M-51

A Simple Debugging Monitor for TAPR's TUC-52
Operations Guide
Issue 1 – January 21, 1998
(For M-51 Ver 980121-3.00a)

Tucson Amateur Packet Radio
8987-309 E. Tanque Verde Rd., #337
Tucson, AZ 85749-9399
Phone: +1 940-383-0000
Fax: +1 940-566-2544
tapr@tapr.org

1. INTRODUCTION

This document describes a version of an 8051 monitor called M-51 that has been customized to run on TAPR's (Tucson Amateur Packet Radio) TUC-52 board. TUC-52 is a universal controller that TAPR makes available for ham radio applications.

The M-51 monitor is like most other simple debug monitor for a single board, stand alone, microcomputer system. A somewhat unusual feature for this simple monitor is that is allows the user to single-step through user programs executing from double mapped (program and data) RAM memory.

This manual describes how to use the monitor, what's needed for hardware, and different options for recovering the lost P0 and P2 ports, which are not available for I/O since they are used to create the address and data bus.

As with all instruction manuals, there may be some errors or sections that do not provide satisfactory explanations. If you find errors or insufficient information, please contact me about these problems and I will work to correct them.

2. OVERVIEW

The monitor offers many commands to help debug a microcomputer program or system. All commands consist of one , two or three letter commands. Commands can be followed by 0 to 16 arguments. All command line arguments are numeric values in base 16 (hexadecimal). With the exception of the GO command, all arguments are separated from the command and other arguments by white-space characters. Arguments shown in angle brackets <LIKE-THIS> are optional. Within this document specific keyboard presses are shown underlined and special characters are shown in braces. For example a space would be shown as {sp}, a return would be shown as {rtn}, an escape would be shown as {esc} and a period when used in a command line would be shown as {.}. All command input lines must be ended with a {rtn) (on modern PCs the rtn key is labeled <-Enter)however some commands will accept a space as a command to

move to the next time (i.e., if considering a substitution for the data at address 12F3, pressing space will cause the monitor to move to address 12F4 and continue with the substitution process. An example of a typical command would be:

```
m51-> dx 8010 81ff  {rtn}
```

2.1 DEFINITIONS AND TERMS

For brevity, I use some abbreviations in this manual. Code space or Program Memory is abbreviated as PM. The single character "p" is used to preface any addresses that refer to PM. Internal Data Memory is abbreviated as IDM. The single character "i" is used to preface any addresses that refer to IDM. External Data Memory is abbreviated as XDM. The single character "x" is used to preface any addresses that refer to XXDM. Special Function Registers are abbreviated as SFR. The single character "f" is used to preface any addresses that refer to SFRs.

2.2 COMMAND SUMMARY

Table 1 below shows a summary for each command. The next section provides the details of each command.

Table 1 -- Command Summary

Command	Description	Usage
BLAST FROM	Blast from XDM	BF adr data
BLAST TO	Blast to XDM	BT adr data
CHECK PROG MEM	Checksum of PM	CP first last
CHECK XDATA	Checksum of XDM	CX first last
DISPLAY PGM	Display Program Memory	DP first last
DISPLAY XDM	Display External Data Memory	DX first last
DISPLAY IDM	Display Internal Data Memory	DI first last
DISPLAY SFR	Display SF Register Contents	DF first last
FIND	Find Bytes in XDM	F data0 <data1> ... <data15>
GO	Go (with opt breakpoints)	G<adr> <bp1> <bp2>
HELP	Print Command Summary	H
JAM	Jam Value to XDM	J first last data
LIST	List Data On Stack	L
MOVE	Move Block of Data in XDM	M first last to
NEXT	Single Step, even through calls	N
POV	Power On Vector	P start
RXINTEL	Receive Intel Hex Code	R offset
VFINTEL	Verify Intel Hex Code	V offset
SUBSTITUTE XDM	Substitute XDM	SX adr {sp/ret}
SUBSTITUTE IDM	Substitute IDM	SI adr {sp/ret}
SUBSTITUTE SFR	Substitute SFR	SF adr {sp/ret}
EXAMINE	Examine Registers	Xrx
ZNEXT	Single Step, skip calls	Z
TEST	Tests various hardware elements	Tn <arg0> … <argN>

2.3 MONITOR COMMANDS IN DETAIL

This section outlines each monitor command in more detail.

2.3.1 BLAST (BT, BF)

The BLAST command is used to repetitively read or write to an XDM location. The BLAST command is followed by a single letter (F for FROM and T for TO) which is followed by one or two arguments. To get data repetitively from a location in data memory enter "BF adr" where "adr" is the address of the external data memory (or I/O) location. The samples will be displayed on the console. You can stop this process by entering either a period {.}, return {rtn} or escape {esc} character from the console. To repetitively write a value to data memory enter "BT adr data" where "adr" is the address of the data memory and "data" is the value you want to send to that address. Again, you can stop this process by entering either a period {.}, return {rtn} or escape {esc} character from the console. If the "adr" value is 00H to FFH then IDM is used. If the "adr" value is 100H or higher then XDM is used.

An example would be:

```
m51-> bt b03f 5a   {rtn}
xB03F: 5A          (outputs rtn w/o lf to overwrite)
m51->
```

2.3.2 CHECKSUM (CP, CX)

The CHECKSUM command is followed by two arguments. CHECKSUM calculates a 16 bit sum (checksum) from "arg0" to "arg1" (inclusive) and prints the sum on the console. Use "CP first last" for Program Memory and "CX first last" for XDM.

An example would be:

```
m51-> cp 1000 1fff   {rtn}
Checksum: 14DE
m51->
```

2.3.3 DISPLAY (DP, DX, DI, DS)

The DISPLAY is used to show the data at multiple consecutive locations of PM, XDM, IDM or SFRs. Use DP for PM, DX for XDM, DI for IDM and DF for Special Function Registers,. The display command may be followed by one or two hexadecimal arguments. If one argument is used, it is rounded down to the nearest 16 byte boundary and the first 16 bytes up from that rounded down address are displayed on one line. If two arguments are used, the first is rounded down to the nearest 16 byte boundary and it becomes the starting address for the display. The second argument is rounded up to the next 16 byte boundary less one and it becomes the ending address for the display. . You can stop the display output process by entering either a period {.}, return {rtn} or escape {esc} character from the console.

The display consists on 16 bytes per line. A letter prefaces each line to remind the user which address space is being represented. Each byte is printed in hex and the 16 values are followed with the ASCII representation for each byte. Values outside the range of 20H and 7FH are printed as dots.

An example would be:

```
m51-> di 32   {rtn}
  i30: 54 55 56 57  58 59 61 62  63 64 65 66  67 68 69 70  NOPQRSabcdefghij
m51-> dx 0023  0038   {rtn}
x0220: 00 20 40 41  42 43 44 45  46 47 48 49  50 51 52 53  . @ABCDEFGHIJKLM
x0230: 54 55 56 57  58 59 61 62  63 64 65 66  67 68 69 70  NOPQRSabcdefghij
m51->
```

2.3.4 EXAMINE (XA, ..., XR7, XP)

The EXAMINE (X) command is used to examine and change user registers. The X command will display all the normal registers with register labels shown below the values.

An example would be:

```
m51-> x   {rtn}
      FF FE 03 97 B6 5F 6A E7 64 C9 48  FF  FF FFFF  FF FF 60 8000  02 80 7C
m51->  A PS RB R0 R1 R2 R3 R4 R5 R6 R7 @R0 @R1 DPTR @DP  B SP PCTR @PC +1 +2
```

To change a register, the X entry would be followed with the register's name. All registers accessed by more or less standard naming.

XA	display and allow changes of the A register
XW	display and allow changes of the PSW register (incl. reg bank)
XK	display and allow changes of just the current reg bank within PSW
XRn	display and allow changes of Rn of the selected register bank (n = 0, 1, …., 7)
XD	display and allow changes of the DPTR register (16 bits)
XB	display and allow changes of the B register
XS	display and allow changes of the SP register
XP	display and allow changes of the PCTR register (16 bits)

Some examples would be:

```
m51-> xa   {rtn}
  ACC: FF 00   {sp}   (space gets next register)
  PSW: 10      {sp}
   RB: 02      {sp}
   R0: FF      {sp}
   R1: C0      {rtn}  (rtn will end)

m51-> xp
 PCTR: 8087 9042   {rtn}
```

2.3.5 FIND (F)

The FIND command may be followed by 1 to 16 hexadecimal arguments separated by spaces. The monitor searches for replications of the argument list in XDM and prints the starting address of each area within XDM where there is an exact data match. Note that only the least significant 8 bits of each argument are used for the comparison. You can stop the find process by entering either a period {.}, return {rtn} or escape {esc} character from the console.. Find is very useful to see where a piece of code has been relocated following re-assembly without benefit of a new listing.

An example would be:

```
m51-> f 44 66 8d 3e 2a   {rtn}
x0025:
x33DE:
xE09C:
m51->
```

2.3.6 GO (G)

The GO command may be followed with up to 3 arguments. The form is "g<argo> <arg1> <arg2>" where all arguments are optional. If "arg0" is included[1], this will be the address that the monitor will begin the user program at. If "arg0" is not used, the monitor will begin execution at the address contained in the user's PC register (shown by the XP command[2]). "arg1" and "arg2" are optional break-points. If included, the monitor will replace the data at these locations with an LCALL instruction to the monitor break-point routine and then begin execution of the user's program. When the user's program encounters the LCALL breakpoint instruction, control will be returned to the monitor and the monitor will return the original user's data to the breakpoint address.

The GO command restores all user registers before going to the user program and it saves all user registers following a break-point.

Be sure to review the section near the end of this document titled "Cautions with GO Breakpoints, NEXT and ZNEXT Commands". This is important information.

2.3.7 HELP (H)

The HELP command will print some notes about the commands to the console. This is handy if you don't have the manual nearby.

2.4 COMMANDS IN DETAIL

This section outlines the commands in more detail.

2.4.1 JAM (J)

The JAM command is used to jam (or fill) an area of XDM with a constant. Usage is "j start end value". The "value" will be loaded into all memory locations from "start" to "end" inclusive.

An example would be:

```
m51-> j c000 c01f 41   {rtn}
m51-> dx c000          {rtn}
xC000: 41 41 41 41  41 41 41 41  41 41 41 41  41 41 41 41  AAAAAAAAAAAAAAAA
m51->
```

[1] Note that in the GO command there is no space between the G and the arg0 value.

[2] See the Examine command.

2.4.2 LIST STACK (L)

The LIST STACK command prints out the values on the user's stack, a few entries above and below the current user's stack pointer location. The label "SP" denotes the current user's stack pointer address and data.

An example would be:

```
m51-> l   {rtn}
ADR: 58 59 5A 5B 5C 5D 5E 5F 60 61 62 63 64 65 66 67 68
     ------------------------SP-----------------------
DAT: DD 13 93 8A 8B 92 98 31 98 3E 93 99 93 DD 13 93 8A
m51->
```

2.4.3 MOVE (M)

The MOVE command is used to move data from one portion of XDM to another portion of XDM. The move command is followed by three arguments. The first is the starting address of the source range; the second is the last address of the source range and the third is the starting address of the destination range.

An example would be:

```
m51-> m 8000 87f4 9001   {rtn}
m51->
```

This would fill the XDM locations 9001 to 97F5 inclusive with the data starting at XDM location 8000.

2.4.4 NEXT (N)

The NEXT command is a "single step" command and is probably the most powerful command available to the user. When the user gives the monitor a NEXT command the monitor will look at the opcode pointed to by the user's PC register, determine the address of the following instruction to be executed and put a trap (actually an LCALL) instruction at that second address. Then the monitor will load the user's registers into the 8051 and jump to the address pointed to by the user's PC register. The 8051 will execute one instruction and then go on to the following instruction where it will then encounter the previously placed monitor trap instruction. This instruction will transfer execution back to the monitor which will, (1) swap the trap instruction with the user's original instruction at the trap address, (2) save the user's registers, (3) print the user's registers to the terminal and (4) wait for the next user's command from the keyboard (which very likely will be another NEXT command). The NEXT command is like a special case of the GO command where monitor will use the user's PC register data to determine where to go and where to set the break-point address such that only one instruction will be executed.

As an example, after several NEXT commands, the screen might look like this:

```
FF 11111110 FE 03 97 B6 5F 6A E7 64 C9 48  FF  FF FFFF  FF FF 60 8000  02 80 7C
FF 11111110 FE 03 97 B6 5F 6A E7 64 C9 48  FF  FF FFFF  FF FF 60 807C  C2 AF 75
FF 11111110 FE 03 97 B6 5F 6A E7 64 C9 48  FF  FF FFFF  FF FF 60 807E  75 D0 10
FF 00010000 10 02 FF FF FF FF FF FF FF FF  FF  FF FFFF  FF FF 60 8081  75 81 60
FF 00010000 10 02 FF FF FF FF FF FF FF FF  FF  FF FFFF  FF FF 60 8084  12 8E 62
FF 00010000 10 02 FF FF FF FF FF FF FF FF  FF  FF FFFF  FF FF 62 8E62  75 98 52
FF 00010000 10 02 FF FF FF FF FF FF FF FF  FF  FF FFFF  FF FF 62 8E65  43 87 80
FF 00010000 10 02 FF FF FF FF FF FF FF FF  FF  FF FFFF  FF FF 62 8E68  75 89 2D
 A CAFBBOFP PS RB R0 R1 R2 R3 R4 R5 R6 R7 @R0 @R1 DPTR @DP  B SP PCTR @PC +1 +2
```

Be sure to review the section on "Cautions with GO Breakpoints, NEXT and ZNEXT Commands". This is important information.

2.4.5 POV (P)

The POV (Power On Vector) command allows the user to tell the monitor that you want it to jump to location "arg0" a few seconds after power or reset. This command is only useful if the external RAM is protected from memory loss during power off states (i.e., battery backed up). To issue the command type "P arg0" from the console. The location for start-up is saved in a reserved location in RAM (it's complement is also saved in several places as a check). At power-on or reset the monitor looks to the console port for a few seconds to see if a period {.} or escape character {esc} are entered. If not, program control is passed to the routine starting at location "arg0". If period {.} or escape character {esc} are entered within a few seconds of reset then program control is retained by the monitor and the user will see the normal sign-on message and be able to operate the monitor until the next power-on or reset event. To clear the vector issue the UPOV command again with "arg0" equal to zero (0). That will clear the vector so that the monitor will function normally and look only to the console for commands after a reset or power-up. The best way to ensure that the monitor will see the period or ESC is to type the character repetitively following a reset.

2.4.6 RXINTEL (R)

The RXINTEL command is used to load INTEL HEX code into memory space. The RXINTEL command may be followed by an optional offset argument. The offset value is added to the address contained in each record of the HEX code. The result of the addition points to where the data portion of the record will be stored in XDM. If the offset is omitted, it is assumed to be zero. When a record is correctly received, a dot (.) will be printed on the console. When the end of file is found the monitor will output a message to indicate that the down load was successful and the last location where data was stored.

An example of a screen following a download might look like this.

```
m51->r 8000   {rtn}
.....................................................
.....................................................
.....................................................
.........................................

RX OK, DATA ENDING AT: 9981

m51->
```

2.4.7 VFINTEL (V)

The VFINTEL command is very much like the RXINTEL command. The only difference is that the downloaded data is compared to the data in memory (as opposed to overwriting the data in memory like RXINTEL). If there is a difference between the downloaded record and the data in memory an error message will be displayed.

2.5 SUBSTITUTE (SX, SI, SF)

The SUBSTITUTE command is used to show and modify XDM, IDM or the Special Function Registers. The SUBSTITUTE command is always followed by an address argument. This argument determines where the substitution will begin. The address will be displayed followed by the data at that location. To change the data enter a new value and type space {sp} to continue at the next location. To go to the next address without changing the data just enter a space {sp} without preceding data. To complete the SUBSTITUTE process enter a return character [ret] to return to the monitor command mode.

An example would be:

```
m51-> sx b000   {rtn}
xB000: 00       {sp}
xB001: 20 30    {sp}
xB003: 40       {rtn}
m51-> dx f000   {rtn}
xB000:  00 30 40 00  00 00 00 00  00 00 00 00  00 00 00 00  .0@.............
m51->
```

2.5.1 ZNEXT (Z)

The ZNEXT command works exactly like the NEXT command except that when a CALL instruction is found, the monitor will not step through the subroutine. It will break at the location the subroutine should return to.

Be sure to review the section on "Cautions with GO Breakpoints and NEXT" This is important information.

2.6 TEST COMMANDS IN DETAIL

This section outlines each test command in more detail. Test commands are used to exercise various hardware elements and are machine specific. These are not expected to be used to debug user's firmware and therefor are not restricted from using Internal Data Memory (IDM) within the 8051 CPU.

2.6.1 Test External Data Memory (TX)

The TX command is used to test some contiguous area of External Data Memory. This is usually the system RAM. The test makes use of the standard WRITE, READ, WRITE command with walking 1s and walking 0s that usually picks up crossed (or heavily capacitivly coupled) address and/or data leads.

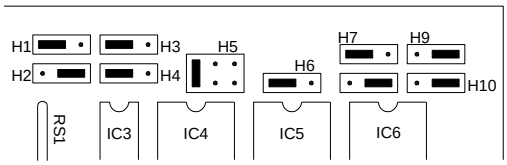
An example would be:

```
m51-> tx 0 1000    {rtn}
TR: Write, Read, Write Test....
TR: Loading initial value, 00
TR: Walking ones, 01 02 04 08 10 20 40 80
TR: Loading initial value, FF
TR: Walking zeros, FE FD FB F7 EF DF BF 7F
TR: Random data test...
TR: Test OK!
m51->
```

3. MEMORY AND PORT MAPPING

The m-51 monitor, as configured for TUC-52, requires that the monitor ROM start at 0H. For TUC-52, this ROM function is provided by the 27c256 installed at IC4. This ROM must output data to the bus when PSEN* is active. PSEN* can be OR-ed with RD* before going to the ROM OE* signal but this isn't a must.

RAM must exist from EF00H to EFFFH and must be double mapped as code and data space (i.e., TUC-52 must OR the PSEN* and RD* signals from the 8032 and connect the output of the OR gate to the RD* input on the RAM). On TUC-52 this RAM is function is provided by the 43256 installed at IC6. When user's programs are executed from RAM, that RAM device must be double mapped as program and data space (i.e., you must OR the PSEN* and RD* signals from the 8032 and connect the output of the OR gate to the RAM's OE* input). On TUC-52 with the shunt settings shown below, the RAM at IC6 will be accessed from 08000H to 0EFFFH. The user should stay out of locations 0EF00 to 0EFCF which are used for monitor functions.



T-Basic & Monitor Jumper Settings

The console communication are provided by the internal UART on the 8032. Because of the way the baud rate generation is provided within the CPU, m-51 for the TUC-52 only runs on the 8032 (or 8052, 8752) CPU) and Dallas 8032 equivalents. It will not operate correctly with an 8051 family part installed at IC1.

Additional memory mapped I/O devices are expected to reside in the space from 0F000H to 0FFFFH. The first 82c55 (IC13) on TUC-52 has a base address of 0F8FCH and the second 82c55 (IC15) on TUC-52 has a base address of 0F9FCH. The 8582 EEPROM on the TUC-52 I2C bus has a base address of 0A2H while the 8583 RTC on the TUC-52 I2C bus has a base address of 0A0H.

4. EMULATION ISSUES

This section outlines some items of importance when using M-51 to emulate your target system.

4.1 ALTERNATIVE FOR LOST PORTS

One problem with using the 8051 in extended memory mode is that ports 0 and 2 are lost. Additionally, P3.6 and P3.7 are lost to !RD and !WR signals. Ports 0 and 2 can be recovered by 8255 I/O ports, to some extent. If possible, assign those I/Os that need to be bi-directional of quickly bit-banged to P1 and the unused sections of P3. Then group all the remaining inputs to one 8255 port (configured as in input) and all the remaining outputs to another 8255 port (configured as an output). In most cases, this will give you a means to recover the lost I/O from Ports 0 and 2. One hardship of using the 8255 I/O is that you can not bit-bang directly to the I/O port. All access to the 8255 ports must be done using byte writes and byte reads. No bit banging permitted! To ease the writing of firmware, you find it helpful to assign two RAM locations in the bit addressable area within the CPU (020H to 028H) to be images of these two 8255 ports. For emulation your firmware can read or write these individual bit values and then have a process, perhaps driven at a 1 ms interrupt rate, that copies the contents of the output RAM image to the 8255 output port and the data read from the 8255 input port and copy it to the RAM input image location. This can work out pretty well if the signals into and out of the 8255 don’t need to move more frequently than once every millisecond.

4.2 SYSTEM STACK POINTER

At reset the monitor's stack pointer is initialized to 31 bytes below the top of Internal Data Memory (IDM). If a processor with 256 bytes of IDM is used the monitor will be initialized to address 0E0H. If a processor with 128 bytes of IDM is used the monitor will be initialized to address 060H. Parts that have more than 256 bytes of internal data memory will be treated like a part with only 256 bytes (i.e., stack initialized to 0E0H). Be aware that following a breakpoint or single step operation that the monitor will not reinitialize the stack pointer. The monitor only sets the stack pointer following a reset (code starts from 0x0000)! Following a break-point, NEXT or ZNEXT instruction the monitor will simply continue to use the stack pointer that was in use when the TRAP instruction was encountered in the user's program. So, if the user's program changed the stack pointer then that's what the monitor will be using following the TRAP. Be sure that your program provides enough stack space for both your program and the monitor. The monitor needs about 22 (decimal) bytes of stack for its operations.

4.3 CAUTIONS WITH GO BREAKPOINTS, NEXT AND ZNEXT COMMANDS

Proper operation of the GO, NEXT and ZNEXT commands demand that the user stack value be pointing to a valid location with about 22 (decimal) bytes of free space. Failure to ensure this will cause the monitor to act flaky or just plain quit. Use the XS command to set the user's stack pointer to somewhere useful following power-on or a system crash (i.e., anytime the XDM used by the monitor may have been overwritten).

If the user's program doesn't eventually get to one of the break-point addresses and the reset button is pressed to stop the user's program the monitor may leave an LCALL instruction at the break-point address(es). Following a reset the monitor will do its best to check for this problem and, if detected, restore the original data at the break-point address(es). However, the monitor isn't always successful at this. You may need to re-download the your code again to ensure that your code is intact.

Recall that the monitor uses a three byte LCALL instruction as its breakpoint trap instruction. It's unfortunate that the 8051 doesn't include a single byte call instruction (like the RSTn of the 8080 family). This would be ideal for use by a simple monitor. The three byte trap instruction causes problems only with short backwards jumps of either 0, 1 or 2 bytes. These are often found in tight DJNZ loops. If you are debugging code in areas such as this you have two options when using the monitor. You can either avoid NEXTing through this part of the code or temporarily add some NOPs so that the backwards jumps are of three or more bytes for debugging purposes. Once you have resolved the programming problem remove the NOPs. This problem of NEXTing to short backwards jumps is discussed below in more detail.

4.3.1 A detailed example

For this example assume that at location PC there's a SJMP instruction to location PC-1. The problem occurs as follows: (1) on receiving a NEXT command the monitor knows where it's going to jump, in this case to PC-1, (2) the monitor then inserts the three byte LCALL instruction at this address with the LCALL occupying locations PC-1, PC-0 and PC+1, (the second byte of the LCALL instruction is the high order address of the function in the monitor where the trap instruction will vector to, the vectored from address will be left on the stack -- that's why a LCALL instruction is used instead of a LJMP), this second byte of the LCALL instruction will be placed at the current PC location (in this example), (3) note that the second byte of the LCALL instruction overwrites the op-code we are now going to execute, (4) the monitor transfers program control to location PC and the high address byte of the LCALL instruction is interpreted by the 8051 as an instruction, (5) at this point operation of the monitor is indeterminate, however it's safe to assume that program control has now been transferred to never-never land. This is why it's best to avoid single stepping through tight loops with small backwards (less than 3 byte) jumps.

4.4 INTERRUPT VECTORS

Because the monitor is configured with ROM at the first 256 bytes of Program Memory the interrupt vectors can not be changed by the user. To solve this problem, the monitor has LJMP instructions placed at the interrupt vector locations in low ROM. These LJMPs point to high RAM locations that the user can load at run time of the emulating firmware. The programmer can use the monitor command "DP 0 FF" to look at the monitor's Program Memory to see where the vectors point to for that particular version of the monitor. Note that the memory locations LJMPed to in high RAM are at three byte intervals, providing enough space to hold another LJMP instruction to any other location in the Program Memory space. In general, the first vector (location 0003H) jumps to EFFDH, the second vector (000BH) jumps to EFFAH, and so on).

At the time this document was written, the interrupt vectors were as follows:

Table 2 -- Interrupt Vectors[3]

ROM Adr:	Jumps to	RAM Adr
00003H	Jumps to	0EFFDH
0000BH	Jumps to	0EFFAH
00013H	Jumps to	0EFF7H
0001BH	Jumps to	0EFF4H
00023H	Jumps to	0EFF1H
0002BH	Jumps to	0EFEEH
00033H	Jumps to	0EFEBH
0003BH	Jumps to	0EFE8H
00043H	Jumps to	0EFE5H
0004BH	Jumps to	0EFE2H
00053H	Jumps to	0EFDFH
0005BH	Jumps to	0EFDCH
00063H	Jumps to	0EFD9H
0006BH	Jumps to	0EFD6H
00073H	Jumps to	0EFD3H
0007BH	Jumps to	0EFD0H

5. BUGS

If you find a bug please submit a bug report in writing and I will attempt to correct the problem.

———ooOoo———

[3] Interrupt vectors at the time this document was written. Although unlike, it's possible that they will change with future revisions of the M-51 monitor firmware.