8086 Monitor

For Use with the SCP 300 CPU Support Board
CONTENTS

Getting Started .............................................. 3
Parameters .................................................... 4
Commands ..................................................... 4
  Boot ......................................................... 4
  Dump ......................................................... 5
  Enter ........................................................ 6
  Fill .......................................................... 6
  Go ............................................................ 7
Input ........................................................... 7
Move ........................................................... 8
Output ........................................................ 8
Register ....................................................... 8
Search ......................................................... 9
Trace .......................................................... 10

8086 Monitor Assembly Listing ............................ 11
Getting Started

Connect an RS-232 terminal to the cable coming from J1 of the CPU support card. The terminal should be set for full duplex at one of the following rates: 19200, 9600, 1200, 300, 150, or 110 baud. The software-selected baud rate feature of the CPU Support card is used to automatically determine the baud rate of the terminal. By hitting the carriage return no more than four times, the sign-on message should appear. If it does not, reset the computer and try again. If it still does not sign on, check all connections carefully.

If Sense Switch 0 is a one (position 1 of S2 is closed), then the monitor will NOT sign on after baud rate selection but instead will automatically boot the disk. This is equivalent to the Boot command with no parameters.

Directly below the sign-on message there will be a greater-than symbol, ">". This is the Monitor prompt, and indicates that the Monitor is ready to accept a command. The input buffer allows commands of up to 80 characters in length. While typing the command line, <backspace> and <rubout> or <delete> may be used back up to correct a mistake, while "@" cancels the line and re-issues the prompt. Typing <carriage return> either causes the command to be executed or an error to be reported. Most errors are syntax errors, and an arrow followed by the word "Error" will appear under the first bad character. If an error occurs, no part of the command is executed (except during boot or flag replacement - see Boot and Register commands).

Monitor commands are available to display, alter and search memory; to do inputs and outputs; to boot the disk; and to aid in debugging 8086 programs. The debugging commands allow the user to execute a program in a controlled manner, observing its behavior. This controlled execution may be done either by single-stepping or through execution with breakpoints.

Single-stepping is done with the Monitor's Trace command. By using 8086 hardware trace mode, a single instruction can be executed, and the resulting effects on the registers or memory displayed. Even ROM may be traced, and every instruction is traced correctly (unlike 8080 or Z80 debuggers).

Execution of any command may be aborted by typing Control-C. Typing Control-S during output will cause the display to pause so it may be read before scrolling away; any key (except Control-C) may be typed to continue.

If a user program is executing as a result of a Boot or Go command and interrupts are enabled, then the console may interrupt the program and return control to the Monitor. Typing any key will cause the interrupt, save program status, and print a register dump; except that Control-C will inhibit the register dump. Note that complete program status is always saved, and execution may be continued with a Go or Trace command.

The Monitor requires 5K of memory at address zero. Specifically, interrupt vectors are kept at locations 4-7, 0CH-0FH, and 64H-67H, while scratch pad ram, input buffer, and stack use less than 256 bytes beginning at 100H. User programs must not modify these locations if the Monitor is to be used for debugging.
Parameters

All commands of the Monitor accept one or more parameters on the line following the command letter. These parameters MAY be separated from each other and the command letter by spaces or commas, but one these delimiters is REQUIRED only to separate consecutive hex values. Most parameters are one of the following types:

<BYTE>, <HEX4>, <ADDRESS> - A hexadecimal number with no more than 2, 4, or 5 digits, respectively. Thus, <BYTE> becomes an 8-bit value, <HEX4> a 16-bit value, and <ADDRESS> a 20-bit value. If too many digits are entered or a non-hex character is typed, the error arrow will point to the mistake. Hex A-F must be in upper case.

<RANGE> - A <RANGE> is either <ADDRESS> <ADDRESS> or <ADDRESS> L <HEX4>. The first form specifies the first and last addresses affected by the command. The second form specifies a starting address and a length. For either form, the maximum length (first address - last address + 1) cannot exceed 10000H, and this limit may be as low as 0FF1H due to limitations of working within a segment. (Specifically, [starting address modulo 16] + length must be ≤ 10000H.) An "RG Error" results if the length is too large. To specify a length of 10000H with only four digits, use a length of zero. Note that the 'L' in this form must be in upper case.

<List> - This is always the last parameter on a line and may extend to the end of the input buffer. It is actually a series of one or more parameters, each of which is either a <BYTE> or a <STRING>.

A <STRING> is any number of characters (except control characters) enclosed by either single (') or double (") quotes. Since the opening and closing quotes must be the same, the other type may appear in the string freely. If the same quote as opened the string needs to appear within it, it must be given as two adjacent quotes. The ASCII values of the characters in the string are used as a list of bytes.

Commands

A command is executed by typing the first letter of its name (upper case only) followed by any parameters. If the first letter on the line is not recognized as a command, the error arrow will point to it. Commands are listed below in alphabetical order, with the forms of all parameters shown.

B
B <ADDRESS> . . . <ADDRESS>

Boot - Loads the first sector of track 0 of the disk into memory starting at 200H. Up to ten 5-digit addresses may be specified; too many will cause a "BP Error". After the sector is loaded, breakpoints will be set at these locations. Then all registers will be set from the register save area, except that the Code Segment will be set to zero, and the Instruction Pointer will be set to 200H - thus a jump will be made to 200H. The user stack pointer MUST be valid for this command to work. See Go command for more information.

This command works in three steps. First, the disk sector is loaded. Next, the Code Segment and Instruction Pointer are set in the register save area. Finally, a Go command is executed. The result is that an error in a breakpoint address will not be found until AFTER the sector is loaded and the register save area changed. Thus it is not necessary to use another Boot command to correct the error; a Go command with the corrected breakpoints will do.
The example below shows how Boot can help test an experimental 8086 program. The program to be tested fits into one 128-byte sector and has been placed on track 0, sector 1 of a disk. The program is loaded with the Boot command but execution does not begin because a breakpoint is set at 200H, the first byte of loaded program. Before testing, the program is moved to 400H, just above the interrupt table, and CS and IP are adjusted.

SCP 8086 Monitor 1.4

> B200

AX=0000 BX=0000 CX=0000 DX=0000 SP=0C00 BP=0000 SI=0000 DI=0000
DS=0040 ES=0040 SS=0040 CS=0000 IP=0200 NV UP EI PL NZ NA PD NC

> M 200 L80 400

> RCS

CS 0000

:40

> R IP

IP 0200

:0

> R

AX=0000 BX=0000 CX=0000 DX=0000 SP=0C00 BP=0000 SI=0000 DI=0000
DS=0040 ES=0040 SS=0040 CS=0040 IP=0000 NV UP EI PL NZ NA PD NC

D <ADDRESS>

D <RANGE>

Dump - Displays memory contents in hex and ASCII. If only a starting address is specified, 80H bytes are dumped; otherwise the specified range is displayed. To help pinpoint addresses, each line (except possibly the first) begins on a 16-byte boundary, and each 8-byte boundary is marked with a '-' in the ASCII dump.
E <ADDRESS> <LIST>

Enter - In the first form, the list of bytes is entered at the specified address, with the command being executed and completed upon hitting <carriage return>. If an error occurs, NO locations are changed.

The second form puts the Monitor into 'Enter Mode', starting at the specified address. After hitting <carriage return>, the address and its current contents will be displayed. The user now has several options:

1) Replace the displayed value with a new value. Simply type in the new value in hex, using <backspace> or <delete> to correct mistakes. If an illegal hex digit is typed or more than two digits are typed, the bell will sound and the character will not be echoed. After entering the new value, type either <space>, '->', or <carriage return>, as defined below.

2) Type <space> to display and possibly replace the next memory location. Every 8-byte boundary will start a new line with the current address.

3) Type '->' to backup to the preceding memory location. This will always start a new line with the address. The '-' will not be echoed.

4) Type <carriage return> to terminate the command.

> E500 24,9, A 'Test', 0
> D 500 L10
00500 24 09 0A 54 65 73 74 00-00 20 00 00 00 40 01 00 Test...
>
> E508
00508 00.
00507 00.
00506 74. 00.49
00508 00.4E 20.47 00.0 00.0 00.0 40.0 01.0 00.
00510 60. 01. 01.76 00.
> D500 513
00500 24 09 0A 54 65 73 74 49-4E 47 00 00 00 00 00 00 $..TestING......
00510 60 01 76 00
>

F <RANGE> <LIST>

Fill - The specified range is filled with the values in the list. If the list is larger than the range, not all values will be used; if the range is larger, the list will be repeated as many times as necessary to fill it. All memory in <RANGE> must be valid for this command to work properly. If bad or non-existent memory is encountered, the error will be propagated into all succeeding locations.

> F400 L28 'Help' A D
> D400 L30
00400 48 65 6C 70 0A 0D 48 65-6C 70 0A 0D 48 65 6C 70 Help..Help..Help
00410 0A 0D 48 65 6C 70 0A 0D-48 65 6C 70 0A 0D 48 65 ..Help..Help..Help
00420 6C 70 0A 0D 48 65 6C 70-FF 7F FF FF FF FF FF FF FF FF FF F7 FF 1F..Help......W.
Go - Sets all registers from the register save area. Since this includes the Code Segment and Instruction Pointer, this implies a jump to the program under test.

This command allows setting up ten breakpoints. Attempting to set more than ten will cause a "BP Error". Breakpoints may be set only at an address containing the first byte of an 8086 opcode. A breakpoint is set by placing an interrupt opcode (OCCH) at the specified address. When that opcode is executed, all registers are saved and displayed, and all breakpoints locations are restored to their original value. If control is not returned to the Monitor by a breakpoint or interrupt, the breakpoints will not be cleared.

The user stack pointer must be valid and have 6 bytes available for this command to work. The jump to the user program is made with an IRET instruction with the user stack pointer set and user Flags, Code Segment register, and Instruction Pointer on the user stack. Thus if the user stack is not valid, the system will "crash".

The program below is an infinite loop of 16 INC AX instructions followed by a jump to its start. First breakpoints are used to execute a few instructions. Then a Go without breakpoints allows continuous, full-speed execution which is terminated by an interrupt from the keyboard - in this case, typing the space bar.

```
> F400 L10 40
> E410 EB EE
> D400 L12
00400 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40
> G410

AX=0010 BX=0000 CX=0000 DX=0000 SP=0C00 BP=0000 SI=0000 DI=0000
DS=0040 ES=0040 SS=0040 CS=0040 IP=0010 NV UP EI PL NZ AC PO NC
> G400 412

AX=0010 BX=0000 CX=0000 DX=0000 SP=0C00 BP=0000 SI=0000 DI=0000
DS=0040 ES=0040 SS=0040 CS=0040 IP=0000 NV UP EI PL NZ AC PO NC
> G

AX=4590 BX=0000 CX=0000 DX=0000 SP=0C00 BP=0000 SI=0000 DI=0000
DS=0040 ES=0040 SS=0040 CS=0040 IP=0000 NV UP EI PL NZ AC PE NC
> 1 <HEX4>

Input - Inputs a byte from the specified port and displays it. A 16-bit port address is allowed.
```
M <RANGE> <ADDRESS>

Move - Moves the block of memory specified by <RANGE> to <ADDRESS>. Overlapping moves are always performed without loss of data, i.e., data is moved before it is overwritten. To do this, all moves from higher addresses to lower ones are done front-to-back, while moves from lower addresses to higher ones are done back-to-front.

> M400 L10 420
> D400 42F
00400 54 45 53 54 49 4E 47 FF-F7 FF FF F6 FF FE FF TESTING.w...v...
00410 FF FF FE FF FF FF FF FF FF FF FF FE FF FE FF TESTING.w...v...
00420 54 45 53 54 49 4E 47 FF-F7 FF FF F6 FF FE FF TESTING.w...v...
>
> M404 40F 405
> D400 L10
00400 54 45 53 54 49 4E 47-FF F7 FF F6 FF FE TESTING.w...v...
>
> M405 410 404
> D400L10
00400 54 45 53 54 49 4E 47 FF-F7 FF F6 FF FE TESTING.w...v...
>
O <HEX4> <BYTE>

Output - <BYTE> is sent to the specified output port. A 16-bit port address is allowed.

R
R <REGISTER NAME>

Register - with no parameters, this command dumps the register save area.

Giving a register name as a parameter allows that register to be displayed and modified. The register name may be AX, BX, CX, DX, SP, BP, SI, DI, DS, ES, SS, CS, IP, PC, or F (upper case only); anything else will result in an "BR Error". IP and PC both refer to the Instruction Pointer and F refers to the Flag register. For all except the Flag register, the current 16-bit value will be printed in hex, then a colon will appear as a prompt for the replacement value. Typing <carriage return> leaves the register unchanged; otherwise type a <HEX4> to replace.

The Flag register uses a system of two-letter mnemonics for each flag, as shown below:

<table>
<thead>
<tr>
<th>FLAG</th>
<th>CLEAR</th>
<th>SET</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overflow</td>
<td>NV No Overflow</td>
<td>OV Overflow</td>
</tr>
<tr>
<td>Direction</td>
<td>UP Up (Incrementing)</td>
<td>DN Down (Decrementing)</td>
</tr>
<tr>
<td>Interrupt</td>
<td>DI Disabled Interrupts</td>
<td>EI Enabled Interrupts</td>
</tr>
<tr>
<td>Sign</td>
<td>PL Plus</td>
<td>NG Negative</td>
</tr>
<tr>
<td>Zero</td>
<td>NZ Not Zero</td>
<td>ZR Zero</td>
</tr>
<tr>
<td>Auxiliary Carry</td>
<td>NA No Auxiliary Carry</td>
<td>AC Auxiliary Carry</td>
</tr>
<tr>
<td>Parity</td>
<td>PO Parity Odd</td>
<td>PE Parity Even</td>
</tr>
<tr>
<td>Carry</td>
<td>NC No Carry</td>
<td>CY Carry</td>
</tr>
</tbody>
</table>
Whenever the Flag register is displayed, all flags are displayed in this order. When the F register is specified with the R command, the flags are displayed and then the Monitor waits for any replacements to be made. Any number of two-letter flag codes may be typed, and only those flags entered will be modified. If a flag has more than one code in the list, a "DF Error" (Double Flag) will result. If any code is not recognized, a "BF Error" (Bad Flag) will occur. In either case, those flags up to the error have been changed, and those after the error have not.

After reset, all registers are set to zero except the segment registers, which are set to 40H, and the Stack Pointer, which is set to 0000H. Flags are all cleared except for interrupts. Execution on a Trace or Go command would thus begin at 400H, which is the first location after the interrupt table.

> R
AX=0000 BX=0000 CX=0000 DX=0000 SP=0C00 BP=0000 SI=0000 DI=0000
DS=0040 ES=0040 SS=0040 CS=0040 IP=0000 NV UP EI PL NZ AC PE NC

> RF
DS=0040 ES=0040 SS=0040 CS=0040 SP=0C00 BP=0000 SI=0000 DI=0000
AX=0106 BX=0000 CX=0000 DX=0000 SP=0C00 BP=0000 SI=0000 DI=0000
DS=0040 ES=0040 SS=0040 CS=0040 IP=0000 NV UP EI PL ZR AC PE NC

S <RANGE> <LIST>
Search - The range is searched for a byte or string of bytes specified by <LIST>. For each occurrence the first address of the match is displayed.

>S 400 L8000 'Help'
00400
00406
0040C
00412
00418
0041E
00424
D 400 L28
00400 48 65 6C 70 0A 0D 48 65-6C 70 0A 0D 48 65 6C 70
00410 0A 0D 48 65 6C 70 0A 0D 48 65 6C 70
00420 6C 70 0A 0D 48 65 6C 70

- 9 -
Trace - The number of instructions specified (default 1) are traced. After each instruction, the complete contents of the registers and flags are displayed. (For the meaning of the flag symbols, see Register command.) Since this command uses the hardware trace mode of the 8086, even ROM may be traced.

> R
AX=0106 BX=0000 CX=0000 DX=0000 SP=0C00 BP=0000 SI=0000 DI=0000
DS=0040 ES=0040 SS=0040 CS=0040 IP=0000 NV DN EI PL ZR AC PE NC
>T
AX=0107 BX=0000 CX=0000 DX=0000 SP=0C00 BP=0000 SI=0000 DI=0000
DS=0040 ES=0040 SS=0040 CS=0040 IP=0001 NV DN EI PL NZ NA PO NC
>T
AX=0108 BX=0000 CX=0000 DX=0000 SP=0C00 BP=0000 SI=0000 DI=0000
DS=0040 ES=0040 SS=0040 CS=0040 IP=0002 NV DN EI PL NZ NA PO NC
>T4
AX=0109 BX=0000 CX=0000 DX=0000 SP=0C00 BP=0000 SI=0000 DI=0000
DS=0040 ES=0040 SS=0040 CS=0040 IP=0003 NV DN EI PL NZ NA PE NC
AX=010A BX=0000 CX=0000 DX=0000 SP=0C00 BP=0000 SI=0000 DI=0000
DS=0040 ES=0040 SS=0040 CS=0040 IP=0004 NV DN EI PL NZ NA PE NC
AX=010B BX=0000 CX=0000 DX=0000 SP=0C00 BP=0000 SI=0000 DI=0000
DS=0040 ES=0040 SS=0040 CS=0040 IP=0005 NV DN EI PL NZ NA PO NC
AX=010C BX=0000 CX=0000 DX=0000 SP=0C00 BP=0000 SI=0000 DI=0000
DS=0040 ES=0040 SS=0040 CS=0040 IP=0006 NV DN EI PL NZ NA PE NC
>
; Seattle Computer Products 8086 Monitor version 1.5 4/24/80
; by Tim Paterson
; This software is not copyrighted.

; To select a disk boot, set one of the following equates
; to 1, the rest to 0.

CROMECO4EDC: EQU 0 ; 1 for 4EDC, 0 for others
NORTHSTARSD: EQU 0 ; North Star single density?
TARBELL: EQU 1 ; Tarbell (single or double)?
OTHER: EQU 0 ; User-defined disk

PUTBASE: EQU 100H
LOAD: EQU 200H
ORG 7FOH
PUT PUTBASE+7FOH
JMP 0, OFF80H ; Power-on jump to monitor

; Baud Rate Table. The 9513 divides 2MHz by these values.
; They are for 9600, 1200, 300, 150, 110 baud

ORG 100H ; RAM area base address

BASE: EQU 0FOH ; CPU Support base port address
STAT: EQU BASE+7 ; UART status port
DATA: EQU BASE+6 ; UART data port
DAV: EQU 2 ; UART data available bit
TBMT: EQU 1 ; UART transmitter ready bit
BUFLEN: EQU 80 ; Maximum length of line input buffer
BPMAX: EQU 10 ; Maximum number of breakpoints
BPLEN: EQU BMAX+BPMAX ; Length of breakpoint table
RECYCLE: EQU 14 ; Number of registers
SEGDIR: EQU 800H ; -OFF800H (ROM address)
PROMPT: EQU ">" ; Working stack area
CAN: EQU "@"

; RAM area.

BRKCT: DS 2 ; Number of breakpoints
TCOUNT: DS 2 ; Number of steps to trace
BPTAB: DS BUFLEN ; Breakpoint table
LINEBUF: DS BUFLEN+1 ; Line input buffer
ALIGN DS 50 ; Working stack area

; Register save area

STACK:

AXSAVE: DS 2
BXSAVE: DS 2
CSAVE: DS 2
DSAVE: DS 2
SSAVE: DS 2
ESAVE: DS 2
RSTACK: ; Stack set here so registers can be saved by pushing

- 11 -
FSAVE: DS 2

;Start of Monitor code
ORG 0
PUT PUTBASE

;One-time initialization
UP
XOR AX,AX
MOV SS,AX
MOV DS,AX
MOV ES,AX
MOV DI,AXSAVE
MOV CX,14
REP
STOW ;Set register images to zero
OR B,[FSAVE+1],2 ;Enable interrupts
MOV CL,4
MOV AL,40H
MOV DI,DSSAVE
REP
STOW ;Set segment reg. images to 40H
MOV B,[FSAVE+1],0CH ;Set user stack to 400H+0C00H
MOV SP,STACK

;Prepare 9513
MOV AL,17H
OUT BASE+5 ;Select Master Mode register
MOV AL,0F3H
OUT BASE+4 ;Low byte of Master Mode
MOV AX,584H ;Output 84H to BASE+4
OUTW BASE+4 ;and 05H to BASE+5

;Master Mode now set to 84F3H:
;Scaler set to BCD division
;Enable data pointer increment
;8-bit data bus
;FOUT=100Hz, dividing F5 by 4 (F5=4MHz/10000)
;Both alarm comparators disabled
;Time-of-day enabled
;Counter 5 selected

;Initialize loop. Ports BASE through BASE+7 are initialized
;from table. Each table entry has number of bytes followed by
;data.

MOV SI,INITTABLE ;Initialization table
MOV DX,BASE ;DX has (variable) port no.
INITPORT:
SEG CS
LODB ;Get byte count
JCXZ NEXTPORT ;No init. for some ports
SEG CL,AL
JGCXZ NEXTPORT ;No init. for some ports

INITBYTE:
SEG CS
LODB ;Get init. data
OUT DX ;Send to port
LOOP INITBYTE ;As many bytes as required

NEXTPORT:
INC DX ;Prepare for next port
CMP DL,BASE+8 ;Check against limit
JNZ INITPORT

;Initialization complete except for determining baud rate.
;Both 8259As are ready to accept interrupts, the 9513 is
;providing 19.2k baud X 16 to the 8251A which is set for
;16X clock and one stop bit.
CALL CHECKB ;Check for correct baud rate
;CHECKB does not return if baud rate is correct
; Initial baud rate (19.2k) was wrong, so run auto-baud routine

INITBAUD:
	MOV SI,BAUD
; First set up 9513 for slower baud rates (<=9600).
; Counter 5 mode register has already been selected.
	MOV AX,OE823H ; Output 23H to BASE+4
	OUTW BASE+4 ; and OE8H to BASE+5
; 23H to BASE+4 sets lower half of Counter 5 mode register.
; Reload from Load, count down repetitively in binary.
; Toggle output.
; OE8H to BASE+5 disables data pointer sequencing

; Select Counter 5 load reg.

INITB:
	SEG CS
	LODW ; Get divisor
	OUT BASE+4 ; Output low byte
	MOV AL,AH
	OUT BASE+4 ; Output high byte
	CALL CHECKB ; Check if baud rate correct
	JP INITB ; Try next rate if not

CHECKB:
	CALL IN ; First byte could be messed up
	CALL IN ; Get carriage return
	CMP AL,13 ; Correct?
	JZ MONITOR ; Don't return if correct
	RET ; Didn't get it yet

; Initialization complete, including baud rate.

MONITOR:
; Do auto boot if sense switch 0 is on.

DOMON:
	IN TEST JZ JMP B,[DI],13 BASE+OFH AL,1 DOMON

COMMAND:
; Re-establish initial conditions

UP
	XOR AX,AX
	MDV DS,AX
	MOV ES,AX
	MOV [64H],INT
	MOV [66H],CS
	MOV AL,PROMPT
	CALL OUT

CALL INBUF ; Get command line
; From now and throughout command line processing, DI points
to next character in command line to be processed.

CALL SCANB ; Scan off leading blanks

JZ COMMAND ; Null command?

MOV AL,[DI] ; AL=first non-blank character

; Prepare command letter for table lookup

SUB AL, "B" ; Low end range check

JC ERR1

CMP AL, "T"+1-H "B" ; Upper end range check

JNC ERR1

INC DI

SHL AL ; Times two

CBW

XCHG BX,AX ; In BX we can address with it
ERR1:  JMP  ERROR

; Get input line

INBUF:  MOV  DI, LINEBUF  ; Next empty buffer location
        XOR  CX, CX  ; Character count
GETCH:  CALL  IN  ; Get input character
        CMP  AL, 20H  ; Check for control characters
           JC  CONTROL  ; RUBOUT is a backspace
        CMP  AL, 7FH  
           JZ  BACKSP  
        CALL  OUT  ; Echo character
           JZ  KILL  ; Cancel line?
STOB:   JMP  GETCH  ; Put in input buffer
        CMP  CX, BUFLEN  ; Bump character count
           JBE  GETCH  ; Buffer full?
           JNO  BACKSP  ; Drop in to backspace if full
BACKSP:  JCXZ  GETCH  ; Can't backspace over nothing
        DEC  DI  ; Drop pointer
        DEC  CX  ; and character count
           JNZ  BACKUP  ; Send physical backspace
           JZ  GETCH  ; Get next char.
CONTROL: CMP  AL, 8  ; Check for backspace
           JC  BACKSP  ; Check for carriage return
        CMP  AL, 13  ; Ignore all other control char.
           JNZ  GETCH  ; Put the car. ret. in buffer
        MOV  DI, LINEBUF  ; Set up DI for command processing

; Output CR/LF sequence

CRLF:   MOV  AL, 13
        CALL  OUT  
        MOV  AL, 10
        JP  OUT

; Cancel input line

KILL:   CALL  CRLF
        JP  COMMAND

; Character input routine

IN:     DI  ; Poll, don't interrupt
        INB  STAT
        TEST  AL, DAV
           JZ  IN
               JNO  DATA
               AND  AL, 7FH
           JNO  RET  ; Only 7 bits
               JNO  RET
; Interrupts OK now

; Physical backspace - blank, backspace, blank

BACKUP: MOV  SI, BACMES

; Print ASCII message. Last char has bit 7 set
PRINTMES:
SEG CS
LODB
CALL OUT
SHL AL
JNC PRINTMES
RET

;Scan for parameters of a command
SCANP:
CALL SCANB
CMP B,[DI],"," ;Get first non-blank
JNE EOLCHK
INC DI
;If not comma, we found param
;Skip over comma

;Scan command line for next non-blank character
SCANB:
MOV AL,""
;Don't disturb CX
PUSH CX
MOV CL,-1
;But scan as many as necessary
REPE SCAB
DEC DI ;Back up to first non-blank
POP ex

EOLCHK:
CMP B,[DI],13
RET

;Print the 5-digit hex address of SI and DS
OUTSI:
MOV DX,DS
;Put DS where we can work with it
MOV AH,0
;Will become high bits of DS
CALL SHIFT4
;Shift DS four bits
ADD DX,SI
;Compute absolute address
JP OUTADD
;Finish below

;Print 5-digit hex address of DI and ES
OUTDI:
;Same as OUTSI above

OUTSI:
MOV DX,ES
;Put ES where we can work with it
MOV AH,0
;Will become high bits of ES
CALL SHIFT4
;Shift ES four bits
ADD DX,DI
;Finish OUTSI here too

OUTADD:
ADC AH,0
;Add in carry to high bits
CALL HIDIG
;Output hex value in AH

;Print out 16-bit value in DX in hex
OUT16:
MOV AL,DH
;High-order byte first
CALL HEX
MOV AL,DL
;Then low-order byte

;Output byte in AL as two hex digits
HEX:
MOV AH,AL
;Save for second digit
;Shift high digit into low 4 bits
PUSH CX
MOV CL,4
SHR AL,CL
POP CX

CALL DIGIT
;Output first digit
HIDIG: MOV AL, AH ;Now do digit saved in AH

DIGIT: AND AL, 0FH ;Mask to 4 bits

;Trick 6-byte hex conversion works on 8086 too.
ADD AL, 90H
DAA
ADD AL, 40H
DAA

;Console output of character in AL

OUT: PUSH AX ;Character to output on stack

OUT1: INB STAT
AND AL, TBMT
JZ OUT1 ;Wait until ready
POP AX
OUTB DATA
RET

;Output one space

BLANK: MOV AL, " "
JP OUT

;Output the number of blanks in CX

TAB: CALL BLANK
LOOP TAB
RET

;Command Table. Command letter indexes into table to get
;address of command. PERR prints error for no such command.

COMTAB:

DW BOOT ;B
DW PERR ;C
DW DUMP ;D
DW ENTER ;E
DW FILL ;F
DW GO ;G
DW PERR ;H
DW INPUT ;I
DW FERR ;J
DW PERR ;K
DW PERR ;L
DW MOVE ;M
DW PERR ;N
DW OUTPUT ;O
DW PERR ;P
DW FERR ;Q
DW REG ;R
DW SEARCH ;S
DW TRACE ;T

;Given 20-bit address in AH:DX, breaks it down to a segment
;number in AX and a displacement in DX. Displacement is
;always zero except for least significant 4 bits.

GETSEG:

MOV AL, DL ;AL has least significant 4 bits
AND AL, 0FH
CALL SHIFT4 ;4-bit left shift of AH:DX
MOV DL, AL ;Restore lowest 4 bits
MOV AL, DH ;Low byte of segment number
XOR DH, DH ;Zero high byte of displacement
RET

;Shift AH:DX left 4 bits

SHIFT4:

SHL DX
RCL AH ; 1
SHL DX
RCL AH ; 2
SHL DX
RCL AH ; 3
SHL DX
RCL AH ; 4

RET2: RET

; RANGE - Looks for parameters defining an address range.
; The first parameter is a hex number of 5 or less digits
; which specifies the starting address. The second parameter
; may specify the ending address, or it may be preceded by
; "L" and specify a length (4 digits max), or it may be
; omitted and a length of 128 bytes is assumed. Returns with
; segment no. in AX and displacement (0-F) in DX.

RANGE:

MOV CX, 5 ; 5 digits max
CALL GETHEX ; Get hex number
PUSH AX ; Save high 4 bits
PUSH DX ; Save low 16 bits
CALL SCANP ; Get to next parameter
CMP B,[DI], "L" ; Length indicator?
JE GETLEN ; Default length
CALL HEXIN ; Second parameter present?
MOV DX, 128 ; Default length
JC RNGRET ; If not, use default
CALL GETHEX ; Get ending address
MOV CX, 5 ; 5 hex digits
MOV CX, DX ; Low 16 bits of ending addr.
PUSH DX ; Low 16 bits of starting addr.
PUSH BX ; BH=hi 4 bits of start addr.
CALL SCANP ; Get to next parameter
CMP B,[DI], "L" ; Length indicator?
JE GETLEN ; Length may have 4 digits
CALL GETHEX ; Get range
MOV CX, DX ; Length
POP DX ; Low 16 bits of starting addr.
POP AX ; AH=hi 4 bits of starting addr.

RNGRET:

MOV CX, DX ; Length
POP AX ; Low 16 bits of starting addr.
POP DX ; BH=hi 4 bits of start addr.
JNC RNGERR ; Finish 20-bit subtract
INC AX ; Range must include ending location
INC CX ; Finish range testing and return

RNGERR:

MOV AX, BX ; Range must include ending location
INC BX ; Finish range testing and return

RNGCHK:

MOV BX, DX ; Low 16 bits of starting addr.
AND BX, 0FH ; Low 4 bits of starting addr.
JNC MAXRNG ; If count=10000H then BX must be 0
ADD BX, CX ; Must be <=10000H
JNC GETSEG ; OK if strictly <

MAXRNG:

; If here because of JCCZ MAXRNG, we are testing if low 4 bits
; (in BX) are zero. If we dropped straight in, we are testing
; for BX+CX=10000H (=0). Either way, zero flag set means
; within range.
RNGERR:
JZ GETSEG
MOV AX,4700H+"R" ;RG ERROR
JMP ERR

;Dump an area of memory in both hex and ASCII

DUMP:
CALL RANGE ;Get range to dump
PUSH AX ;Save segment
CALL GETEOL ;Check for errors
POP DS ;Set segment
MOV SI,DX ;SI has displacement in segment

ROW:
CALL OUTSI ;Print address at start of line
PUSH SI ;Save address for ASCII dump

BYTE:
CALL BLANK ;Space between bytes

BYTE1:
LODB ;Get byte to dump
CALL HEX ;And display it
PUSH DX ;DX has start addr. for ASCII dump
DEC CX ;Drop loop count
JZ ASCII ;If through do ASCII dump
MOV AX,SI
TEST AL,0FH
JZ ENDROW ;On 16-byte boundary?
FUSH DX
TEST AL,7
JNZ BYTE ;Didn't need ASCII addr. yet
MOV AL,\"\n\" ;On 8-byte boundary?
BYTE AL,\".\"
;Mark every 8 bytes

ENDROW:
CALL ASCII ;Show it in ASCII
JP ROW ;Loop until count is zero

ASCII:
PUSH CX ;Save byte count
MOV AX,SI ;Current dump address
MOV SI,DX ;ASCII dump address
SUB AX,DX ;AX=length of ASCII dump

;Compute tab length. ASCII dump always appears on right side
;screen regardless of how many bytes were dumped. Figure 3
;characters for each byte dumped and subtract from 51, which
;allows a minimum of 3 blanks after the last byte dumped.

;Length times 2
MOV BX,AX
;Length times 3
SHL AX
;Amount to tab in CX
ADD AX,BX
MOV CX,51
SUB CX,AX

;ASCII dump length back in CX

ASCDMP:
LODB ;Get ASCII byte to dump
AND AL,7FH ;ASCII uses 7 bits
CMP AL,7FH
JZ NOPRT ;Don't try to print RUBOUT
JNC PRIN ;Check for control characters

NOPRT:
MOV AL,\"\n\" ;If unprintable character

PRIN:
CALL OUT ;Print ASCII character
LOOP ASCDMP ;CX times
POP CX ;Restore overall dump length
JMP CRLF ;Print CR/LF and return

;Block move one area of memory to another. Overlapping moves
;are performed correctly, i.e., so that a source byte is not
;overwritten until after it has been moved.
MOVE:

CALL RANGE ;Get range of source area
PUSH CX ;Save length
PUSH AX ;Save segment
MOV SI,DX ;Set source displacement
MOV CX,5 ;Allow 5 digits
CALL GETHEX ;Get destination address
CALL GETBOL ;Check for errors
CALL GETSEG ;Convert dest. to seg/disp
MOV DI,DX ;Get destination displacement
POP BX ;Source segment
MOV DS,BX
POP CX
CMP DI,SI ;Check direction of move
SBB AX,BX ;Extend the CMP to 32 bits
JB COPYLIST ;Move forward into lower mem.

;Otherweise, move backward. Figure end of source and destination
;areas and flip direction flag.

COPYLIST:

MOVB ;Do at least 1 - Range is 1-10000H not 0-FFFFH
DEC CX ;End of source area
REP MOVB ;Block move
RET

;Fill an area of memory with a list values. If the list
;is bigger than the area, don't use the whole list. If the
;list is smaller, repeat it as many times as necessary.

FILL:

CALL RANGE ;Get range to fill
PUSH CX ;Save length
PUSH AX ;Save segment number
PUSH DX ;Save displacement
CALL LIST ;Get list of values to fill with
POP DI ;Displacement in segment
POP ES ;Segment
POP CX ;Length
CMP BX,CX ;BX is length of fill list
MOV SI,LINEBUF ;List is in line buffer
JCXZ BIGRNG ;If list is big, copy part of it
JAE COPYLIST

BIGRNG:

SUB CX,BX ;How much bigger is area than list?
XCHG CX,BX ;CX = length of list
PUSH DI ;Save starting addr. of area
REP MOVB ;Move list into area
POP SI

;The list has been copied into the beginning of the
;specified area of memory. SI is the first address
;of that area, DI is the end of the copy of the list
;plus one, which is where the list will begin to repeat.
;All we need to do now is copy [SI] to [DI] until the
;end of the memory area is reached. This will cause the
;list to repeat as many times as necessary.

MOV CX,BX ;Length of area minus list
PUSH ES ;Different index register
POP DS ;Requires different segment reg.
JP COPYLIST ;Do the block move

;Search a specified area of memory for given list of bytes.
;Print address of first byte of each match.
SEARCH:
CALL RANGE ;Get area to be searched
PUSH CX ;Save count
PUSH AX ;Save segment number
PUSH DX ;Save displacement
CALL LIST ;Get search list
DEC BX ;No. of bytes in list-1
POP DI ;Displacement within segment
POP ES ;Segment
POP CX ;Length to be searched
SUB CX,BX ;minus length of list
SCAN:
MOV SI,LINEBUF ;List kept in line buffer
LDB ;Bring first byte into AL
DOSCAN:
SCAB 
LOOPNE DOSCAN ;Do at least once by using LOOP
JNZ RET ;Exit if not found
PUSH BX ;Length of list minus 1
XCHG BX,CX
PUSH DI ;Will resume search here
REPE
CMPB
MOV CX,BX ;Compare rest of string
PUSH BX ;Area length back in CX
XCHG CX,BX
PUSH DI ;Next search location
DJNZ TEST ;Continue search if no match
DEC DI ;Match address
CALL OUTDI ;Print it
INC DI ;Restore list length
CALL CRLF ;Restore search address
TEST:
JNZ RET ;Look for next occurrence
HEX: CALL SCANP ;Scan to next parameter
GETHEX: CALL SCANP ;Scan to next parameter
GETHEX1:
XOR DX,DX ;Initialize the number
MOV AH,DH
CALL HEXIN ;Get a hex digit
JC ERROR ;Must be one valid digit
MOV DL,AL ;First 4 bits in position
GETLP:
INC DI ;Next char in buffer
DEC CX ;Digit count
CALL HEXIN ;Get another hex digit?
JC ERROR ;All done if no more digits
JZ ERROR ;Too many digits?
CALL SHIFT4 ;Multiply by 16
OR DL,AL ;and combine new digit
JP GETLP ;Get more digits
HEXIN:
MOV AL,[DI] ;Check if AL has a hex digit and convert it to binary if it is. Carry set if not.
HEXCHK:
SUB AL,"O" ;Kill ASCII numeric bias
;Process one parameter when a list of bytes is required. Carry set if parameter bad. Called by LIST

LISTITEM:
CALL SCANP ;Scan to parameter
CALL HEXIN ;Is it in hex?
JC STRINGCHK ;If not, could be a string
MOV CX,2 ;Only 2 hex digits for bytes
CALL GETHEX ;Get the byte value
MOV [BX],DL ;Add to list
INC BX
GRET: CLC ;Parameter was OK
RET

STRINGCHK:
MOV AL,[DI] ;Get first character of param
CMP AL,"" ;String?
JZ STRING ;Either quote is all right
CMP AL,"" ;Not string, not hex - bad
JZ STRING
STC
RET

STRING:
MOV AH,AL ;Save for closing quote
INC DI
STRNGLP:
MOV AL,[DI] ;Next char of string
INC DI
CMP AL,13 ;Check for end of line
JZ ERROR ;Must find a close quote
CMP AL,AH ;Check for close quote
JNZ STOSTRG ;Add new character to list
JNZ GRET ;If not, we're done
INC DI ;Yes - skip second one
STOSTRG:
MOV [BX],AL ;Put new char in list
INC BX
JP STRNGLP ;Get more characters

;Get a byte list for ENTER, FILL or SEARCH. Accepts any number
;of 2-digit hex values or character strings in either single
;'(' or double ("") quotes.

LIST:
MOV BX,LINEBUF ;Put byte list in the line buffer

LISTLP:
CALL LISTITEM ;Process a parameter
JNC LISTLP ;If OK, try for more
SUB BX,LINEBUF ;BX now has no. of bytes in list
JZ ERROR ;List must not be empty

;Make sure there is nothing more on the line except for
;blanks and carriage return. If there is, it is an
;unrecognized parameter and an error.

GETEOL:
CALL SCANB ;Skip blanks
JNZ ERROR ;Better be a RETURN
;Command error. DI has been incremented beyond the
;command letter so it must decremented for the
;error pointer to work.

PERR:      DEC DI

;Syntax error. DI points to character in the input buffer
;which caused error. By subtracting from start of buffer,
;we will know how far to tab over to appear directly below
;it on the terminal. Then print "^ Error".

ERROR:
SUB DI,LINEBUF-1 ;How many char processed so far?
MOV CX,DI       ;Parameter for TAB in CX
CALL TAB        ;Directly below bad char
MOV SI,STNERR   ;Error message

;Print error message and abort to command level

PRINT:
CALL PRINTMES
JMP COMMAND

;Short form of ENTER command. A list of values from the
;command line are put into memory without using normal
;ENTER mode.

GETLIST:
CALL LIST       ;Get the bytes to enter
POP DI          ;Displacement within segment
POP ES          ;Segment to enter into
MOV ST,LINEBUF  ;List of bytes is in line buffer
MOV CX,BX       ;Count of bytes
REP MOVB        ;Enter that byte list
RET

;Enter values into memory at a specified address. If the
;line contains nothing but the address we go into "enter
;mode", where the address and its current value are printed
;and the user may change it if desired. To change, type in
;new value in hex. Backspace works to correct errors. If
;an illegal hex digit or too many digits are typed, the
;bell is sounded but it is otherwise ignored. To go to the
;next byte (with or without change), hit spacebar. To
;back up to a previous address, type "^ n". On
;every 8-byte boundary a new line is started and the address
;is printed. To terminate command, type carriage return.
;Alternatively, the list of bytes to be entered may be
;included on the original command line immediately following
;the address. This is in regular LIST format so any number
;of hex values or strings in quotes may be entered.

ENTER:
MOV CX,5        ;5 digits in address
CALL GETHEX    ;Get ENTER address
CALL GETSEG     ;Convert to seg/disp format

;Adjust segment and displacement so we are in the middle
;of the segment instead of the very bottom. This allows
;backing up a long way.

;Adjust segment 32K down

;And displacement 32K up

;Save for later

;Any more parameters?

;If not end-of-line get list

;Displacement of ENTER

;Segment
GETBYTE:

GETDIG:

WAIT:

NOHEX:

BS:

STORE:

NOSTO:

EOL:

NEXT:
INC CX ;Leave a space plus two for
INC CX ; each digit not entered
CALL TAB
MOV AX,DI ;Next memory address
AND AL,7 ;Check for 8-byte boundary
JNZ GETBYTE ;Take 8 per line
NEWROW:
CALL CRlf ;Terminate line
JMP GETROW ;Print address on new line
PREV:
CALL STORE ;;Enter the new value
;DI has been bumped to next byte. Drop it 2 to go to previous addr
DEC DI
DEC DI
JP NEWROW ;;Terminate line after backing up
;Perform.register dump if no parameters or set register if a
;register designation is a parameter.
REG:
CALL SCANP
JZ DISPREG
MOV DI,[DI]
INC DI
MOV DI,[DI]
CMP DH,13
JZ FLAG
INC DI
CALL GETEOL
CMP DH,""
JZ FLAG
MOV DI,RECTAB
XCHG AX,DX
PUSH CS
POP ES
MOV CX,RECTABLEN
REPNZ SCAW
JNZ BADREG
OR CX,CX
JNZ NOTPC
DEC DI
DEC DI
SEG CS
MOV AX,[DI-2]
RET3: RET
BADREG:
MOV AX,5200H+"B"
BR ERROR
DISPREG:
0496 BE 06 07 MOV SI,RECTAB
0499 BB 08 00 MOV BX,AXSAVE
049C B9 05 00 MOV CX,8
049F EB 05 00 CALL DISPREGLINE
04A2 EB 4F FC CALL CRLF
04A5 B9 05 00 MOV CX,5
04A8 EB 5C 00 CALL DISPREGLINE
04AA EB 65 FC CALL BLANK
04AE EB 93 00 CALL DISPFLAGS
04B1 EB 40 00 JMP CRLF
04B4 BB 9C 01 MOV BX,AXSAVE
04B7 93 05 MOV CX,8
04B9 BB 65 00 CALL DISPREGLINE
04BE EB 67 FC CALL OUT
04C1 EB 7D FB CALL INBUF
04C4 EB 8E 00 CALL SCANB
04C7 33 DB XOR BX,BX
04CD 8B 16 01 MOV AX,[FSAVE]
04CB 08 01 CALL OUT
04CD 8B 46 00 MOV CX,32
04D0 08 01 POP ES
04D2 0F 60 MOV AX,1
04D4 BB 66 00 CALL OUT
04D7 74 66 CALL OUT
04D9 BB 06 00 MOV DI,FLAGS
04DC BB 20 00 MOV CX,32
04DF 0E 07 POP ES
04E1 F2 REPNE
04E2 8D 07 MOV SI,DI
04E3 75 0A CMP DL,11
04E5 8A E9 MOV CH,CL
04E7 80 01 0F AND CL,0FH
04EA BB 01 00 MOV AX,1
04ED 3C OD CMP AL,13
04F1 76 33 JNZ REPFLG
04F3 0B D8 OR BX,AX
04F5 0B D0 OR DX,AX
04F7 F6 C5 10 TEST CH,16
04FA 75 02 JNZ NEXFLG
04FC 33 D0 XOR AX,DX
04FE 0B 0E MOV SI,DI
0500 1E PUSH DS
0501 07 POP ES
0502 EB 17 FC CALL SCANP
0505 EB C6 JP GETFLG
0507 2E SEG CS
0508 AD LODW
0509 EB 5C FC CALL OUT
050C EB C4 MOV AL,AH
050E EB 57 FC CALL OUT
0511 BB 03 0D MOV AL,"=
0513 EB 52 FC CALL OUT
0516 EB 17 MOV DX,[BX]
0518 43 INC BX
0519 43 INC BX
051A EB 2F FC CALL OUT16
051D EB 53 FC CALL BLANK
0520 EB 50 FC CALL BLANK
0523 EB E2 LO E2 LOOP DISPREGLINE
0525 C3 RET
0526 BB 44 46 MOV AX,4600H+"D"
0529 FERR:

- 25 -
CALL SAVCHG
ERR:
CALL OUT
MOV AL, AH
CALL OUT
MOV SI, ERRMES
JMP PRINT

SAVCHG:
MOV [FSAVE], DX

FLGERR:
MOV AX, 4600H + 11B11 ; BF ERROR
JP FERR

DISPFLAGS:
MOV SI, FLAGTAB
MOV CX, 16
MOV DX, [FSAVE]

DFLAGS:
SEG CS
LODW
SHL DX
JC FLAGSET
SEG CS
MOV AX, [SI + 30]
FLAGSET:
OR AX, AX
JZ NEXTFLG
CALL OUT
MOV AL, AH
CALL OUT
CALL BLANK

NEXTFLG:
LOOP DFLAGS
RET

; Trace 1 instruction or the number of instruction specified
; by the parameter using 8086 trace mode. Registers are all
; set according to values in save area

TRACE:
CALL SCANP
CALL HEXIN
MOV DX, 1
MOV [TCOUNT], DX
CALL GETEOL
STEP:
MOV [BRKCNT], 0
OR B, [FSAVE + 1], 1
EXIT:
MOV [12], BREAKFIX
MOV [14], CS
MOV [4], REENTER
MOV [6], CS
DI
MOV [64H], REENTER
MOV [66H], CS
MOV SP, STACK
POP AX
POP EX
POP CX
POP DX
POP BP
POP SI
POP DI
POP ES
0588 07  POP ES
0589 17  POP SS
058A 88 26 A4 01  MOV SP,[SPSAVE]
058E FF 36 B6 01  PUSH [FSAVE]
05C2 FF 36 B2 01  PUSH [CSSAVE]
05C6 FF 36 B4 01  PUSH [IPSSAVE]
05CA 88 1E AC 01  MOV DS,[DSSAVE]
05CF EB B1
STEPL: JP STEP

; Re-entry point from breakpoint. Need to decrement instruction
; pointer so it points to location where breakpoint actually
; occurred.

BREAKFIX:
XCHG SP,BP
DEC [BP]
XCHG SP,BP

; Re-entry point from trace mode or interrupt during
; execution. All registers are saved so they can be
; displayed or modified.

REENTER:
SEG CS
MOV [SPSAVE+SEGDIF],SP
SEG CS
MOV [SSSAVE+SEGDIF],SS
XOR SP,SP
MOV SS,SP
MOV SP,RSTACK
PUSH ES
PUSH DS
PUSH DI
PUSH SI
PUSH BP
DEC SP
PUSH AX
PUSH SS
POP DS
MOV SP,[SPSAVE]'
MOV AX
MOV ES
PUSH [FSAVE],AX
MOV [SPSAVE],SP
MOV [IPSSAVE],SP
MOV [SPSAVE]
MOV SP,[SPSAVE]
MOV SP,[SPSAVE+SEGDIF]
MOV SP,[SSSAVE+SEGDIF]
DEC [IPSAVE]
MOV SP,[SPSAVE]
MOV [CSSAVE]
MOV [CSSAVE]
MOV AX
AND AH,0FEH
MOV [FSAVE],AX
MOV [SPSAVE],SP
PUSH DS
PUSH ES
PUSH SS
MOV SP,STACK
MOV [64H],INT
MOV AL,20H
OUT BASE+2
CALL CRLF
CALL DISPREG
DEC [TCOUNT]
JNZ STEPL

ENDGO:
MOV SI,BPTAB
MOV CX,[BRKCNT]
JNZ STEP1

- 27 -
0639 8B 54 14
063C 50
063E E8 62 FB
0641 8E CO
0643 8B FA
0645 58
0646 AA
0647 E2 F0
0649 E9 3B FA

**CLEARBP:**

```assembly
MOV DX, [SI+BPLEN]
LODW
PUSH AX
CALL GETSEG
MOV ES, AX
MOV DI, DX
POP AX
STOB
LOOP CLEARBP
```

**COMJMP:**

```assembly
JMP COMMAND
```

;Input from the specified port and display result

**INPUT:**

```assembly
MOV CX, 4
CALL GETHEX
INB DX
CALL HEX
JMP CRLF
```

;Output a value to specified port.

**OUTPUT:**

```assembly
MOV CX, 4
CALL GETHEX
PUSH DX
MOV CX, 2
MOV [BX], DX
XCHG AX, DX
OUTB DX
RET
```

;Jump to program, setting up registers according to the
;save area. Up to 10 breakpoint addresses may be specified.

**GO:**

```assembly
MOV BX, LINEBUF
XOR SI, SI
```

**GO!:**

```assembly
CALL SCANP
JZ EXEC
BE 04 01
MOV SI, BPTAB
SETBP:
MOV DX, [SI+BPLEN]
LODW
CALL GETSEG
MOV DS, AX
MOV DI, DX
MOV AL, [DI]
MOV B, [DI], OCCH
PUSH ES
POP DS
```

- 28 -
;Console input interrupt handler. Used to interrupt commands for programs under execution (if they have interrupts enabled). Control-S causes a loop which waits for any other character to be typed. Control-C causes abort to command mode. All other characters are ignored.

INT: PUSH AX ;Don't destroy accumulator

;Output End-of-Interrupt commands to slave 8259A. This wouldn't be necessary if Automatic End of Interrupt mode worked like it was supposed to!
    MOV AL, 20H
    OUT BASE+2
    IN DATA
    AND AL, 7FH ;ASCII has only 7 bits
    CMP AL, 11
    S11 - 11 @ 11
    JNZ NO STOP
    CALL IN
    CMP JZ AL, "C"-"@" ;Check for Control-C
    BREAK
    ;Just ignore interrupt - restore AX and return
    POP AX
    IRET

BREAK:
    CALL CRLF
    JMP COMMAND

REGTAB:
    DB "AXBXCDXSPBPSIDIDSESSCSIPPC"

REGDIF: EQU AXSAVE-REGTAB

;Flags are ordered to correspond with the bits of the flag register, most significant bit first, zero if bit is not a flag. First 16 entries are for bit set, second 16 for bit reset.

FLAGTAB:
    DW 0
    DW 0
    DW 0
    DW 0
    DW 0
    DB "OV"
    DB "DN"
    DB "EI"
    DW 0
    DB "NC"
    DB "ZR"
    DW 0
    DB "AC"
    DW 0
    DB "PE"
    DW 0
    DB "CY"
    DW 0
    DW 0
    DW 0
    DW 0
    DB "NV"
    DB "UP"
    DB "DT"
;Initialization table. First byte of each entry is no. of bytes to output to the corresponding port. That many initialization bytes follow.

INITTABLE:

;Port BASE+0 - Master 8259A. Initialization Command Word (ICW)
;One sets level-triggered mode, multiple 8259As, require ICW4.

DB 1
DB 19H

;Port BASE+1 - Master 8259A. ICW2 sets vector base to 10H
;ICW3 sets a slave on interrupt input 1; ICW4 sets buffered mode, as a master, with Automatic End of Interrupt, 8086 vector; Operation Command Word (OCW) One sets interrupt mask to enable line 1 (slave 8239A) only.

DB 4
DB 10H, 2OH, OFDH

;Port BASE+2 - Slave 8259A. ICW1 sets level-triggered mode, multiple 8259As, require ICW4.

DB 1
DB 19H

;Port BASE+3 - Slave 8259A. ICW2 sets vector base to 18H
;ICW3 sets slave address as 1; ICW4 sets buffered mode, as a slave, with Automatic End of Interrupt (which doesn’t work in slaves), 8086 vector; OCW1 sets interrupt mask to enable line 1 (serial receive) only.

DB 4
DB 18H, 1OH, OFDH

;Port Base+4 - 9513 Data. 9513 has previously been set up for Counter 5 mode register with auto increment. Thus Fr (4 MHz), reload from load or hold, count down repetitively
;in binary, with output toggle. Load register is set to 0007H, and Hold register is set to 0006H. Thus we alternately divide by 7 and 6, which is divided by 2 by the output toggle, thus providing a square wave of 4 MHz/13 = 307.7 kHz, which divided by 16 in the 8251A provides 19,230 baud (0.16% high).

DB 6
DB 63H, 0BH, 7, 6, 0

;Port BASE+5 - 9513 Control. Load and arm counter 5, enabling baud rate generation. Then select counter 5 mode register, in case baud rate wasn’t right.

DB 2
DB 70H, 5

;Port BASE+6 - 8251A Data. No initialization to this port.

DB 0

;Port BASE+7 - 8251A Control. Since it is not possible to know whether the 8251A next expects a Mode Instruction or a Command Instruction, a dummy byte is sent which could safely be interpreted as either but guarantees it is now expecting a Command. The command sent is Internal Reset which causes it to start expecting a mode. The mode sent is for 2 stop bits, no parity, 8 data bits, 16X clock. This is followed by the command to error reset, enable transmitter and receiver, set RTS and DTR to +12V.

DB 4
DB 087H, 77H, 0CEH, 37H

HEADER: DM 13, 10, "SCP 8086 Monitor 1.5", 13, 10
; Disk boot. Select one of the following routines by setting the equates at the start of this program.

BOOT:     PUSH DI

;*******************************************************************************
; Boot for Cromemco 4FDC disk controller with either large or small disks. Loads track 0, sector 1 into LOAD.
;*******************************************************************************
 IF CROMEMCO4FDC

DISK:    EQU 30H

 MOV AL, 1
 OUT 2 ;Reset 4FDC serial I/O

 MOV AL, 84H ;and set for 300 baud

 MOV AL, 7FH
 OUT 4

 MOV DL, 21H

 RETRY:   MOV AL, ODH
 OUTB DISK

 READY:   INB DISK
 ROR AL
 JC READY
 XOR DL, 10H
 MOV AL, DL
 OUTB DISK+4

 MOV DI, LOAD

 HOME:    INB DISK+4
 ROR AL
 MOV AL, 1

 OUTB DISK+2

 MOV CX, 80H
 MOV AL, DL
 OR AL, 80H
 OUTB DISK+4

 MOV AL, 8CH
 OUTB DISK

 READ:    INB DISK+4
 ROR AL
 MOV AL, 1

 OUTB DISK+2

 INB DISK+3
 STOB

 LOOP READ

 WSTAT:   INB DISK+4
 ROR AL

 JNC WSTAT

 - 31 -
DONE:
   INB  DISK
   AND  AL,9CH
   JNZ  RETRY
   ENDF
   ENDF
   ENDF
   ENDF
   ENDF
   ENDF
   ENDF
   MOV  [CSSAVE],0
   MOV  [IPSAVE],LOAD
   POP  DI
   JMP  GO

Error Count = 0

; Boot for North Star disk, single density.
; Loads track 0, sector 0 into address LOAD
; Bug in North Star boot fixed 5-26-81.

; Disk command equates

SEL:    EQU 1
STP1:   EQU 9
STP2:   EQU 8
NOP:    EQU 10H
SEC:    EQU 14H
STPOUT: EQU 1CH
RD:     EQU 40H
BST:    EQU 20H

PUSH DS
MOV AX,OFEB8H
MOV DS,AX
MOV AL,[SEL]
CALL SECTOR
LOOP MOTOR

TEST B,[STPOUT],1
JNZ ONTRACK
MOV AL,[STP1]
AAM
MOV AL,[STP2]
CALL SECTOR
CALL SECTOR
JP CHKTRK

MOV AL,[SEC]    ; Reset sector flag.

MOV AL,[NOP]    ; Wait for sector flag.

TEST B,[NOP],80H
JZ SECLP

SECLP:

MOV AL,[SEC]

RET

MOV DI,LOAD
MOV CX,280
MOV BX,RD+NOP
CALL SECTOR
TEST B,[BST+NOP],OFH    ; Test for sector zero.
JNZ GETSEC

GETSYNC:    TEST B,[NOP],4
LOOPZ GETSYNC

- 32 -
Error Count = 0

;***********************************************************************
;Boot for Tarbell disk controllers. Load track 0, sector 1 into LOAD.

IF TARBELL

DISK: EQU 78H

DCOM: JP RETRY

OUTB DISK

MOV AL,50

HEC AL, Hold

JNZ HOLD

RET

RETRY:

MOV AL,000H

CALL DCOM

READY:

INB DISK

ROR AL

JC READY

MOV DL,LOAD

MOV AL,0EH ;Home command @ 10ms/track

CALL DCOM

INB DISK+4

INB DISK

AND AL,9BH

JNZ RETRY

MOV AL,1

OUTB DISK+2

MOV CX,80H

MOV AL,8CH

CALL DCOM

INB DISK+4

ROL AL

JNC DONE

INB DISK+3

STOB

LOOP READ

- 33 -
WSTAT:
INB DISK+4
ROL AL
JC WSTAT

DONE:
INB DISK
AND AL,9CH
JNZ RETRY

;Successful read
MOV [CSSAVE],0
MOV [IPSAVE],LOAD
POP DI
JMP GO

Error Count = 0

IF OTHER
;User may insert customized disk boot here. All
;registers are available, stack pointer is valid
;and interrupts are enabled. Stack should be at
;same level on fall-through to code below. Last
;address available is 07DF hex.
ORG 7EOH ;Simulate boot of maximum length
ENDIF

;Successful read
MOV [CSSAVE],0
MOV [IPSAVE],LOAD
POP DI
JMP GO

Error Count = 0