |

4.) Try not to use the SWTBUG® RAM any more than necessary. With the exception of using it as stack storage and memory diagnostics, there is no real reason to use the SWTBUG® RAM area.
5.) Define the stack area at the beginning of the program. Example: Start LDS #$ A042. Relocating the stack location to A042 at the beginning of each of your programs will prevent you from having to reload the program counter addresses A048 and A049 each time you RESET and restart your program.
6.) Most programs should have a provision for exiting them without hitting the RESET button. A jump to CONTRL (7E E0E3) instruction in your program will cause SWTBUG® control to resume when executed.

## MEMORY DIAGNOSTICS

The earlier memory diagnostics ROBIT, MEMCON and CDAT supplied by SWTPC were compatible only with MIKBUG®. The new versions ROBIT 2, MEMCON 3 and CDAT 2 are compatible with both MIKBUG® and SWTBUG®.

## PROGRAM DESCRIPTION

Although the source listing of SWTBUG® is well commented, the following subroutine by subroutine description should be of use to those who wish to gain the maximum advantage of its routines.

### TEMPORARY STORAGE LOCATIONS

| | |
|---|---|
| IRQ (A000) | This location is used by the standard IRQ interrupt request feature. When an interrupt is generated, processor control will jump to the location stored in IRQ. |
| BEGA (A002) | This location is where the beginning address is stored for the punch and end of tape routines. |
| ENDA (A004) | This location is where the ending address is stored for the punch and end of tape routines. It is also used by the byte search routine. |
| NMI (A006) | NMI is used by the non-maskable interrupt (NMI) function. When an NMI is generated, processor control will jump to the location stored in NMI. |

| | |
|---|---|
| SP (A008) | Temporary storage location for the stack pointer. SP is used in the register dump subroutines and by the breakpoint function. |
| PORADD (A00A) | This location contains the port address used for SWTBUG's I/O routines. |
| PORECH (A00C) | This byte tells SWTBUG® 's input routines whether or not to echo. |
| XHI (A00D) | Temporary index register storage used by numerous routines. |
| XLOW (A00E) | Temporary index register storage used by numerous routines. |
| XTEMP (A010) | Temporary index register storage for input and output routines. |
| SWIJMP (A012) | When a SWI instruction is encountered, processor control will transfer to the location stored in SWIJMP. |
| BKPT (AQ14) | Temporary breakpoint address storage. |
| BKLST (A016) | Temporary data storage for the breakpoint routine. |
| TW (A044) | Temporary storage location for load/punch. |
| TEMP (A046) | Temporary storage location for punch and load. |
| BYTECT (A047) | Temporary storage location for load/punch. |

## SWTBUG@SUBROUTINE AND TEXT STRING DESCRIPTION

IRQV (E000) — This is the entry point for regular IRQ interrupts. Processor control is given of the service routine whose address is stored in IRQ.

JUMP (E005) — This is the service routine for the **J** command. BADDR is used to input the address and a jump then occurs to the correct address.

CURSOR (E009) — Home-up and erase to end of frame characters for CT 1024.

LOAD (E00C) — Load is the ASCII loading routine. Load uses a number of other SWTBUG® subroutines.

BADDR (E047) — BADDR is a subroutine to input a 4-digit hexadecimal number, such as 137D, from the control terminal. BADDR uses the subroutines BYTE, INCH and INHEX and uses temporary storage locations XHI, XLOW, CKSM, both accumulators and the index register. When BADDR is called it will look for four hex numbers to be entered from the terminal. If a non-hex value, such as **H,** is entered, SWTBUG® control will resume. If all characters entered are valid hex, the results will be stored in XHI, XLOW and the index register. Accumulator **A** will contain of XLOW. If 137D is entered the results will be as follows-

| | |
|---|---|
| ACC A | 7D |
| ACC B | CKSM |
| IXR | 137D |
| XHI | 13 |
| XLOW | 7D |

- CKSM and ACC B are used internally to generate a check sum for the PUNCH routine.

BYTE (E055) — BYTE is similar to BADDR, but inputs only two hex characters from the terminal to generate one 8-bit byte equivalent. BYTE uses the subroutines

INHEX and INCH, temporary storage locations CKSM and both accumulators. If a non-hex value is entered, SWTBUG® control will resume. When BYTE is called as a subroutine, the computer will wait for two hex characters to be entered thru the control port. If a 3C is entered, the results will be as follows:

| | |
|---|---|
| ACC A | 3C |
| ACC B | CKSM |
| IXR | UNCHANGED |
| CKSM | Prior CKSM + check sum generated inside BYTE |

| | |
|---|---|
| OUTHL (E067)<br>OUTHR (E06B) | These subroutines are used by OUT2HS and OUT4HS to output hexadecimal numbers |

OUTEEE
OUTCH
OUTEE
(E1D1)

This is the character output routine used by PDATA1, OUT4HS, OUT2HS and most programs written for SWTBUG/MIKBUG® to output one character from the computer to the control port (I/O # 1). OUTEEE, OUTCH and OUTEE1 are all functional equivalents- OUTEE1 is the main output routine with OUT-CH and OUTEEE being jumps to OUTEE1. When using this routine, OUTEEE (E1D1) should be used to maintain compatibility with MIKBUG® systems.
To use OUTEEE the character to be output should be placed in the A accumulator in its ASCII form. To output the letter A on the control terminal, the following program could be used.

```
LDA A        #$41
JSR          OUTEEE
```

The processor's registers are affected as follows.

| | |
|---|---|
| ACC A | changed internally |
| ACC B | not affected |
| IXR | not affected |

OUTEEE is an 8-bit output routine and does not generate a parity bit.

INEEE,
INCH,
INEEE1
(E1AC)

The locations are all functionally equivalent to SWTBUG® 's character input routine. This routine will look for one character from the control terminal (I/O # 1) and store it in the A accumulator. Once called, INEEE will loop within it-self until a character has been input. Anytime input is desire, the call JSR INEEE should be used.
INEEE automatically sets the 8th bit to 0 and does not check for parity. When using INEEE the processor's registers are affected as follows:

| | |
|---|---|
| ACC A | loaded with the character input from the terminal |
| ACC B | not affected |
| IXR | not affected |

INCH8 (E1F6)

INCH8 is functionally the same as INEEE except that the 8th bit is not set to 8. This subroutine should be used whenever full 8-bit input is desired, such as in binary loader programs.

INHEX (E0AA)

INHEX is the subroutine used by BYTE and BADDR that will input one hexadecimal character from the control terminal. If a non-hex character is entered, SWTBUG® control will resume. If a hex character, such as an E is entered, the results will be as follows:

| | |
|---|---|
| ACC A | 0E |
| ACC B | not affected |
| IXR | not affected |

PDATA1 (E07E)

PDATA1 is the subroutine used to output a string of text on the control terminal. PDATA1 will start outputting data that is pointed to by the index register and will continue until a 04 is seen. For example, if you wanted to print HELLO on the terminal the following could be used.

```
                    ORG $100
          START     LDX #TEXT
                    JSR PDATA1
                    JMP CONTRL
          TEXT      FCB $0D, $0A
                    FCC /HELLO/
                    FCB 4
                    END
```

- The accumulator and register status after using PDATA1 is as follows:

| | |
|---|---|
| ACC A | Changed during the operation |
| ACC B | UNCHANGED |
| IXR | Contains the memory location of the 04 |

CHANGE (E088) CHANGE is SWTBUG's memory examine and change function. Change uses a number of other SWTBUG® subroutines.

OUT4HS (E0C8) OUT4HS is used to output a four-digit (16-bit) hexadecimal number onto the control terminal. The address of the most significant byte to be output should be loaded into the index register before calling OUT4HS. For example, to out-put the 16-bit hex number stored in memory locations 1000 and 1001 whose most significant byte is in 1000 while the least significant byte is in 1001 the following sequence should be used:

```
          LDX # $1000
          JSR OUT4HS
```

- If location 1000 contained a hex 3C and 1001 contained a hex 0B, an ASCII, 3C0B would be displayed on the screen when OUT4HS is called. Remember memory data is handled in hex but must be output as ASCII characters which have a different hex value than those stored in the computer's memory. The registers are affected as follows:

| | |
|---|---|
| ACC A | Changed during the operation |
| ACC B | UNCHANGED |
| IXR | Incremented by two. (1002 in this example) |

OUT2HS (E0CA) OUT2HS is similar to OUT4HS, but outputs only two hex characters (one byte). For example, to display the byte stored at 2008 the following sequence would be used:

```
          LDX # $2000
          JSR OUT2HS
```

- An ASCII 6C would be output to the control terminal if location 2001 contained a hex 6C. The registers are affected as follows:

| | |
|---|---|
| ACC A | Changed during the operation |
| ACC B | UNCHANGED |
| IXR | Incremented by one. (1001in this example) |

OUTS (E0CC) OUTS is a subroutine that outputs one space to the control terminal.

START (E0D0) START is the beginning of a sequence of steps that initialize the system during power up or reset. First, the stack pointer is set to be A042 and is stored in SP. Next, an FF is stored in location A943. This sets the interrupt mask bit in the stack so that when a **G** command is given the processor will not respond to interrupts until told to do so. The type of port being used (ACIA or PIA) is then determined by initializing it as a PIA and looking to see if this worked. If not, an ACIA is assumed and the proper ACIA initialization routine, AL2, is selected. The program then goes to CONTRL sequence.

| | |
|---|---|
| CONTRL (E0E3) | This MIKBUG® equivalent sequence again resets the stack to A042. PORECH is cleared to enable echo and the subroutine SAVGET is selected to get the correct port number and type. Next, the routines PNCHOF and RDOFF generate punch and reader off commands. A carriage return, line feed, erase to end of line ($15_{16}$) and a $ is then output to the control terminal. At this point SWTBUG® is ready for command input. |
| SFEl (E124) | SFEl is the entry point for non user-vectored software interrupt instructions. If vectored software interrupts are selected, a jump is executed to the proper location. If breakpoints are in use the stack pointer is not changed, the processor's registers are displayed and SWTBUG® is instructed to look for the next command. If neither breakpoints or vectored software interrupts are selected a register dump occurs and the CONTRL sequence is initiated. |
| PRINT (E130) | PRINT is the routine that actually does the dumping of the processor's registers. |
| LOOK (E173) | LOOK is the routine that inputs a character from the terminal and jumps to the appropriate location in SWTBUG® if it is a valid command. |
| SFE (E18B) | Entry point for software interrupt instructions. |
| S9 (E190) | S and 9 string for the end of tape routine. |
| MTAPE1 (E193) | This is the character string containing a carriage return, line feed, erase to end of line, four nulls, a S1 for tape control and 04 for PDATA1 control. |
| MCL (E19D) | This string contains a carriage return, line feed, three nulls, a $ and a 04 for PDATA1 control. |
| ElA5 (E1A5) | ElA5 is a special entry location which is used by the binary load routine on some SWTPC binary tapes. |
| NMIV (E1A7) | This routine fetches the correct jump location for a NMI. |
| SEARCH (E1AE) | Byte searching routine. |
| GOT0 (E1D0) | GOT0 contains the RTI instruction that is used by the G command to execute a user program. |
| SAVGET (E1D3) | This routine saves the index register in XTEMP and tests for the appropriate interface location and type. The index register is then loaded with the address of this interface. |
| ISACIA (E1D9) | ISACIA is the routine that sees if an ACIA or PIA is present. |
| BILD (E1F3) | BILD is a special sequence of increment stack pointer instructions used only by the binary loader on some SWTPC binary tapes. |
| ACIAIN (E1FF) | This is the ACIA input routine. PORECH is polled for the desired echo/don't condition. The character in the A accumulator is then output, the index register and B accumulator restored and an RTS instruction executed by the RES routine. |
| ACIOUT (E212) | This is the ACIA output routine which outputs the character in the A accumuator, and recovers the B accumulator and index register. |
| IN1 (E223) | IN1 is the PIA input routine which inputs a character from the control terminal and stores it in the A accumulator. The correct echo/non echo condition is selected and the B accumulator and index register are restored. |

| | |
|---|---|
| IOUT (E240) | IOUT is the PIA output routine which outputs the character in the A accumulator. |
| OPTL (E269) | OPTL is the service routine that sets up PORECH for I/O on port 0. The appropriate command P, E or L is then selected. |
| PIAECH (E27D) | This routine disables the echo on a PIA type interface. |
| PIAINI (E284) | This routine is used to initialize PIA type interfaces. |
| DELAY (E202) | DELAY is a general purpose delay loop. If desired, the index register can be pre-loaded with a number other than FFFF for shorter delays. The entry point in this case is DELAY1 (E2C5). |
| CLEAR (E2CC) | This routine generates a home up, erase to end of frame command for SWTPC CT 1024 and similar terminal systems. |
| BREAK (E2D9) | BREAK is the routine that is used to enter software breakpoints. First, the ad-dress is input by BADDR. If breakpoints were previously in use the data is replaced at the previous breakpoint address and the new breakpoint is inserted. |
| PNCHS9 (E31E | This routine selects A048 and A049 as the beginning and ending address for the punch routine and punches their contents. A S9 is then punched to tape followed by a short delay. |
| RDON (E334)<br>RDOFF (E347)<br>PNCHON (E34D)<br>PNCHOF(E353) | These subroutines are used to turn a reader/punch on and off. At the beginning of each routine the A accumulator is loaded with the ASCII value of the particular function that is normally decoded by a Teletype or other terminal system. The B accumulator is then loaded with a special bit pattern that will cause a predetermined line to be toggled on PIA type interfaces. ACC A is then out-put using OUTEEE and if a PIA type interface is being used the proper PIA line is strobed by the STROBE routine. The proper line is determined by the con-tents of ACC B. The subroutine RDON also clears the location PORECH and re-configures PIA type interfaces to be sure that the echo function of the character input routine is disabled. Both accumulators and the index register are used and are not retained. |

| ACC A | ACC B | FUNCTION |
|---|---|---|
| 11 | 20 | Reader ON |
| 13 | 10 | Reader OFF |
| 12 | 04 | Punch ON |
| 14 | 08 | Punch OFF |

| | |
|---|---|
| STROBE (E357) | This is the routine that actually generates the pulses on the unused lines of a PIA type interface for reader/punch control. |
| PUNCH (E376) | PUNCH is the ASCII punching routine of SWTBUG, PUNCH consists of several parts and uses various SWTBUG's subroutines. |
| TABLE (E3D1) | This is the command table used by the lookup routine. The table is arranged in three byte blocks. The first byte is the ASCII value of the command. The next two bytes are the address of the routine that will service the command. |

```
                        NAM    SWTBUG
               *      VERSION 1.00

                        OPT    PAG
               ****************************************************
               *REPLACEMENT FOR MIKBUG ROM
               *FOR SWTPC 6800 COMPUTER SYSTEM
               *COPYRIGHT 1977
               *SOUTHWEST TECHNICAL PROD. CORP.
               *AUGUST, 1977
               ****************************************************


A000                    ORG    $A000
A000           IRQ      RMB    2          IRQ POINTER
A002           BEGA     RMB    2          BEGINNING ADDR PNCH
A004           ENDA     RMB    2          ENDING ADDR PNCH
A006           NMI      RMB    2          NMI INTERRUPT VECTOR
A008           SP       RMB    1          S HIGH
A009                    RMB    1          S LOW
A00A           PORADD   RMB    2          PORT ADDRESS
A00C           PORECH   RMB    1          ECHO ON/OFF FLAG
A00D           XHI      RMB    1          XREG HIGH
A00E           XLOW     RMB    1          XREG LOW
A00F           CKSM     RMB    1          CHECKSUM
A010           XTEMP    RMB    2          X-REG TEMP STGE
A012           SWIJMP   RMB    2          SWI JUMP VECTOR
A044           TW       EQU    $A044      TEMPORARY STORAGE
A046           TEMP     EQU    $A046      TEMPORARY STORAGE
A047           BYTECT   EQU    $A047      BYTECT AND MCONT TEMP.
8004           CTLPOR   EQU    $8004      CONTROL PORT ADDRESS
C000           PROM     EQU    $C000      JUMP TO PROM ADDRESS
A014           BKPT     RMB    2          BREAKPOINT ADDRESS
A016           BKLST    RMB    1          BREAKPOINT DATA

A042                    ORG    $A042
A042           STACK    RMB    1          SWTBUG STACK

E000                    ORG    $E000

               *I/O INTERRUPT SEQUENCE
E000 FE A0 00  IRQV     LDX    IRQ
E003 6E 00              JMP    0,X

               *JUMP TO USER PROGRAM
E005 8D 40     JUMP     BSR    BADDR
E007 6E 00              JMP    0,X

E009 10        CURSOR   FCB    $10,$16,4 CT-1024 CURSOR CONTROL
E00A 16 04

               *ASCII LOADING ROUTINE
E00C BD E3 34  LOAD     JSR    RDON       READER ON, DIS ECHO, GET P#
E00F 8D 67     LOAD3    BSR    INCH
E011 81 53              CMP A  #'S
```

```
E013 26 FA              BNE    LOAD3     1ST CHAR NOT S
E015 8D 61              BSR    INCH      READ CHAR
E017 81 39              CMP A  #'9
E019 27 29              BEQ    LOAD21
E01B 81 31              CMP A  #'1
E01D 26 F0              BNE    LOAD3     2ND CHAR NOT 1
E01F 7F A0 0F           CLR    CKSM      ZERO CHECKSUM
E022 8D 31              BSR    BYTE      READ BYTE
E024 80 02              SUB A  #2
E026 B7 A0 47           STA A  BYTECT    BYTE COUNT
               *BUILD ADDRESS
E029 8D 1C              BSR    BADDR
               *STORE DATA
E02B 8D 28     LOAD11   BSR    BYTE
E02D 7A A0 47           DEC    BYTECT
E030 27 09              BEQ    LOAD15    ZERO BYTE COUNT
E032 A7 00              STA A  0,X       STORE DATA
E034 A1 00              CMP A  0,X       DATA STORED?
E036 26 08              BNE    LOAD19
E038 08                 INX
E039 20 F0              BRA    LOAD11
E03B 7C A0 0F  LOAD15   INC    CKSM
E03E 27 CF              BEQ    LOAD3
E040 86 3F     LOAD19   LDA A  #'?
E042 8D 31              BSR    OUTCH
E044 7E E2 D4  LOAD21   JMP    RDOFF1

               *BUILD ADDRESS
E047 8D 0C     BADDR    BSR    BYTE      READ 2 FRAMES
E049 B7 A0 0D           STA A  XHI
E04C 8D 07              BSR    BYTE
E04E B7 A0 0E           STA A  XLOW
E051 FE A0 0D           LDX    XHI       LOAD IXR WITH NUMBER
E054 39                 RTS

               *INPUT BYTE (TWO FRAMES)
E055 8D 53     BYTE     BSR    INHEX     GET HEX CHAR
E057 48        BYTE1    ASL A
E058 48                 ASL A
E059 48                 ASL A
E05A 48                 ASL A
E05B 16                 TAB
E05C 8D 4C              BSR    INHEX
E05E 1B                 ABA
E05F 16                 TAB
E060 FB A0 0F           ADD B  CKSM
E063 F7 A0 0F           STA B  CKSM
E066 39                 RTS

E067 44        OUTHL    LSR A            OUT HEX LEFT BCD DIGIT
E068 44                 LSR A
E069 44                 LSR A
E06A 44                 LSR A
E06B 84 0F     OUTHR    AND A  #$F       OUT HEX RIGHT BCD DIGIT
```

```
E06D 8B 30              ADD A  #$30
E06F 81 39              CMP A  #$39
E071 23 02              BLS    OUTCH
E073 8B 07              ADD A  #$7

                *OUTPUT ONE CHAR
E075 7E E1 D1  OUTCH   JMP    OUTEEE
E078 7E E1 AC  INCH    JMP    INEEE

                *PRINT DATA POINTED TO BY X REG
E07B 8D F8     PDATA2  BSR    OUTCH
E07D 08                INX
E07E A6 00     PDATA1  LDA A  0,X
E080 81 04             CMP A  #4
E082 26 F7             BNE    PDATA2
E084 39                RTS              STOP ON HEX 04

E085 7E E1 4A  C1      JMP    SWTCTL

                *MEMORY EXAMINE AND CHANGE
E088 8D BD     CHANGE  BSR    BADDR
E08A CE E1 9D  CHA51   LDX    #MCL
E08D 8D EF             BSR    PDATA1    C/R L/F
E08F CE A0 0D          LDX    #XHI
E092 8D 34             BSR    OUT4HS    PRINT ADDRESS
E094 FE A0 0D          LDX    XHI
E097 8D 31             BSR    OUT2HS    PRINT OLD DATA
E099 8D 31             BSR    OUTS      OUTPUT SPACE
E09B 8D DB     ANOTH   BSR    INCH      INPUT CHAR
E09D 81 20             CMP A  #$20
E09F 27 FA             BEQ    ANOTH
E0A1 81 0D             CMP A  #$D
E0A3 27 E0             BEQ    C1
E0A5 81 5E             CMP A  #'^       UP ARROW?
E0A7 20 2C             BRA    AL3       BRANCH FOR ADJUSTMENT
E0A9 01                NOP

                *INPUT HEX CHARACTER
E0AA 8D CC     INHEX   BSR    INCH
E0AC 80 30     INHEX1  SUB A  #$30
E0AE 2B 4C             BMI    C3
E0B0 81 09             CMP A  #$9
E0B2 2F 0A             BLE    IN1HG
E0B4 81 11             CMP A  #$11
E0B6 2B 44             BMI    C3        NOT HEX
E0B8 81 16             CMP A  #$16
E0BA 2E 40             BGT    C3        NOT HEX
E0BC 80 07             SUB A  #7
E0BE 39        IN1HG   RTS

E0BF A6 00     OUT2H   LDA A  0,X       OUTPUT 2 HEX CHAR
E0C1 8D A4     OUT2HA  BSR    OUTHL     OUT LEFT HEX CHAR
E0C3 A6 00             LDA A  0,X
E0C5 08                INX
```

```
E0C6 20 A3              BRA    OUTHR     OUTPUT RIGHT HEX CHAR

E0C8 8D F5      OUT4HS  BSR    OUT2H     OUTPUT 4 HEX CHAR + SPACE
E0CA 8D F3      OUT2HS  BSR    OUT2H     OUTPUT 2 HEX CHAR + SPACE

E0CC 86 20      OUTS    LDA A  #$20      SPACE
E0CE 20 A5              BRA    OUTCH     (BSR & TRS)

                *ENTER POWER ON SEQUENCE
E0D0 8E A0 42   START   LDS    #STACK
E0D3 20 2C              BRA    AL1       BRANCH FOR ADDRESS COMPATIBIL

                ********************************************
                *PART OF MEMORY EXAMINE AND CHANGE
E0D5 26 07      AL3     BNE    SK1
E0D7 09                 DEX
E0D8 09                 DEX
E0D9 FF A0 0D           STX    XHI
E0DC 20 AC              BRA    CHA51
E0DE FF A0 0D   SK1     STX    XHI
E0E1 20 02              BRA    AL4

E0E3 20 6D      EOE3    BRA    CONTRL    BRANCH FOR MIKBUG EQUIV. CONT

E0E5 81 30      AL4     CMP A  #$30
E0E7 25 A1              BCS    CHA51
E0E9 81 46              CMP A  #$46
E0EB 22 9D              BHI    CHA51
E0ED 8D BD              BSR    INHEX1
E0EF BD E0 57           JSR    BYTE1
E0F2 09                 DEX
E0F3 A7 00              STA A  0,X       CHANGE MEMORY
E0F5 A1 00              CMP A  0,X
E0F7 27 91              BEQ    CHA51     DID CHANGE
E0F9 7E E0 40           JMP    LOAD19    DIDN'T CHANGE
E0FC BE A0 08   C3      LDS    SP
E0FF 20 49              BRA    SWTCTL
                **************************************************

                *CONTINUE POWER UP SEQUENCE
E101 BF A0 08   AL1     STS    SP        INIT TARGET STACK PTR.
E104 86 FF              LDA A  #$FF
E106 BD E3 08           JSR    SWISET
                *CONFIGURE FOR PIA AND SEE IF OK
E109 CE 80 04           LDX    #CTLPOR
E10C BD E2 84           JSR    PIAINI    INIT PIA
E10F A6 00              LDA A  0,X
E111 A1 02              CMP A  2,X
E113 20 02              BRA    AL2

E115 20 19              BRA    PRINT     BRA FOR BILOAD

E117 26 39      AL2     BNE    CONTRL
```

```
               *INITIALIZE AS ACIA
E119 86 03             LDA A  #3         ACIA MASTER RESET
E11B A7 00             STA A  0,X
E11D 86 11             LDA A  #$11
E11F A7 00             STA A  0,X
E121 20 2F             BRA    CONTRL

               *ENTER FROM SOFTWARE INTERRUPT
E123 01        SF0     NOP
E124 BF A0 08  SFE1    STS    SP         SAVE TARGETS STACK POINTER
               *DECREMENT P COUNTER
E127 30                TSX
E128 6D 06             TST    6,X
E12A 26 02             BNE    *+4
E12C 6A 05             DEC    5,X
E12E 6A 06             DEC    6,X
               *PRINT CONTENTS OF STACK.
E130 CE E1 9D  PRINT   LDX    #MCL
E133 BD E0 7E          JSR    PDATA1
E136 FE A0 08          LDX    SP
E139 08                INX
E13A 8D 8E             BSR    OUT2HS     COND CODES
E13C 8D 8C             BSR    OUT2HS     ACC B
E13E 8D 8A             BSR    OUT2HS     ACC A
E140 8D 86             BSR    OUT4HS     IXR
E142 8D 84             BSR    OUT4HS     PGM COUNTER
E144 CE A0 08          LDX    #SP
E147 BD E0 C8          JSR    OUT4HS     STACK POINTER
E14A FE A0 12  SWTCTL  LDX    SWIJMP
E14D 8C E1 23          CPX    #SF0
E150 27 19             BEQ    CONTR1

E152 8E A0 42  CONTRL  LDS    #STACK     SET CONTRL STACK POINTER
E155 CE 80 04          LDX    #CTLPOR    RESET TO CONTROL PORT
E158 FF A0 0A          STX    PORADD
E15B 7F A0 0C          CLR    PORECH     TURN ECHO ON
E15E 8D 73             BSR    SAVGET     GET PORT # AND TYPE
E160 27 03             BEQ    POF1
E162 BD E2 7D          JSR    PIAECH     SET PIA ECHO ON IF MP-C INTER
E165 BD E3 53  POF1    JSR    PNCHOF     TURN PUNCH OFF
E168 BD E3 47          JSR    RDOFF      TURN READER OFF
E16B CE E1 9C  CONTR1  LDX    #MCLOFF
E16E BD E0 7E          JSR    PDATA1     PRINT DATA STRING
E171 8D 39             BSR    INEEE      READ COMMAND CHARACTER

               *COMMAND LOOKUP ROUTINE
E173 CE E3 D1  LOOK    LDX    #TABLE
E176 A1 00     OVER    CMP A  0,X
E178 26 07             BNE    SK3
E17A BD E0 CC          JSR    OUTS       SKIP SPACE
E17D EE 01             LDX    1,X
E17F 6E 00             JMP    0,X
E181 08        SK3     INX
E182 08                INX
```

```
E183 08                 INX
E184 8C E3 F8           CPX    #TABEND+3
E187 26 ED              BNE    OVER
E189 20 BF      SWTL1   BRA    SWTCTL

                *SOFTWARE INTERRUPT ENTRY POINT
E18B FE A0 12   SFE     LDX    SWIJMP    JUMP TO VECTORED SOFTWARE INT
E18E 6E 00              JMP    0,X

E190 53         S9      FCB    'S,'9,4   END OF TAPE
E191 39 04

                ***************************************************
E193 0D         MTAPE1  FCB    $D,$A,$15,0,0,0,'S,'1,4 PUNCH FORMAT
E194 0A 15
E196 00 00
E198 00 53
E19A 31 04

E19C 13         MCLOFF  FCB    $13       READER OFF
E19D 0D         MCL     FCB    $D,$A,$15,0,0,0,'$,4
E19E 0A 15
E1A0 00 00
E1A2 00 24
E1A4 04

E1A5 20 4C      EIA5    BRA    BILD      BINARY LOADER INPUT
                ***************************************************


                *NMI SEQUENCE
E1A7 FE A0 06   NMIV    LDX    NMI       GET NMI VECTOR
E1AA 6E 00              JMP    0,X

E1AC 20 40      INEEE   BRA    INEEE1

                *BYTE SEARCH ROUTINE
E1AE BD E0 47   SEARCH  JSR    BADDR     GET TOP ADDRESS
E1B1 FF A0 04           STX    ENDA
E1B4 BD E0 47           JSR    BADDR     GET BOTTOM ADDRESS
E1B7 BD E0 55           JSR    BYTE      GET BYTE TO SEARCH FOR
E1BA 16                 TAB
E1BB A6 00      OVE     LDA A  0,X
E1BD FF A0 0D           STX    XHI
E1C0 11                 CBA
E1C1 27 02              BEQ    PNT
E1C3 20 21              BRA    INCR1
E1C5 CE E1 9D   PNT     LDX    #MCL
E1C8 BD E0 7E           JSR    PDATA1
E1CB CE A0 0D           LDX    #XHI
E1CE 20 10              BRA    SKP0
                ***************************************************

                *GO TO USER PROGRAM ROUTINE
```

```
E1D0 3B         GOTO    RTI
E1D1 20 3A      OUTEEE  BRA    OUTEE1



                *SAVE IXR AND LOAD IXR WITH CORRECT
                *PORT NUMBER AND TEST FOR TYPE
E1D3 FF A0 10  SAVGET  STX    XTEMP     STORE INDEX REGISTER
E1D6 FE A0 0A  GETPT1  LDX    PORADD
E1D9 37         ISACIA  PSH B
E1DA E6 01              LDA B  1,X
E1DC E1 03              CMP B  3,X
E1DE 33                 PUL B
E1DF 39                 RTS
                ***************************************************

                *CONTINUATION OF SEARCH ROUTINE
E1E0 BD E0 C8  SKP0    JSR    OUT4HS
E1E3 FE A0 0D          LDX    XHI
E1E6 BC A0 04  INCR1   CPX    ENDA
E1E9 27 9E              BEQ    SWTL1
E1EB 08                 INX
E1EC 20 CD              BRA    OVE

E1EE 8D 06      INEEE1  BSR    INCH8     INPUT 8 BIT CHARACTER
E1F0 84 7F              AND A  #%01111111 GET RID OF PARITY BIT
E1F2 39                 RTS

E1F3 31         BILD    INS              FIX UP STACK WHEN USING
E1F4 31                 INS              BINARY LOADER ON SWTPC TAPES
E1F5 31                 INS

                *INPUT ONE CHAR INTO ACC B
E1F6 37         INCH8   PSH B            SAVE ACC B
E1F7 8D DA              BSR    SAVGET    SAVE IXR, GET PORT# AND TYPE
E1F9 26 28              BNE    IN1       INPUT FROM PIA IF NOT
E1FB 86 15              LDA A  #$15      RECONFIG FOR 8 BIT, 1 SB
E1FD A7 00              STA A  0,X
E1FF A6 00      ACIAIN  LDA A  0,X
E201 47                 ASR A
E202 24 FB              BCC    ACIAIN    NOT READY
E204 A6 01              LDA A  1,X       LOAD CHAR
E206 F6 A0 0C          LDA B  PORECH
E209 27 07              BEQ    ACIOUT    ECHO
E20B 20 11              BRA    RES       DON'T ECHO

                *OUTPUT ONE CHARACTER
E20D 37         OUTEE1  PSH B            SAVE ACC B
E20E 8D C3              BSR    SAVGET
E210 26 2E              BNE    IOUT

E212 C6 11      ACIOUT  LDA B  #$11
E214 E7 00              STA B  0,X
E216 E6 00      ACIOU1  LDA B  0,X
```

```
E218 57                 ASR B
E219 57                 ASR B
E21A 24 FA              BCC    ACIOU1    ACIA NOT READY
E21C A7 01              STA A  1,X       OUTPUT CHARACTER
E21E 33        RES      PUL B            RESTORE ACC B
E21F FE A0 10           LDX    XTEMP
E222 39                 RTS

               *PIA INPUT ROUTINE
E223 A6 00     IN1      LDA A  0,X       LOOK FOR START BIT
E225 2B FC              BMI    IN1
E227 8D 3A              BSR    DDL       DELAY HALF BIT TIME
E229 C6 04              LDA B  #4        SET DEL FOR FULL BIT TIME
E22B E7 02              STA B  2,X
E22D 58                 ASL B            SET UP CNTR WITH 8
E22E 8D 2A     IN3      BSR    DEL       WAIT ONE CHAR TIME
E230 0D                 SEC
E231 69 00              ROL    0,X
E233 46                 ROR A
E234 5A                 DEC B
E235 26 F7              BNE    IN3
E237 8D 21              BSR    DEL       WAIT FOR STOP BIT
E239 F6 A0 0C           LDA B  PORECH    IS ECHO DESIRED?
E23C 27 13              BEQ    IOUT2     ECHO
E23E 20 DE              BRA    RES       RESTORE IXR,ACCB
               *PIA OUTPUT ROUTINE
E240 8D 23     IOUT     BSR    DDL1      DELAY ONE HALF BIT TIME
E242 C6 0A              LDA B  #$A       SET UP COUNTER
E244 6A 00              DEC    0,X       SET START BIT
E246 8D 16              BSR    DE        START TIMER
E248 8D 10     OUT1     BSR    DEL       DELAY ONE BIT TIME
E24A A7 00              STA A  0,X       PUT OUT ONE DATA BIT
E24C 0D                 SEC
E24D 46                 ROR A            SHIFT IN NEXT BIT
E24E 5A                 DEC B            DECREMENT COUNTER
E24F 26 F7              BNE    OUT1      TEST FOR 0
E251 E6 02     IOUT2    LDA B  2,X       TEST FOR STOP BITS
E253 58                 ASL B            SHIFT BIT TO SIGN
E254 2A C8              BPL    RES       BRA FOR 1 STOP BIT
E256 8D 02              BSR    DEL       DELAY FOR STOP BITS
E258 20 C4              BRA    RES
E25A 6D 02     DEL      TST    2,X       IS TIME UP
E25C 2A FC              BPL    DEL
E25E 6C 02     DE       INC    2,X       RESET TIMER
E260 6A 02              DEC    2,X
E262 39                 RTS

E263 6F 02     DDL      CLR    2,X       HALF BIT DELAY
E265 8D F7     DDL1     BSR    DE
E267 20 F1              BRA    DEL


               *OPTIONAL PORT ROUTINE
E269 8D 83     OPTL     BSR    INEEE1
```

```
E26B 16                 TAB
E26C 7F A0 0B           CLR    PORADD+1  SET I/O ADDRESS FOR $8000
E26F FE A0 0A           LDX    PORADD
E272 8D 10              BSR    PIAINI    INITIALIZE PIA
E274 8D 07              BSR    PIAECH    SET ECHO
E276 CE E3 EF           LDX    #TABLE1   P, L OR E
E279 17                 TBA
E27A 7E E1 76           JMP    OVER      LOOK AT TABLE FOR E, L OR P

E27D 86 34      PIAECH  LDA A  #$34      SET DDR
E27F A7 03              STA A  3,X
E281 A7 02              STA A  2,X
E283 39         NOOPT   RTS

                *PIA INITIALIZATION ROUTINE
E284 6C 00      PIAINI  INC    0,X       SET DDR
E286 86 07              LDA A  #$7
E288 A7 01              STA A  1,X
E28A 6C 00              INC    0,X
E28C A7 02              STA A  2,X
E28E 39                 RTS

                *MINIFLOPPY DISK BOOT
E28F 7F 80 14   DISK    CLR    $8014
E292 8D 2E              BSR    DELAY
E294 C6 0B              LDA B  #$0B
E296 8D 25              BSR    RETT2
E298 E6 04      LOOP1   LDA B  4,X
E29A C5 01              BIT B  #1
E29C 26 FA              BNE    LOOP1
E29E 6F 06              CLR    6,X
E2A0 8D 1D              BSR    RETURN
E2A2 C6 9C              LDA B  #$9C
E2A4 8D 17              BSR    RETT2
E2A6 CE 24 00           LDX    #$2400
E2A9 C5 02      LOOP2   BIT B  #2
E2AB 27 06              BEQ    LOOP3
E2AD B6 80 1B           LDA A  $801B
E2B0 A7 00              STA A  0,X
E2B2 08                 INX
E2B3 F6 80 18   LOOP3   LDA B  $8018
E2B6 C5 01              BIT B  #1
E2B8 26 EF              BNE    LOOP2
E2BA 7E 24 00           JMP    $2400
E2BD E7 04      RETT2   STA B  4,X
E2BF 8D 00      RETURN  BSR    RETT1
E2C1 39         RETT1   RTS

                *GENERAL PURPOSE DELAY LOOP
E2C2 CE FF FF   DELAY   LDX    #$FFFF
E2C5 09         DELAY1  DEX
E2C6 8C 80 14           CPX    #$8014    STOP AT 8014
E2C9 26 FA      DUM     BNE    DELAY1
E2CB 39                 RTS
```

```
               *CLRAR SCREEN FOR CT-1024 TYPE TERMINALS
E2CC CE E0 09  CLEAR   LDX    #CURSOR
E2CF BD E0 7E          JSR    PDATA1
E2D2 8D F1             BSR    DELAY1    DELAY
E2D4 BD E3 47  RDOFF1  JSR    RDOFF
E2D7 20 58             BRA    C4

               *BREAKPOINT ENTERING ROUTINE
E2D9 CE E1 23  BREAK   LDX    #SF0
E2DC BC A0 12          CPX    SWIJMP    BREAKPOINTS ALREADY IN USE?
E2DF 27 1A             BEQ    INUSE
E2E1 08                INX
E2E2 8D 32     BREAK0  BSR    STO1
E2E4 BD E0 47          JSR    BADDR
E2E7 FF A0 14          STX    BKPT
E2EA A6 00             LDA A  0,X
E2EC B7 A0 16          STA A  BKLST
E2EF 86 3F             LDA A  #$3F
E2F1 A7 00             STA A  0,X
E2F3 CE E1 23          LDX    #SF0
E2F6 8D 1E             BSR    STO1
E2F8 7E E1 6B          JMP    CONTR1
E2FB FE A0 14  INUSE   LDX    BKPT
E2FE B6 A0 16          LDA A  BKLST
E301 A7 00             STA A  0,X
E303 CE E1 24          LDX    #SFE1
E306 20 DA             BRA    BREAK0

E308 B7 A0 43  SWISET  STA A  STACK+1   FIX POWER UP INTERRUPT
E30B FE A0 12          LDX    SWIJMP
E30E 8C E1 23          CPX    #SF0
E311 27 06             BEQ    STORTN
E313 CE E1 24  STO     LDX    #SFE1
E316 FF A0 12  STO1    STX    SWIJMP
E319 39        STORTN  RTS

E31A 8D 5A     PUNCH1  BSR    PUNCH
E31C 20 0F             BRA    POFC4

               *FORMAT END OF TAPE WITH PGM. CTR. AND S9
E31E CE A0 49  PNCHS9  LDX    #$A049
E321 FF A0 04          STX    ENDA
E324 09                DEX
E325 8D 52             BSR    PUNCH2
E327 CE E1 90          LDX    #S9
E32A BD E0 7E  PDAT    JSR    PDATA1
E32D 8D 24     POFC4   BSR    PNCHOF
E32F 8D 91             BSR    DELAY
E331 7E E1 52  C4      JMP    CONTRL

E334 73 A0 0C  RDON    COM    PORECH    DISABLE ECHO FOR ACIA
E337 86 11             LDA A  #$11      RON CHAR.
```

```
E339 C6 20              LDA B  #$20      STROBE CHAR
E33B 8D 1A              BSR    STROBE
E33D BD E1 D9           JSR    ISACIA    CHECK TO SEE IF PIA
E340 27 04              BEQ    RTNN
E342 86 3C              LDA A  #$3C      DISABLE PIA ECHO IF PIA
E344 A7 03              STA A  3,X
E346 39         RTNN    RTS

E347 86 13      RDOFF   LDA A  #$13      TURN READER OFF
E349 C6 10              LDA B  #$10
E34B 20 0A              BRA    STROBE

E34D 86 12      PNCHON  LDA A  #$12
E34F C6 04              LDA B  #4
E351 20 04              BRA    STROBE

E353 86 14      PNCHOF  LDA A  #$14
E355 C6 08              LDA B  #$8

                *PIA STROBING ROUTINE FOR PUNCH/READ ON/OFF
E357 BD E0 75   STROBE  JSR    OUTCH
E35A BD E1 D6           JSR    GETPT1
E35D 27 16              BEQ    RTN1
E35F 86 02              LDA A  #2
E361 CA 01              ORA B  #1
E363 8D 0C              BSR    STR2
E365 8D 08              BSR    STR1
E367 86 02              LDA A  #2
E369 C6 01              LDA B  #1
E36B E7 00              STA B  0,X
E36D 8D 02              BSR    STR2
E36F 86 06      STR1    LDA A  #6
E371 A7 01      STR2    STA A  1,X
E373 E7 00              STA B  0,X
E375 39         RTN1    RTS

                *PUNCH FROM BEGINNING ADDRESS (BEGA) THRU
                *ENDING ADDRESS (ENDA)
E376 FE A0 02   PUNCH   LDX    BEGA
E379 FF A0 44   PUNCH2  STX    TW
E37C 8D CF              BSR    PNCHON
E37E B6 A0 05   PUN11   LDA A  ENDA+1
E381 B0 A0 45           SUB A  TW+1
E384 F6 A0 04           LDA B  ENDA
E387 F2 A0 44           SBC B  TW
E38A 26 04              BNE    PUN22
E38C 81 10              CMP A  #16
E38E 25 02              BCS    PUN23
E390 86 0F      PUN22   LDA A  #15
E392 8B 04      PUN23   ADD A  #4
E394 B7 A0 47           STA A  BYTECT
E397 80 03              SUB A  #3
E399 B7 A0 46           STA A  TEMP
                *PUNCH C/R L/F NULLS S1
```

```
E39C CE E1 93           LDX    #MTAPE1
E39F BD E0 7E           JSR    PDATA1
E3A2 5F                 CLR B
                *PUNCH FRAME COUNT
E3A3 CE A0 47           LDX    #BYTECT
E3A6 8D 24              BSR    PUNT2     PUNCH 2 HEX CHARACTERS
                *PUNCH ADDRESS
E3A8 CE A0 44           LDX    #TW
E3AB 8D 1F              BSR    PUNT2
E3AD 8D 1D              BSR    PUNT2
                *PUNCH DATA
E3AF FE A0 44           LDX    TW
E3B2 8D 18      PUN32   BSR    PUNT2     PUNCH ONE BYTE
E3B4 7A A0 46           DEC    TEMP
E3B7 26 F9              BNE    PUN32
E3B9 FF A0 44           STX    TW
E3BC 53                 COM B
E3BD 37                 PSH B
E3BE 30                 TSX
E3BF 8D 0B              BSR    PUNT2     PUNCH CHECKSUM
E3C1 33                 PUL B            RESTORE STACK
E3C2 FE A0 44           LDX    TW
E3C5 09                 DEX
E3C6 BC A0 04           CPX    ENDA
E3C9 26 B3              BNE    PUN11
E3CB 39         RTN5    RTS

                *PUNCH 2 HEX CHAR, UPDATE CHECKSUM
E3CC EB 00      PUNT2   ADD B  0,X
E3CE 7E E0 BF           JMP    OUT2H     OUTPUT 2 HEX CHAR AND RTS

                *COMMAND TABLE
E3D1 47         TABLE   FCB    'G        GOTO
E3D2 E1 D0              FDB    GOTO
E3D4 5A                 FCB    'Z        GOTO PROM
E3D5 C0 00              FDB    PROM
E3D7 4D                 FCB    'M        MEMORY EXAM AND CHANGE
E3D8 E0 88              FDB    CHANGE
E3DA 46                 FCB    'F        BYTE SEARCH
E3DB E1 AE              FDB    SEARCH
E3DD 52                 FCB    'R        REGISTER DUMP
E3DE E1 30              FDB    PRINT
E3E0 4A                 FCB    'J        JUMP
E3E1 E0 05              FDB    JUMP
E3E3 43                 FCB    'C        CLEAR SCREEN
E3E4 E2 CC              FDB    CLEAR
E3E6 44                 FCB    'D        DISK BOOT
E3E7 E2 8F              FDB    DISK
E3E9 42                 FCB    'B        BREAKPOINT
E3EA E2 D9              FDB    BREAK
E3EC 4F                 FCB    'O        OPTIONAL PORT
E3ED E2 69              FDB    OPTL
E3EF 50         TABLE1  FCB    'P        ASCII PUNCH
E3F0 E3 1A              FDB    PUNCH1
```

```
E3F2 4C                 FCB    'L        ASCII LOAD
E3F3 E0 0C              FDB    LOAD
E3F5 45         TABEND  FCB    'E        END OF TAPE
E3F6 E3 1E              FDB    PNCHS9

E3F8                    ORG    $E3F8
E3F8 E0 00              FDB    IRQV      IRQ VECTOR
E3FA E1 8B              FDB    SFE       SOFTWARE INTERRUPT
E3FC E1 A7              FDB    NMIV      NMI VECTOR
E3FE E0 D0              FDB    START     RESTART VECTOR

A048                    ORG    $A048
A048 E0 D0              FDB    START
                        END

ERROR(S) DETECTED
```

```
   SYMBOL TABLE:

ACIAIN E1FF   ACIOU1 E216   ACIOUT E212   AL1    E101   AL2    E117
AL3    E0D5   AL4    E0E5   ANOTH  E09B   BADDR  E047   BEGA   A002
BILD   E1F3   BKLST  A016   BKPT   A014   BREAK  E2D9   BREAK0 E2E2
BYTE   E055   BYTE1  E057   BYTECT A047   C1     E085   C3     E0FC
C4     E331   CHA51  E08A   CHANGE E088   CKSM   A00F   CLEAR  E2CC
CONTR1 E16B   CONTRL E152   CTLPOR 8004   CURSOR E009   DDL    E263
DDL1   E265   DE     E25E   DEL    E25A   DELAY  E2C2   DELAY1 E2C5
DISK   E28F   DUM    E2C9   EIA5   E1A5   ENDA   A004   EOE3   E0E3
GETPT1 E1D6   GOTO   E1D0   IN1    E223   IN1HG  E0BE   IN3    E22E
INCH   E078   INCH8  E1F6   INCR1  E1E6   INEEE  E1AC   INEEE1 E1EE
INHEX  E0AA   INHEX1 E0AC   INUSE  E2FB   IOUT   E240   IOUT2  E251
IRQ    A000   IRQV   E000   ISACIA E1D9   JUMP   E005   LOAD   E00C
LOAD11 E02B   LOAD15 E03B   LOAD19 E040   LOAD21 E044   LOAD3  E00F
LOOK   E173   LOOP1  E298   LOOP2  E2A9   LOOP3  E2B3   MCL    E19D
MCLOFF E19C   MTAPE1 E193   NMI    A006   NMIV   E1A7   NOOPT  E283
OPTL   E269   OUT1   E248   OUT2H  E0BF   OUT2HA E0C1   OUT2HS E0CA
OUT4HS E0C8   OUTCH  E075   OUTEE1 E20D   OUTEEE E1D1   OUTHL  E067
OUTHR  E06B   OUTS   E0CC   OVE    E1BB   OVER   E176   PDAT   E32A
PDATA1 E07E   PDATA2 E07B   PIAECH E27D   PIAINI E284   PNCHOF E353
PNCHON E34D   PNCHS9 E31E   PNT    E1C5   POF1   E165   POFC4  E32D
PORADD A00A   PORECH A00C   PRINT  E130   PROM   C000   PUN11  E37E
PUN22  E390   PUN23  E392   PUN32  E3B2   PUNCH  E376   PUNCH1 E31A
PUNCH2 E379   PUNT2  E3CC   RDOFF  E347   RDOFF1 E2D4   RDON   E334
RES    E21E   RETT1  E2C1   RETT2  E2BD   RETURN E2BF   RTN1   E375
RTN5   E3CB   RTNN   E346   S9     E190   SAVGET E1D3   SEARCH E1AE
SF0    E123   SFE    E18B   SFE1   E124   SK1    E0DE   SK3    E181
SKP0   E1E0   SP     A008   STACK  A042   START  E0D0   STO    E313
STO1   E316   STORTN E319   STR1   E36F   STR2   E371   STROBE E357
SWIJMP A012   SWISET E308   SWTCTL E14A   SWTL1  E189   TABEND E3F5
TABLE  E3D1   TABLE1 E3EF   TEMP   A046   TW     A044   XHI    A00D
XLOW   A00E   XTEMP  A010
```

## SWTPC 6800 Short Memory Address Convergence MEMCON-3
## MODIFIED FOR MIKBUG® OR SWTBUG®

This Memory Convergence diagnostic is one designed to check for and locate address convergence problems in SWTPC 6800 Computer System memory boards. The program itself uses $56_{10}$ words and is meant to be loaded within the 128 word RAM used by the SWTBUG® operating system on the MPA Microprocessor System board making the program independent of the MPM RAM memory. The diagnostic may be loaded from either tape or from the terminal instruction by instruction using SWTBUG® starting from address $A014_{16}$ thru $A033_{16}$ and then from $A048_{16}$ thru $A051_{16}$. The program must be loaded in two parts to avoid interfering with the system's pushdown stack. The section of memory to be tested is set by loading the most significant byte of the lower memory address into $A002_{16}$ the least significant byte of the lower memory address into $A003_{16}$ the most significant byte of the upper memory address into A00416 and the least significant byte of the upper memory address into $A005_{16}$ using SWTBUG® just as is done for SWTBUG® punch routine. The lower and upper addresses are inclusive and may be any addresses between $0000_{16}$ and $FFFF_{16}$ with the only requirement that the lower address be less than or equal to the upper address. Since addresses $A05F_{16}$ thru $A07F_{16}$ of the SWTBUG® RAM are still available for program use, the diagnostic may run on these locations just to make sure the diagnostic itself is functioning correctly, Since the program counter is set when the program 1s initially loaded, the routine is initiated by typing **G** for “Go To User Program”. Once initiated, the program can be stopped only by depressing the “RESET” button. The program may then be restarted after setting the program counter to $A014_{16}$ at A048 and A049.

The test sequence starts by loading a continuous stream of 256 sequential binary numbers from the low memory address to the high memory address, inclusive. It then goes back and sequentially reads the data in each of the locations and compares it to what actually should be there. If it finds any discrepancies within the memory cycle, one X is printed and the cycle is restarted, otherwise a # is printed to indicate successful cycle completion. Since the actual location of any detected errors does not point to the source of the problem, no provision is made for indicating the addresses of detected errors. It must also be noted that the program is not 100% effective. It would be possible to set bits in multiple locations that coincidentally would have been set anyway. However, each cycle puts different data in each memory location, so the chances of a missed problem are reduced. The program loops forever and may be exited when desired by depressing the “RESET” switch which loads the SWTBUG® control program.

If you wish to eliminate the cyclic printout of the "# " sign you can do so by changing the data in address locations A059, A05A and A05B to NOP instructions ($01_{16}$) using SWTBUG®. This way you only get a printout of the error cycles, if any.

MIKBUG® is a registered trademark of Motorola Inc.
SWTBUG® is a registered trademark of Southwest Technical Products Corporation.

```
                          NAM    MEMCON3
                  *SHORT MEMORY ADDRESS CONVERGENCE TEST
                  *FOR SWTPC 6800 COMPUTER SYSTEMS
                  *MIKBUG AND SWTBUG COMPATIBLE

A002              LOMEM   EQU    $A002     STARTING ADDRESS
A004              HIMEM   EQU    $A004     ENDING ADDRESS
E1D1              OUTEEE  EQU    $E1D1     CHARACTER OUTPUT

A014                      ORG    $A014
A014 F7 A0 5F     START   STA B  BSTORE    STORE ACCCB
A017 FE A0 02             LDX    LOMEM     LOAD LOW MEMORY ADDRESS
A01A E7 00        LOOP1   STA B  0,X
A01C BC A0 04             CPX    HIMEM     END OF MEMORY?
A01F 27 04                BEQ    CHECK     CHECK IF FINISHED
A021 08                   INX
A022 5C                   INC B
A023 20 F5                BRA    LOOP1
A025 F6 A0 5D     CHECK   LDA B  BSTORE    CHECKS ALL LOCATIONS FOR CORR
A028 FE A0 02             LDX    LOMEM
A02B E1 00        LOOP2   CMP B  0,X
A02D 26 21                BNE    ERROR
A02F BC A0 04             CPX    HIMEM
A032 20 16                BRA    JUMP

A048                      ORG    $A048
A048 A0 14                FDB    START
A04A 27 0B        JUMP    BEQ    CYCLE
A04C 08                   INX
A04D 5C                   INC B
A04E 20 DB                BRA    LOOP2
A050 86 58        ERROR   LDA A  #'X       PRINT X IF ERROR FOUND
A052 BD E1 D1             JSR    OUTEEE
A055 20 BD                BRA    START     START OVER
A057 86 23        CYCLE   LDA A  #'#       PRINT # FOR CORRECT CYCLE
A059 BD E1 D1             JSR    OUTEEE
A05C 5A                   DEC B
A05D 20 B5                BRA START
A05F 00           BSTORE  FCB    0
                          END
```

## Dual Address Memory Test CDAT
**By John Christensen**

The CDAT memory diagnostic can be used to help locate memory problems in a SWTPC 6800 computer system that MEMCON and ROBIT may miss. The program itself resides entirely within the 128 byte SWTBUG® RAM. The program must be loaded in two parts to avoid interfering with the systems push down stack. The contiguous section of memory to be tested is set by loading the most significant byte of the lower memory address into A002, the least significant byte into A003, the most significant byte of the upper memory address in A004 and its least significant byte in A005. The low address must be less than or equal to the upper address.

The test starts from the low address and writes a 00 into all memory up to the high address. An FF is then written into the first address and all other locations are checked to be sure they contain 00. If all are OK the FF is replaced with a 00 and an FF is written in the next memory location. This pattern continues until all memory is checked or an error is found. If the computer returns to SWTBUG®, then no errors were found.

If the program displays a register dump then a problem was discovered on the board. The register dump should look similar to the following:

```
      ADDRESS    ERROR MSG.
            V       V
F3 00 FF 0400    A079 A042.
```

The important parts of the dump are the ADDRESS and the ERROR MSG. areas, as denoted above. The error messages are interpreted as follows:

| | |
|---|---|
| A077 | Error on initial test pattern (can't write 0's into mem.) |
| A078 | Error on second test pattern (can't write FF's into mem.) |
| A079 | Dual address error low |
| A07A | Dual address error high |

If a dual address error is found then writing into one memory location affects another. For example, if ADDRESS = 0400 and A016 contains 0410 then writing into 0400 will change the contents of 0418 or vice-versa. (A014 is a temporary index register storage location within the program that you can compare with ADDRESS in the register dump to see which two memory locations caused the error). The IC assignments table included with the memory board instructions can then be used to help locate the problem by comparing the bit pattern of the locations in error.

The CDAT program takes some time to run, so run the diagnostic over only one complete board at a time.

| MEM. SIZE | APPROX. RUN TIME |
|---|---|
| 1K | 29 sec. |
| 2K | 1 min. 53 sec. |
| 3K | 4 min. 13 sec. |
| 4K | 7 min. 29 sec. |
| 8K | more than 30 min. |

```
                        NAM    CDAT-2
                *MEM DIAGNOSTIC (JOHN CHRISTENSEN'S)
                *MODIFIED FOR MIKBUG AND SWTBUG OPERATION
E0E3            CONTRL  EQU    $E0E3
A002                    ORG    $A002
A002            LOTEMP  RMB    2         STARTING ADDRESS
A004            HITEMP  RMB    2         ENDING ADDRESS
A014                    ORG    $A014
A014            IXRTMP  RMB    2         IXR TEMPORARY STORAGE
A016 FE A0 02   START   LDX    LOTEMP
A019 B6 A0 7E           LDA A  INIPAT
A01C A7 00      LOOP1   STA A  0,X
A01E A1 00              CMP A  0,X
A020 26 55              BNE    ERPNT1
A022 BC A0 04           CPX    HITEMP
A025 27 03              BEQ    LOAPAT
A027 08                 INX
A028 20 F2              BRA    LOOP1
A02A FE A0 02   LOAPAT  LDX    LOTEMP
A02D F6 A0 7F           LDA B  TESPAT
A030 E7 00      LOOP4   STA B  0,X
A032 20 16              BRA    CHECK
A048                    ORG    $A048
A048 A0 16              FDB    START
A04A E1 00      CHECK   CMP B  0,X
A04C 26 2A              BNE    ERPNT2
A04E FF A0 14   CHKLOW  STX    IXRTMP
A051 BC A0 02   LOOP2   CPX    LOTEMP
A054 27 07              BEQ    CHCKHI
A056 09                 DEX
A057 A1 00              CMP A  0,X
A059 26 1E              BNE    ERPNT3
A05B 20 F4              BRA    LOOP2
A05D FE A0 14   CHCKHI  LDX    IXRTMP
A060 BC A0 04           CPX    HITEMP
A063 27 16              BEQ    END
A065 08         LOOP3   INX
A066 A1 00              CMP A  0,X
A068 26 10              BNE    ERPNT4
A06A BC A0 04           CPX    HITEMP
A06D 26 F6              BNE    LOOP3
A06F FE A0 14   RESTRE  LDX    IXRTMP
A072 A7 00              STA A  0,X
A074 08                 INX
A075 20 B9              BRA    LOOP4
A077 3F         ERPNT1  SWI              ERROR ON INITIAL PATTERN
A078 3F         ERPNT2  SWI              ERROR ON TEST PATTERN
A079 3F         ERPNT3  SWI              DUAL ADDRESS ERROR LOW
A07A 3F         ERPNT4  SWI              DUAL ADDRESS ERROR LO
A07B 7E E0 E3   END     JMP    CONTRL

A07E 00         INIPAT  FCB    0
A07F 00         TESPAT  FCB    0
                        END
```

## SWTPC 6800 Rotating Bit RAM Memory Diagnostic ROBIT-2
## Modified for MIKBUG® or SWTBUG® use

This rotating bit memory diagnostic is designed to check for and locate memory retaining problems in SWTPC 6800 Computer System memory boards. The program itself uses 8510 words and is meant to be loaded within the 128 word RAM used by the SWTBUG® operating system on the MP-A Microprocessor System board. This makes the program independent of external RAM memory. The diagnostic may be loaded from either tape or from the terminal instruction by instruction using SWTBUG® starting from address $A014_{16}$ thru $A067E_{16}$. The program must be loaded in two parts to avoid interfering with the system's pushdown stack. The contiguous section of memory to be tested is set by loading the most significant byte of the lower memory address into $A002_{16}$, the least significant byte of the lower memory address into $A003_{16}$, the most significant byte of the upper memory address into $A004_{16}$, and the least significant byte of the upper memory address into $A005_{16}$ using SWTBUG® just as is done for the SWTBUG® punch routine. The lower and upper addresses are inclusive and may be any addresses between $0000_{16}$ and $FFFF_{16}$ with the only requirement being that the lower address be less than or equal to the upper address. Since the program counter is set when the program is initially loaded, the routine is initiated after loading by typing G for "Go To User's Program". Once initiated, the program may then be restarted after setting the program counter to $A014_{16}$ at A048 and A049.

The test sequence starts from the lower address and loads that address with a binary 0000 0001 or $01_{16}$ The data in this location is then read and verified. If accurate the "one" bit is shifted left to form a binary 0000 0010 or $02_{16}$ and is then again tested. This shift left sequence continues until a binary 1000 0000 or $80_{16}$ has been loaded and verified, at which time the entire sequence is repeated at the next sequential memory address. This sequence continues until the selected upper memory address is reached. The program then prints a "+" on the control terminal to indicate cycle completion and proceeds to repeat Itself. The program loops forever and may be exited when desired by depressing the "RESET" switch which loads the SWTBUG® control program. When an error is detected, the memory address followed by what data should have been followed by what the memory data was, are printed out on the control terminal in hexadecimal (base 16) form. Example:

```
$0110 02 00
```

When converted to binary this means that when address 0110, which is located in the first 1,024 words of RAM memory, was loaded with a binary 0000 0010 it was read back as containing a binary 0000 0000 which indicates a possible problem in the 21 bit memory chip in the lower 1,024 words of memory or a possible problem in the 21 bit of the memory board data transceiver or a variety of other possibilities. The best way to tell for sure is to look for a pattern in the indicated errors. Take note that once one bit error has been located at a specific memory address, the one error is printed in the form shown above and the program increments to the next address without searching for more errors in the already defective address.

If you wish to eliminate the cyclic printout of the "+" sign you can do so by changing the data in address locations A076, A077 and A078 to NOP instructions ($01_{16}$) using SWTBUG. This way you only get a printout of the error locations; that is if there are any. The running time of this program is very fast. It will cycle thru 2,048 words of memory in less than one second.

```
                        NAM    ROBIT-2
                *PRTATING BIT MEMORY TEST FOR MIKBUG
                *OR SWTBUG 6800 COMPUTER SYSTEM

A002            LOTEMP  EQU    $A002
A004            HITEMP  EQU    $A004
E07E            PDATA1  EQU    $E07E
E0C8            OUT4HS  EQU    $E0C8
E0CA            OUT2HS  EQU    $E0CA
E19D            MCL     EQU    $E19D
E1D1            OUTEEE  EQU    $E1D1

A014                    ORG    $A014
A014 FE A0 02   START   LDX    LOTEMP
A017 86 01      LODREG  LDA A  #1         STORE 1 IN MEMORY
A019 A7 00              STA A  0,X
A01B A1 00              CMP A  0,X        WAS 1 WRITTEN
A01D 26 0D              BNE    ERRPNT
A01F 48         LOOP1   ASL A
A020 68 00              ASL    0,X
A022 A1 00              CMP A  0,X
A024 26 06              BNE    ERRPNT
A026 81 80              CMP A  #$80       SHIFT UNTIL 80
A028 26 F5              BNE    LOOP1
A02A 20 3F              BRA    INCR1
A02C FF A0 7B   ERRPNT  STX    INXMSB
A02F CE E1 9D           LDX    #MCL       PRINT C/R L/F
A032 20 16              BRA    SKIP1

A048                    ORG    $A048
A048 A0 14              FDB    START
A04A B7 A0 7D   SKIP1   STA A  ACCA
A04D BD E0 7E           JSR    PDATA1
A050 CE A0 7B           LDX    #INXMSB    LOAD ERROR ADDRESS
A053 BD E0 C8           JSR    OUT4HS
A056 CE A0 7D           LDX    #ACCA
A059 BD E0 CA           JSR    OUT2HS     OUTPUT WHAT SHOULD BE STORED
A05C FE A0 7B           LDX    INXMSB
A05F BD E0 CA           JSR    OUT2HS
A062 CE E1 9D           LDX    #MCL
A065 BD E0 7E           JSR    PDATA1
A068 FE A0 7B           LDX    INXMSB
A06B BC A0 04   INCR1   CPX    HITEMP     COMPARE TO END ADDRESS
A06E 27 03              BEQ    FINISH
A070 08                 INX
A071 20 A4              BRA    LODREG
A073 B6 A0 7E   FINISH  LDA A  FLAG
A076 BD E1 D1           JSR    OUTEEE
A079 20 99              BRA    START
A07B            INXMSB  RMB    1
A07C            INXLSB  RMB    1
A07D            ACCA    RMB    1
A07E 2B         FLAG    FCB    '+
                END
```

## SUMTEST - 2 -- Address Variable Memory Test

By Chris Courtney, RESPCO Inc.

The SUMTEST memory diagnostic can be used to supplement the ROBIT, MEMCON and CDAT tests. The program itself resides entirely within the 128 byte MIKBUG®/SWTBUG® RAM and must be loaded in two parts to avoid interfering with the system's push down stack. The contiguous section of memory to be tested is set by loading the most significant byte of the lower memory address into A002, the least significant byte into A003, the most significant byte of the upper memory address in A004 and its least significant byte in A005. When inputting the upper address use 1+ (upper location you want to check). TO check the entire lowest 4K board, for example, you would use an upper memory address of 1000, not OFFF. When loading the diagnostic the program counter is automatically set to the starting address of the program. Typing G will initiate the program.

If no errors are found in the memory being checked a + will be displayed on the screen. To completely test an area of memory the diagnostic must be allowed to run until 256 +'s have been displayed on the screen. Each time a + is displayed on the screen SUMTEST has successfully cycled through memory storing and reading a different pattern. If an error is detected the output wilt be similar to the following:

```
$G +++++
$06 20 16A0
(PATTERN #) (ERRANT BITS) (ADDRESS)
```

An error message such as this says that SUMTEST cycled thru memory five times without error, but on the sixth try a pattern was used that detected an error. The 06 tells what pattern number SUMTEST was working on when the error was detected. The 20 tells which bits were in error. $20_{16}$ converted to binary is 00100000 - the location of the 1 is the bit that is in error, in this case bit 5. Bit numbers start from 0 as shown:

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| $20_{16}$ = | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | |
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | BIT# |

The 16A0 is the address where the error was detected. This address may not store a particular number or possibly writing into another address, such as 16B0, changed the contents of 16A0.

The IC assignment table supplied with the memory board should be used to help locate the problem. In the above case on an MP-8M 8K memory board the bit # 5 IC in the upper 4K of memory should be suspected.

Be sure to re-load the program before testing another area of memory. SUMTEST runs fast enough that it can be used on an entire 32K system in an acceptable amount of time.

```
                         NAM    SUMTEST2
               *IMPROVED MEMORY TEST FOR SWTPC 6800
               *BYTES STORED IN MEMORY ARE THE SUM OF THE
               *MSB AND LSB OF THE MEMORY POINTER, THEREFORE
               *ADJACENT MEMORY LOCATIONS AND ADJACENT
               *PAGES CONTAIN UNIQUE CONTENTS
               *INITIALIZE LOWEST MEMORY ADDRESS IN LOTEMP
               *AND HIGHEST MEMORY ADDRESS+1 IN HITEMP
               *MODIFIED FOR MIKBUG OR SWTBUG

A002                     ORG    $A002
A002           LOTEMP    RMB    2
A004           HITEMP    RMB    2
E07E           PDATA1    EQU    $E07E
E1D1           OUTEEE    EQU    $E1D1
E0CA           OUT2HS    EQU    $E0CA
E0C8           OUT4HS    EQU    $E0C8
E19D           MCL       EQU    $E19D

A014                     ORG    $A014
A014 00        CTR       FCB    0          PASS COUNTER
A015 00        STORE     FCB    0          BIT MISMATCHED
A016 00        INXMSB    FCB    0
A017 00        INXLSB    FCB    0
A018 FE A0 02  START     LDX    LOTEMP
A01B 8D 39     LOOP1     BSR    INCRX      INCREMENT INDEX
A01D A7 00               STA A  0,X
A01F 08                  INX
A020 BC A0 04            CPX    HITEMP     END OF MEMORY?
A023 26 F6               BNE    LOOP1
A025 FE A0 02            LDX    LOTEMP
A028 8D 2C     LOOP2     BSR    INCRX
A02A A8 00               EOR A  0,X
A02C 26 35               BNE    ERROR
A02E 08        RETURN    INX
A02F BC A0 04            CPX    HITEMP
A032 20 16               BRA    SKIP1      BRANCH AROUND HOLE

A048                     ORG    $A048
A048 A0 18               FDB    START
A04A 26 DC     SKIP1     BNE    LOOP2
A04C 86 2B               LDA A  #$2B
A04E BD E1 D1            JSR    OUTEEE
A051 7C A0 14            INC    CTR
A054 20 C2               BRA    START
A056 FF A0 16  INCRX     STX    INXMSB
A059 B6 A0 16            LDA A  INXMSB
A05C BB A0 17            ADD A  INXLSB     ADD IN ADDR LSB
A05F BB A0 14            ADD A  CTR        ADD IN COUNTER
A062 39                  RTS
A063 B7 A0 15  ERROR     STA A  STORE      STORE ERRANT BIT
A066 CE E1 9D            LDX    #MCL
A069 BD E0 7E            JSR    PDATA1     DO C/R L/F
A06C CE A0 14            LDX    #CTR
A06F BD E0 CA            JSR    OUT2HS     COUNTER, IN HEX
A072 BD E0 CA            JSR    OUT2HS     ERRANT BITS, IN HEX
A075 BD E0 C8            JSR    OUT4HS     ADDRESS, IN HEX
A078 FE A0 16            LDX    INXMSB
A07B 20 B1               BRA    RETURN
```