TRITON SOFTWARE

- HUMBUG V5.1 - A new more powerful monitor

The original TRITON monitor program was written to give the machine code programmer a tool to enter, modify and run programs. It provided routines to drive the keyboard and vdu for Tiny BASIC and routines to enable the user to dump and load programs from a bulk storage medium, namely cassette tape. This first monitor was written to fit into a 1K EPROM (2708) and so it could only provide limited facilities. The author decided soon after building his TRITON that a better monitor was needed to develop machine code programs. At the same time Don Scales (who configured Tiny BASIC for TRITON) decided to use part of the fourth EPROM to extend the facilities of Tiny BASIC. It was therefore a logical step to write an extended monitor for the TRITON using the rest of this EPROM. ROMBUS 5.1 is the firmware package incorporating the upgraded BASIC and HUMBUG. While HUMBUG does not provide all the facilities one would like in a monitor program (eg single step execution), it goes a long way to making life easier for the programmer. The extended Tiny BASIC allows machine code subroutines to be called, so to make the most of your TRITON you will probably want to develop machine code subroutines with the aid of the monitor.

The new monitor contains the original seven functions and eight new ones in addition. The new functions are register dump and modify after a breakpoint, continue from a breakpoint, ASCII string creation and display, formatted hexadecimal dump, tape motor on/off control, base conversion between decimal and hexadecimal and vice versa. The final function allows a choice of output device between the vdu and a serial output port. The device onto which output appears depends on the contents of memory location 1401H. In addition hitting the Reset button does not cause the memory automatically to be cleared as we shall see.

The new monitor resides in Read Only Memory (EPROM) from locations 0000H upto 03FFH and from 0DBEH upto 0FFFH. When you switch your TRITON on a power up reset is generated. This is a signal to the computer to start obeying instructions from location 0000H. At this location the computer finds the instructions which set it up ready for you to use. The first of these initialises the stack pointer. This is a pointer to an area of memory which is used to store parameters and return addresses for subroutines. The program then selects the vdu as the output device, clears the screen and announces itself saying:

HUMBUG V5.1
INITIALISE?
The monitor now waits for a response from you. If the response is 'Y', your TRITON executes a memory test routine. This does a checker board memory test. This involves storing the bit patterns 01010101 and 10101010 in every memory location and checking that each pattern is correctly stored. When a read back error occurs, the address is stored in locations 1481 and 1482H. This address is used by BASIC to determine the size of the workspace available. The routine then initialises the variable (1402,3H) which determines the Baud rate of the serial interface to 110 Baud and sets up a jump table which is used by Tiny BASIC to access the input and output routines in the monitor.

On returning from this routine or if any response other than 'Y' is typed, the prompt:
FUNCTION? P C I O L W T R C A D H V M B
is printed. When you type in a character the TRITON checks it against those in the prompting list, and if a match occurs, it jumps to a routine to obey the function. Otherwise the message:
INVALID
is printed and the TRITON waits for another character which it processes in the same way. When a function terminates the TRITON re-displays the prompting message listing the functions available. In some cases if an invalid character is received by one of the functions, the INVALID message will appear and the TRITON will wait for a new function to be entered.

Most of the subroutines from monitor V4.1 have been retained but they have all been relocated in order to obtain a more compact program. All of the standard monitor utilities have been maintained and a new one has been created. This new one, called ECHOCH is used by means of a CALL instruction and it fetches a character from the keyboard, echoes it on the selected output device and returns the ASCII code for the character in the accumulator.

The characteristics of the input and output subroutines have been changed slightly. As you will know if you have used version 4.1 monitor, the TRITON accepts unshifted letters as upper case letters and shifted letters as graphic characters. In the old input routine blocks of 32 characters were shifted by the software. This meant that the symbols @,[,],^ and _ required unexpected action of the shift key in order to access them and their corresponding graphics. With HUMBUG this action is not needed and all characters are accessed as depicted on the keyboard. (ie @ requires shift to be depressed while [and] do not.)

The character output routine checks the keyboard to see if CONTROL S has been typed berore it outputs a character. In the ASCII code this character is called X-OFF and it is used here to temporarily suspend output. If CONTROL S has not been typed, output continues as normal, but if it has the character will not be displayed. Instead the TRITON will do nothing except wait for you to type another character. If the character is CONTROL C, then the computer will re-initialise with the function prompt. If you type CONTROL Q (ASCII code for X-ON) the output will resume with the current character and then continue as normal until the output ends or another CONTROL S is received. If you type any other character it is ignored. This facility to interrupt the output stream is particularly useful when you are using the hex dump function or listing a BASIC program in order to give you time to read the information before it scrolls off the screen.

You can abort from most of the functions by typing CONTROL C although the computer must be expecting an input character when you type this. This is not the case during the tape input and output functions after a tape has been started, if a machine code program gets stuck in a loop and during a hex dump. You can abort the latter by typing CONTROL S (to stop the output) and then CONTROL C to quit. To quit from the first three cases you can use interrupt 2 or reset. It is alright to use reset since HUMBUG will not clear memory unless you instruct it to, as explained above. If you have used interrupt 2 you will have noticed that your TRITON prints some more information besides FUNCTION etc.. This information is a list of the contents of all the 8080 registers after it obeyed the instruction during which you hit the interrupt button. This is especially useful when you use the interrupt to get your TRITON out of a loop because the contents of the program counter tell you where the loop is in your program. The program counter is a pointer used by the 8080 to tell it the address in memory of the next instruction to be obeyed. Every time an instruction is executed the contents of this pointer are updated. A typical display fter using interrupt 2 is:

```
F  A  C  B  E  D  L  H   SP   PC
46 20 00 00 14 14 00 38 1470 39A7
FUNCTION? P G I O L W T R C A D H V M B
```

There are several other important differences to note in HUMBUG compared with monitor V4.1. When you switch your TRITON on or execute a reset and ask HUMBUG to initialise, the memory will not end up filled with 00H in every byte as before. Instead even addressed bytes (eg 1600H) will contain AAH and the odd addressed bytes will contain 55H. This is because the new memory test subroutine uses a checker board technique as explained above to test for short circuits between adjacent data bits. It does not matter that these bit patterns are

left in memory and they will not affect your programs when you enter them. the stack pointer of the monitor has been changed from 1480H to 1470H so that the sixteen bytes between these values can be used by the BASIC to permit greater flexibility. If you have used 1480H as the initial value of the stack pointer in any of your programs, you should change it to 1470H. Finally the result of interrupts 3-7 is a jump to a location between 1430H and 143FH instead of a jump to somewhere between 1618H and 1638H. This has been done to allow you to use interrupts in BASIC programs without having the problem of the interrupts being vectored into your BASIC source statements. If you have already written programs utilising these interrupts or the RST 3-7 instructions you must set up the vectors at the addresses in the table below before running your program.

It was mentioned earlier that a serial output port has been provided. This utilises bit 8 of output port 06H as a serial output line. The data is formatted into a serial fashion using a software routine. The description given above for the character output routine was slightly simplified. After checking the keyboard for CONTROL S, the routine checks the contents of location 1401H. This location is called the display switch and if it contains 55H then the output is directed to the serial output port. If there is any other value then output occurs on the vdu. Having assertained where the output must be directed the routine performs the appropriate operations. The serial output port produces a pulse train using negative logic (ie a zero is represented as +5V and a one as 0V). The framing format is one start bit (zero) and two stop bits (one's) as shown in figure 1.

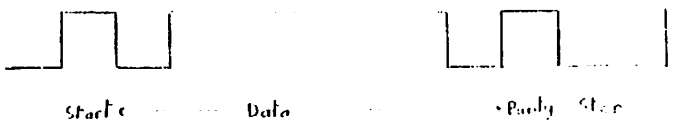


Figure 1 Output format for ASCII 'A'

A fake parity bit is generated for the last bit of the eight bit data field. This format is suitable for TELETYPES etc. where parity checking is not required. To allow time for the mechanics of a printer to perform a carraige return cycle a delay of three character periods is introduced after a carraige return code is sent to the

serial port. During the delay a continuous stop level is sent. When the computer is initialised the bit time is set to 9.09 mS which corresponds to a bit rate of 110 Baud or a data rate of 10 characters/second. If you wish to use a printer or vdu at any other speed it is a simple matter to change the speed by changing the sixteen bit value in locations 1402-3H. The required value can be calculated from the following formulae.

$$T_{bit} = \frac{1}{\text{Baud rate}} \text{ sec}$$

$$t_{cy} = \frac{9}{\text{clock rate}} \text{ sec}$$

t_{cy} = 1.255uS for 7.168MHz clock

$$n = \frac{\frac{T_{bit}}{t_{cy}} - 107}{24}$$

eg for 110 Baud

$$T_{bit} = \frac{1}{110} = 9.09\text{mS}$$

$$n = \frac{\frac{9.09\text{mS}}{1.26\text{uS}} - 107}{24}$$

$$= 297 \text{ or } 0129\text{H}$$

The minimum value of n is 0001H which corresponds to about 6K Baud. Hopefully by now you will have realised that it would be possible to change from outputing on the vdu to outputing on a printer whilst your programs are running. Listed below in listings 1 and 2 are the machine code and BASIC instructions to do this. The machine code version includes the hex instruction codes.

```
3E 55          MVI A,55H
32 01 14       STA A,1401H     ;SET PRINTER MODE

3E AA          MVI A,AAH
32 01 14       STA A,1401H     ;SET VDU MODE
```

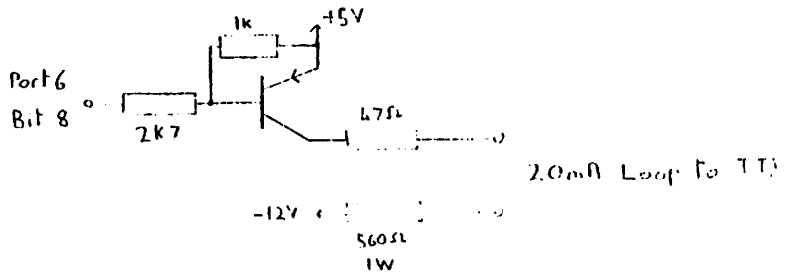
Listing 1 Machine code instructions to change o/p mode

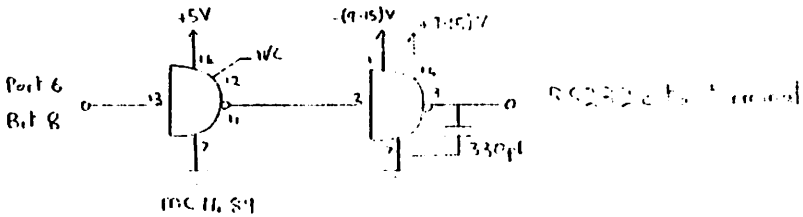

```
A=21760
POKE 5120,A              ;REM SET PRINTER MODE

B=-22016
POKE 5120,B              ;REM SET VDU MODE
```

Listing 2 BASIC instructions to change o/p mode

Obviously the BASIC instructions can use any variables When the POKE 5120 instructions are used location 1400H has 00H written into it. This is the same as the value put there when the character input routine is entered. If you wish to use the serial output facility you must connect an interface between the output port and your peripheral device. It is quite likely that you will require either an RS232c or 20mA loop signal and circuits to generate these are given below in figure 2. For other interface signals you will have to design your own interface. In this case remember that the port produces negative logic.

Figure 2





We will now look at the functions in turn and see exactly how to use each one. Except for the R and B functions you only need to type a single letter to invoke a function. You should not type carraige return.

P - Program

The purpose of this function is to allow you to enter machine code programs into your TRITON's memory. When you type P in response to the function prompt, HUMBUG will respond with:
PROG START = .
You should now reply with four hex digits to specify the address at which you wish to start programming, followed by carraige return. If you type a wrong character you can correct it by typing CONTROL H until the cursor points to the wrong character. You must then retype the rest of the address again. When you type carraige return, HUMBUG will print the address on a new line followed by a space and two hex digits to represent the data held in that address and finally another space. Your TRITON will now wait for you. You can modify the contents of the memory location by typing two hex digits followed by carraige return. Once again errors can be corrected by typing CONTROL H and retyping the character(s) deleted. Whether you change the contents of the memory location or not, you have the choice of stepping on to the next location by typing carraige return or stepping back to the previous location by typing up arrow (SHIFT ~). This facility gives you the chance to correct errors in a memory location after you have moved on to another location. When you do not want to change the contents simply type carraige return or up arrow depending on which location you want to open next. After you have finished type CONTROL C to quit back to the monitor loop.

G - Go to program

This function enables you to start programs anywhere in memory. After typing G, the prompt:
PROG START =
is displayed. Once again you must specify a four digit hex address and this can be corrected as before. After you have typed carraige return, your TRITON will immediately jump to the address specified. (eg Type G 0400 and your TRITON will jump to Tiny BASIC which starts at address 0400H.)

I - Input from cassette tape

The function enables you to input a program from a previously recorded audio tape. After typing I, HUMBUG will ask you for the header code of the program that you wish to load with the prompt:
TAPE HEADER = .
When you have typed the name and carraige return, the tape recorder motor is started automatically and then FILES FOUND: is printed on the display. When a header code is found it is printed on the next line. If this is a perfect match with the name that you specified, then the cursor will stay on the same line and the program will be loaded to 1600H. At the end of the file, the motor is turned off and HUMBUG will say END then re-initialise with the FUNCTION? prompt. If the header code is different the cursor will move to the beginning of the next line and the computer will search for the next header code and the process repeats. By specifying a header code which does not exist on the tape (eg just type carraige return) you can produce a list of the programs stored on the tape. The programs are recorded onto the tape at 300 Baud so you cannot use the tape input function when you are directing the output to the serial port at a speed which is less than or equal to this, since characters from the tape recorder will be lost.

O - Output a program to tape

This is complementry to the above function and it allows you to dump programs to tape. Once again you have to specify the tape header code. This code is recorded onto the tape as an identifier before the program. BASIC programs are recorded onto tape automatically but you must set the programs end address for machine code programs before you enter the O function. This involves putting the address of the first location after your program into 1600 and 1601H in the normal 8080 format with the least significant byte first. When the whole program has been dumped, HUMBUG responds with END and the function list prompt. The recording format is:

About 5-6 seconds of mark tone
64 bytes of leader code (0DH)
Program end address
Program
About 5-6 seconds of mark tone

L - List memory

This function allows you to list the contents of fourteen memory locations at once, but not change them. HUNBUG asks you for a start address and after you have typed an address and carraige return your TRITON will print the address and contents of the next fourteen locations down the left hand side of the display. When the listing stops, you will see the prompt MORE? at the end. If you type Y(es), the contents of the next fourteen locations are printed and so on. If instead you type one of the function letters, that function will immediately be invoked. Typing anything else will cause HUNBUG to display the INVALID prompt.

W - Typewriter mode

After typing W your TRITON will act as if it were only a keyboard connected to a display. Everything you type is echoed onto the selected output device. When you use the vdu all the graphics are displayed and the cursor control codes act as normal. To get out of this mode you must type CONTROL C.

T - Tiny BASIC

When you type T your TRITON executes a jump to Tiny BASIC and you will see the message:

```
BASIC L5.1
OK
>
```

You should read the separate section to find out how to use the BASIC interpreter.

R - Register dump and modify

This function is probably the most useful one for debugging machine code programs. After typing R nothing will appear to happen. In fact the computer is waiting for you to type another character. If you now type SPACE the register contents will be dumped in the same format as described earlier in connection with interrupt 2. It should be realised that these are not the contents of the 8080 registers at the present moment but the contents of a set of virtual registers. They were the contents of the registers last time an interrupt 2 occured.

To modify any register you must type type the letter that represents the register as the second letter of the function call. For the processor status word (flags) use F, for the stack pointer use S, and for the program counter use P. The display will show a two or four digit hex code as appropriate followed by a space. If you wish

to change the value you can type in a new hex number of the correct number of digits (the same number as displayed). Alternatively, to leave the contents unchanged, simply type carraige return. You would normally use this function in conjunction with the continue function and breakpoints. We shall see how later.

C - Continue

This function loads the values of the virtual registers into the 8080 machine registers. Since the program counter is one of the registers which is loaded, the computer effectively jumps to the value put into the virtual program counter. If you wish to run a piece of program a number of times you can set up the value of RP as the start address of the program and simply press C each time to run the program. A more important use of this facility is when you want to test a part of a progam starting with values which would have been left in the registers by the previous part of the program. To do this use Rx commands to set up the registers and the program counter and type C to start.

A - ASCII string insert

The use of this function is to put ASCII strings into memory for use by machine code programs. Typically you will want to have prompts etc. displayed by your programs and this allows you to insert the ASCII strings without having to look up the hex code for each character. After you type A, HUNBUG will ask for the starting address of the string. When you have entered this the cursor will step onto the next line. Anything you now type, including cursor control codes, will be stored in memory and echoed back. CONTROL H will delete characters from memory. When you have finished type CONTROL D (EOT code). The EOT code will be stored in memory and then your TRITON will re-initialise. If you do not want the EOT code to be stored, terminate the string with CONTROL C which will not be stored either. The EOT code allows you to display the strings by calling PDATA from your programs.

D - Display ASCII string

This is the reverse of the A function. It allows you to print out the ASCII string starting at the address that you specify. This enables you to check that you have entered strings correctly. You should not use this function for strings that do not end with EOT code, because this function does not terminate until it finds this code.

H - Hexadecimal dump

This function produces a formatted hex dump of a section of memory. After typing H, the computer asks for the start address in the normal way. When you have entered this a request is made for an end address with the prompt:
PROG END = .
After you have specified this, the start address is printed at the start of the next line, followed by upto sixteen data bytes. The number of data bytes on the first line is such that the next line starts at address xyz0H. On each line of the dump the address of the first byte on the line is printed followed by the sixteen data bytes. The function is a useful way of obtaining a printed listing of a machine code program.

V - Vdu switch

This function is used to select the serial output port when output is on the vdu and vice-versa. After you have typed V nothing else happens on the current output device until it is reselected. The FUNCTION? prompt is printed on the newly selected output device. You can then carry on as normal with all output appearing on the newly selected output device but note the restriction above for the tape input routine. If the function fails to work it means that the data in the display switch is corrupted. Set the value in location 1401H to AAH and try again.

M - Motor control

In the original TRITON design the tape recorder was automatically controlled by the tape routines and there was a manual overide facility for fast winding. The automatic control of the tape recorder has been retained in HUMBUG but the over-ride facility has been incorporated into the software. If the motor is turned off and you type M, the motor will be turned on and vice-versa. This facility is particularly useful when you try to input from tape a non-existant file and have to abort by using a reset or interrupt 2. Doing this leaves the tape motor running. Typing M will stop the motor. This facility makes the front panel over-ride switch redundant so you can disconnect the switch and use it for one of two things. Firstly it can be used as a pause button and secondly it can be used for interrupt 3.

B - Base conversion

This function is provided mainly for the convenience of the BASIC language programmer. The PEEK, POKE, READ, WRITE and CALL commands each require an address which must be specified in decimal. It is most likely that the address or data that you know will be in hex so this routine, which works with sixteen bit two's complement numbers, saves you the trouble of carrying out the calculation on paper in binary! After typing B the computer will wait for you to type a D or a H. This second letter represents the base to which you wish to convert. Hence to convert from hex to decimal type D. Any other letter that you type will be flagged as an error. If you typed a D, HUMBUG will be expecting you to type a four digit hex number. If you wish to convert a two digit number, you must insert two leading zeroes. When you type carraige return the decimal equivelent of the number is displayed including a negative sign where appropriate. If you typed H, HUMBUG will expect a decimal number to be input in free form. This means that you can specify a sign if necessary and need not specify leading zeroes. Try BH -1 and you should get FFFF as the hex value returned. The formatting of numbers has been made compatible with the representation of sixteen bit numbers elsewhere in ROMBUS 5.1 (ie four digit hex numbers in HUMBUG and free format decimal numbers in BASIC).

Breaking and entering

We have now seen what all of HUMBUG's functions can do and we will now see how to make use of them to debug a program. Enter the program shown in listing 3 exactly as shown, (yes it has got an error in it!), using the P function. Try using the up arrow facility (SHIFT ~) to step back down memory at some point. After you have entered it obtain a hex dump using the H function. The start address should be 1600H and the end address should be 1610H. Note that you can stop the output by typing CONTROL S and restart it by typing CONTROL Q. Now carefully check the listing given against the dump you have obtained. If there are any errors go back and correct them with the P function. The aim of the program is to clear the vdu screen and then print the alphabet on the top line of the screen. Now using the A function, enter the alphabet from A-Z starting at 1700H. Terminate the string by typing CONTROL D. Type G 1600 and you should see the screen clear and the letter A appear in the top left hand corner and the re-initialisation message appear on the next line. This is not what the program should do and we must now debug it to get it working correctly. Clearly the jump to NXTCHR is not being executed. We will now set a breakpoint so that we can see what is happening when this point in the program

is reached. A breakpoint is a way of stopping execution of a program at a particular point and returning to the monitor in such a way that the previous execution of the program can be ascertained. As we have already seen an interrupt 2 causes the states of all the registers to be saved and displayed on the screen. The interrupt signal can be generated by software using the RST 2 instruction (D7H). Because interrupt 2 causes the registers to be dumped into the virtual registers, you must not use this interrupt to terminate a function when you are working with breakpoints or you will lose the values of the registers used by the program. Use CONTROL c instead. It is a good idea to get into a habit of doing this always. To set the breakpoint use the P function to change the contents of address 160BH from CAH to D7H. Now start the program again from 1600H. Again the screen will clear and the A will be printed and then the registers will be dumped as described before. The registers that are of interest are A,F,D and E. You will see that the stack pointer is set to 1470H (the monitor stack area) by default. The program counter is pointing to the address of the RST 2 instruction plus one (160CH). The value in the DE register pair is 1701H as we would expect. (We loaded it with 1700H and incremented it once.) The value of the accumulator is 41H which is the value loaded from the data buffer. Obviously 41H is not equal to 5AH and if we look at the flag byte we will see that the zero flag is cleared to indicate this. The format of the flag byte is:

SIGN ZERO xx AUX CY xx PARITY xx CARRY

Clearly the sequence of execution will not be affected by the jump if zero instruction. We must change this instruction to a jump if not zero. Using the P function again, change the instruction at 160BH to C2H. Now using the RP function change the program counter to 160BH, the address we wish to continue from. By typing R SPACE you can confirm that A=41, PC=160B, D=17, and E=01H. If we now type C the register values will be reloaded and the program will continue from 160B as if it had never been interrupted. The computer encounters the JNZ instruction and since the flags indicate non-zero the jump will be executed. What you should see is the C that you just typed followed by the alphabet from B-Z and then the re-initialisation message on the next line. If this is what you see then the program works (It has already done the rest before we interrupted it.) and if you type G 1600, the program will do what we said it would.

```
1600            RST 1           CF      ;CLEAR SCREEN
1601            LXI D,1700H     11      ;SET POINTER
1602              -             00      ;TO DATA
1603              -             17
1604    NXTCHR: LDAX D          1A      ;GET CHAR
1605            INX D           13      ;BUMP PTR
```



```
1606            CALL OUTCH      CD      ;PRINT A
1607              -             13      ;CHARACTER
1608              -             00
1609            CPI 5AH         FE      ;WAS IT Z?
160A              -             5A
160B            JZ NXTCHR       CA      ;NO SO GET
160C              -             04      ;NEXT LETTER
160D              -             16
160E            JMP START       C3      ;YES SO GO
160F              -             60      ;MONITOR
1610              -             00
```

Memory and port designations

To help those of you who wish to use TRITON for machine code programming here are the addresses of memory blocks and ports. Also included is a list of useful subroutines and utility programs that are included in HUMBUG V5.1 which will simplify programming.

Memory addresses

0000 - 03FF	1K EPROM	(holds HUMBUG V5.1A)
0400 - 07FF	"	(holds BASIC L5.1A)
0800 - 0BFF	"	(holds BASIC L5.1B)
0C00 - 0FFF	"	(holds HUMBUG V5.1B)
1000 - 13FF	1K RAM	(vdu memory)
1400 - 14FF	1/4K RAM	(holds stack and tables for HUMBUG from 1400 -1480; 1480 onwards is used by BASIC)
1500 - 15FF	"	(Used by BASIC only)
1600 - 1FFF	2 1/2K RAM	(work area for BASIC or user programs.Note that locations 1600-1 are used by the tape routines to store end of file address. Hence user programs for saving should start at 1602.)
2000 - FFFF	56K	(Available for "off board" extensions)

Port designations

```
00      Keyboard input (note special routine is necessary)
01      Tape I/O UART status input
02      Tape I/O UART data strobe output
03      LED output port (note LED's are on for "0")
04      Tape I/O UART receive data enable (receive data) input
05      Vdu output (note strobe -bit 8- has to be formatted by
                    software)
06      Serial output port (note bit 8 is used only. Bit 7 is
                            available as a spare line)
07      Relay output (note bit ( is used only. Bit 7 is spare)
08 - FF Available for "off board" extensions
```

Monitor utilities

0000	RST0	Reset address. Enables interrupts and initialises memory if requested with a set of default values.
0008	RST1	Reacts to interrupt 1 clearing vdu and resetting cursor.
000B	INCH	A CALL routine which inputs character from keyboard. Stays in keyboard loop until key is depressed. Character returned in accumulator.
0010	RST2	Reacts to interupt 2 displaying register contents and jumping to the monitor.
0013	OUTCH	A CALL routine which outputs character from accumulator to vdu.
0018	RST3	Reacts to interupt 3 and re-vectors to 1430 for jump to user routine.
001B	INDATA	A CALL routine which allows a string of characters to be entered to any place in memory. Start location of string pointed to by DE pair which must be pre-loaded before calling INDATA. Returns on carraige return.
0020	RST4	Reacts to interupt 4 and re-vectors to 1433 for jump to user routine.
0023	PDATA	A CALL routine which allows strings of characters to be printed from anywhere in memory. DE register pair must be pre-loaded with start address before calling. Routine returns when it sees EOT terminator (04H) but DE steps to one address beyond terminator allowing immediate recall for a further string to be printed.
0028	RST5	Reacts to interupt 5 and re-vectors to 1436 for jump to user routine.
002B	PSTRNG	A CALL routine identical to PDATA, except that it outputs carraige return/line feed prior to printing string.
0030	RST6	Reacts to interupt 6 and re-vectors to 1439 for jump to user routine
0033	PCRLF	A CALL routine which prints carraige return/line feed, and then returns the original contents of the registers intact.
0038	RST7	Reacts to interupt 7 and re-vectors to 143C for jump to user routine.
003B	ECHOCH	A CALL routine which inputs a character from the keyboard and then echoes it on the output device and returns it in the accumulator.

The above routines are fixed location utilities whereas the following are within the main body of HUMBUG and are liable to be relocated if a new generation monitor is written. They are however very useful and can save a lot of redundant instructions elsewhere. Note that you can only guarentee using these if you have HUMBUG V5.1 on board!

003E	URTOUT	Waits for a ready signal from the UART and then outputs the value of the accumulator. This should used with a CALL. It is useful for programs that need to store data on the tape.
004A	TAPOFF	When called switches the tape control relay off. Returns with 00H in the accumulator.
0060	START	Re-starts the monitor resetting the stack and printing the re-initialisation prompt. This should be used as the destination of a JMP instruction. It is useful to put at the end of programs instead of HALT.
00BC	DLY	When called introduces approximately 3mS time delay and returns with all registers intact except flags.
00C5	FIVSEC	When called introduces a delay of 5-6 seconds. It returns with all registers and flags intact.
00E3	CLRSCN	Clears screen and resets cursor. Approximately 200mS delay is built into this routine. Returns all registers and flags intact.
0150	IN	BASIC input routine. Reads keyboard port. Returns A=00H if no key is pressed, else it returns the ASCII code. Flags set accordingly.
01F2	PRTADR	Prints the contents of the HL register pair. Returns HL intact.
0206	OUT	BASIC output routine. Outputs a character from A. If the character is carraige return, then line feed is sent as well.
0240	PRTDAT	Prints the contents of the accumulator. Returns the contents intact.
03F6	TAPON	When called switches the tape control relay on. Note that the routine returns 80H in the accumulator.
0E62	UARTIN	When called waits for a character to be received from the UART and then returns it in the accumulator.
0F55	ACKA	Start address of string which prints INVALID. Use should be made of PSTRNG utility to insert carraige return and line feed.
0F69	ACKB	Start address of string which prints START = . (Use via PSTRNG)
0FAC	ACKC	Start address of string which prints HEADER = . (Use via PSTRNG)
0FB5	ACKD	Start address of string which prints END. (Use via PSTRNG)

Notes for use with the examples in the TRITON manual

1. Using interupt 3
Interrupt 3 now vectors to 1430H so you should first insert the bytes C3, 18, 16H into locations 1430-2H before hitting the interrupt button.

2. Duo-decimal string program
Change the instruction at 1615H to JMP START by changing the bytes at 1616H and 1617H to 60H and 00H.

3. Keyboard/LED program version 2
Change the instruction at 160EH in the alternative ending to JZ START by changing the bytes at 160FH and 1610H to 60H and 00H.

4. Alphabet twelve times over using I/O
Change the instruction at 1615H as in 2 above.

5. Alphabet using memory mapping
Change the instruction at 1626H to JMP START by changing the bytes at 1627-8H to 60H and 00H.

6. Modem echo test
Change the instruction at 1602H to CALL URTOUT by changing the bytes at 1603-4H to 3EH and 00H. Change the instruction at 1606H to CALL UARTIN by changing the bytes at 1607-8H to 62H and 0EH.

7. Test tape playback program
Change the instruction at 1600H to CALL TAPON by changing the bytes at 1601-2H to F6H and 03H. Change the instruction at 1603H to CALL UARTIN by changing the bytes at 1604-5H to 62H and 0EH.

8. Test tape record program
Change the instruction at 1602, 160D, 1612H to CALL URTOUT by changing the bytes following each of these addresses to 3EH and 00H.

Implementation of ROMBUS 5.1

ROMBUS 5.1 consists of a set of four EPROMS which are available ready programmed from TRANSAM COMPONENTS LTD. at 12, Chapel Street,London, NW1. To obtain these you should send your old EPROMS back and they will be re-programmed with the new software and a fourth EPROM supplied. The EPROMS are labeled HUMBUG A and B and BASIC A and B. The BASIC should be plugged into the same position as the original BASIC. HUMBUG A should be plugged into the socket where MONITOR V4.1 was and HUMBUG B should be plugged into the fourth EPROM socket.

THE EXTENDED TRITON BASIC INTERPRETER

by

D.M.SCALES

The extended BASIC Interpreter contains several new commands which greatly increase the power of the Interpreter.

These extensions enhance the editing features, improve the exit, allow stop and start of printout, and give the programmer greater access to all areas of memory and input output ports.

They also allow the BASIC to interface with machine code subroutines.

DIRECT COMMAND EXTENSIONS

The control C exit from the BASIC Interpreter has been extended to improve it.

When control C is used during programme execution,control is returned to the BASIC Interpreter rather than the Monitor. When control C is used during programme coding or editing, control is then returned to the Monitor.

Two extra control commands have been added to the keyboard routine.

These are:

Control S

Control S can be used to stop BASIC printout.

Control Q

Control Q is used to restart the BASIC printout again.

During Control S, Control C can be used to exit to the Monitor.

EXAMPLE 1

Enter the following programme

```
10  FOR   I = 1 to 10000
20  PRINT I
30  NEXT  I
```

Now enter

```
RUN
```

Now while the programme is printing on the screen press the control and the S key together. This should stop the printout.

To start the printing again press the control and the Q key and it should start.

Now press control S again and when it stops press control C to exit to the monitor.

EDIT

EDIT allows the programmer to EDIT a line of BASIC code.

EDIT 100 will print statement 100 and place the cursor on the first character of the statement after the statement number.

By typing in new characters, the old statement can be changed by replacement.

Not all characters have to be changed, by using control I and control H the cursor can be forward and backward spaced along the line to the required characters.

EXAMPLE 2

Enter the following code

100 ABCDEFGHIJ

then EDIT 100

the Interpreter will print

100 ABCDEFGHIJ

The cursor is under the A

Now type in 123456 Return

followed by LIST

the Interpreter will print

100 123456

Now EDIT 100 again

This time type control I three times followed by ABCD Return

and LIST again

the Interpreter will print

100 123ABCD

This example shows replacement with shorter code and partial replacement with extra code.

It is also possible to delete characters using the Delete Key, insert by using the Escape key, followed by the character to insert, and to terminate the Edit part of the way along the line without deleting the rest of the line, by using control]

EXAMPLE 3

Enter the following code

100 ABCDEFGHIJ

then EDIT 100

Now press the Delete key twice, this should delete the characters A and B.

Now press control I twice and position the cursor under the E

Now press the Escape key followed by W

This will place the W between the D and E

Now press Return followed by LIST

The line should now read

100 CDWEFGHIJ

Repeat all the above instructions except after inserting the W press control] instead of Return.

The line should now read

100 CDWEFGHIJ

This example shows deletion and insertion and how the EDIT command can be used to change characters in the middle of a statement without completely retyping it.

COMMAND EXTENSIONS

Five new commands have been added to the BASIC Interpreter together with spare commands and dummy entries to enable further user extensions.

CALL

The CALL command is used to call a machine code subroutine.

CALL 8 will call the subroutine at decimal address 8.

(This subroutine will clear the screen and return after the appropriate delay).

The subroutine location can be numeric, a variable or an expression.

```
10 LET A = 9432
20 CALL A + 48
```

This will call a subroutine at decimal location 9480 (For decimal to hexadecimal and hexadecimal to decimal conversions use the B monitor function).

EXAMPLE 4

Using the P monitor function, enter the following machine code at location !E00

```
!E00    CD    INCHAR    TSTC    ', ', ERR
!E01    BD
!E02    09
!E03    2C
!E04    0A
!E05    CD              CALL    TSTV
!E06    8B
!E07    09
!E08    D5              PUSH    D
!E09    EB              XCHG
!E0A    CD              CALL    INDATA
!E0B    1B
!E0C    00
!E0D    D1              POP     D
!E0E    C9              RET
!E0F    C3              ERR     JMP     QWHAT
!E10    32
!E11    09
```

Now enter BASIC and enter the following

```
10  CALL 8
20  CALL 7680,A
30  VDU  513,A
40  VDU  514,A/256
50  VDU  515,B
60  VDU  516,B/256
70  GOTO 20
```

An explanation of the example is as follows:

Statement 10 will clear the screen.

Statement 20 will call the machine code routine at address 1E00.

The BASIC Interpreter uses the DE register pair to point to the current position in the programme, the other registers can be altered without problems.

The machine code programme first checks for a comma following the call routine address and will go to the error exit QWHAT if no comma is found.

On finding a comma, the routine obtains the address of the variable following the comma. DE is saved and the address of the variable placed in DE. The input character string routine is then called. This will accept characters until Return is pressed. The characters will be placed, two at a time, in the Basic variables starting at the one specified in the call.

After the input, DE is restored and control returned to the Basic Interpretor.

The next Basic commands then print the contents of variables A and B using the VDU command.

Type RUN and when the screen is clear enter AB12 Return.

AB12 should appear on the screen and the programme loop waiting for the next input.

This quite complicated example shows how to call a machine code routine, which in turn uses the BASIC Interpreters own routine to decode the information following the subroutines location.

Note that the machine code routine will have to be entered before the programme will run when loading from tape unless the BASIC end of text pointer at location 1600 is adjusted before writing the programme to tape.

PEEK

The PEEK command is used to copy any two bytes of memory into any of the Tiny Basic Variables.

```
10  PEEK  8192, C
```

This statement will copy the contents of decimal Memory locations 8192 and 8193 into the variable C.

The memory address can be numeric, a variable or an expression.

EXAMPLE 5

```
10   FOR I=0 TO 1024   STEP 16
20   FOR J=I TO I+14   STEP 2
30   PEEK J,D
40   E=0; IF D < 0 D=32767+D+1, E=1
50   A=D/4096,D=D-A*4096
60   B=D/256, D=D-B*256
70   C=D/16,  D=D-C*16
80   IF E=1   A=A+8
90   Z=C;GOSUB 200
100  Z=D;GOSUB 200
110  Z=A;GOSUB 200
120  Z=B;GOSUB 200
130  PRINT '  ',
140  NEXT J
150  PRINT
160  NEXT I
170  STOP
200  IF Z < 10 PRINT  #1,Z,;RETURN
210  VDU 0, 55+Z
220  RETURN
RUN
```

This example uses PEEK to enable Basic to list the first part of the Monitor in a Hexadecimal Dump.

Note the STEP 2 on statement 20 to index the address by 2 bytes each time.

POKE

The POKE command is used to write a numeric value, the contents of a variable or the result of an expression into any two bytes of memory (RAM).

```
10 POKE 7680,C+12
```

This statement will copy the contents of variable C plus twelve into decimal memory locations 7680 and 7681.

EXAMPLE 6

```
10    FOR    @(1) = 1 TO 26
20    POKE   @(1) * 2 + 5249,  @(1)
30    NEXT   @(1)
40    INPUT  'ENTER ANY LETTER' @(1)
50    PRINT  'THE LETTER IS NUMBER ',@(1),' IN THE ALPHABET '
60    GOTO   40
RUN
```

This example uses POKE to initialize all the BASIC variables A to Z with 1 to 26.

READ

The READ command enables the Basic user to read information from any port on his system into any of the Tiny Basic Variables.

```
10  READ  16, @(1)
```

This statement will read a byte from port 16 and place it in the variable @(1). Note that the high order byte is zeroed and the data is placed in the low order byte.

The port number is the least significant byte of a numeric constant, a variable or an expression.

The data must be read into a variable.

EXAMPLE 7

```
10    READ  0,A
20    IF A > 127 GOTO  10
30    READ  0,A
40    IF A < 128 GOTO  30
50    A=A-128
60    B=A
70    IF A > 96  B=B-32;GOTO 90
80    IF A > 64  B=B+32
90    VDU 0,B
100   GOTO 10
RUN
```

This example reads a character from the keyboard and prints it on the screen.

Statements 10 and 20 check that no key is depressed by looping until the strobe bit (the high order bit) is off. Once this is so, the programme loops on statements 30 and 40 waiting for a key to be pressed. When a key is pressed, the strobe bit is removed and the input corrected for the shift bias of the keyboard. The character is displayed using VDU.

READ can also be used to examine the keyboard in 'Real Time'.

This enables the user to write Interactive programmes.

EXAMPLE 8

```
10  READ  0,A
20  IF A < 128   GOTO 10
30  VDU 0  A-128
40  GOTO  10
RUN
```

In this example, the keyboard is examined each time round the loop. If a key is depressed, the character is printed. As long as the key is depressed, the character will be printed each time round the loop. This example does not correct the shift bias and letters can be obtained using the shift key.

WRITE

The WRITE command enables the Basic user to write a byte of information to any port on his system.

```
10  WRITE 64, D*2
```

This statement will write the least significant byte resulting from the expression D*2 to port 64.

The port number is the least significant byte of a numeric constant, a variable or an expression.

The data is also the least significant byte of a numeric constant, a variable or an expression.

EXAMPLE 9

```
10  PRINT  'THE TRITON LIGHT SHOW'
15  FOR J=1 TO 30
20  A=1
30  FOR I=1 TO 8
40  WRITE 3,A
50  A=A*2
60  NEXT I
65  NEXT J
70  FOR I=1 TO 500
75  FOR J=1 TO 75;NEXT J
80  WRITE 3,73
85  FOR J=1 TO 75;NEXT J
90  WRITE 3,146
95  FOR J=1 TO 75;NEXT J
100 WRITE 3,36
110 NEXT I
RUN
```

This example will run two patterns through the LED's on port 3.

OTHER BASIC OPTIONS

As well as the extra options just described, the Tiny Basic Interpreter has four spare commands and six dummy entries in its various tables. These enable the user to test new Basic commands before burning them into an EPROM.

Two of the spare commands are called SPRA and SPRB and are found in the main command table TAB2.

They are used in BASIC by just coding

```
10  SPRA
20  SPRB
```

When these are encounted by the Interpreter, it branches to addresses in high core. For SPRA the interpreter jumps to 1FFD and SPRB to 1FFA. At these addresses there is room for three bytes to contain a jump instruction to the appropriate machine code extension. At the end of the machine code extension should be a jump to FINISH in the interpreter at address 090B

The extensions can make full use of the interpreter routines using CALL's. Examples of the use of these routines can be obtained by examining VDU, PEEK and POKE in the Basic Listing.

Variables and expressions can follow the command as long as the machine code routine decodes these and leaves the DE register pair pointing after them.

```
10  SPRA  5,X+3
```

The parameters 5 and X+3 can be decoded using CALL EXPR in the Basic.

The two other spare commands are found in the function table TAB3. SPRC jumps to 1FF7 and SPRD jumps to 1FF4. These can be used to test new functions

```
10   B = SPRC(A,B)
```

This is an example of how SPRC could be coded.

EXAMPLE 10

This example uses SPRC(A,B) to perform MOD(A,B). Using the P monitor function. Enter the following programme:

```
1E00    CDBD09    MOD    TSTC    '(',ERR
1E03    28
1E04    13
1E05    CD5D07           CALL    EXPR
1E08    E5               PUSH    H
1E09    CDBD09           TSTC    ',',ERR
1E0C    2C
1E0D    0A
1E0E    CD6108           CALL    PARNP
1E11    EB               XCHG
1E12    E3               XTHL
1E13    CDB008           CALL    DIVIDE
1E16    D1               POP     D
1E17    C9               RET
1E18    C33209    ERR    JMP     QWHAT
```

The using P again enter this jump instruction

```
1FF7    C3001E           JMP     MØD
```

Now enter the Basic Programme

```
10  INPUT  'ENTER A NUMBER'A
20  FOR I=1 TO 10
30  B = SPRC(A,I)
40  IF B=0 PRINT    #1,A,' IS DIVISIBLE BY ', I
50  NEXT I
60  GOTO 10
```

The dummy entries are found in TAB2,3 and 6.

The names are coded as 7F7F7F7F and the addresses as FFFF. This enables the user with an EPROM Burner to modify these without needing new EPROMS.

Do test any new routines VERY carefully before modifying your EPROMS.

INTERCEPT JUMPS

To allow the user further access to the BASIC, there are five jump instructions coded into RAM.

When the BASIC gets to the end of TAB2,3 or 6 without a match, it jumps out of the interpreter to one of these jumps before returning to the appropriate error or default routines. These jumps can be modified to jump to Machine Code routines before going to the error exit.

TAB2 jumps to address 1471

TAB3 jumps to address 1474

TAB6 jumps to address 1477

The other two jumps are for BASIC IN and BASIC OUT.

BASIC OUT jumps to address 147A before jumping to the monitor.

BASIC IN jumps to address 147D before jumping to the monitor.

If different routines are coded for these, the routines should not corrupt registers D and E. To return to BASIC use the RET instruction.

Note that the jump instructions are restored in RAM only when the memory check is used.

BOUNCE 2

```
10    CALL 8
20    A=1,B=1,I=1
30    C=1,D=1,W=1
40    X=3,Y=3
50    READ O,E
60    IF E < 176 GOTO 130
70    IF E > 183 GOTO 130
80    I= E - 176
90    Y= I - 1
100   IF Y>3 Y=3
110   X= 7 - I
120   IF X>3 X = 3
130   A= A + X * C
140   IF A > 64 C = -C, A=A-X-X
150   IF A < 1  C = -C, A=A+X+X
160   B= B + Y * D
170   IF B > 16 D = -D, B=B-Y-Y
180   IF B < 1  D = -D, B=B+Y+Y
190   Z= A + (B-1)*64
200   VDU W,32
210   VDU Z,42
220   W=Z
230   GOTO 50

RUN
```

BOUNCE 2 is similar to the original BOUNCE programme where a symbol appears to bounce around the screen.

In BOUNCE 2, the user can modify the direction of the symbol, via the READ command, by pressing any of the keys 1 to 7.

If no key is pressed or any other key pressed the direction remains unchanged.

MISSILE 2

```
10      N = 0
20      FOR J = 1 TO 10
30      @(1) = 976, @(2) = 976
40      @(3) = 992, @(4) = 992
50      @(5) = 1008, @(6)=1008
60      CALL 8
70      Y = (RND(16)-1)*64,Z=1
80      A = - 1
100     FOR X = 1 TO 64
110     VDU Z, 32
120     W = Y + X
130     VDU W,126
140     Z = W
150     READ O,B
160     IF B < 128 GOTO 180
170     IF A < 5  A=A+2
180     IF A < 0  GOTO 300
200     FOR I = 1 TO A STEP 2
210     VDU @(I+1),32
220     IF @(1) < 1 GOTO 280
230     VDU @(I),100'
240     @(I+1) = @(I)
250     @(I) = @(I) - 64
260     IF @(I+1) = W GOTO 400
280     NEXT I
300     NEXT X
310     GOTO 500
400     VDU W,42
410     N = N + 1
500     NEXT J
510     PRINT 'HITS',N
```

MISSILE 2 is similar to the original MISSILE programme.

The enemy aircraft will still fly in at a random height between 1 and 16 miles. Your radar is still not working yet (it might be fixed by MISSILE 3) but the missiles ignition has been wired from the silos to the display keyboard. To fire a missile just press any key for a short period. If you keep the key pressed, all three missiles will fire.