THE ZAPPLE MONITOR

A. INTRODUCTION

This monitor system is a result of many years of work by many people. It was first used on an 8008 system (Intellec 8 by Intel), and then later modified for the 8080. Although the actual code has been greatly modified, the basic concept has been retained in this Z-80 implementation.

There are many approaches that can and have been used regarding a "SYSTEM MONITOR". The one that is used here is probably the most desirable for either the industrial or the hobbyist/experimenter environment. This monitor may be classified as a "DEBUG" monitor. That is, it contains all the needed tools to fully debug both hardware & software, as well as support the I/O used by the system or transient USER programs. It should be painfully obvious to anyone that has watched this field grow that each person has his own idea as to what he wants (or can afford) for his Input/Output devices. This makes exchanging software extremely difficult, and takes some of the fun out of it. TDL's ZAPPLE monitor may help solve this problem. All of our resident software contains NO I/O routines whatsoever! What we are suggesting is that since everyone's I/O is different, let's make the applications software "I/O INDEPENDENT", and then supply a universal monitor system that each person can CUSTOM-FIT to his own particular needs. As long as the I/O VECTORS are honored, then a BASIC or a FORTRAN or a TEXT EDITOR (etc.) program does not have to be concerned whether you have a parallel input keyboard & video display or a model 33.

The ZAPPLE system monitor program may be called a "CHARACTER ORIENTED" system. This means that there are no buffers needed to buffer keyboard input, or hold data waiting for output. It handles all I/O through vectors at the beginning of the program. It also contains features that have come about as the need for them was felt during it's use by the author. This monitor will come to be the most important piece of software in your system.

The Zapple Monitor occupies 2K of memory, is relocatable, expandable, and ROMable.

The expandability feature is of tremendous user importance as it allows the user to attach his own additional monitor routines at the end of the monitor. Such routines often include I/O drivers. Typical additions might be a VDM driver routine or cassette driver routine. Specific routines will be published in the TDL User's group newsletter from time to time.

The monitor also includes many useful subroutines that may be used by user written programs (see the Assembly listing).

B. LOADING PROCEDURE

All TDL software is relocatable. That is, it may be loaded and run from any address the user chooses, providing only that sufficient memory space above the starting address is provided. Loading of all this software is accomplished via the Monitor. The monitor however requires a bitswitched loader for its loading.

If you now have the 1-K ZAP monitor in running in your system, it is a simple matter to load the 2-K ZAPPLE. Set the tape up on the reader AFTER the binary boot-strap, and type: R,F000(cr), and start the reader. If ZAP is now located at 0F000H, you may load a temporary copy at some other location, and then load ZAPPLE up at 0F000H. Remember, this is the location that future software expects to find the monitor. (although it may be located elsewhere, and the I/O vectors modified in any future software accordingly.)

ZAPPLE will initially be set up for the old MITS standard, I.E.:

DEVICE	STATUS	DATA	TEST BIT
MAIN CONSOLE	0	1	0=RDA, TBE=7
CRT	4	5	SAME
CASSETTE	6	7	SAME

The test bits are active low. I.E.

Receiver Data Available (RDA)= 0 (there is data)
Transmitter Buffer Empty (TBE)= 0 (You may load the buffer.)

Once the program has been loaded, the user is free to modify any of the drivers, using the software listings included with the Monitor documentation. This will be adequate as all current and future programs reference the monitor system for its I/O handling. This establishes a large degree of hardware independence for the applications software.

To initially load the monitor, a small "bootstrap" program is required. This program may use any I/O port, with any "Data Available" test bit, and may have any polarity. Once loaded, if standards other than the previously mentioned ones were used, bit-switching will be needed to at least bring up the main console device. Once this is done, the monitor itself may be used to modify the other I/O drivers.

Since the Zapple Monitor is relocatable, it is necessary to tell the loader where it is to be loaded. This is done by setting the SENSE switches to the starting PAGE desired. For example, to load the monitor to run at 0F000H, set the sense switches to 11110000. It is recommended that the monitor be loaded in the highest available location in memory, so that both the monitor and its stack area be "out

of the way" of your other software. Additionally, it is wise to leave some blank memory above the monitor so that your own user routines may be added on. For our in-house systems we typically load the monitor in one 4K board, addressed from F000 to FFFF. Thus, the monitor will be addressed from F000 to F7FF, leaving F800 to FFFF available for additional I/O drivers.

It should be noted that the monitor does NOT require contiguous memory below it. It will function equally well with itself located at F000 and the remainder of your RAM from 0 to 1FFF for example.

TDL software is sent out assuming the monitor resides at F000 (hex) or 360:000 (crazy octal) or 170000 (octal) or 61470 (decimal). If this address is either not possible or inconvenient, you may subtract the size of the monitor (800 hex) from the top address available. For example, if you have a 12K system, and you wish to put the monitor in at 10K, then you would subtract 800 hex from 3000 hex. This means setting the sense switches at 28 Hex.

The LOADER for ZAPPLE has been modified from the one used with the ZAP monitor. It is now easier to set-up for almost any type of device that may be used to read paper tape. The following is the minimum procedure required:

1. Load the following bootstrap program in at address 0.
(This boot program would be used with the old MITS I/O standards. Examples of this and other possible boot loaders are contained in appendix A.)

```
addr   0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F

0000  C3 1A 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0010  00 00 00 00 00 00 00 00 00 00 31 00 02 21 F3 01
0020  CD 2B 00 BD 28 FA 2D 77 20 F6 E9 DB 00 E6 01 20
0030  FA DB 01 C9
```

2. Set the sense switches to the page address where you wish the Monitor to reside.

3. Hit RESET on the processor.

4. Place the tape in the reader device.

5. Start the reader, and WHILE the initial pattern of "1110011" is being read THEN:

6. Hit RUN on the processor.

After the program loads, at the end of the tape it will "sign on" to the console device. If you are using non-standard I/O, (that is, not as per the standard already

stated) it will be sitting in a loop, attempting to sign on. At this time, stop the processor, and modify the routines to handle your own I/O set-up.

This short program actually loads a larger, more sophisticated loader which actually performs the relocating. It is also a checksumed loader. If while loading a checksum error is detected, the Programmed Output lights on an IMSAI will all flash at a rate of about 1 HZ. The machine may be stopped and the tape backed up to an area before the error was detected, (2-3 feet) and the machine reset and started from zero. If you do this, do not change the sense switches.

The relocating checksumed loader listing is given later in this manual. It is similar to the "R" command in the monitor, the primary difference being that the loader at the beginning of the tape gets its address reference from the sense switches. That of the Monitor itself gets the data from the operator console.

COMMANDS

The following is a list of commands for the Zapple Monitor. Precise definitions and usage notes are covered in the next section.

A - ASSIGN reader, punch, console or list device options from the console.
B - BYE (system shut down).
C - COMPARE the contents of memory with the reader input and display any differences.
D - DISPLAY the contents of any defined memory area in Hex.
E - END OF FILE statement generator.
F - FILL any defined area of memory with a constant.
G - GOTO an address and execute. With breakpointing.
H - HEX MATH. Gives the sum and difference of two Hex numbers.
I * USER DEFINED.
J - JUSTIFY MEMORY - a non-destructive test for hard memory failures.
K * USER DEFINED.
L - LOAD a binary file.
M - MOVE a defined memory area to another starting address.
N - NULLS to the punch device.
O * USER DEFINED.
P - PUT ASCII characters into memory from the keyboard.
Q - QUERY I/O ports - may output or input any value to or from any I/O port.
R - READ a Hex file. Performs checksum, relocating, offsetting, etc.
S - SUBSTITUTE and/or examine any value at any address (in hex).
T - TYPEs the contents of a defined memory block in their ASCII equivalent.
U - UNLOAD a binary tape to the punch device.
V - VERIFY the contents of a defined memory block against that of another block and display the differences.
W - WRITE a checksummed hex file to the punch device.
X - eXAMINE and/or modify any or all registers including the special Z-80 registers.
Y - "Yis there". Search memory for defined byte strings and display all the addresses where they are found.
Z - "Z end". Locate and display the highest address in memory.

D. COMMAND SET USAGE

The following section lists the commands, and describes their format and their use. It should be noted that the Zapple Monitor recognizes both upper and lower case letters for its commands, and that in general, a command which is printing can be stopped with a CONTROL C, which is checked during a carriage return - line feed sequence. The following EXAMPLES show a comma [,] as a delimiter between parameters, however a space may also be used. If an error is made while inputting a command from the keyboard, it may be terminated by a rubout and the command re-typed. An asterisk is displayed indicating an ABORT of some kind.

COMMAND — DESCRIPTION

A — ASSIGNMENT OF I/O DEVICES: The monitor system is capable of supporting up to 4 logical devices, these being: The CONSOLE, The READER, the PUNCH, and the LIST DEVICE. To these may be connected 4 different actual I/O devices, for a total of 16 direct combinations of I/O device and function. The specific permutations are:

LOGICAL DEVICE	ASSIGNED DEVICES
CONSOLE	TTY CRT BATCH USER (user defined)
READER	TTY CASSETTE PAPER (HIGH SPEED READER user written) USER (user defined)
PUNCH	TTY CASSETTE PAPER (HIGH SPEED PUNCH user written) USER (user defined)
LIST DEVICE	TTY CRT LINE PRINTER (user written) USER (user defined)

The default mode for each logical device is always the teleprinter.

Assignments are made using the following format:

EXAMPLE: AC=C(cr)

assigns the console equal to the Crt (video terminal) device. similarly:

EXAMPLE: AR=T(cr)

assigns the reader device to be the teleprinter.

While performing a command which requires a reader input (C,L,R), if the assigned reader is the Teleprinter, the software will look for a character from the TTY input. If a character is not recieved within a few seconds, it will ABORT, printing an asterisk [*], and return to the command mode. Similarly, if the assigned reader is the Cassette device, and you WISH to abort for some reason, changing the position of any of the SENSE switches will force an ABORT. On the external reader routines, returning with the carry set indicates an abort (or OUT OF DATA) condition.

When assigning a device, only the first letter initial of its name is required.

The Monitor itself is set-up to support the TTY, CRT and Cassette routines. The other assignments require the addition of user's routines. These are addressed via the commands, which vector to starting addresses.

EXAMPLE: AL=L(cr)

assigns the list device to be the line printer. It vectors to (start address) +812H, or 12H above the end of the monitor. That would be the address for the line printer routine. For details of these arrangements, see the Source Documentation.

Within the above, the assign console equals batch "AC=B(cr)" deserves further mention. In BATCH mode, the READER is made the Keyboard input, and the LIST DEVICE is made the console output. This allows the running of a job directly from the reader input, with the result being output to the list device.

A typical use of this assignment would be the reconstruction of a lengthly text editing job where the text and your editing commands have all been saved on paper tape. With the BATCH MODE, you may assign the reader equals the TTY, the List device equals the TTY, and Console equals BATCH. Running the tape through the reader is the same as you redoing the entire text editing by hand, and the output will go to the TTY and be printed. On a very lengthly job, you could even start the process, and go away until it's done. Its usefullness is limited only by your imagination.

B BYE. This command completly shuts down the system. It is useful where children might have access to the system, where a telephone communications link is established under remote control, or anytime when the operator wishes to make the system inaccessible to unauthorized use.

EXAMPLE: B

completly kills the keyboard. Recovery from the shut-down is accomplished simply by inputting a CONTROL-SHIFT N from the keyboard. (ASCII equivalent is a Record Separator - "RS"; HEX character is a 1EH.) The monitor will sign on and print a greater-than sign (>), however the register storage area will not be cleared.

C COMPARE the reader input with memory. This command is useful for verifying correct loads, verifying that a dumped tape matches with its source etc.

EXAMPLE: C1000,2000(cr,start reader)

compares the memory block 1000H to 2000H with the input from the reader device.

For those with automatic readers, the operation is very simple. Assign the Reader equal to the device you wish to enter the data against, type C(starting address),(ending address)(cr), and the reader will start. The first character read by the reader will be the one matched with the starting address. If any discrepencies are encountered, the reader will stop, and the address (in hex) of the error will be printed on the display. The reader will restart, and continue in this fashion until the entire tape is compared.

If your reader cannot operate automatically, start the reader manually. If an error is encountered, however, while the incorrect address is being printed, the reader will continue, and get "out of sync" with the compare action. Therefore, it is necessary to manually stop the reader if an error is encountered, and manually reposition the tape to the byte following the error. (An excellent article on how to convert ASR33 type readers to automatic operation was recently presented in INTERFACE magazine.)

D DISPLAY memory contents. This command displays the contents of memory in Hex. Memory is displayed

16 bytes per line, with the starting address of the line given as the first piece of data on the line.

EXAMPLE: D100,1FF(cr)

will display in hex the values contained in the memory block 100H to 1FFH.

E END OF FILE. This command generates the end of file pattern for the checksum loader. It is used after punching a block of memory to the punch device using the "W" command. An address parameter for the end of file may be given if so desired.

EXAMPLE: E(cr)

will generate an "end of file marker".

EXAMPLE: E100(cr)

generates the EOF marker with the address parameter "100H". When loading such a file, upon completion, the address contained in the End of File will be placed in the "P" register. Execution of the program may then be initiated by typing "G(cr)".

F FILL command. This command fills a block of memory with a specific value. It is quite handy for initializing a block to a specific value (such as for tests, zeroing memory when starting up, etc.) *NOTE: Avoid doing this over the monitor's stack area. This area may be determined as being between the value you get when typing the Z command, and the value in the S register upon sign-on. It is approximately 60H bytes below the "Top of memory" (Z).

The format for the command is:

EXAMPLE: F100,1FF,FF

fills memory block 100H to 1FFH with the value FFH.

G GOTO command. This command allows the user to cause the processor to GOTO an address and execute the program from that address. In the actual performing of the G command, a program, which has been placed in the stack area during the sign-on of the monitor, is executed. This program will first take all of the values in the register storage area (displayed with the X command), and stuff them in their correct registers in the CPU, and finally JMP to the program address being requested by the

operator. If this short program up in the stack has been destroyed (as a result of a "blow-up", or the F or M commands, etc.) the monitor will not be able to GO anywhere, and a manual restart of the monitor will be required. Whenever the monitor is restarted at the initialization point (first address I.E. 0F000H), the contents of the registers are set to ZERO with the exception of the S (stack), which contains a valid stack address. This actual value depends on the amount of memory in the system, etc. In its simplest form, the letter "G" accompanied by a parameter causes the processor to go to that address and start execution.

EXAMPLE: G1000

would cause the processor to goto address 1000(H) and execute from that address.

Additionally, one or two breakpoints may be set.

EXAMPLE: G1000,1005,1010

would cause the program to start execution at address 1000H, and IN THE EVENT that the program gets to address 1005, OR 1010, the program will stop execution, and return to the monitor, printing an "at" sign, and the address of the breakpoint that was executed. (I.E. @1010) It then prints the ">" prompt, awaiting further instructions. This action also cancels any breakpoints previously set.

Breakpoints must be set at locations containing an instruction byte. This is a SOFTWARE breakpoint system, and requires either RAM at RST 7 (restart 7, addr. 0038H), or if using ROM, a permanent JMP to the monitor TRAP address (0F01EH) at 0038H. Remember, this is a SOFTWARE breakpoint system, and the program being debugged must be in non-protected Read/Write memory.

```
EXAMPLE:    *C2   JNZ   1234H
             34
             12
            *3E   MVI   A,CR
             0D
            *21   LXI   H,1000H
             00
             10
            *77   MOV   M,A
            *23   INX   H
            *CD   CALL  5678H
             78
             56
```

The asterisks (*) mark the bytes that may be used as breakpoints.

H HEX MATH. This command allows the execution of hexidecimal arithmetic directly from the console. it will give the sum and difference of any two hex numbers entered.

```
EXAMPLE:    H1000,1010(cr)
            2010 FFF0
            >
```

2010H being the sum, and FFF0 being the difference of the two hex values.

J The J command is a non-destructive memory test. The command reads any given byte, complements it, writes into the location the complement, compares the complement with the accumulator, and rewrites the original byte into the location. The command is used with two parameters, delineating the block of memory to be checked.

```
EXAMPLE:    J1000,1FFF
```

would perform the above test on the block 1000H to 1FFFH.

If errors are detected, the address at which the error is found and the error are displayed on the console before the test is continued.

```
EXAMPLE:    J1000,1FFF(cr)
            1F00  00001000
            >
```

would indicate that the 4th bit (D3) at location 1F00H did not correctly complement itself.

This test is useful for the discovery of hard memory failures, and also serves as a quick check for accidentally protected memory. A fully protected memory block would print out as entirely "1s". (11111111)

L LOAD BINARY FILE. This command loads a binary file from either a cassette or paper tape.

```
EXAMPLE:    L1000(cr)
```

would load the tape at address 1000H. This would require that the program be an absolute program, designed for address 1000H. The start-of-file mark (automatically generated by the "U" command) is a series of 8 0FFH's (rubouts). When this is detected at the start of file, the bell will ring on the TTY to indicate the start of the load process. When the end-of-file is detected (again, a series of 8 rubouts) the load is terminated, and the address of

the NEXT location that would have been loaded is printed on the console. There are two constraints on this type of file system. The middle of the program cannot contain more than 6 0FF's (11111111) in a row (an unusual occurence), and if 0FFH is the LAST data byte in the file, it will be ignored. This too is unusual, and only a minor inconvenience.
Binary programs loaded at other than their design address will not run. The "L" command does not perform checksum functions, and cannot handle relocatable files. This is a pure and simple byte-for-byte binary loader (see "U" command).

M MOVE COMMAND. This command is used to move a block of memory from one location to another. The original block is NOT affected by the move, remaining intact so long as the block moved into does not overlap with the block currently occupied. This command, like the "F" command should be used with some caution as moving a block into an area occupied by the stack, or the program or the monitor will cause unpredictable results.

EXAMPLE: M1000,1FFF,2000(cr)

moves the contents of memory contained in the block 1000H to 1FFFH to a starting address of 2000H. The new block has the limits 2000H to 2FFFH.

This command is very useful for working on programs without destroying the original, verifying blocks of memory loaded with existing memory, etc.

N NULL. This command punches nulls to the punch device. 72 nulls are punched whenever the command is used. It may be used repetitively for any desired leader length.

EXAMPLE: (N)
*Note: The "N" or "n" will NOT echo, so as to not spoil the paper tape.

It will punch 72 nulls to the punch device.

P PUT ASCII characters into memory. This command allows ASCII characters to be written directly into memory. It is useful for placing labels in files etc.

EXAMPLE: P1000(cr)

activates the command, and any further inputs via the keyboard would be placed into memory in their ASCII equivalent. The command is terminated by a CONTROL D character, with the address of the

location following the last entry printed on the console (the Control-D is NOT stored). Recovery of the input data is affected by use of the "T" or "U" command.

Q QUERY INPUT/OUTPUT PORTS. This command allows any value to be output to any I/O port, and allows the value in binary on any I/O port to be read on the console.

EXAMPLE: Q01,7(cr)

would output an ASCII "7" to I/O PORT 1. (ASCII seven is a "bell" so on a TTY, the bell would ring.)

EXAMPLE: QI1(cr) 00001101

inputs the value at port 1, in the illustration above, we see that bits 0,2 and 3 are high, the others low. This is useful for observing the condition of status bits and other diagnostic activities.

R READ A CHECKSUMMED HEX FILE. This command reads checksummed hex files in the INTEL format, as well as being capable of loading the relocatable TDL files at any selected address and bias offset. When reading an ABSOLUTE file (INTEL format), there may be only a BIAS added. These files cannot be relocated. The format is: R[bias],[relocation](cr).

If a checksum error or a failure to write the data to memory occurs, the loading process is stopped, an asterisk is printed (indicating some error condition), and the address that was attempting to be written will be displayed on the console device. This is to assist in determining the failure.

EXAMPLE: R(cr, start reader)

will load a hex file at its absolute address.

EXAMPLE: R,1000(cr,start reader)

will load a TDL relocatable hex file at address 1000H and modify the program to run at address 1000H.

EXAMPLE: R1000,100(cr,start reader)

loads the file set up to run at 100H, but with a positive BIAS of 1000H added to it. Thus, the file, set up to run at 100H will be loaded at 1100H.

EXAMPLE: R1000(cr)

will load the file, set up to run at address 0000H, at address 1000. In other words, using the TDL relocating format, you may load any program, to execute anywhere in memory, anywhere in memory. (Think about it.....)

S SUBSTITUTE and examine. This command allows any address in memory to be examined directly, and allows substituion of one value for another at that address if desired.

EXAMPLE: SF810(sp)00-(sp)1A-(sp)C3-(sp)(cr)
>

In this case the "S" command examines address F810H. The hitting of the space bar (sp) displays the value at that address. (assuming value 00H at that address.) Hitting the space bar again displays the NEXT location in memory (F811H), and so forth. Simply typing S(sp) starts display from address 0000H. By repetitive typing of (sp), all of memory could be displayed one address at a time.

EXAMPLE: SF810(sp)00-(kb)FF(cr)

This command examines address F810H, showing the value 00H at that address. Immediately typing in FFH from the keyboard SUBSTITUTES FFH for 00H at that address. Repeating the example above would show:

EXAMPLE: SF810(sp)FF-

When an address is being examined, the address being examined may be moved BACKWARD by entering a backarrow (ba) or SHIFT-O, or underline, depending on the terminal used.

EXAMPLE: SF810(sp)00-(ba)AA-

shows that at address F80FH, the value AA exists. Typing a space bar will examine F810H again.

T TYPE ASCII characters from memory. This command allows the contents of memory to be displayed in their ASCII equivalents. All non-printing characters will be displayed as periods [.]. It is may used to display the results of the "P" command which allows keyboard entry of ASCII characters directly into memory. Also useful for finding text strings and messages in software. The initial address is first displayed, then the first 64 characters, the next address, etc. until the upper limit has been reached.

EXAMPLE: T1000,2000(cr)

displays the ASCII equivalents of memory locations 1000H to 2000H. If the "P" command had been used to place a "message" into memory somewhere in that memory block, it would soon be apparent on the console display.

U UNLOAD BINARY. This command simply dumps core to the punch device. It may be used with a cassette system as well, with no start-up problems. It does not generate a checksum. The format which is generated will be a leader, eight 0FFHs, binary data, eight 0FFHs, and a trailer. The 0FFHs are "rubouts" and are called file cues. These are detected and counted to determine the start and the end of files.

EXAMPLE: U00,FF(cr,start reader)

will generate a binary tape, formated as described above, of the values contained in memory locations 00H to FFH.

V VERIFY. This command allows the user to verify the contents of one memory block against the contents of another memory block. This is very useful for functions such as verifying that a file generated from a program is a duplicate of the actual program, etc.

EXAMPLE: V1000,2000,3000

will compare the contents of the memory block 1000H to 2000H against the contents of the memory block commencing at 3000H and extending to 4000H. Any differences will be displayed.

EXAMPLE: V1000,2000,3000
100F 00 FF

indicates that the contents of address 100FH is a 00 while that at 300FH is an FF.

W WRITE Hex file. This command dumps memory to the punch device in the standard "Intel-style" hex file format. Both start and end of file parameters are required. The proper "end of file" (EOF) is generated by the "E" command.

EXAMPLE: W00,FF(cr,start punch)
(after punching)
E(cr)

will generate a checksummed hex file of the values in the memory block 00H to FFH. If the assigned punch and console are the same, the program will pause and wait for the operator to turn on the punch (ASR33, etc.). Use of the "N" command at either the beginning and/or end of the file is optional, but recommended.

X eXAMINE REGISTERS. The "X" command allows the user to examine and/or modify all of the Z80 registers.

A=Accumulator
B,C,D,E,H,L=CPU REGISTERS
M=Memory (pointed to by H&L)
P=Program Counter (PC)
S=Stack Pointer (SP)
I=Interrupt Register
X=Index (IX)
Y=Index (IY)
R=Refresh Register

EXAMPLE: X(cr)

displays the contents of MAIN registers A,B,C,D,E,F,H,L,M,P,S and I, in hex.

EXAMPLE: X'(cr)

displays the contents of PRIME registers A,B,C,D,E,F,H,L,M,X,Y and R.

Typing the letter "X" (or X'), followed by a specific register letter will display the contents of that register. Entering a new value via the keyboard (kb) will substitute the new value in the specific register. Hitting the space bar will display the next register in which you may then perform substitutions, etc. In the unique case of the "M" register, you may modify the 16 bit pointer (H&L) to that memory location.

```
EXAMPLE:    XA 00-(kb)FF(cr)
            XA FF-(sp)00-(kb)FF(cr)
            XA FF-(sp)FF-(cr)
            >
```

first examines the contents of register "A" (00H), then substitutes an FF. In the next line, the FF is displayed, a space character displays the next register (again a 00H), and substitutes an FF for this value. The last line displays both registers as containing FFHs.

Y SEARCH. This command allows unique byte strings, from one up to 255 bytes to be searched for in

memory, and the addresses where they are found to be displayed. It is advisable to search for unique patterns rather than single bytes. The search operation may be stopped with a control-C.

```
EXAMPLE:    YC3,21,F3,01(cr)
            0081
            00B2
            0F08
            >
```

indicates that the byte string (in hex) C3, 21, F3, 01, is found in memory at locations 0081H, 00B2H and 0F08H. This routine will search all 65-K of memory for a unique sequence of bytes in less than one second.

Z Z TOP OF MEMORY. This command locates and gives the highest address of available memory in your system.

```
EXAMPLE:    Z
            7FFF
            >
```

indicates that the highest available memory is at address 7FFFH. Note that NO carriage return is required. Also, If only one 1K board were in the system, and it was adressed to have its top byte at address 7FFFH, the Z command would so indicate regardless of the absence of lower memory.