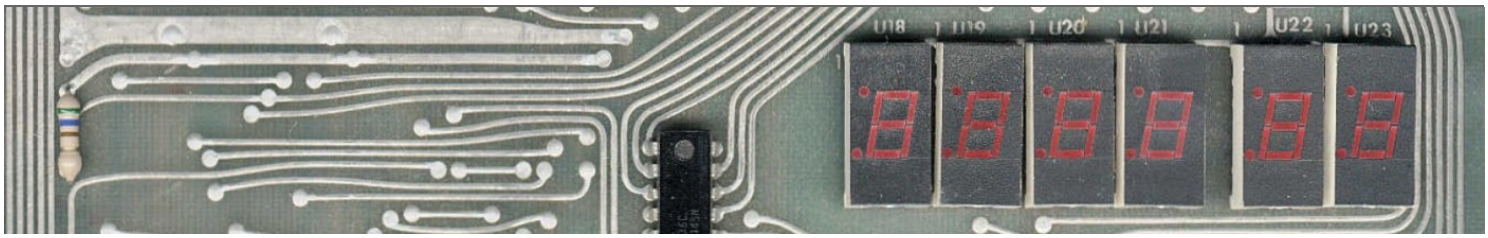
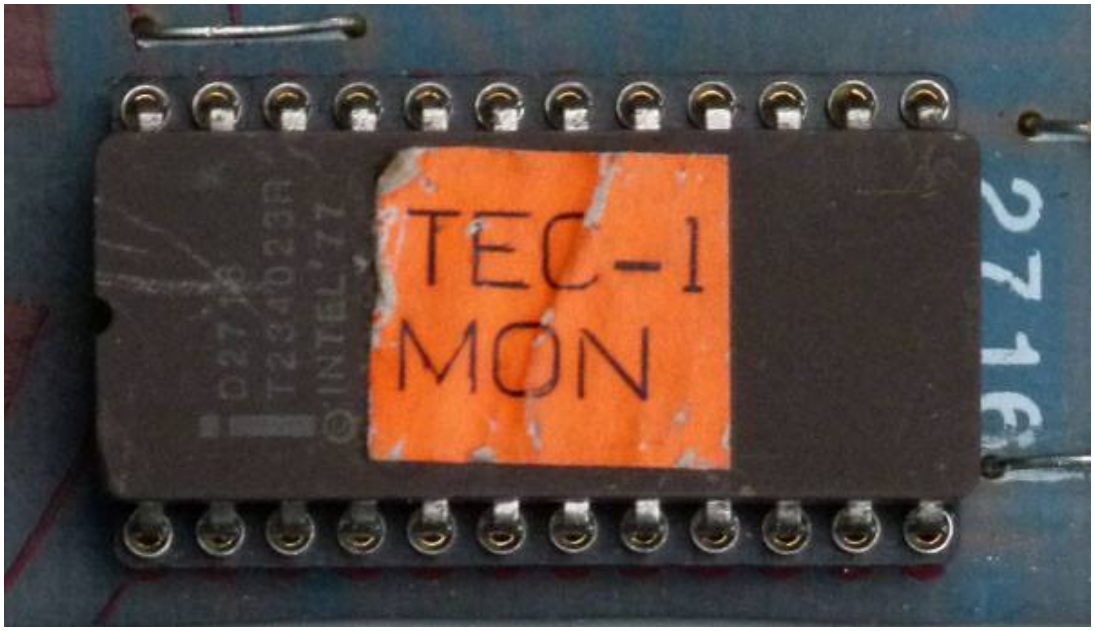


TEC-1 SYSTEM ROMS



Archive with all ROM images, sources, instructions, articles

Mon-1
by John Hardy
The orginal ROM that shipped with the TEC-1 in 1983
Included:
– original BIN binary of ROM
listing in HEX format
listing in disassembled LST format
Reconstructed, labelled and annotated listing in Z80 ASM format
Licensed under the GPL public license v3.0

Search ...

ALL PAGES

KIM clone
Home
Sales!
News
Contact
My sites
6502
The Digital Group 6501 CPU board
6501
65XX IC's
65XX Datasheets
6530-6532
6530 Commodore
6530 KIM-1 clone
Gottlieb sound board
KIM-1 6530 Replacement
6530-004 TIM
A Christmas Story About A Tiny TIM

Mon-1A
by John Hardy with portions written by Ken Stone
This ROM added a sequencer routine at 0x05b0
Included:
– original BIN binary of ROM
– listing in HEX format
– listing in disassembled LST format
– reconstructed, labelled and annotated listing in Z80 ASM format
Licensed under the GPL public license v3.0

Mon-1B
by John Hardy with portions written by Ken Stone
A maintenance release which cleans up some junk at 0x07E2 that was added with Mon-1A
Included:
– original BIN binary of ROM
– listing in HEX format
– listing in disassembled LST format
– reconstructed, labelled and annotated listing in Z80 ASM format
Licensed under the GPL public license v3.0

Mon-2
by Ken Stone
A rewrite of the monitor ROM
Included:
– original BIN binary of ROM
– listing in HEX format
– listing in disassembled LST format
– reconstructed, labelled and annotated listing in Z80 ASM format
Licensed under the GPL public license v3.0

JMON Monitor
by Jim Robertson
Included:
– original BIN binary of ROM
– listing in HEX format
– listing in disassembled LST format
– Listing in Z80 ASM format

– Fully commented source listing by Jim Roberton (PDF)
Licensed under the MIT license

Utilities / Disassembler ROM
by Jim Robertson and Mike Donaghy
Included:
– original BIN binary of ROM
– Listing in Z80 ASM format (needs more work)
– Fully commented source listing by Jim Roberton (PDF)
Licensed under the MIT license

The program will then jump to **80F** and draw the second side of the diamond.

On the next pass, the register pair DE will be looking at locations **0864, 0865, 0866,** and **0867.** This will change locations **089A, 089B, 089D** and **089E** to **2D 34 2D 34** and thus the third side of the diamond will be drawn.

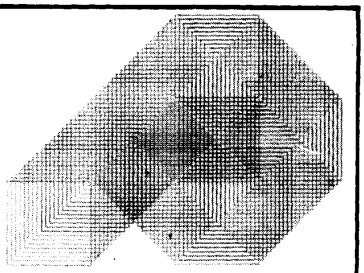
Via the same reasoning, the 4th side of the diamond will be completed.

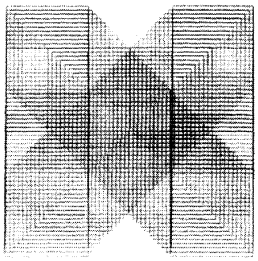
860	00	0	880	49	= I	88C	38	= 8	898	0D	= CR
861	34	4	881	2C	= ,	88D	30	= 0	899	4D	= Move
862	2D	—	882	44	= Draw	88E	2C	= ,	89A	00	= 0
863	34	4	883	38	= 8	88F	30	= 0	89B	34	= 4
864	2D	—	884	30	= 0	890	2C	= ,	89C	2C	= ,
865	34	4	885	2C	= ,	891	2D	= —	89D	00	= 0
866	2D	—	886	30	= 0	892	38	= 8	89E	34	= 4
867	34	4	887	2C	= ,	893	30	= 0	89F	0D	= CR
868	2D	—	888	38	= 8	894	2C	= ,	8A0	FF	= end
869	34	4	889	30	= 0	895	30	= 0			
86A	00	0	88A	2C	= ,x	896	2C	= ,			
86B	34	4	88B	2D	= —	897	30	= 0			

The Direction Change table and Drawing table with decoded values.

Try experimenting and changing this program to produce other patterns. We have included two examples on the right and will be continuing with these and more 'add-ons' in the next issue.

The aim of COMPUTER GRAPHICS is to be able to produce 'forms' and ruled work for invoices etc. and place information in correct locations. It also gives you an understanding of ROBOT movement, an area we all would like to investigate.





MON-1B

LOOKING AT THE REGISTERS

The MONITOR ROM for the TEC-1A (it can also be fitted to the TEC-1) is a MON-1B. This ROM has the facility for looking at the registers.

This ROM is the result of a number of requests from readers who needed to look at the contents of the various registers during the running of a program.

If you would like one of these updated ROMs, send your MON-1 or MON-1A plus $3.00 postage and we will re-burn your EPROM to include the additional instructions.

There is a limit to when and where you can use the register facility but it can help enormously with debugging programs.

For instance, it can let you know the progress of a program or delay routine simply by interrupting it part way through.

The way this facility works is as follows:

If you reset the computer while it is executing a program, by pressing the reset button ONCE, the contents of each of the registers is pushed onto a stack.

This stack starts at **0FF0** and increases downwards to **0FD8.**

To look at any of the registers, press reset once and key the address of the register you want to look at.

The following list identifies the location of each register:

Mem ADD:	Reg:	
0FF0		
0FEF	A	
0FEE	F	
0FED	B	
0FEC	C	
0FEB	D	
0FEA	E	
0FE9	H	
0FE8	L	
0FE7	IX	MSB
0FE6	IX	LSB
0FE5	IY	MSB
0FE4	IY	LSB
0FE3	A'	
0FE2	F'	
0FE1	B'	
0FE0	C'	
0FDF	D'	
0FDE	E'	
0FDD	H'	
0FDC	L'	
0FDB	I	
0FDA	—	
0FD9	Stack	MSB
0FD8	Pointer	LSB

Note: Reset clears the I register and thus it will always equal 00.

Use **JP 0000** if you wish to look at **I.**

An alternate method of saving the registers is to insert a **JP 00 00** instruction in the program at the position you wish to investigate. This will cause a JUMP to the beginning of the MONITOR ROM where it will find a jump to the register-save routine.

This will enable you to exit a program at a pre-determined point and look at the registers. The contents will be shown in the data displays.

Pushing Reset twice will destroy the information.

This is the program at **05F0** which performs the 'REGISTER-SAVE' operation. Don't forget the Monitor ROM has an instruction at 0000 to Jump to **05F0.**

```
05F0   ED   LD (0FD8),SP
05F4   31   LD SP,0FF0
05F7   F5   PUSH AF
05F8   C5   PUSH BC
05F9   D5   PUSH DE
05FA   E5   PUSH HL
05FB   DD   PUSH IX
05FD   FD   PUSH IY
05FF   08   EX AF,AF'
0600   D9   EXX
0601   F5   PUSH AF
0602   C5   PUSH BC
0603   D5   PUSH DE
0604   E5   PUSH HL
0605   ED   LD A,I
0607   F5   PUSH AF
0608   C3   JP 0580
060B   FF   RST 38H
```

INTRODUCTION

A discussion on Talking Electronics latest monitor for the TEC computer

Article and monitor by Jim Robertson

JMON is a big step ahead for the TEC.

Some of the contents of JMON are: a highly improved Monitor Program, a versatile Tape Storage Program, software for driving a liquid crystal display, a Menu Driver for utilities, a Perimeter Handler, User Reset Patch, Single Stepper and Break Pointer with register display software, and simplified access to utilities and user routines.

JMON also uses indirect look-up tables stored in RAM. This idea leaves the door open for many possibilities.

All the above and more is contained in 2k bytes.

The following is a description of the major blocks in the ROM.

THE MONITOR PROGRAM

To support new features added to the TEC, a new interactive monitor program has been written. The new monitor is, by itself, a considerable upgrade over previous monitors and when combined with other software in the monitor ROM, gives great features for the TEC user.

Major improvements have been made in the MONitor software, to allow quicker entry and editing of code. This has been achieved by adding such features as auto key repeat and auto increment. If you add the LCD, its larger display and cursor control software open up a second level of improvement.

THE TAPE SOFTWARE

The TAPE SAVE facility is versatile and reliable.

Some of the functions include: 300 and 600 baud tape SAVE, auto execution, LOAD selected file, LOAD next file, LOAD at optional address, TEST tape to memory block and TEST tape check sum. Both tests may be combined with other options.

The tape software uses the universal MENU driver and perimeter handler. These routines allow easy selection of cassette functions (e.g. Load, Save, etc.) and easy passing of variables to the tape software.

The tape software contains check-sum error detection that allows the user to know if the load has failed. A check-sum compare is performed after every page (256 bytes) and also after the leader is loaded. This means the user does not need to wait until the end of a load or test for error detection.

Each full page to be loaded, tested or saved, is displayed on the TEC LED display. Up to 16 pages are displayed.

Upon completion of a tape operation, the MENU is re-entered with an appropriate display showing:-END -S (END SAVE); PASS CS (CHECK SUM); PASS Tb (TEST BLOCK); PASS Ld (LOAD); FAIL CS (CHECK SUM); FAIL Tb (TEST BLOCK); FAIL Ld (LOAD).

The one exception is when an auto execute is performed after a successful load.

The tape software will display each file as it is found and also echo the tape signal.

LIQUID CRYSTAL SOFTWARE

This software is called from the monitor program. It is possible to de-select this software to allow the liquid crystal display to perform a user-defined purpose while the monitor is being used.

The Liquid Crystal Display is being accessed as a primary output device to the user during the execution of the monitor. Eight data bytes are displayed at a time and a space between each for the prompt (it appears as a "greater than" sign). Four digits in the top left hand corner show the address of the first byte.

In the bottom left hand corner is a current mode indicator and this lets you know which particular mode JMON is in. E.g. Data mode, Address mode etc.

The prompt points to the next location to have data entered, or if at the end of the 8 bytes being displayed, the prompt parks at the top left corner indicating a screen change will occur on the next data key press. This allows revision before proceeding.

It is possible to use the monitor with only the LCD unit, the only drawback being the actual current value of the address pointer is not displayed (the value shown in the address portion of the LED display). However this is only minor.

MENU DRIVER

This is a universal routine used to select various utilities routine from JMON. It is already used by the tape software and the utilities ROM. It may also be easily used by the TEC user.

The Menu Driver displays names of functions in the TEC LED display. The number of different names is variable and may be user defined. It is possible to step forward and backward through these names.

A 3-byte jump table with an entry for each name provides the address of the required routine. A jump is performed upon "GO."

To have a look, call up the cassette software by pressing SHIFT and ZERO together. If you have not fitted a shift key, the cassette software can be addressed by pressing the address key, then the plus key, then zero.

To move forward through the MENU, press "+". To move backward, press "-". Notice the automatic FIRST-TO-LAST, LAST-TO-FIRST wrap around.

Pressing "GO" will take you into the perimeter handler.

PERIMETER HANDLER

Like the Menu Driver, this is a universal program and may be easily used by the user.

This routine allows variables to be passed to routines in an easy manner. The variables are typically the start and end address of a block of memory that is to be operated on, such as a load, shift, copy, etc.

A 2-character name for each 2-byte variable is displayed in the data display while the actual variable is entered and displayed in the address display.

The number of variables may be from 1 to 255 and is user definable.

The data display is also user definable.

It is possible to step forwards and back through the perimeter handler in the same fashion as the MENU driver.

When a "GO" command is received, control is passed to the required routine

via a 2-byte address stored at 0888 by the calling routine.

The SINGLE STEPPER and BREAK POINT handler.

A single stepper program can be important when de-bugging a program. It effectively "runs" the program one step at a time and lets you know the contents of various registers at any point in the program.

If you have ever produced a program that doesn't "run", you will appreciate the importance of a single stepper. Many times, the program doesn't run because of an incorrect jump value or an instruction not behaving as the programmer thinks.

The single stepper runs through the program one instruction at a time and you can halt it when ever you wish. By looking at the contents of the registers, you can work out exactly what is happening at each stage of the program.

The single stepper operates by accessing a flip flop connected to the Maskable Interrupt line of the Z-80. It can be operated in the manual mode, in which a single instruction is executed after each press of the "GO" key. In the auto mode, 2 instructions are executed per second.

BREAK POINTS

Break points work with groups of instructions. They allow register examination in the same way as a single stepper. The advantage of break points is that there is no time wasted stepping slowly through a program. This is particularly important as some programs contain delay loops and they may take weeks to execute at 2Hz!

Break points are one of the most effective ways to debug a program!

STARTING WITH JMON

JMON is straight forward to use. Some new habits must be learnt, however they are all quite easy.

JMON has 4 modes of operation. They are:

DATA MODE, ADDRESS MODE, SHIFT MODE and FUNCTION MODE.

The data address and shift modes are not new but have been, in part, changed in their operation. The function mode is new to the TEC and I am sure you will find it useful. Below is a description of each mode.

THE DATA MODE

The data mode is used to enter, examine and edit, hex code into RAM memory. It is identified by one or two dots in the data display and the word "DATA" in the bottom left hand corner of the LCD display. It is similar to the data mode on all previous MONitors.

The data mode has a sub-mode called AUTO INCREMENT. This is a default setting, meaning that it is set to auto increment on reset. The user may turn off the auto increment sub-mode if desired.

When in the auto increment mode, the current address pointer in the address display is automatically pre-incremented on each third data key press.

A SINGLE DOT in the RIGHTMOST LED display indicates the current address will be incremented BEFORE the next nibble received from the keyboard is stored.

This allows the user to review the byte just entered. If an incorrect nibble is entered, the internal nibble counter MUST BE RESET by pressing the ADDRESS KEY TWICE. Then two nibbles may be entered at that location. This is a slight annoyance at first, but it is a small price to pay for such a powerful feature as auto increment!

After two nibbles have been entered, the prompt on the LCD is IMMEDIATELY updated and points to the next memory location, or in the case of the last byte on the LCD, the prompt PARKS AT THE TOP LEFT CORNER signifying an entire screen update UPON THE NEXT DATA KEY PRESS.

This allows the user to revise the entered code before continuing.

You must be in the data mode to perform a program execution with the "GO" key. (Actually, you can be in the SHIFT mode also.)

Because of the auto key repeat, and "auto increment", it is possible to fill memory locations with a value by holding down a data key. This may be useful to fill short spaces with FF's or zero's.

Because the LCD prompt is advanced immediately after the second nibble being entered while the LED display is advanced on the third nibble received, the "+" key will advance only the LED display while the "-" key will shift the LCD prompt back two spaces, if either are pressed immediately after the second nibble is entered. This may seem strange but is the result of a clever design which allows for revision of entered code on either display before proceeding.

ADDRESS MODE

This is identified by 4 dots appearing in the address display of the LED display and "ADDR" in the LCD bottom corner.

The address key is used to toggle in and out of this mode.

October 2017
July 2017
June 2017
May 2017
March 2017
February 2017
January 2017
December 2016
September 2016
August 2016
May 2016
April 2016
March 2016
September 2015
August 2015
June 2015
May 2015

CATEGORIES

6502
kim-1
tec-1
TIM
website
Z80

The address mode will be entered by an address key press from either the data or function mode. An address key pressed while in the address mode will result in a return to the data mode.

While in the address mode, data keys are used to enter an address while the control keys (+, -, GO) are used to enter the function mode. No auto zeroing has been included, therefore 4 keystrokes are required to enter any address.

SHIFT MODE

This mode allows easy manual use of the cursor. The shift works by holding down the shift key and at the same time, pressing a data key.

The monitor must be in the data mode and only data keys work with the shift.

Sixteen functions are available but only ten have been used in this monitor.

The shifts are:

Shift-zero: Cassette MENU is displayed.

Shift-one: Cursor back one byte.

Shift-two: Start single stepper at current address.

Shift-four: Cursor forward 4 bytes.

Shift-five: Break from shift lock (see function mode).

Shift-six: Cursor back 4 bytes.

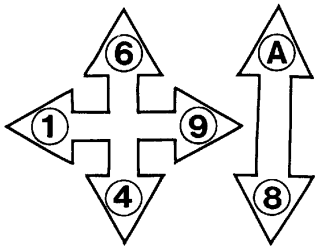
Shift-seven: Enter register examination routine.

Shift-eight: Cursor forward 8 bytes.

Shift-nine: Cursor forward 1 byte.

Shift-A: Cursor back 8 bytes.

Note that 1, 4, 6 and 9 form a cross and 8 and A form an arrow and each is positioned to correspond to their cursor movement.



Keys 1, 4, 6 and 9 move the cursor LEFT, RIGHT, UP AND down on the LCD.
Key "A" shifts the screen back to display the previous eight bytes.
Key "8" shifts the screen forward eight bytes.

When editing a program, the shift enables fast movement through the memory. Data entry is achieved by releasing the shift key.

The shift mode is not identified explicitly on either display.

THE FUNCTION MODE

This has been provided to enable a quick way to call commonly used routines. Only three keystrokes are required invoke up to 48 different routines.

The function mode is broken up into 3 sections.

They are: Function select-1, Function select-2 and Function select-3.

Each is identified by a single dot in the address display: right-most for function 1, second right for function 2 and third right for function 3. On the LCD display, the functions are identified by: Fs - 1, Fs - 2, or Fs - 3 in the bottom left corner.

Fs-1, Fs-2 and Fs-3 are entered FROM THE ADDRESS MODE by pressing the "+" key for Fs-1, the "-" key for Fs-2, the "GO" key for Fs-3.

It is possible to swap between sections without coming out of the current function mode by pressing the required function select key. After entering the required section, A DATA KEY IS THEN USED TO SELECT ONE OF SIXTEEN ROUTINES.

The address of these routines are stored in a look-up table.

SECTION-1 - the SHIFT-LOCK FEATURE.

Section-1 is selected FROM THE ADDRESS MODE by pushing the "+" key. The keys 0, 1, 2, 4, 5, 6, 7, 8, 9 and A then have the functions as listed in the shift mode. (Key 5 has the function of returning to the data mode.)

Cursor control routines return back to section-1 to enable continuous cursor movement (shift-lock).

The look-up table for the jump addresses for section-1 is at 07E0.

SECTION-2

Section-2 is selected from the address mode by pushing the "-" key. This is unused by any existing software and is available to the user.

HERE'S HOW TO USE IT:

Using the section-2 is very easy. All that is required is to enter the address(es) of the required routines in a table. The table begins at 08C0. The first two bytes at 08C0 correspond to the zero key in section 2. While the second two (08C2) correspond to key one etc.

Here is a short program as an example:

08C0: 00 09 04 09 08 09

(These are the addresses of the routines).

0900: 3E EB 18 06 3E 28 18 02
0908: 3E CD D3 02 3E 01 D3 01
0910: 76 C9

Now push ADdress, "-", "0" and the routine at 0900 will be CALLED from the MONitor. Reset the TEC and try ADdress, "-", "1" and ADdress, "-", 2.

Because these routines are CALLED from the MONitor, you may use a return (RET, C9 or RET NZ etc.) instruction to re-enter the MONitor in the same state as you left it. e.g. in the function select-2 mode.

SECTION-3

This has been reserved for the utilities ROM at 3800. The table for Section-3 is at 3820.

USING THE SINGLE STEPPER

Getting the single stepper to work is simple enough, however there is some skill required to understand its limitations and knowing how to avoid them.

To start with, you need a program that you require to be SINGLE STEPPED.

This program may be anywhere in memory except in the lowest 2k (the MON ROM).

This is because the MON select line is used as part of the timing. You may call into the MON ROM but only the first instruction will single step, and when returning out of the ROM, the next instruction will also not be stepped. (However they will be executed at normal speed.)

Programs that use the TEC's keyboard require careful attention as you cannot step them in the normal way. This is because there is no way to distinguishing between key-presses for the single stepper and those for the subject program.

This reduces the usefulness of the single stepper a little however thoughtful software design enables a fair degree of flexibility and this problem may be side-stepped.

The key use of the single stepper is as a de-bugging aid. When you are writing programs. effective use of the single stepper usually requires that while writing your programs, you allow for the use of the single stepper by leaving room to place one byte instructions that turn ON and OFF the single stepper.

Programs using the keyboard may be stepped by turning OFF the stepper. This allows areas requiring use of the keyboard to run in real time while other areas may be single stepped. This applies only to programs that use the keyboard routines provided inside JMON.

The only disadvantage here is that after completing your program you may have NOPs left.(from where you blanked over the single stepper control bytes).

The keyboard controls for the single stepper are as follows:

To start single stepping from the current address, this is what to do: From the data mode, press shift-2. This will start the single stepper. The first instruction will be performed and the address will be displayed as "PC" (Program Counter) on the single stepper. To examine the registers, press "+". The left two nibbles correspond to the high order byte and in the case of register pairs, the left-hand register. You may go backwards also by pressing "-". The registers displayed are : PC, AF, BC, DE, HL, IX, IY, AF', BC', DE', HL' and SP, in this order.

To step the next instruction, press GO. You can also step continuously at about 2Hz by pressing any data key.

When in the auto step mode, you can stop at any time and examine the registers by pressing "+" or "-", or bring it back to the manual mode by pressing GO.

The address key resets back to the MONitor unconditionally. The control bytes for the single stepper are as follows:

To stop single stepping in a program: F3 (disable interrupt).

To restart in a program: EF (restart 28). This causes a restart to 0028 where a routine passes the start address (which is actually the return address of the restart 28 instruction) to the single stepper. It also enables the interrupts and then returns to the next instruction which is then single stepped.

This SINGLE STEPPER is only a first model. Hopefully, when more room is available, some improvements can be added. One improvement on the "cards," is allowing it to be interfaced with a utilities ROM. This ROM will extend the display capabilities, allow editing while stepping and to disassemble on the LCD each instruction as it is stepped. If you have any ideas or requirements, write in and tell us.

BREAK POINTS

Break points are locations in a program where execution is stopped and the registers are examined in the same fashion as the single stepper. The advantages over single stepping include real time execution and less or no control bytes in a program. They also usually allow much quicker fault finding.

As a trade-off move, only a simple (but effective!) form of break-point is available with JMON. This allows for more MONitor functions and also eliminates the need for extra hardware.

More complex methods automatically remove the break-point control byte and re-insert the correct op-code and allows re-entry to the program.

USING JMON BREAK-POINTS

Break points are achieved by using a restart 38 instruction. The op-code for this is FF and all that is required is for it to be placed where ever you require your break point.

Before running your program, make sure the TEC is reset to 0900. This is necessary to clear the auto-repeat on the stepper/break-point register display. (This is explained in the LCD section).

Simply run your program as normal. When the break point is reached, the register display routine is entered. The value of the program counter display WILL NOT BE VALID on the first occurring break unless you provided the address of the break point at 0858. This minor flaw was unavoidable without considerable additional software which would have "eaten" memory like there's no tomorrow!

If you allowed for break commands in your program, you may then have multiple breaks and step to the next break with the GO key.

However if you placed a break command over an existing instruction then no further breaks will be valid and you should never try multiple breaks in this case AS YOU MAY CRASH THE MEMORY.

In the above case, make a note of the contents of the registers and return to the monitor via the address key and then examine memory locations, if required. (You may enter the register examination routine via shift-7). Further breaks should be done by removing the existing break and placing it where required and re-executing the program from the start.

Some other good ideas are to load the stack away from the MONitor's RAM area. (08F0 is good but make certain that 08FF does not contain AA - as this prevents the MONitor rebooting its variables on reset and your stack may have accidentally crashed these variables.) Also, if you are using the LED display scan routine in the MONitor ROM, shift your display buffer to 08F0 by putting this address into 082C/2D. Now you can examine your stack and display values after returning to the MONitor.

There is a conditional way to cause breaks. To do this requires a conditional jump relative with FF as the displacement. If the condition is met, the jump is made and jumps back onto the displacement which then becomes the next op-code! Remember this as it is a very useful idea. You cannot continue on with multiple breaks after a break caused by this method.

Break points are a quick way to debug a program. It is very important that you familiarize yourself with them. They have been the single most important programming aid used when writing most of JMON and the utilities ROM.

SUMMARY:

Clear the auto-repeat via the reset.

: Use FF to cause a break

: PC not valid on first break.

: For multiple breaks, provide spaces for the break control byte.

: Shift stack and display buffer (optional)

: Use FF as displacement for conditional breaks.

Finally, make sure you write down when, where and why, each time you insert a break-point.

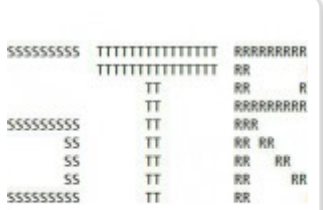
ACKNOWLEDGMENT

Thanks to MR. C PISTRIN of Traralgon VIC. His SINGLE STEPPER program for the MON-1B inspired me to include one in JMON and provided me with a circuit for the hardware section.

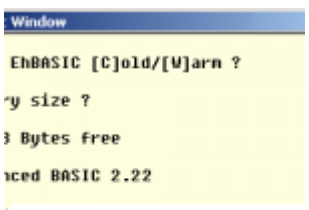
See page 47 for the circuit

Related Posts:

DOS65 manuals, sources listings



DOS65 application: ASTRID and Viditel



EhBASIC language reference

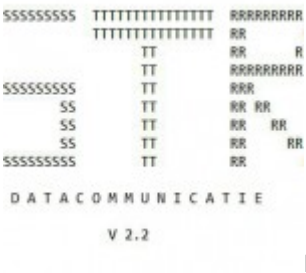
KIM 6502 UP Kenner

ARCHIVES

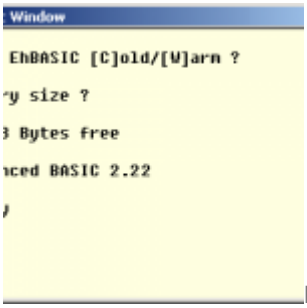
LICENSE

RELATED POSTS

DOS65 manuals, sources, listings



DOS65 application: ASTRID and Viditel



EhBASIC language reference

March 2016

September 2015

August 2015

June 2015

May 2015

KIM 6502 UP Kenner