```
; This is an monitor for the S100computers.com 8086 board
; It started from the simple monitor in Byte on Nov 1980 but is enlarged and
; modified to work with the S100Computers 8086 Board, IDE Board, ZFDC board, PIC-RTC Board and other
; hardware as well.
; More recently it has been extensively enlarged to contain the interrupt based functions required
; to run a Microsoft's MS-DOS (V4.01) or FreeDOS emmulating an IBM-PC BIOS ROM.
;
;               John Monahan, Danville, CA
;       History
;       V1.0                    ;Origional version sometime in 1982
;       V2.1    3/12/1983
;       V2.2    11/12/09        ;Modified for simple I/O. Ports info command added
;       V2.3    11/18/09        ;Reset FAR jump to start of monitor, added Register display
;       V2.31   11/19/09        ;Allow intersegment FAR jump with G command
;       V2.4    2/18/10         ;Write version to reside at F:F000H (to be loaded with a CPM loader from disk)
;       V2.5    8/26/10         ;Add S100Computers Serial IO Board & RTC Board. Input IOBYTE (EFh) for JMP to 0:500H
;       V2.51   8/26/10         ;Stack & Flag below EPROM in high memory
;       V2.52   8/27/10         ;Stack etc in low memory. AP/PM for clock (with DAS opcode)
;       V2.53   8/27/10         ;Check if after a reset a direct jump to CPM86 in RAM is required
;       V2.6    5/7/11          ;Added IDE Board diagnostic package
;       V2.7    5/9/11          ;Aruba trip, complete overhaul while traveling.
;       V2.8    5/14/11         ;Finished IDE drive additions
;       V2.9    5/17/11         ;Switched over to using SI and DI registers for memory move etc functions
;       V3.0    5/31/11         ;Corrected Sector display routines
;       V3.1    6/1/11          ;Corrected memory Map display and move memory routines
;       V3.2    6/7/11          ;Corrected CF Drive A=B Verify routine
;       V3.3    6/8/11          ;Corrected CF Drive A->B copy routine
;       V3.4    6/8/11          ;BP used for all IDE routines thus freeing dependance on a fixed RAM location
;       V3.5    6/10/11         ;Corrected Disk format routine. Corrected Drive ID routine
;       V3.6    6/19/11         ;Corrected Monitor signon message at start
;       V4.0    7/20/11         ;Splice in IBM-PC/MS-DOS Interrupt routines. Enlarged Monitor now starts at FC000H
;       V4.1    7/31/11         ;Correct CICO routine so it is not case sensitive
;       V4.2    8/3/11          ;Vector Int's 0 & 1 working OK.
;       V4.3    8/4/2011        ;MS-DOS 2.01 loading from floppy fine on 5" SS Disks (only)!
;       V4.4    8/20/2011       ;Added special MS-DOS FFDC commands to read DDDS DOS Disks. Now works with IBMs PC-DOS;
        V4.51   8/23/2011       ;Corrected length check with GET5DIGIST etc.
;       V4.52   8/23/2011       ;Added "PATCH" to quickly test RAM/Debug versions of this code.
;       V5.0    8/26/2011       ;MS-DOS hard disk caspability
;       V5.1    9/1/2011        ;Corrected bug in IDE (WR_LBA) routine. Was not sending High Cylinder!
;       V5.2    9/6/2011        ;Move to 27C256 EEPROMS (Will no longe fit in 27C64's). Address starts at F000:8000H
;       V5.3    9/7/2011        ;Last version written for Digital Research ASM86 assembler. (get symbol overflow)
;       V6.0    9/8/2011        ;Major rewrite to work with NASM Assembler. (Sorry I did not do this earlier.
;       V6.1    9/11/2011       ;Corrected IDE disk compare routine.
;       V6.2    9/12/2011       ;Added IDE memu options to test LBA & CHS display on IDE Board HEX display LED's
;       V6.3    9/13/2011       ;Added IBM-BIOS menu option the Read/Write a block of contigous sectors the the IDE Drive
;       V6.4    9/16/2011       ;Added cursor addressing to video output functions. FDISK now displays correctly.
;       V6.5    9/17/2011       ;Corrected printer I/O

;       Notes...
;       This fairly extensive 8086+ monitor consists of 3 main sections. It assumes an 8086 (does not use opcodes of ;
        the 80286+)

;       Section 1.      This is a classical monitor. Display, change RAM/ports etc.
;       Section 2.      This is a self contained set of routines run diagnostic tests on the S100Computers/N8VEM IDE
;                       board.
;       Section 3.      This fairly complex section. It emmulates most of the IBM-PC ROM BIOS interrupts (hard & soft)
;                       such that MS-DOS (V4.01)/FreeDOS can be run on the system - without DOS modifications.

;       For debugging/testing this monitor will reside in RAM at F000:2000H (with the stack at F000:1FFFH & IDE RAM at
;       F000:1000H).
;       In the final EEPROM it will be placed at F000:8000H (and the stack at F000:7FFFH & IDE RAM at F000:7000H).
;
;       This monitor needs a valid stack in RAM. It first checks if there is valid RAM in high memory just below the
;       ROM
;       (The EEROM is usually at F000:8000H). If so it will set the SS to F000H and the SP to 7FFBH. This puts it out ;
        of the way of everything in low RAM. If it does not detect RAM there, it will search for a valid segment at top
;       of RAM downwards and  put the stack there.
;
```

```
;       That is all the RAM the main monitor needs. However the IDE drive diagnostic routines require much more (sector
;       buffers etc)
;       I have set the SS: the (monitor ORG) - 1000H, normally F000:7000H, (or F000:1000H in the RAM based debugging ;
        version)  and uses SS:BP throughout to access that RAM.

;       Most monitor commands are modeled after the old TDL/Zapple/Z80 commands.  Because we are now dealing with ;
        potentially up to 1MG of RAM for many commands, the start,end RAM locations etc. can take up to 5 digits.
;       However the actual span/range for any command is limited to 64K.
;
;       The following example fills RAM with 76H from 1A000H  to 21234H.
;       F1A000,21234,76
;       Of course for the lowest 64K of RAM the "normal" 4,3,2 or 1 byte formats can be used
;       F123,456,76
;       Note because of the 64K range limitation the following will give an error
;       F1A000,31234,76   or F1A000,1B000,76
;
;       In general hitting the ESC key will abort any long display/command function.
;       In all cases, to accept data, finish entry with CR. So if the display says "xxxxH" you enter up to 4 hex digits
;       than a CR.
;
;       To test/load the monitor in RAM...
;       There are a  number of ways to test/run this monitor. Until you actully have this monitor in EEPROM, you
;       can assemble it with a origin in low RAM (say MonitorORG = 2100H).  Until you have a working version you should
;       have your 8086 after reset jump to the debugging monitor in RAM.  The debugging version can be anywhere in RAM
;       but the easiest location is something like 2100H. It needs to be well above 100H, because the Stack & Data ;
        areas are BELOW the ORG or the final EPROM code.
;
;       For Old (< V5.2) Versions Assembled with Digital Research's ASM86 see this section in those files.
;
;       For all New Versions (> V6.0) Assemble to a binary file with the excellent/free MSDOS/FreeDOS, NASM.EXE
;       Assembler
;       NASM -f bin 8086.A86 -o 8086.bin -l 8086.lst

;       This will make a 8 bit format .bin file
;       Move it across to your CPM80 system (Telnet/Modem/serial connection, whatever)
;       SID 8086.bin
;       This will place the code at 2100H to ~7xxxH
;       After switching to your 8086  Board (IN port EDH), have the 8086 jump to there with
;       @FFFF0H:- EA 00 21 00 00
;       (either in its onboard EEPROM or done by hand in RAM. The monitor should come up.
;
;       Note in the code there are a few FAR JMP's in the code, make sure the segment is 0000H for this "low RAM ;
        debugging version".
;
;       As soon as you get things going, burn a EEPROM version that resides at F000:8000H. After switching to your
;       8086 Board (IN port EDH), the Monitor should immediatly come up.
;
;       From then on, it is best to keep RAM test versions up in the 8086 high RAM. That way you can test MSDOS etc.
;       I use the location F000:2000H. You can use the Monitor "Y" Command to Move 2100H-71FFH up to F000:2000H and
;       JMPF to that code. This saves keystrokes, for the many times you do this!
;
;       To burn a 28C256 EEPROM's with a Wellon VP280 Programmer...
;       Load .BIN file. Select Even bytes (1st of 2) for one ROM and "From File HEX address" and "Buffer Address"
;       leave 0000 in the load dialog boxes.  (The Edit BOX the code should appear at 4000H-7FFFH).
;       Repeat for ODD addresses.


SCROLL          EQU     01H             ; Set scrool direction UP.
BELL            EQU     07H
SPACE           EQU     20H
TAB             EQU     09H             ; TAB ACROSS (8 SPACES FOR SD-BOARD)
CR              EQU     0DH
LF              EQU     0AH
FF              EQU     0CH
QUIT            EQU     11H             ; Turns off any screen enhancements (flashing, underline etc).
ESC             EQU     1BH
DELETE          EQU     7FH
BACKS           EQU     08H
CLEAR           EQU     1AH             ; TO CLEAR SCREEN
```

```
TRUE            equ 1
FALSE           equ TRUE-TRUE

MONITOR_ROM     EQU     FALSE           ;<<<<<<< TRUE = Monitor at F000:8000H, FALSE = Monitor in a RAM area (F000:2000H)

;Propeller Console IO S-100 board or SD SYSTEMS VIDIO BOARD FOR CONSOLE I/O(<---These must configured for your
hardware)

KEYSTAT         EQU     0H
KEYIN           EQU     01H             ;Console input port. Normally the Propeller Driven S-100 Console-IO Board
KEYOUT          EQU     01H             ;Console output port. Normally the Propeller Driven S-100 Console-IO Board



;-------------- THIS IS MY PORT TO OUTPUT DATA TO HP 4050T LASAR PRINTER  (IMSAI 8PIO Board)

PRINTER_STATUS EQU      5               ;IN, HP PARRELL PORT
PRINTER_OUT     EQU     5               ;OUT
PRINTER_STROBE EQU      4               ;OUT
DIAG_LEDS       EQU     5               ;OUT (Will use this port initially for diagnostic LED display)


;-------------- S100Computers Serial I/O BOARD PORT ASSIGNMENTS  (A0-AC)

BCTL            EQU     0A0H            ;CHANNEL B CONTROL PORT ASSIGNMENTS OF THE ZILOG SCC CHIP
ACTL            EQU     0A1H            ;CHANNEL A CONTROL
BDTA            EQU     0A2H            ;CHANNEL B DATA
ADTA            EQU     0A3H            ;CHANNEL A DATA
;
PortA_8255      EQU     0A8H            ;A port of 8255 ;<--- Adjust as necessary
PortB_8255      EQU     0A9H            ;B port of 8255
PortC_8255      EQU     0AAH            ;C Port of 8255
PortCtrl_8255   EQU     0ABH            ;8255 configuration port

AinBout8255cfg EQU      10011000b       ;Set 8255 ports:- A input, B output,
                                        ;C(bits 0-3) output, (bits 4-7)input

USB_DATA        EQU     0ACH            ;PORT ASSIGNEMENT FOR DLP-USB Controller chip
USB_STATUS      EQU     0AAH            ;Status port for USB port (Port C of 8255, bits 6,7)

USB_RXE         EQU     80H             ;If Bit 7 = 0, data available to recieve by S-100 Computer
USB_TXE         EQU     40H             ;If Bit 6 = 0 data CAN be written for transmission to PC

IOBYTE          EQU     0EFH            ;IOBYTE Port on S100Computers SMB Board.


;------------ S100Computers PIC/RTC BOARD PORT ASSIGNMENTS  (A0H-ACH, 20H-21H)

RTCSEL          EQU     0A4H            ;58167 RTC Ports ports on S100 PIC/RTC Board
RTCDATA         EQU     0A5H

NS_EOI          equ     20h             ;Non specific end of interrupt command
MASTER_PIC_PORT         equ     20h             ;Hardware port the 8259A is assigned (two ports 20H & 21H)

MasterICW1      equ     00010111B       ;EDGE triggered, 4 bytes, single Master,ICW4 needed
MasterICW2      equ     8H              ;Base address for 8259A Int Table (IBM-PC uses 8X4 = 20H)
MasterICW3      equ     0H              ;No slave
MasterICW4      equ     00000011B       ;No special mode, non buffer, Auto EOI, 8086.  ;<<<<,


;-------------- S100Computers IDE BOARD PORT ASSIGNMENTS  (30-34H)

;Ports for 8255 chip. Change these to specify where the 8255 is addressed,
;and which of the 8255's ports are connected to which IDE signals.
;The first three control which 8255 ports have the IDE control signals,
;upper and lower data bytes.  The forth one is for mode setting for the
;8255 to configure its ports, which must correspond to the way that
;the first three lines define which ports are connected.
```

```
IDEportA        EQU     030H            ;lower 8 bits of IDE interface
IDEportB        EQU     031H            ;upper 8 bits of IDE interface
IDEportC        EQU     032H            ;control lines for IDE interface
IDECtrlPort     EQU     033H            ;8255 configuration port
IDEDrivePort    EQU     034H            ;To select the 1st or 2nd CF card/drive

IDE_Reset_Delay         EQU     020H            ;Time delay for reset/initilization (~66 uS, with 8MHz 8086, 1 I/O wait
state)

READcfg8255     EQU     10010010b       ;Set 8255 IDEportC out, IDEportA/B input
WRITEcfg8255    EQU     10000000b       ;Set all three 8255 ports output

;IDE control lines for use with IDEportC.

IDEa0line       EQU     01H             ;direct from 8255 to IDE interface
IDEa1line       EQU     02H             ;direct from 8255 to IDE interface
IDEa2line       EQU     04H             ;direct from 8255 to IDE interface
IDEcs0line      EQU     08H             ;inverter between 8255 and IDE interface
IDEcs1line      EQU     10H             ;inverter between 8255 and IDE interface
IDEwrline       EQU     20H             ;inverter between 8255 and IDE interface
IDErdline       EQU     40H             ;inverter between 8255 and IDE interface
IDErstline      EQU     80H             ;inverter between 8255 and IDE interface
;
;Symbolic constants for the IDE Drive registers, which makes the
;code more readable than always specifying the address pins

REGdata         EQU     IDEcs0line
REGerr          EQU     IDEcs0line + IDEa0line
REGseccnt       EQU     IDEcs0line + IDEa1line
REGsector       EQU     IDEcs0line + IDEa1line + IDEa0line
REGcylinderLSB  EQU     IDEcs0line + IDEa2line
REGcylinderMSB  EQU     IDEcs0line + IDEa2line + IDEa0line
REGshd          EQU     IDEcs0line + IDEa2line + IDEa1line    ;(0EH)
REGcommand      EQU     IDEcs0line + IDEa2line + IDEa1line + IDEa0line       ;(0FH)
REGstatus       EQU     IDEcs0line + IDEa2line + IDEa1line + IDEa0line
REGcontrol      EQU     IDEcs1line + IDEa2line + IDEa1line
REGastatus      EQU     IDEcs1line + IDEa2line + IDEa1line + IDEa0line

;IDE Command Constants.  These should never change.

COMMANDrecal    EQU     10H
COMMANDread     EQU     20H
COMMANDwrite    EQU     30H
COMMANDinit     EQU     91H
COMMANDid       EQU     0ECH
COMMANDspindown EQU     0E0H
COMMANDspinup   EQU     0E1H
;
; IDE Status Register:
;  bit 7: Busy 1=busy, 0=not busy
;  bit 6: Ready 1=ready for command, 0=not ready yet
;  bit 5: DF   1=fault occured on the IDE drive
;  bit 4: DSC  1=seek complete
;  bit 3: DRQ  1=data request ready, 0=not ready to xfer yet
;  bit 2: CORR 1=correctable error occured
;  bit 1: IDX  vendor specific
;  bit 0: ERR  1=error occured

MAXSEC          EQU     3DH             ;Sectors per track for CF my Memory drive, Kingston CF 8G. (CPM format, 0-3CH)
                                        ;translates to LBA format of 1 to 3D sectors, for a total of 61 sectors/track.
                                        ;This CF card actully has 3F sectors/track. Will use 3D for my CPM86 system
because
                                        ;my Seagate drive has 3D sectors/track. Don't want different CPM86.SYS files
around
                                        ;so this program will also work with a Seagate 6531 IDE drive

DOS_MAXSEC      EQU     3FH             ;For MS-DOS BIOS Setting "Hard Disk" to Custom type (CF Card, 63 Sectors/track)
DOS_MAXHEADS    EQU     10H             ;16 head(s)
```

```
DOS_MAXCYL_L   EQU     0FFH            ;Low Byte maximum cylinder (sent via INT 13H's in CH)
DOS_MAXCYL     EQU     1024            ;Max cylinders
DOS_MAXSEC_CYL EQU     0FFH            ;3FH, maximum sector number (bits 5-0)+ two Cyl High Bits (Sectors numbered
1....x)


;--------------S100Computers PORTS FOR FOR Z80/WD2793 FDC Board

S100DATAA      EQU     10H             ;IN,  S100 Data port to GET data to from FDC Board
S100DATAB      EQU     10H             ;OUT, S100 Data port to SEND data to FDC Board
S100STATUSA    EQU     11H             ;Status port for A
S100STATUSB    EQU     11H             ;Status port for B
RESETZFDCPORT  EQU     13H             ;Port to reset ZFDC Z80 CPU.

STATUSDELAY    EQU     20              ;Time-out for waiting for ZFDC Board handshake signal (Now, ~0.5 seconds @ 8MHz
8086)
SECTOR_TIMEOUT EQU     400H            ;Value for sector R/W status check countdown (For 6-8MHz 8086, not critical)

ZFDCUNINITILIZED EQU   0FFH            ;If ZFDC is not yet initilized
ZFDCNOTWORKING EQU     0FEH            ;If ZFDC is not working
ZFDCNOTPRESENT EQU     0FDH            ;If ZFDC board is absent
ZFDCINITILIZED EQU     000H            ;If ZFDC is initilized OK

STD8IBM        EQU     1               ;ZFDC Board Format table # for IBM 8" SDSS Disk
MSDOS2         EQU     13H             ;Disk format type # for ZFDC board  (MS-DOS V2.0 Disk, 512 X 9 Sec/Track)
IBM144         EQU     15H             ;Disk format type # for 1.4M DDDS, 18 X 512 Byte Sectors, 80 Tracks

CMD_SET_FORMAT EQU     4H              ;This will select a specified drive and assign a disk format table to that drive
CMD_SET_DRIVE  EQU     5H              ;This will select a specified drive (0,1,2,3)
CMD_SET_TRACK  EQU     7H              ;This will set head request to a specified track
CMD_SET_SIDE   EQU     8H              ;This will set side request to a specified side
CMD_SET_SECTOR EQU     9H              ;This will set sector request to a specified sector
CMD_SET_HOME   EQU     0AH             ;This will set head request to Track 0 of CURRENT drive
CMD_SEEEK_TRACK        EQU     0EH             ;Seek to track to (IY+DRIVETRACK) with the track verify bit set on CURRENT
drive/format
CMD_FORMAT_TRACK EQU   16H             ;Format the floppy disk in the of the CURRENT drive using the current format
assigned to that disk
CMD_HANDSHAKE  EQU     21H             ;Handshake command only sent during board initilization/testing

                                       ;These new commands are required for R/W MSDOS Double sided disks
CMD_DOS_RD_MULTI_SEC    EQU 2BH                ;MS-DOS, Read data from multiple sectors starting at the CURRENT sector.
CMD_DOS_WR_MULTI_SEC    EQU 2CH                ;MS-DOS, Write data to multiple sectors starting at the CURRENT sector.
CMD_GET_SIDE           EQU 2DH         ;Get the current selected side of the current selected drive
CMD_DOS_SET_SECTOR     EQU 2EH         ;MS-DOS, Set current sector for the next sec R/W



;Possible ERROR codes returned from the ZFDC Board:-
;These will be translated into ASCII strings in the error reporting function.
;See the ZFDC code for a complete set of possible error coded returned byt the ZFDC Board

NO_ERRORS_FLAG EQU     00H             ;No Errors flag for previous cmd, sent back to S-100 BIOS
CONFIRM_FORMAT EQU     32H             ;Confirm disk format cmd request
DISK_WP_ERR    EQU     31H             ;Sector write error, Disk is write protected
ABORT_FLAG     EQU     3AH             ;Special error flag to signify the user aborted a command

ZFDC_ABSENT    EQU     3BH             ;If ZFDC Board is absent
ZFDC_INIT_ERROR        EQU     3CH             ;If ZFDC initilization error

TIMEOUT_ERROR  EQU     3DH             ;Error flag to signify the previous command timed out
CMD_RANGE_ERR  EQU     3EH             ;CMD out or range.

MAX_ERRORS     EQU     3FH             ;0 to 3FH errors only

                                       ;Meanings for disk status (as returned by IBM BIOS ROM)
seekerr        equ     40h             ;seek failed
hdwerr         equ     20h             ;controller chip failed
crcerr         equ     10h             ;crc error
dmaerr         equ     09h             ;DMA across 64k boundary
```

```
wpterr          equ     03h             ;write protected disk
rnferr          equ     04h             ;sector not found
timerr          equ     80h             ;Floppy time out error
cmderr          equ     01h             ;Floppy bad command for controller

msize           equ     280H            ;Total RAM memory size, (640K)
romdat          equ     0h              ;Data area for ROM usage  (DS will be set to 0H for data at 400H....)

COUNTS_SEC      equ     18              ;Timer tick conversion valuse
COUNTS_MIN      equ     1092
COUNTS_HOUR     equ     7               ;Adjustment for hours


;--------------Other Hardware Equates

SW86            EQU     0EDH            ;INPUT FROM THIS PORT SWITCHES THE 8086/80286 BACK to the Z80 in hardware



;================================ Start of BIOS code segment ================================================

                CPU 8086                ;No 80286/386 opcodes
                [BITS 16]

 SECTION        .text
                org     0H              ;<-- This must be 0. All addresses relative to this location
                                        ;For debugging/testing this monitor will reside in RAM ath F000:8000H with
                                        ;the stack at F000:7FFFH.  In the final EEPROM it will be placed at
                                        ;F000:C000H and the stack at F000:BFFFH.
                                        ;The IBM PC starts its ROM monitor at F000:E000H.
        %if     MONITOR_ROM
        TIMES   8000H   DB      0H      ;<--- The program will run here. F000:8000H for (part of) 256K EPROMS's
        %else
        TIMES   2000H   DB      0H      ;At F000:2000H in RAM for testing/debugging
        %endif

BEGIN:  jmp     INIT                    ;Reset all registers, initilize hardware
        jmp     WARM_INIT               ;warm start
        jmp     CI                      ;console input
        jmp     RI                      ;reader output
        jmp     CO                      ;console output (Character in CL)
        jmp     POO                     ;punch output
        jmp     LIST_OUT                ;printer output (Character in CL)
        jmp     CSTS                    ;consol status
        jmp     CICO                    ;console in with echo
        jmp     LIST_STATUS             ;printer status



INIT:   cld                             ;Set direction up. Through this monitor this is the default direction
        cli                             ;Disabel interrupts initially

        MOV     AL,00000000B            ;ALL LED's ON, for VISUAL DIAGNOSTIC we are alive
        OUT     DIAG_LEDS,AL            ;LED's will go off one at a time

        IN      AL,IOBYTE               ;If bit 8 of Port EFH is 0, Then force jump to this Monitor (Note, RAM disk with
CPM3 will be invalid)
        AND     AL,80H                  ;If bit 1 is 1 then see if CPM86/MSDOS is present in RAM at 0000:0500H
        JZ      ToMonitor               ;IF, present jump to that loction.


        MOV     BX,500H                 ;Normally my CPM86.COM (or MSDOS.COM) program will have 90H,90H in RAM at 500H.
        MOV     AX,0                    ;Check this value is here. If so, chances are we have CPM86 or MSDOS loaded in
RAM
        MOV     DS,AX                   ;If not then skip to this monitor
        CMP     word[BX],9090H          ;Was it a reset requiring CPM86.
        JNZ     ToMonitor               ;Set pointers for IBM-PC BIOS interrupt vectors in low RAM

CPM86Boot:                              ;If 90H,90H at 500H in RAM
```

```
        JMP     word  0000H:500H      ;Far Jump to 500H in RAM (where CPM86 resides)

ToMonitor:                          ;If not, then jump to this 8086 Monitor in ROM
        cld                         ;Set direction up. Through this monitor this is the default direction
        cli                         ;Disabel interrupts

        MOV     AL,10000000B        ;1st LED off, for VISUAL DIAGNOSTIC we are alive
        OUT     DIAG_LEDS,AL        ;LED's will go off one at a time

                                    ;We will now set up a valid stack. Normally there will be 1MG of RAM
                                    ;in the system so there will be RAM just below this EEPROM at F000:C000H
                                    ;If so, we will place the stack just below the EPROM.
                                    ;If however there is less memory we will find the highest RAM and place
                                    ;the stack at the top of availabel RAM

        mov     ax,cs               ;Note cs will be F000H
        mov     ds,ax               ;DS will also be CS:F000H
        mov     es,ax               ;As will ES
        mov     ss,ax               ;For now, SS also set to CS:F000H

        %if     MONITOR_ROM
        mov     bp,7000H            ;BP for IDE RAM variables (will normally be SS:[BP])
        MOV     BX,8000H-2          ;F000:BFFEH, check if we have RAM immediatly below this PROM

TOP_OF_RAM:
        MOV     AX,[SS:BX]
        NOT     AX
        MOV     [SS:BX],AX
        CMP     [SS:BX],AX          ;Is there real RAM there.
        JNZ     NO_RAM              ;If no RAM then search for lower memory

        MOV     SP,BX
        JMP     DoneStack

NO_RAM: MOV     AX,SS               ;Try 64K lower....
        SUB     AX,1000H
        MOV     SS,AX
        OR      AX,AX
        JNZ     TOP_OF_RAM

        MOV     SP,4FEH             ;Special case if <= 64K RAM
                                    ;Point to a RAM area (0000:4FE), assume we have at least this ammount.
                                    ;In this case BP will be at 0000:7000H. If no RAM there we are out of luck
        %else
        mov     bp,1000H
        MOV     SP,1FFFH
        %endif

DoneStack:
        MOV     AL,11000000B        ;2nd LED off, VISUAL DIAGNOSTIC for Stack done
        OUT     DIAG_LEDS,AL

        MOV     AL,0FFH             ;Clear Printer strobe, comes up 0 on a reset
        OUT     PRINTER_STROBE,AL

        MOV     AL,11100000B        ;3rd LED off, VISUAL DIAGNOSTIC
        OUT     DIAG_LEDS,AL

        mov     bx,SIGNON           ;Signon notice
        call    PRINT_STRING        ;Note up until now stack was not used

        MOV     AL,11110000B        ;4th LED off, VISUAL DIAGNOSTIC
        OUT     DIAG_LEDS,AL

        mov     bx,SMSG
        call    STOMM               ;Speak out the message

        MOV     AL,11111000B        ;5 LED's off if speech sent
        OUT     DIAG_LEDS,AL
```

```
        CALL    TIME                    ;PRINT TIME ON CRT
        CALL    CRLF

        MOV     AL,11111100B            ;6 LED's off if Time is obtained
        OUT     DIAG_LEDS,AL

        CALL    PRINT_SEG_REGISTERS     ;Display all four segment registers

        MOV     AL,11111110B            ;7 LED's off, if all initilization is done.
        OUT     DIAG_LEDS,AL
        JMP     MAINLOOP
WARM_INIT:
        cld                             ;Set direction up
        cli                             ;Disabel interrupts
        mov     ax,cs                   ;Note cs always will be F000H
        mov     ds,ax                   ;DS & ES will  be set to F000H as default values within this monitor
        mov     es,ax

MAINLOOP:
        mov     bx,CLEANUP              ;Clear line and '>'
        call    PRINT_STRING

        call    CICO                    ;Get a command from Console
        mov     ah,0
        cmp     al,'A'
        jb      WARM_INIT               ;must be A to Z
        cmp     al,'Z'
        jg      WARM_INIT
        sub     al,'A'                  ;calculate offset
        shl     al,1                    ;X 2
        add     ax,ctable
        mov     bx,ax
        mov     ax,[CS:BX]              ;get location of routine
        CALL    AX                      ;<-----------This is the Main Monitor CMD call
        jmp     WARM_INIT               ;finished


;************************************ Basic Monitor Commands ****************************************

;------- PRINT MENU ON CRT

KCMD:   MOV     BX,MAIN_MENU
        CALL    PRINT_STRING
        JMP     INIT                    ;Will re-initilize everything just in case!

;-------------------MAP the 1MG Adderss space ------------------------------------------------------

MAP:    call    CRLF                    ;Display complete memory map woth R=ram, P=PROM and "." empty space
        mov     ax,0
        mov     ds,ax                   ;Must start in first segment in DS:
        mov     dl,64                   ;character count
        mov     dh,4                    ;segment counter(4 lines per segment)
        mov     SI,ax                   ;need to reset bx (ds = 0 already)
        call    SHOW_ADDRESS_DS                 ;start with address, Send to console the address DS+SI

map1:   mov     ax,[SI]                 ;remember ds is assumed
        not     ax                      ;complement data
        mov     [SI],ax
        cmp     ax,[SI]                 ;did it change
        jne     not_ram
        not     ax                      ;correct data
        mov     [SI],ax
        mov     cl,'R'
        jmp     nextbk                  ;get next block

not_ram:cmp     ax,0                    ;ffff->0  must be rom if not 0
        jne     prom
        mov     cl,'.'                  ;no need to correct data for here
```

```
        jmp     nextbk                  ;get next block
prom:   mov     cl,'p'
nextbk: call    CO                      ;send the R,P or "."

        ADD     SI,100H                 ;check every 100h at a time
        dec     dl                      ;64X1000H across
        jnz     map1                    ;one line of 64K done

        mov     dl,64                   ;reset counter for next line
        dec     dh                      ;segment counter
        jnz     noseg
        mov     ax,ds
        add     ax,1000h
        jc      mapdone
        mov     ds,ax
        mov     dh,4
noseg:  CALL    CRLF_CHECK              ;Print current address at start of each line
        call    SHOW_ADDRESS_DS
        jmp     map1
mapdone:ret




;-------Fill memory with a constant value. Up to 64K bytes from xxxxxH to xxxxxH--------------------------

FILL:   CALL    GET5DIGITS              ;Get (up to) 20 bit parameter. 16 bit value (4 digits) to DI.
                                        ;If 5 digits, then the first digit is put in ES (highest nibble)
        PUSH    ES
        PUSH    DI                      ;Save start address for now = ES:DI

        CALL    GET5DIGITS

        MOV     SI,DI                   ;Put end address in DS:SI
        MOV     AX,ES
        MOV     DS,AX                   ;If 5 digits, then the first digit is put in DS

        POP     DI
        POP     ES                      ;Start=ES:DI  End=DS:SI

        CALL    CLENGTH                 ;Length cx = (ds:si-es:di)+1, if >64K then err

        CALL    GET2DIGITS              ;Fill value to AL (CX unaltered)

                                        ;ES:DI = start address, (DS:SI = end address, not used), CX = count, AL = fill
value
filoop: mov     [ES:DI],al              ;Note RAM is es:[di], count in CX
        inc     DI
        CMP     DI,0                    ;Check if we are crossing a segment boundry
        JNZ     filoop1
        MOV     AX,ES
        ADD     AX,1000H
        MOV     ES,AX
filoop1:loop    filoop                  ;Dec CX to 0
        ret



;--------------Display memory contents

DISPLAY_RAM:
        CALL    GET5DIGITS              ;Get (up to) 20 bit parameter. 16 bit value (4 digits) to DI.
                                        ;If 5 digits, then the first digit is put in ES (highest nibble)
        PUSH    ES
        PUSH    DI                      ;Save start address for now.

        CALL    GET5DIGITS
        MOV     SI,DI                   ;Put end address in SI
```

```
        MOV     AX,ES
        MOV     DS,AX                   ;If 5 digits, then the first digit is put in DS

        POP     DI
        POP     ES                      ;Start=ES:DI  End=DS:SI

        AND     DI,0FFF0h               ;even up printout
        OR      SI,000Fh                ;also nice ending for Ray G.

        call    CLENGTH                 ;Length cx = (ds:si-es:di)+1, if length > 64K then err

dloop6: CALL    CRLF_CHECK              ;Note BX,CX is saved, ESC at keyboard will abort
        call    SHOW_ADDRESS_ES                 ;Send start address

        MOV     DL,16                   ;First print a line of 16 Hex byte values
        PUSH    CX
        PUSH    DI
        PUSH    ES

dloop1: mov     al,[es:di]              ;Will increment DI
        call    AL_HEXOUT
        call    BLANK
        call    Inc_DI_boundry_check    ;Will increase DI
        DEC     DL                      ;Have we done 16 bytes yet
        jnz     dloop1
                                        ;Now print ascii for those 16 bytes
        mov     cx,6                    ;first send 6 spaces
        call    TABS

        MOV     DL,16                   ;16 across again
        POP     ES
        POP     DI
        POP     CX
dloop2: mov     al,[es:DI]
        and     al,7fh
        cmp     al,' '                  ;filter out control characters
        jnc     dloop3
dloop4: mov     al,'.'
dloop3: cmp     al,'~'
        jnc     dloop4
        PUSH    CX
        mov     cl,al
        call    CO
        POP     CX
        loop    dloop5                  ;--CX has total byte count
        ret
dloop5: call    Inc_DI_boundry_check
        DEC     DL                      ;Have we done 16 bytes yet
        jnz     dloop2
        JMP     dloop6


Inc_DI_boundry_check:                   ;Check if we are crossing a segment boundry
        inc     DI                      ;If so, inc [ES]
        CMP     DI,0
        JNZ     bounds1
        MOV     AX,ES
        ADD     AX,1000H
        MOV     ES,AX
bounds1:RET


Inc_SI_boundry_check:                   ;Check if we are crossing a segment boundry
        inc     SI                      ;If so, inc [DS]
        CMP     DI,0
        JNZ     bounds2
        MOV     AX,DS
        ADD     AX,1000H
        MOV     DS,AX
```

```
bounds2:RET

;----------------DISPLAY ASCII in Memory ------------------------------------------------------------

DISPLAY_ASCII:
        CALL    GET5DIGITS              ;Get (up to) 20 bit parameter. 16 bit value (4 digits) to DI.
                                        ;If 5 digits, then the first digit is put in ES (Highest nibble)
        PUSH    ES
        PUSH    DI                      ;Save start address for now.

        CALL    GET5DIGITS
        MOV     SI,DI                   ;Put end address in DS:SI
        MOV     AX,ES
        MOV     DS,AX                   ;If 5 digits, then the first digit is put in DS

        POP     DI
        POP     ES                      ;Start=ES:DI  End=DS:SI

        AND     DI,0FFF0H               ;even up printout
        OR      SI,000FH                ;also nice ending for Ray G.

        call    CLENGTH                 ;Length cx = (ds:si-es:di)+1, if length > 64K then err

aloop6: CALL    CRLF_CHECK              ;Note BX,CX is saved, ESC at keyboard will abort
        call    SHOW_ADDRESS_ES                 ;Send start address

        mov     dl,64                   ;64 characters across
aloop2: PUSH    CX                      ;put length on stack
        mov     al,[es:di]
        and     al,7fh
        cmp     al,' '                  ;filter out control characters
        jnc     aloop3
aloop4: mov     al,'.'
aloop3: cmp     al,'~'
        jnc     aloop4
        mov     cl,al
        call    CO
        POP     CX
        LOOP    aloop5                  ;--CX has total byte count
        ret
aloop5: call    Inc_DI_boundry_check
        dec     dl                      ;Have we done 64 characters across yet
        jnz     aloop2
        JMP     aloop6



;----------------MOVE Memory ------------------------------------------------------------

MOVE:   CALL    GET5DIGITS              ;Get (up to) 20 bit parameter. 16 bit value (4 digits) to DI.
                                        ;If 5 digits, then the first digit is put in ES (highest nibble)
        PUSH    ES                      ;Do everything relative to first ES value
        PUSH    DI                      ;Save start address for now.

        CALL    GET5DIGITS
        MOV     SI,DI                   ;Put end address in SI
        MOV     AX,ES
        MOV     DS,AX                   ;If 5 digits, then the first digit is put in DS

        POP     DI
        POP     ES                      ;Start=ES:DI  End=DS:SI

        call    CLENGTH                 ;Length cx = (ds:si-es:di)+1, if length > 64K then err

        PUSH    ES
        PUSH    DI                      ;Save Start Address ES:DI
        PUSH    CX                      ;Save length

        CALL    GET5DIGITS              ;For Destination, get (up to) 20 bit parameter. 16 bit value (4 digits) to DI.
```

```
        MOV     SI,DI                   ;Put destination address in DS:SI
        MOV     AX,ES
        MOV     DS,AX                   ;If 5 digits, then the first digit is put in DS


        POP     CX                      ;Get length
        POP     DI                      ;Get start ES:DI  destination DS:SI
        POP     ES                      ;Get back the initial ES value (often 0)


MOVE1:  MOV     AL,[ES:DI]              ;Note cannot use MOVS opcode because of segment boundries
        MOV     [DS:SI],AL
        CALL    Inc_DI_boundry_check    ;Check if we are crossing a segment boundry
        CALL    Inc_SI_boundry_check
MOVE3:  LOOP    MOVE1
        RET



;----------------SUBSTITUTE Memory ------------------------------------------------------------

SUBSTITUTE:
        CALL    GET5DIGITS              ;Get (up to) 20 bit parameter. 16 bit value (4 digits) to DI.
                                        ;If 5 digits, then the first digit is put in ES (highest nibble)
        CALL    CRLF
nusloop:call    SHOW_ADDRESS_ES
        mov     cx,8
sloop:  call    BLANK
        mov     al,[es:DI]
        push    cx
        push    ax
        call    AL_HEXOUT
        mov     cl,'-'
        call    CO
        pop     ax
        pop     cx
        call    GET2DIGITS              ;Get 8 bit value (2 digits) to AL. (BX, CX & DX Unchanged), terminator in AH
        cmp     ah,CR                   ;CR signals we are done
        je      qtest
        cmp     ah,ESC                  ;Also ESC
        je      qtest
        cmp     ah,' '                  ;is a SP so skip to next byte
        je      snext1
        mov     [es:DI],al
snext1: inc     DI
        CMP     DI,0
        JNZ     snext2
        MOV     AX,ES
        ADD     AX,1000H
        MOV     ES,AX
snext2: loop    sloop
        jmp     nusloop
qtest:  ret



;-------------------------Verify Memory Contents ------------------------------------

VERIFY: CALL    GET5DIGITS              ;Get (up to) 20 bit parameter. 16 bit value (4 digits) to DI.
                                        ;If 5 digits, then the first digit is put in ES (highest nibble)
        PUSH    ES                      ;Do everything relative to first ES value
        PUSH    DI                      ;Save start address for now.

        CALL    GET5DIGITS
        MOV     SI,DI                   ;Put end address in DS:DI
        MOV     AX,ES
        MOV     DS,AX                   ;If 5 digits, then the first digit is put in DS
```

```
        POP     DI
        POP     ES                      ;Start=ES:SI  End=DS:DI

        call    CLENGTH                 ;Length cx = (ds:si-es:di)+1, if length > 64K then err

        PUSH    ES
        PUSH    DI                      ;Save Start Address
        PUSH    CX                      ;Save length

        CALL    GET5DIGITS              ;For Destination, get (up to) 20 bit parameter. 16 bit value (4 digits) to DI.
        MOV     SI,DI                   ;Put destination address in DS:SI
        MOV     AX,ES
        MOV     DS,AX                   ;If 5 digits, then the first digit is put in DS

        POP     CX
        POP     DI
        POP     ES                      ;Get back the initial ES value (often 0)

        MOV     BX,0                    ;Count of mis-matches

VERIFY1:MOV     AL,[ES:DI]              ;cannot use cmps because of segments
        CMP     AL,[DS:SI]
        JZ      MATCH_OK
        call    verr
MATCH_OK:
        INC     DI
        CMP     DI,0                    ;Check if we are crossing a segment boundry
        JNZ     VERIFY2
        MOV     AX,ES
        ADD     AX,1000H
        MOV     ES,AX

VERIFY2:INC     SI
        CMP     SI,0                    ;Check if we are crossing a segment boundry
        JNZ     VERIFY3
        MOV     AX,DS
        ADD     AX,1000H
        MOV     DS,AX

VERIFY3:LOOP    VERIFY1
        CMP     BX,0                    ;Was there any errors
        JNZ     TOTAL_MISMATCHES
        MOV     BX,MATCHES_OK
        CALL    PRINT_STRING
TOTAL_MISMATCHES:
        RET

verr:   CMP     BX,0                    ;Save count, print error
        JNZ     SKIP_DIFF_MSG

        PUSH    DS
        MOV     AX,CS
        MOV     DS,AX
        MOV     BX,DIFF_Header_Msg
        CALL    PRINT_STRING
        POP     DS

SKIP_DIFF_MSG:
        CALL    CRLF                    ;There is a mis-match show values
        call    SHOW_ADDRESS_ES
        PUSH    CX
        MOV     CX,3
        call    TABS
        POP     CX
        mov     al,[ES:DI]
        mov     dh,al                   ;store AL in DH for comparison below
        call    AL_HEXOUT
        PUSH    CX
        MOV     CX,7
```

```
        call    TABS
        POP     CX
        call    SHOW_ADDRESS_DS
        PUSH    CX
        MOV     CX,7
        call    TABS
        POP     CX
        mov     al,[ds:SI]
        push    ax
        call    AL_HEXOUT
        PUSH    CX
        MOV     CX,8
        call    TABS
        POP     CX
        pop     ax
        xor     al,dh                   ;stored from above
        call    AL_BINOUT
        call    CTRL_CHECK
        INC     BX                      ;This prevents the header being show each time
        ret

;---------------- Simple test of RAM (Continous)----------------------------------------

TEST_RAM:
        mov     bx,JMSG                 ;Will test memory forever
        call    PRINT_STRING

        CALL    GET5DIGITS              ;Get (up to) 20 bit parameter. 16 bit value (4 digits) to DI.
                                        ;If 5 digits, first digit entered to ES (Highest nibble)
        PUSH    ES
        PUSH    DI                      ;Save start address for now.

        CALL    GET5DIGITS
        MOV     SI,DI                   ;Put end address in DS:SI
        MOV     AX,ES
        MOV     DS,AX                   ;If 5 digits, then the first digit is put in DS

        POP     DI
        POP     ES                      ;Start=ES:BX  End=DS:DX

        CALL    CLENGTH                 ;Length cx = (ds:SI-es:DI)+1, if length > 64K then err

        MOV     DX,0                    ;Test loop count

        PUSH    CX                      ;CX has length
        mov     bx,STARTJMSG            ;Test memory until ESC
        call    PRINT_STRING
        POP     CX

mtest1: push    cx
        push    di

mtloop: mov     al,[es:DI]
        mov     ah,al                   ;Store value currently in RAM
        not     al
        mov     [es:DI],al
        mov     al,[es:DI]
        not     al
        cmp     al,ah
        jne     terr
        mov     [es:DI],ah
tnext:  inc     DI
        CMP     DI,0
        JNZ     tnext2
        mov     AX,ES
        ADD     ax,1000H
        MOV     ES,AX
tnext2: call    CTRL_CHECK              ;See if an abort is requested
        loop    mtloop                  ;Repeat for "length" number of bytes
```

```
        mov     bx,RAM_Test_Count
        CALL    PRINT_STRING
        inc     dx
        MOV     AX,DX
        CALL    AX_HEXOUT
        mov     bx,H_MSG              ;H.
        CALL    PRINT_STRING
        pop     di                    ;Repeat the whole process
        pop     cx
        jmp     mtest1                ;test forever


terr:   push    DX
        mov     dx,ax                 ;save data in dx
        CALL    CRLF
        call    SHOW_ADDRESS_ES
        mov     ax,dx                 ;get back data
        xor     al,ah                 ;identify bits
        mov     dx,ax
        call    AL_HEXOUT
        call    BLANK
        mov     ax,dx
        call    AL_BINOUT
        pop     dx
        jmp     tnext



;------------- QUERY PORTS -----------------------------------------------

QUERY:  call    CICO                  ;is it input or output
        cmp     al,'I'
        jz      input
        cmp     al,'O'
        jz      output
        jmp     ERR                   ;if not QI or QO then error

input:  call    GET4DIGITS            ;Get 8 or 16 bit value (2 or 4 digits) to DI, terminator in AH
        call    CRLF
        mov     dx,DI
        in      al,dx                 ;Note will assume here we have just an 8 bit port
        push    ax
        call    AL_HEXOUT             ;Show value in HEX
        call    BLANK
        pop     ax
        call    AL_BINOUT             ;Show value in binary
        ret


output: call    GET4DIGITS            ;Get 8 or 16 bit value (2 or 4 digits) to DI, terminator in AH
        mov     dx,DI
        CALL    GET2DIGITS            ;Output value to AL (BX unaltered)
        PUSH    AX
        CALL    CRLF
        POP     AX
        out     dx,al                 ;Send 8 bit value in AL to port at [DX]
        RET



;------------- GO TO A RAM LOCATION ----------------------

GOTO:   mov     bx,GET_SEG_MSG        ;Segment=
        call    PRINT_STRING
        call    GET4DIGITS            ;Get (up to) 16 bit value (4 digits) to BX.
        PUSH    DI                    ;Save Segment (in [DI]) on stack)
```

```
        mov     bx,GET_OFFS_MSG              ;Offset=
        call    PRINT_STRING
        call    GET4DIGITS
        PUSH    DI                   ;Save Offset (in [DI]) on stack)
        RET                          ;Will pop offset, then CS and go there



;------------- SWITCH CONTROL BACK TO Z80 (Master) ------------------

Z80:    in      al,SW86              ;This switches control back over to Z80
        nop
        nop
        nop
        nop
        nop
        JMP     BEGIN



;------------ HEX MATH ----------------------------------------------

HEXMATH:mov     bx,MATH_MSG          ;HEX MATH
        call    PRINT_STRING
        call    GET4DIGITS
        push    DI                   ;save data for the moment
        call    GET4DIGITS
        push    DI
        mov     bx,MATH_HEADER
        call    PRINT_STRING
        pop     DI                   ;get back data2 (DI=data2)
        pop     BX                   ;and data1  (BX=data1)

        push    DI                   ;save them again for below
        push    BX
        add     BX,DI
        call    BX_HEXOUT            ;Show addition (data1+data2)
        mov     cx,2                 ;skip over 2 spaces
        call    TABS
        pop     BX                   ;get back data1 one more time
        pop     DI                   ;and data2
        sub     bx,DI                ;data1-data2
        call    BX_HEXOUT
        ret


;--------------- JUMP to 500H IN RAM
                                     ;"W" command
JMP_500H:
        JMP     word  0000H:500H     ;Far Jump to whatever is at 0;500H


;---------------- "PATCH" This is a Q&D patch to test RAM versions of this code loaded via CPM.

PATCH:  mov     bx,PATCH_MSG
        call    PRINT_STRING
        MOV     AX,0H
        MOV     DS,AX
        MOV     AX,0F000H
        MOV     ES,AX
        MOV     SI,2100H
        MOV     DI,2000H
        MOV     CX,2F00H             ;2X Count of bytes to move (F000:2000 ~ F000:7C00H)
        STD

PATCH1: MOV     AX,[DS:SI]           ;cannot seem to get REP       movsw to work!
```

```
        MOV     [ES:DI],AX
        INC     SI
        INC     SI
        INC     DI
        INC     DI
        LOOP    PATCH1

        JMP     word  0F000H:2000H    ;Far Jump to F000H:2000H
                                      ;DB 0EAH,00H,20H,00H,0F0H


;-------------- Display all active IO input ports in the system -------------
;               Will first do lower 256 8 bit ports, then 64K 16 bit ports

INPORTS:mov     bx,PORTS8_MSG
        call    PRINT_STRING
        MOV     CH,4                  ;Display 4 ports across
        MOV     DX,00FFH              ;Will contain port number
LOOPIO: CMP     DL,SW86               ;INPUT FROM THIS PORT SWITCHES THE 80286 BACK to THE Z80
        JZ      SKIP                  ;So skip it
        IN      AX,DX                 ;Get port data (note, 8 bits port Input will be in AH)
        CMP     AH,0FFH               ;No need for 0FF's
        JZ      SKIP
        MOV     BH,AH                 ;Store for now

        MOV     AL,DL                 ;Print Address (lower 8 bits)
        CALL    AL_HEXOUT

        MOV     CL,'-'                ;Put in a "->"
        CALL    CO
        MOV     CL,'>'
        CALL    CO
        MOV     AL,BH
        CALL    AL_HEXOUT

        MOV     CL,TAB
        CALL    CO

        DEC     CH                    ;4 ports across per line
        JNZ     SKIP
        MOV     CH,4
        CALL    CRLF
SKIP:   DEC     DL                    ;Next Port
        JNZ     LOOPIO

                                      ;NOW do the the 16 bit ports.

        mov     bx,PORTS16_MSG
        call    PRINT_STRING
        MOV     CH,4                  ;Display 4 ports across
        MOV     DX,00FFH              ;Will contain port number
LOOPIO16:
        CMP     DL,SW86               ;INPUT FROM THIS PORT SWITCHES THE 80286 BACK to THE Z80
        JZ      SKIP16                ;So skip it
        IN      AX,DX                 ;Get port data (0FFFFH if none)
        CMP     AX,0FFFFH             ;No need for 0FF's
        JZ      SKIP16
        MOV     BX,AX                 ;Store for now

        MOV     AL,DH                 ;Print Address
        CALL    AL_HEXOUT
        MOV     AL,DL
        CALL    AL_HEXOUT

        MOV     CL,'-'                ;Put in a "->"
        CALL    CO
        MOV     CL,'>'
        CALL    CO
        MOV     AL,BH
```

```
        CALL    AL_HEXOUT
        MOV     AL,BL
        CALL    AL_HEXOUT

        MOV     CL,TAB
        CALL    CO

        DEC     CH                      ;4 ports across per line
        JNZ     SKIP16
        MOV     CH,4
        CALL    CRLF
SKIP16: DEC     DL                      ;Next Port
        CMP     DL,0FFH                 ;Have we done a full loop
        JNZ     LOOPIO16

        MOV     CH,4
        CALL    CRLF
        CALL    CRLF
        INC     DH                      ;Next batch of 256 ports
        PUSH    BX
        PUSH    CX
        PUSH    DX
        mov     bx,MORE_MSG
        call    PRINT_STRING
        call    CICO
        POP     DX
        POP     CX
        POP     BX
        cmp     al,'Y'
        JNZ     DONEP16
        CALL    CRLF
        CMP     DX,0
        JNZ     LOOPIO16
DONEP16:RET



;------THIS IS A ROUTINE TO GET THE TIME DATA FROM THE 58167 chip on a S100Computers Serial IO Board -----------

TIMEC:  MOV     BX,TMMSG
TIMECC: CALL    PRINT_STRING
        CALL    TIME
        CALL    CRLF
        RET

UPDATE: CALL    TIMEC
        MOV     AL,15H                  ;GO register
        OUT     RTCSEL,AL
        MOV     AL,0H
        OUT     RTCDATA,AL
        MOV     BX,MSG30                ;ADJ TIME SIGNON
        JMP     TIMECC
                                        ;Because the CPM3 Time driver (origionally from the SD Systems
                                        ;SIO-6 S-100 board) & the S100Computers Serial IO S-100 board
                                        ;(which I use) stores its own data in registers on the clock chip to calculate
                                        ;dates, I cannot use the registers directly here for Dates.
                                        ;Time is OK however.  See HBOOT3.ASM in the CPM3 folder for more info
TIME:
;       MOV     AL,07H                  ;Point to MONTH
;       CALL    CLKREG                  ;Print it
;       MOV     CL,'/'
;       CALL    CO
;
;       MOV     AL,06H                  ;Point to Day
;       CALL    CLKREG                  ;Print it
;       MOV     BX,MSG16T               ;"20" for year 200xx
;       CALL    PRINT_STRING
;
;       MOV     AL,09H                  ;Point to RAM store of YEAR (0-99)
```

```
;       CALL    CLKREG                  ;Print it
;       MOV     BX,MSG12T               ;Space between date and time
;       CALL    PRINT_STRING

        MOV     AH,0H                   ;For PRINT_REG Below
        MOV     AL,04H                  ;Point to HOURS
        CALL    CLKREG                  ;Print BCD in [AL]
        MOV     CL,':'
        CALL    CO

        MOV     AL,03H                  ;Point to MINS
        CALL    CLKREG                  ;Print BCD in [AL]
        MOV     CL,':'
        CALL    CO

        MOV     AL,02H                  ;Point to SEC
        CALL    CLKREG                  ;Print BCD in [AL]
        RET

CLKREG:                                 ;Get a clock Register
        OUT     RTCSEL,AL
        IN      AL,RTCDATA
PRINT_REG:                              ;;Print BCD in [AL]
        PUSH    AX
        MOV     CL,4
        RCR     AX,CL
        AND     AL,0FH
        ADD     AL,30H
        MOV     CL,AL                   ;Write high byte mins to CRT
        CALL    CO
        POP     AX

        AND     AL,0FH
        ADD     AL,30H
        MOV     CL,AL
        CALL    CO
        RET


;Run diagnostic tests on the 8259A PIC.
;Configured below for the S100Computers PIC/RTC S-100 Board

TEST_8259:
        mov     bx,PIC_SIGNON           ;Send a signon message
        call    PRINT_STRING
        CALL    CI                      ;Wait for stat key
        mov     bx,CRLFMSG              ;Send a CRLF
        call    PRINT_STRING

         mov     ax,cs
         mov     ds,ax
         sub     ax,ax
         mov     es,ax
        CLD                             ;Default to direction up

        mov     cx,256                  ;fill all 8086 interrupts initially with a default error trapping pointer
         sub     di,di                  ;clear destination register
Tloop:  mov     ax,BAD_SOFT_INT                 ;Send warning "Bad Interrupt call" and iret
         stosw
         mov     ax,cs                  ;Interrupt segment pointer to here.
         stosw
         loop    Tloop

        mov     al,MasterICW1           ;Initilize the 8259A PIC Controller
        out     MASTER_PIC_PORT,al
        mov     al,MasterICW2           ;Ints starts at 120H in RAM
        out     MASTER_PIC_PORT+1,al
        mov     al,MasterICW4           ;No slaves above, so 8259 does not expect ICW3
        out     MASTER_PIC_PORT+1,al
```

```
mov     al,0h                   ;NO mask (i.e. all 8 int lines will be accepted)
out     MASTER_PIC_PORT+1,al

MOV     DI,3FCH                 ;Location of INT FF (8259A is not putting vector on bus)
MOV     AX,TrapFFInt
MOV     [DI],AX

MOV     DI,000CH                ;Location for single byte 8086 CC debug trap
MOV     AX,DebugTrap            ;Location of Hardware Int V0 routine
MOV     [DI],AX
INC     DI
INC     DI
MOV     [DI],CS
                                ;Now setup >>>> 8259A HARDWARE <<<< INT vectors in low mamory
                                ;Now setup the 8 jump locations for the hardware
MOV     DI,MasterICW2*4                 ;<---- Location of 8259A INT table at end of this monitor ROM

MOV     AX,V0int                ;Location of Hardware Int V0 routine (Note DS: = 0)
MOV     [DI],AX                 ;We will now (one by one) put in the 8
INC     DI                      ;interrupt jump locations.
INC     DI
MOV     [DI],CS                 ;This will normally be 0F000H
INC     DI
INC     DI

MOV     AX,V1int                ;Location of Hardware Int V1 routine
MOV     [DI],AX
INC     DI
INC     DI
MOV     [DI],CS
INC     DI
INC     DI

MOV     AX,V2int                ;Location of Hardware Int V2 routine
MOV     [DI],AX
INC     DI
INC     DI
MOV     [DI],CS
INC     DI
INC     DI

MOV     AX,V3int                ;Location of Hardware Int V3 routine
MOV     [DI],AX
INC     DI
INC     DI
MOV     [DI],CS
INC     DI
INC     DI


MOV     AX,V4int                ;Location of Hardware Int V4 routine
MOV     [DI],AX
INC     DI
INC     DI
MOV     [DI],CS
INC     DI
INC     DI

MOV     AX,V5int                ;Location of Hardware Int V5 routine
MOV     [DI],AX
INC     DI
INC     DI
MOV     [DI],CS
INC     DI
INC     DI

MOV     AX,V6int                ;Location of Hardware Int V6 routine
MOV     [DI],AX
```

```
        INC     DI
        INC     DI
        MOV     [DI],CS
        INC     DI
        INC     DI

        MOV     AX,V7int              ;Location of Hardware Int V7 routine
        MOV     [DI],AX
        INC     DI
        INC     DI
        MOV     [DI],CS
        INC     DI
        INC     DI
                                      ;All Hardware Ints are setup, now allow int's
INT_LOOP:
        cli                           ;Just to be safe, stop ints during consol I/O
        mov     cl,'.'                ;For testing puroses just put the 8086 in a loop
        call    CO                    ;put continous series of dots on screen
        call    CTRL_CHECK            ;See if an abort is requested, if so JMP to err: will occour, then to start of
monitor.
        sti                           ;Are there any Ints?
        NOP                           ;Allow extra time
        JMP     INT_LOOP              ;Forever until you hit reset or ESC was entered


;-----------  Location of Monitor default Interrupt vector handeling routines -------------

;       Note this is for the 8259A Monitor HARDWARE Interrupt test/trap routine.
;       It is not the same as that used when the IBM-PC BIOS functions are used. See below.

BAD_SOFT_INT:
        PUSH    AX
        PUSH    BX
        PUSH    CX
        MOV     BX,TrapIntMSG         ;Announce we got a bad Interrupt call
         jmp     NO_INT_SUPPORT

TRAP_INT:
        Cli                           ;Critical area****
        MOV     BX,TrapIntMSG
        jmp     Int_msg

TrapFFInt:
        Cli                           ;Critical area****
        MOV     BX,TrapFFIntMSG
        jmp     Int_msg

DebugTrap:
        Cli                           ;Critical area****
        MOV     BX,DebugTrapMSG
        CALL    PRINT_STRING          ;General info dump routine
        IRET

NO_INT_SUPPORT:                               ;Common for warning about un-implemented int
        CALL    PRINT_STRING          ;Send msg pointed to by CS:BX
        POP     CX                    ;Note this routine is also used by the MS-DOS BIOS section
        POP     BX
        POP     AX
        iret                          ;Remember IRET collects the saved Flags


V0int:  Cli                           ;Critical area****
        MOV     BX,Int0MSG     ;Will arrive here from int vector at 20H in RAM
        jmp     Int_msg

V1int:  Cli                           ;Will arrive here from int vector at 24H in RAM
        MOV     BX,Int1MSG
        jmp     Int_msg
```

```
V2int:  Cli                         ;Will arrive here from int vector at 28H in RAM
        MOV     BX,Int2MSG
        jmp     Int_msg

V3int:  Cli
        MOV     BX,Int3MSG
        jmp     Int_msg

V4int:  Cli
        MOV     BX,Int4MSG
        jmp     Int_msg

V5int:  Cli
        MOV     BX,Int5MSG
        jmp     Int_msg

V6int:  Cli
        MOV     BX,Int6MSG
        jmp     Int_msg

V7int:  Cli
        MOV     BX,Int7MSG
        jmp     Int_msg

Int_msg:
        CALL    PRINT_STRING        ;General info dump routine
;       MOV     AL,00001011B        ;Send OCW3 (Read 8259A Interrupt Service Reg)
;       OUT     MASTER_PIC_PORT,AL
;       IN      AL,MASTER_PIC_PORT  ;Get and show Bit pattern returned.
;       CALL    ZBITS               ;Send bit pattern along with a CR/LF
        MOV     AL,NS_EOI           ;8259A End of Interrupt command, can now allow another interrupt
        OUT     MASTER_PIC_PORT,AL
        IRET



;*************************************************************************************************
;
;       Module to Test and diagnose the www.S100Computers.com IDE Board
;
;       Instead of using the CPM86 style DS:[BX] format, we will use SS:[BP] so the the buffers
;       can reside at the top segment of available RAM. Normally this will be F000:7000H but the monitor
;       will not assume the full 1MG address space is avalable.
;       See the monitor initilization section where BP is setup.
;
;*************************************************************************************************

MYIDE:  MOV     BP,DISPLAY_FLAG             ;Do we have detail sector data display flag on or off
        MOV     AL,0FFH             ;Set default to detailed sector display
        MOV     [BP],AL

        MOV     BX,IDE_HARDWARE             ;"Initilizing IDE Drive hardware"
        CALL    PRINT_STRING

        CALL    SET_DRIVE_A         ;Select the first Drive/CF card
        CALL    IDEinit             ;Initialize the board and drive 0. If there is no drive abort
        JZ      INIT1_OK

        MOV     BX,INIT_ERROR
        CALL    PRINT_STRING
        CALL    SHOWerrors
        RET

INIT1_OK:
        CALL    SET_DRIVE_B         ;Select the second Drive/CF card
        CALL    IDEinit             ;Initialize drive 1. If there is no drive abort
        JZ      INIT2_OK

        MOV     BX,DRIVE2_ERR       ;Warn second IDE drive did not initilize
```

```
        CALL    PRINT_STRING

INIT2_OK:
        CALL    SET_DRIVE_A             ;Back to first drive/CF Card

        CALL    DRIVE_ID                ;Get the drive 0 id info. If there is no drive just abort
        JZ      INIT3_OK

        MOV     BX,ID_ERROR
        CALL    PRINT_STRING
        CALL    SHOWerrors
        RET

INIT3_OK:                               ;Set default position will be first sector block
        MOV     BP,RAM_SEC
        MOV     word[BP],0H             ;Sec 0
        MOV     BP,RAM_TRK
        MOV     word[BP],0H             ;Track 0

        MOV     BP,RAM_DMA
        MOV     word[BP],IDE_Buffer     ;DMA initially to IDE_Buffer,
                                        ;Note DMA segment will always be SS: (in valid RAM)

        CALL    IDEinit                 ;For some reason this need to be here after getting the drive ID.
                                        ;otherewise sector #'s are off by one! (Probably ebacase on non-LBA reads)
        CALL    WR_LBA                  ;Update LBA on "1st" drive



;------------- MAIN IDE DRIVE DIAGNOSTIC MENU ---------------------------

IDE_LOOP:
        MOV     AX,CS                   ;Just in case somehow they changed somewhere below
        MOV     DS,AX
        MOV     ES,AX

        MOV     BX,IDE_SIGNON0          ;List IDE command options
        CALL    PRINT_STRING

        MOV     BP,CURRENT_IDE_DRIVE
        MOV     AL,[BP]
        OR      AL,AL
        JNZ     SIGN_B
        MOV     BX,CURRENT_MSG_A
        JMP     IDE_LOOP0
SIGN_B: MOV     BX,CURRENT_MSG_B
IDE_LOOP0:
        CALL    PRINT_STRING

        MOV     BX,IDE_SIGNON4          ;List IDE command options
        CALL    PRINT_STRING

        MOV     BP,DISPLAY_FLAG                 ;Do we have detail sector data display flag ON or OFF
        MOV     AL,[BP]                 ;NZ = on
        OR      AL,AL
        JNZ     IDE_LOOP1
        MOV     BX,IDE_SIGNON1          ;"ON"
        JMP     IDE_LOOP2
IDE_LOOP1:
        MOV     BX,IDE_SIGNON2          ;"OFF"
IDE_LOOP2:
        CALL    PRINT_STRING
        MOV     BX,IDE_SIGNON3          ;List IDE command options
        CALL    PRINT_STRING

        CALL    DISPLAY_POSITION        ;Display current Track,sector,head#

        CALL    CRLF
        MOV     BX,IDE_MENU             ;Enter a command
```

```
        CALL    PRINT_STRING

        call    CICO                    ;Get a command from Console
        mov     ah,0
        CMP     AL,ESC                  ;Abort if ESC
        JNZ     NOT_ESC
        JMP     INIT                    ;Back to start of Monitor

NOT_ESC:cmp     al,'A'                  ;Find meuu option from table
        jb      IDE_LOOP                ;must be A to Z
        cmp     al,'Z'
        jg      IDE_LOOP
        sub     al,'A'                  ;calculate offset
        shl     al,1                    ;X 2
        add     ax,IDE_TABLE            ;Note DS:=CS:
        mov     bx,ax
        CALL    CRLF
        mov     ax,[cs:bx]              ;get location of routine CS:[BX]
        call    ax                      ;<----------- This is the IDE Menu CMD call
        jmp     IDE_LOOP                ;finished



;               INDIVIDUAL IDE DRIVE MENU COMMANDS

;---------------Select Drive/CF card ----------------------------------------
SET_DRIVE_A:                            ;Select First Drive
        MOV     AL,0
SELECT_DRIVE:
        MOV     BP,CURRENT_IDE_DRIVE
        MOV     [BP],AL
        OUT     IDEDrivePort,AL                 ;Select Drive 0 or 1
        RET

SET_DRIVE_B:                            ;Select Drive 1
        MOV     AL,1
        JMP     SELECT_DRIVE


;---------------- Do the IDEntify drive command, and display the IDE_Buffer ------------
DRIVE_ID:
        CALL    IDEwaitnotbusy
        JNB     L_5
        XOR     AX,AX
        DEC     AX                      ;NZ if error
        RET                             ;If Busy return NZ

L_5:    MOV     DH,COMMANDid
        MOV     DL,REGcommand
        CALL    IDEwr8D                 ;issue the command

        CALL    IDEwaitdrq              ;Wait for Busy=0, DRQ=1
        JNB     L_6
        JMP     SHOWerrors

L_6:    MOV     CH,0                    ;256 words
        MOV     BP,IDE_Buffer           ;Store data here
        CALL    MoreRD16                ;Get 256 words of data from REGdata port to ss:[BP]


        MOV     BX,msgmdl               ;print the drive's model number
        CALL    PRINT_STRING
        MOV     BP,(IDE_Buffer + 54)
        MOV     CH,10                   ;character count in words
        CALL    Print_ID_Info           ;Print [HL], [B] X 2 characters
        CALL    CRLF
                                        ; print the drive's serial number
        MOV     BX,msgsn
        CALL    PRINT_STRING
```

```
        MOV     BP,(IDE_Buffer + 20)
        MOV     CH,5                    ;Character count in words
        CALL    Print_ID_Info
        CALL    CRLF
                                        ;PRINT_STRING the drive's firmware revision string
        MOV     BX,msgrev
        CALL    PRINT_STRING
        MOV     BP,(IDE_Buffer + 46)
        MOV     CH,2
        CALL    Print_ID_Info           ;Character count in words
        CALL    CRLF
                                        ;print the drive's cylinder, head, and sector specs
        MOV     BX,msgcy
        CALL    PRINT_STRING
        MOV     BP,(IDE_Buffer + 2)
        CALL    Print_ID_Info
        MOV     BX,msghd
        CALL    PRINT_STRING
        MOV     BP,(IDE_Buffer + 6)
        CALL    Print_ID_Info
        MOV     BX,msgsc
        CALL    PRINT_STRING
        MOV     BP,(IDE_Buffer + 12)
        CALL    Print_ID_Info
        CALL    CRLF
        XOR     AX,AX                   ;Ret Z
        RET


; Print a 16 bit number, located [BX] (Used only by the above DISK ID routine)
Print_ID_Info:
        MOV     AL,[BP+1]
        CALL    AL_HEXOUT
        MOV     AL,[BP]
        CALL    AL_HEXOUT
        RET



;-------------- Read the current selected sector (based on LBA) to the IDE Buffer
READ_SEC:
        MOV     AX,CS
        MOV     DS,AX
        MOV     BP,RAM_DMA
        MOV     word [BP],IDE_Buffer  ;DMA initially to IDE_Buffer

        CALL    READSECTOR

        JZ      Main1B
        CALL    CRLF                    ;Here if there was a problem
        RET

Main1B: MOV     BX,msgrd                ;Sector read OK
        CALL    PRINT_STRING

        MOV     BP,DISPLAY_FLAG                 ;Do we have detail sector data display flag on or off
        MOV     AL,[BP]                 ;NZ = on
        OR      AL,AL
        JNZ     SHOW_SEC_RDATA
        RET

SHOW_SEC_RDATA:
        MOV     BP,RAM_DMA
        MOV     word [BP],IDE_Buffer  ;DMA initially to IDE_Buffer
        CALL    DISPLAY_SEC
        MOV     BX,CR_To_Continue
        CALL    PRINT_STRING
        CALL    CI
        RET
```

```
;----------------- Write the current selected sector (based on LBA) from the IDE Buffer
WRITE_SEC:
        MOV     AX,CS
        MOV     DS,AX
        MOV     BX,CONFIRM_WR_MSG      ;Are you sure?
        CALL    PRINT_STRING
        CALL    CICO
        CMP     AL,'Y'
        JZ      WR_SEC_OK1
        CALL    CRLF                   ;Here if there was a problem
        RET

WR_SEC_OK1:
        MOV     BP,RAM_DMA
        MOV     word [BP],IDE_Buffer   ;DMA initially to IDE_Buffer

        CALL    WRITESECTOR            ;Will write whatever is in the IDE_Buffer

        JZ      Main2B
        CALL    CRLF                   ;Here if there was a problem
        RET

Main2B: MOV     BX,msgrd               ;Sector written OK
        CALL    PRINT_STRING

        MOV     BP,DISPLAY_FLAG               ;Do we have detail sector data display flag on or off
        MOV     AL,[BP]                ;NZ = on
        OR      AL,AL
        JNZ     SHOW_SEC_WDATA
        RET


SHOW_SEC_WDATA:
        MOV     BP,RAM_DMA
        MOV     word [BP],IDE_Buffer   ;DMA initially to IDE_Buffer
        CALL    DISPLAY_SEC
        MOV     BX,CR_To_Continue
        CALL    PRINT_STRING
        CALL    CI
        RET


;----------------------  Set a new LBA value from imputted Track/Sec info. Send to drive
SET_LBA:MOV     AX,CS
        MOV     DS,AX
        MOV     BX,SET_LBA_MSG         ;Set new LBA and send to drive
        CALL    PRINT_STRING
        CALL    GEN_HEX32_LBA          ;Get new CPM style Track & Sector number and put them in RAM at RAM_SEC & RAM_TRK
        JB      main3b                 ;Ret C set if abort/error
        CALL    WR_LBA                 ;Update LBA on drive
main3b: CALL    CRLF
        RET


;---------------------- Toggle detailed sector display on/off
DISPLAY:
        MOV     AX,CS
        MOV     DS,AX
        MOV     BP,DISPLAY_FLAG        ;Do we have detail sector data display flag on or off
        MOV     AL,[BP]                ;NZ = on
        NOT     AL
        MOV     [BP],AL
        RET


;---------------------- Point current sector to next sector
```

```
NEXT_SECT:
        CALL    GET_NEXT_SECT
        JNZ     AT_END
        RET
AT_END:
        MOV     BX,AT_END_MSG           ;Tell us we are at end of disk
        CALL    PRINT_STRING
        RET

;---------------------- Point current sector to previous sector
PREV_SECT:
        CALL    GET_PREV_SECT
        JNZ     AT_START
        RET
AT_START:
        MOV     BX,AT_START_MSG                 ;Tell us we are at start of disk
        CALL    PRINT_STRING
        RET


;---------------------- Sequentially read sectors from disk starting at current LBA position
SEQ_SEC_RD:
        MOV     AX,CS
        MOV     DS,AX
        CALL    IDEwaitnotbusy
        JNB     MORE_SEC
        JMP     SHOWerrors

MORE_SEC:
        CALL    CRLF
        MOV     BP,RAM_DMA              ;Set DMA initially to IDE_Buffer

        MOV     CL,'<'
        CALL    CO
        MOV     AX,BP
        CALL    AX_HEXOUT

        MOV     word [BP],IDE_Buffer
        MOV     CL,'.'
        CALL    CO
        MOV     AX,[BP]
        CALL    AX_HEXOUT
        MOV     CL,'>'
        CALL    CO

        CALL    READSECTOR              ;If there are errors they will show up in READSECTOR
        JZ      SEQOK

        MOV     BX,CONTINUE_MSG                 ;If an error ask if we wish to continue
        CALL    PRINT_STRING
        CALL    CICO
        CMP     AL,ESC                  ;Abort if ESC
        JNZ     SEQOK
        RET

SEQOK:  CALL    DISPLAY_POSITION        ;Display current Track,sector,head#

        MOV     BP,DISPLAY_FLAG                 ;Do we have detail sector data display flag on or off
        MOV     AL,[BP]                 ;NZ = on
        OR      AL,AL
        JZ      MORES2
        MOV     BP,RAM_DMA              ;Point DMA to IDE_Buffer again
        MOV     word [BP],IDE_Buffer
        CALL    DISPLAY_SEC

MORES2: CALL    CSTS                    ;Any keyboard character will stop display
        JZ      NO_WAIT
        CALL    CI
        MOV     BX,CONTINUE_MSG
```

```
        CALL    PRINT_STRING
        CALL    CI
        CMP     AL,ESC
        JNZ     NO_WAIT
        RET                             ;Bug, is returning to monitor, must be a stack problem!
NO_WAIT:CALL    GET_NEXT_SECT           ;Point LBA to next sector
        JZ      MORE_SEC                ;Note will go to last sec on disk unless stopped
        RET




;--------------- Read N Sectors to disk
;Note unlike the normal sector read, this routine increments the DMA address after each sector read

N_RD_SEC:
        MOV     AX,CS
        MOV     DS,AX
        MOV     BX,READN_MSG
        CALL    PRINT_STRING
        CALL    GET2DIGITS              ;Hex to AL

        MOV     BP,SECTOR_COUNT                 ;store sector count
        MOV     [BX],AL

        MOV     BP,RAM_DMA_STORE
        MOV     word [BP],IDE_Buffer    ;DMA_STORE initially to IDE_Buffer

NextRSec:
        MOV     BX, READN_MSG
        CALL    PRINT_STRING
        CALL    WR_LBA                  ;Update LBA on drive
        CALL    DISPLAY_POSITION        ;Display current Track,sector,head#

        MOV     BP,RAM_DMA_STORE
        MOV     AX,[BP]                 ;Get last value of DMA address
        MOV     BP,RAM_DMA
        MOV     [BP],AX                 ;Store it in DMA address


        CALL    READSECTOR              ;Actully, Sector/track values are already updated

        MOV     BP,RAM_DMA
        MOV     AX,[BP]                 ;Store it in DMA_STORE address
        MOV     BP,RAM_DMA_STORE
        MOV     [BP],AX

        MOV     BP,SECTOR_COUNT
        MOV     AL,[BP]
        DEC     AL
        MOV     [BP],AL
        JNZ     NEXT_SEC_NRD
        RET
NEXT_SEC_NRD:
        CALL    GET_NEXT_SECT
        JZ      NextRSec
        MOV     BX,AT_END_MSG           ;Tell us we are at end of disk
        CALL    PRINT_STRING
        RET




;------------------ Write N Sectors to disk
;Note unlike the normal sector write routine, this routine incriments the DMA address after each write.

N_WR_SEC:
        MOV     AX,CS
```

```
        MOV     DS,AX
        MOV     BX,CONFIRM_WR_MSG      ;Are you sure?
        CALL    PRINT_STRING
        CALL    CICO
        CMP     AL,'Y'
        JZ      WR_SEC_OK2
        CALL    CRLF                   ;Here if there was a problem
        RET

WR_SEC_OK2:
        MOV     BX,WRITEN_MSG
        CALL    PRINT_STRING
        CALL    GET2DIGITS             ;Hex to AL

        MOV     BP,SECTOR_COUNT                ;store sector count
        MOV     [BP],AL

        MOV     BP,RAM_DMA_STORE
        MOV     word [BP],IDE_Buffer   ;DMA_STORE initially to IDE_Buffer

NextWSec:
        MOV     BX, WRITEN_MSG
        CALL    PRINT_STRING
        CALL    WR_LBA                 ;Update LBA on drive
        CALL    DISPLAY_POSITION       ;Display current Track,sector,head#

        MOV     BP,RAM_DMA_STORE
        MOV     AX,[BP]                ;Get last value of DMA address
        MOV     BP,RAM_DMA
        MOV     [BP],AX                ;Store it in DMA address

        CALL    WRITESECTOR            ;Actully, Sector/track values are already updated

        MOV     BP,RAM_DMA
        MOV     AX,[BP]                ;Store it in DMA_STORE address
        MOV     BP,RAM_DMA_STORE
        MOV     [BP],AX

        MOV     BP,SECTOR_COUNT
        MOV     AL,[BP]
        DEC     AL
        MOV     [BP],AL
        JNZ     NEXT_SEC_NWR
        RET
NEXT_SEC_NWR:
        CALL    GET_NEXT_SECT
        JZ      NextWSec
        MOV     BX,AT_END_MSG          ;Tell us we are at end of disk
        CALL    PRINT_STRING
        RET


;-------------- Format current disk
FORMAT:
        MOV     AX,CS
        MOV     DS,AX
        MOV     BP,CURRENT_IDE_DRIVE
        MOV     AL,[BP]
        OR      AL,AL
        JNZ     FORM_B
        MOV     BX,FORMAT_MSG_A
        JMP     FORM_X
FORM_B: MOV     BX,FORMAT_MSG_B
FORM_X: CALL    PRINT_STRING
        MOV     BX, CONFIRM_WR_MSG     ;Are you sure?
        CALL    PRINT_STRING
        CALL    CICO
        CMP     AL,'Y'
        JZ      FORMAT_OK
```

```
        RET

FORMAT_OK:
        MOV     AX,0                    ;Back to CPM sector 0
        MOV     BP,RAM_SEC              ;Get Current Sector
        MOV     [BP],AX                 ;0 to CPM Sectors

        MOV     BP,RAM_TRK              ;And track
        MOV     [BP],AX

        MOV     AX,0E5E5H               ;First set Sector pattern to E5's
        CALL    RAM_FILL
        CALL    CRLF

NEXT_FORMAT:
        MOV     BP, RAM_DMA             ;Point DMA to the area
        MOV     word [BP],IDE_Buffer

        CALL    WRITESECTOR             ;Will return error if there was one
        JZ      NEXTF1                  ;Z means the sector write was OK

        MOV     BX,FORMAT_ERR           ;Indicate an error
        CALL    PRINT_STRING
        CALL    SHOW_TRACK_SEC          ;Show current location of error
        CALL    CRLF
        JMP     FNEXTSEC3

NEXTF1: MOV     BP,RAM_SEC              ;Get Current Sector
        MOV     AX,[BP]
        OR      AX,AX                   ;At start of each track give an update
        JNZ     FNEXTSEC2

        CALL    SHOW_TRACK

FNEXTSEC2:
        CALL    CSTS                    ;Any keyboard character will stop display
        JZ      FNEXTSEC1
        CALL    CI                      ;Flush character
FNEXTSEC3:
        MOV     BX,CONTINUE_MSG
        CALL    PRINT_STRING
        CALL    CICO
        CMP     AL,ESC
        JNZ     FNEXTSEC1
F_DONE: MOV     AL,0                    ;Login drive A:
        CALL    SELECT_DRIVE
        MOV     BP,CURRENT_IDE_DRIVE
        MOV     [BP],AL
        RET

FNEXTSEC1:
        CALL    GET_NEXT_SECT
        JZ      NEXT_FORMAT
        MOV     BX,AT_END_MSG           ;Tell us we are at end of disk
        CALL    PRINT_STRING
        JMP     F_DONE

;---------------- Copy Drive A: to Drive B:  -----------------------------
COPY_AB:
        MOV     AX,CS
        MOV     DS,AX
        MOV     BX,DiskCopyMsg
        CALL    PRINT_STRING
        CALL    CICO
        CMP     AL,'Y'
        JZ      COPY_AB1
        JMP     C_DONE

COPY_AB1:
```

```
        MOV     BP, RAM_SEC             ;Start with CPM sector 0
        MOV     AX,0
        MOV     [BP],AX
        MOV     BP,RAM_TRK              ;Start with CPM Track 0
        MOV     AX,0
        MOV     [BP],AX                 ;High & Low Track to 0
        CALL    CRLF
        CALL    CRLF

NextDCopy:
        MOV     AL,0                    ;Login drive A:
        CALL    SELECT_DRIVE

        CALL    WR_LBA                  ;Update LBA on "A:" drive

        MOV     BP,RAM_DMA
        MOV     word [BP],IDE_Buffer    ;DMA initially to IDE_Buffer

        CALL    READSECTOR              ;Get sector data from A: drive to buffer

        MOV     AL,1                    ;Login drive B:
        CALL    SELECT_DRIVE

        CALL    WR_LBA                  ;Update LBA on "B:" drive

        MOV     BP,RAM_DMA
        MOV     word [BP],IDE_Buffer    ;DMA initially to IDE_Buffer

        CALL    WRITESECTOR             ;Write buffer data to sector on B: drive
        JZ      COPY_OK1

        MOV     BX,COPY_ERR             ;Indicate an error
        CALL    PRINT_STRING
        CALL    SHOW_TRACK_SEC          ;Show current location of error
        CALL    CRLF
        JMP     COPY_OK3

COPY_OK1:
        MOV     BP,RAM_SEC              ;Get Current Sector
        MOV     AX,[BP]
        OR      AX,AX                   ;At start of each track give an update
        JNZ     COPY_OK2

        CALL    SHOW_TRACK

COPY_OK2:
        CALL    CSTS                    ;Any keyboard character will stop display
        JZ      C_NEXTSEC1
        CALL    CI                      ;Flush character
COPY_OK3:
        MOV     BX,CONTINUE_MSG
        CALL    PRINT_STRING
        CALL    CICO
        CMP     AL,ESC
        JNZ     C_NEXTSEC1
C_DONE: MOV     AL,0                    ;Login drive A:
        CALL    SELECT_DRIVE
        MOV     BP,CURRENT_IDE_DRIVE
        MOV     [BP],AL
        RET

C_NEXTSEC1:
        CALL    GET_NEXT_SECT           ;Update to next sector/track
        JNZ     C_NEXTSEC2
        JMP     NextDCopy

C_NEXTSEC2:
        MOV     BX,CopyDone             ;Tell us we are all done.
        CALL    PRINT_STRING
```

```
        JMP     C_DONE


;-------------- Verify Drive A: = B: -------------------------------
VERIFY_AB:
        MOV     AX,CS
        MOV     DS,AX
        MOV     BX,DiskVerifyMsg
        CALL    PRINT_STRING

        MOV     BP,RAM_SEC              ;Start with CPM sector 0
        MOV     AX,0
        MOV     [BP],AX
        MOV     BP,RAM_TRK              ;Start with CPM Track 0
        MOV     AX,0
        MOV     [BP],AX                 ;High & Low Track to 0

        CALL    CRLF
        CALL    CRLF

NextVCopy:
        MOV     AL,0                    ;Login drive A:
        CALL    SELECT_DRIVE

        CALL    WR_LBA                  ;Update LBA on "A:" drive

        MOV     BP,RAM_DMA
        MOV     word [BP],IDE_Buffer    ;DMA initially to IDE_Buffer

        CALL    READSECTOR              ;Get sector data from A: drive to buffer

        MOV     AL,1                    ;Login drive B:
        CALL    SELECT_DRIVE

        CALL    WR_LBA                  ;Update LBA on "B:" drive

        MOV     BP,RAM_DMA
        MOV     word [BP],IDE_Buffer2   ;DMA initially to IDE_Buffer2

        CALL    READSECTOR

        MOV     DI,IDE_Buffer2
        MOV     SI,IDE_Buffer
        MOV     CX,512                  ;Length of sector in words

NEXT_CMP:
        MOV     AL, [SS:DI]             ;Note we have to use SS:
        CMP     AL, [SS:SI]
        JNZ     VER_ERROR
        INC     DI
        INC     SI
        LOOP    NEXT_CMP                ;CX will contain count of words done so far, (0 if done OK)
        JMP     VERIFY_OK

VER_ERROR:
        MOV     BX,VERIFY_ERR           ;Indicate an error
        CALL    PRINT_STRING
        CALL    SHOW_TRACK_SEC          ;Show current location of error
        MOV     BX,DRIVE1_MSG           ;'  Drive A',CR,LF
        CALL    PRINT_STRING

        MOV     SI,IDE_Buffer
        MOV     CX,512                  ;Length of sector in words
VER_SOURCE:
        MOV     AL, [SS:SI]             ;Note we have to use SS:
        CALL    AL_HEXOUT
        INC     SI
        LOOP    VER_SOURCE
        CALL    CRLF
```

```
        CALL    SHOW_TRACK_SEC          ;Show current location of error
        MOV     BX,DRIVE2_MSG           ;'  Drive B',CR,LF
        CALL    PRINT_STRING

        MOV     SI,IDE_Buffer2
        MOV     CX,512                  ;Length of sector in words
VER_DEST:
        MOV     AL, [SS:DI]             ;Note we have to use SS:
        CALL    AL_HEXOUT
        INC     DI
        LOOP    VER_DEST
        CALL    CRLF
        JMP     VERIFYT                 ;Do not ask for a continue message here. Just continue
                                        ;If you want it change to VERIFYT1
VERIFY_OK:
        MOV     BP,RAM_SEC              ;Get Current Sector
        MOV     AX,[BP]
        OR      AX,AX                   ;At start of each track give an update
        JNZ     VERIFYT

        CALL    SHOW_TRACK

VERIFYT:CALL    CSTS                    ;Any keyboard character will stop display
        JZ      V_NEXTSEC1
        CALL    CI                      ;Flush character
VERIFYT1:
        MOV     BX,CONTINUE_MSG
        CALL    PRINT_STRING
        CALL    CICO
        CMP     AL,ESC
        JNZ     V_NEXTSEC1
        JMP     V_NEXTSEC3
V_NEXTSEC1:
        CALL    GET_NEXT_SECT           ;Update to next sector/track
        JNZ     V_NEXTSEC2
        JMP     NextVCopy
V_NEXTSEC2:
        MOV     BX,VerifyDone           ;Tell us we are all done.
        CALL    PRINT_STRING
V_NEXTSEC3:
        MOV     AL,0                    ;Login drive A:
        CALL    SELECT_DRIVE
        MOV     BP,CURRENT_IDE_DRIVE
        MOV     [BP],AL
        RET




;------------------ Fill RAM buffer with 0's

RAMCLEAR:
        MOV     AX,CS
        MOV     DS,AX
        MOV     AX,0
RAM_FILL:
        MOV     BP,IDE_Buffer
        MOV     CX,256                  ;512 bytes total
CLEAR1: MOV     [BP],AX                 ;Note this will be SS:BP
        INC     BP
        INC     BP
        LOOP    CLEAR1

        MOV      BX,FILL_MSG
        CALL    PRINT_STRING
        RET


;----------------  Power up a Hard Disk
```

```
SPINUP: MOV     DH,COMMANDspinup
spup2:  MOV     DL,REGcommand
        CALL    IDEwr8D
        CALL    IDEwaitnotbusy
        JNB     L_7
        JMP     SHOWerrors
L_7:    OR      AL,AL                   ;Clear carry
        RET


;-------------------- Tell the Hard disk to power down

SPINDOWN:
        CALL    IDEwaitnotbusy
        JNB     L_8
        JMP     SHOWerrors
L_8:    MOV     DH,COMMANDspindown
        JMP     spup2


;-------------------- Back to parent 8086 Monitor commands
QUIT_IDE:
        JMP     INIT




;==================== Support Routines FOR IDE MODULE ==================================

;Generate an LBA sector number with data input from CPM style Track# & Sector#

GEN_HEX32_LBA:
        MOV     BX,ENTERRAM_SECL        ;Enter sector number, low
        CALL    PRINT_STRING
        CALL    GET2DIGITS              ;Get 8 bit value (2 digits) to AL. (BX, CX & DX Unchanged)
        MOV     BP,RAM_SEC
        MOV     [BP],AL                 ;Note: no check that data is < MAXSEC
        CALL    CRLF

        MOV     BX,ENTERRAM_TRKL        ;Enter low byte track number
        CALL    PRINT_STRING
        CALL    GET2DIGITS              ;Get 8 bit value (2 digits) to AL. (BX, CX & DX Unchanged)
        MOV     BP,RAM_TRK
        MOV     [BP],AL
        CALL    CRLF

        MOV     BX,ENTERRAM_TRKH        ;Enter high byte track number
        CALL    PRINT_STRING
        CALL    GET2DIGITS              ;Get 8 bit value (2 digits) to AL. (BX, CX & DX Unchanged)
        MOV     BP,RAM_TRK+1
        MOV     [BP],AL
        XOR     AL,AL
        OR      AL,AL                   ;To return NC
        RET


DISPLAY_POSITION:                       ;Display current track,sector & head position
        MOV     BX,msgCPMTRK            ;Display in LBA format
        CALL    PRINT_STRING            ;---- CPM FORMAT ----
        MOV     BP,RAM_TRK+1
        MOV     AL,[BP]                 ;High TRK byte
        CALL    AL_HEXOUT
        DEC     BP
        MOV     AL,[BP]                 ;Low TRK byte
        CALL    AL_HEXOUT

        MOV     BX,msgCPMSEC
        CALL    PRINT_STRING            ;SEC = (16 bits)
```

```
        MOV     BP,RAM_SEC+1            ;High Sec
        MOV     AL,[BP]
        CALL    AL_HEXOUT
        DEC     BP
        MOV     AL,[BP]                 ;Low sec
        CALL    AL_HEXOUT

                                        ;---- LBA FORMAT ----
        MOV     BX,msgLBA
        CALL    PRINT_STRING            ;(LBA = 00 (<-- Old "Heads" = 0 for these drives).

        MOV     BP,RAM_DRIVE_TRK+1      ;High "cylinder" byte
        MOV     AL,[BP]
        CALL    AL_HEXOUT
        DEC     BP
        MOV     AL,[BP]                 ;Low "cylinder" byte
        CALL    AL_HEXOUT

        MOV     BP,RAM_DRIVE_SEC
        MOV     AL,[BP]
        CALL    AL_HEXOUT
        MOV     BX,MSGBracket           ;)$
        CALL    PRINT_STRING
        RET

SHOW_TRACK_SEC:                                 ;Display current (CPM) track,sector
        MOV     BX,msgCPMTRK
        CALL    PRINT_STRING            ;---- CPM FORMAT ----
        MOV     BP,RAM_TRK+1
        MOV     AL,[BP]                 ;High TRK byte
        CALL    AL_HEXOUT
        DEC     BP
        MOV     AL,[BP]                 ;Low TRK byte
        CALL    AL_HEXOUT
        MOV     BX,msgCPMSEC
        CALL    PRINT_STRING

        MOV     BP,RAM_SEC              ;Low Sec (Only)
        MOV     AX,[BP]
        CALL    AL_HEXOUT
        MOV     BX,H_Msg
        CALL    PRINT_STRING
        RET

SHOW_TRACK:
        MOV     BX,msgCPMTRK
        CALL    PRINT_STRING            ;---- CPM FORMAT ----
        MOV     BP,RAM_TRK+1
        MOV     AL,[BP]                 ;High TRK byte
        CALL    AL_HEXOUT
        MOV     BP,RAM_TRK
        MOV     AL,[BP]                 ;Low TRK byte
        CALL    AL_HEXOUT
        MOV     BX,OK_CR_MSG
        CALL    PRINT_STRING            ;---- CPM FORMAT ----
        RET


DISPLAY_SEC:                            ;Print a DISPLAY_SEC of the data in the 512 byte IDE_Buffer (RAM_DMA)
        CALL    CRLF                    ;Note written so it can be easily converted to a "normal: DS: based" routine
        MOV     BP,RAM_DMA              ;Get Current DMA Address
        MOV     SI,[BP]                 ;Both DS:DI & SI point to buffer
        MOV     DI,SI
        MOV     DH,32                   ;print 32 lines

SF172:  CALL    CRLF
        call    SHOW_ADDRESS_SS                 ;Show SS:SI
        mov     cx,2                    ;send 2 spaces
        call    TABS
```

```
        MOV     DL,16                   ;32 characters across
SF175:  MOV     AL,[SS:SI]
        CALL    AL_HEXOUT               ;Display A on CRT/LCD
        MOV     AL,'~'
        CALL    CO
        INC     SI
        DEC     DL
        JNZ     SF175

        mov     cx,3                    ;first send 3 spaces
        call    TABS

        MOV     DL,16                   ;24 across again
Sloop2: mov     al,[SS:DI]
        and     al,7fh
        cmp     al,' '                  ;filter out control characters
        jnc     Sloop3
Sloop4: mov     al,'.'
Sloop3: cmp     al,'~'
        jnc     Sloop4
        mov     cl,al
        call    CO
        INC     DI
        DEC     DL
        JNZ     Sloop2
        DEC     DH
        JNZ     SF172                   ;--DH has total byte count
        CALL    CRLF
        ret



;Point to next sector.  Ret Z if all OK      NZ if at end of disk
GET_NEXT_SECT:
        MOV     BP,RAM_SEC              ;Get Current Sector
        MOV     AX,[BP]
        INC     AX
        MOV     [BP],AX                 ;0 to MAXSEC CPM Sectors
        CMP     AX,MAXSEC-1             ;Assumes < 255 sec /track
        JNZ     NEXT_SEC_DONE

        MOV     AX,0                    ;Back to CPM sector 0
        MOV     [BP],AX

        MOV     BP,RAM_TRK              ;Bump to next track
        MOV     AX,[BP]
        INC     AX
        CMP     AX,100H                 ;Tracks 0-0FFH only
        JZ      AT_DISK_END
        MOV     [BP],AX
NEXT_SEC_DONE:
        CALL    WR_LBA                  ;Update the LBC pointer
        XOR     AX,AX
        RET                             ;Ret z if all OK
AT_DISK_END:
        XOR     AX,AX
        DEC     AX
        RET



;Point to previous sector.  Ret Z if all OK
GET_PREV_SECT:
        MOV     BP,RAM_SEC              ;Get Current Sector
        MOV     AX,[BP]
        CMP     AX,0
        JZ      PREVIOUS_TRACK
        DEC     AX
        MOV     [BP],AX                 ;0 to MAXSEC CPM Sectors
```

```
        JMP     PREVIOUS_SEC_DONE

PREVIOUS_TRACK:
        MOV     AX,MAXSEC-1             ;Back to CPM last sector on previous track
        MOV     [BP],AX

        MOV     BP,RAM_TRK              ;Bump to next track
        MOV     AX,[BP]
        CMP     AX,0                    ;If On track 0 already then problem
        JNZ     AT_00
        DEC     AX
        MOV     [BP],AX
PREVIOUS_SEC_DONE:
        CALL    WR_LBA                  ;Update the LBC pointer
        XOR     AX,AX                   ;Return Z if all OK
        RET
AT_00:  MOV     BX,ATHOME_MSG
        CALL    PRINT_STRING
        XOR     AX,AX
        DEC     ax                      ;NZ if problem
        RET

;
SHOWerrors:
        CALL    CRLF
        MOV     DL,REGstatus            ;Get status in status register
        CALL    IDErd8D
        MOV     AL,DH
        AND     AL,1H
        JNZ     MoreError               ;Go to  REGerr register for more info
                                        ;All OK if 01000000

        PUSHF                           ;<<< Save for return below
        AND     AL,80H
        JZ      NOT7
        MOV     BX,DRIVE_BUSY           ;Drive Busy (bit 7) stuck high.   Status =
        CALL    PRINT_STRING
        JMP     DONEERR

NOT7:   AND     AL,40H
        JNZ     NOT6
        MOV     BX,DRIVE_NOT_READY      ;Drive Not Ready (bit 6) stuck low.  Status =
        CALL    PRINT_STRING
        JMP     DONEERR

NOT6:   AND     AL,20H
        JNZ     NOT5
        MOV     BX,DRIVE_WR_FAULT       ;Drive write fault.    Status =
        CALL    PRINT_STRING
        JMP     DONEERR

NOT5:   MOV     BX,UNKNOWN_ERROR
        CALL    PRINT_STRING
        JMP     DONEERR

MoreError:                              ;Get here if bit 0 of the status register indicted a problem
        MOV     DL,REGerr               ;Get error code in REGerr
        CALL    IDErd8D
        MOV     AL,DH
        PUSHF                           ;<<<< Save flags for below

        AND     AL,10H
        JZ      NOTE4
        MOV     BX,SEC_NOT_FOUND
        CALL    PRINT_STRING
        JMP     DONEERR

NOTE4:  AND     AL,80H
        JZ      NOTE7
```

```
        MOV     BX,BAD_BLOCK
        CALL    PRINT_STRING
        JMP     DONEERR

NOTE7:  AND     AL,40H
        JZ      NOTE6
        MOV     BX,UNRECOVER_ERR
        CALL    PRINT_STRING
        JMP     DONEERR

NOTE6:  AND     AL,4H
        JZ      NOTE2
        MOV     BX,INVALID_CMD
        CALL    PRINT_STRING
        JMP     DONEERR

NOTE2:  AND     AL,2H
        JZ      NOTE1
        MOV     BX,TRK0_ERR
        CALL    PRINT_STRING
        JMP     DONEERR

NOTE1:  MOV     BX,UNKNOWN_ERROR1
        CALL    PRINT_STRING
        JMP     DONEERR

DONEERR:POPF                            ;>>>>> get back flags
        PUSH    AX
        CALL    AL_BINOUT               ;Show error bit pattern
        CALL    CRLF
        POP     AX
        XCHG    AL,AH
        OR      AL,AL                   ;Set Z flag
        STC                             ;Set Carry flag
        RET


;=============================================================================================
; IDE Drive BIOS Routines written in a format that can be used with CPM86 (Note MSDOS/DOS has its own
; modules see further below. However instead of using DS:[BX] (as we do in the CPM86 BIOS), throughout we
; will use SS:[BP] so the the buffers can reside at the top segment of available RAM.
; Normally this will be F000:7000H  (1K below the ROM) but the monitor will not assume that there is a
; full 1MG address space avalable and may put them lower.  See monitor initilization code at start.
;=============================================================================================

IDEinit:                                ;Initilze the 8255 and drive then do a hard reset on the drive,
                                        ;By default the drive will come up initilized in LBA mode.
        MOV     AL,READcfg8255          ;10010010b
        OUT     IDECtrlPort,AL          ;Config 8255 chip, READ mode

        MOV     AL,IDErstline
        OUT     IDEportC,AL             ;Hard reset the disk drive

        MOV     CH,IDE_Reset_Delay      ;;Time delay for reset/initilization (~66 uS, with 8MHz 8086, 1 I/O wait state)
ResetDelay:
        DEC     CH
        JNZ     ResetDelay              ;Delay (IDE reset pulse width)
        XOR     AL,AL
        OUT     IDEportC,AL             ;No IDE control lines asserted

        CALL    DELAY_32                ;Allow time for CF/Drive to recover

        MOV     DH,11100000b            ;Data for IDE SDH reg (512bytes, LBA mode,single drive,head 0000)
;       MOV     DH,10100000b            ;For Trk,Sec,head (non LBA) use 10100000 (This is the mode we use for MSDOS)
                                        ;Note. Cannot get LBA mode to work with an old Seagate Medalist 6531 drive.
                                        ;have to use the non-LBA mode. (Common for old hard disks).
        MOV     DL,REGshd               ;00001110,(0EH) for CS0,A2,A1,
        CALL    IDEwr8D                 ;Write byte to select the MASTER device
```

```
        MOV     CH,0FFH             ;<<< May need to adjust delay time
WaitInit:
        MOV     DL,REGstatus        ;Get status after initilization
        CALL    IDErd8D             ;Check Status (info in [DH])
        MOV     AL,DH
        AND     AL,80H
        JZ      DoneInit            ;Return if ready bit is zero

        PUSH    CX
        MOV     CX,0FFFFH
DELAY2: MOV     DH,2                ;May need to adjust delay time to allow cold drive to
DELAY1: DEC     DH                  ;to speed
        JNZ     DELAY1
        DEC     CX
        MOV     AL,CL
        OR      AL,CH
        JNZ     DELAY2
        POP     CX

        DEC     CH
        JNZ     WaitInit
        CALL    SHOWerrors          ;Ret with NZ flag set if error (probably no drive)
        RET
DoneInit:
        XOR     AL,AL
        RET

DELAY_32:
        MOV     AL,40               ;DELAY ~32 MS (DOES NOT SEEM TO BE CRITICAL)
DELAY3: MOV     BL,0
M0:     DEC     BL
        JNZ     M0
        DEC     AL
        JNZ     DELAY3
        RET
                                    ;Read a sector, specified by the 4 bytes in LBA
                                    ;Z on success, NZ call error routine if problem
READSECTOR:
        CALL    WR_LBA              ;Tell which sector we want to read from.
                                    ;Note: Translate first in case of an error otherewise we
                                    ;will get stuck on bad sector
        CALL    IDEwaitnotbusy      ;make sure drive is ready
        JNB     L_19
        JMP     SHOWerrors          ;Returned with NZ set if error

L_19:   MOV     DH,COMMANDread
        MOV     DL,REGcommand
        CALL    IDEwr8D             ;Send sec read command to drive.
        CALL    IDEwaitdrq          ;wait until it's got the data
        JNB     L_20
        JMP     SHOWerrors

L_20:   MOV     BP,RAM_DMA          ;Get Current DMA Address at SS:RAM_DMA
        MOV     AX,[BP]             ;Note SS: is assumed here
        MOV     BP,AX
        MOV     CH,0                ;Read 512 bytes to [HL] (256X2 bytes)

MoreRD16:
        MOV     AL,REGdata          ;REG regsiter address
        OUT     IDEportC,AL

        OR      AL,IDErdline        ;08H+40H, Pulse RD line
        OUT     IDEportC,AL

        IN      AL,IDEportA         ;Read the lower byte first
        MOV     [BP],AL
        INC     BP
        IN      AL,IDEportB         ;THEN read the upper byte
        MOV     [BP],AL
```

```
        INC     BP

        MOV     AL,REGdata              ;Deassert RD line
        OUT     IDEportC,AL
        DEC     CH
        JNZ     MoreRD16

        MOV     DL,REGstatus
        CALL    IDErd8D
        MOV     AL,DH
        AND     AL,1H
        JZ      L_21
        CALL    SHOWerrors              ;If error display status
L_21:   RET


                                ;Write a sector, specified by the 3 bytes in LBA (_ IX+0)",
                                ;Z on success, NZ to error routine if problem
WRITESECTOR:
        CALL    WR_LBA                  ;Tell which sector we want to read from.
                                ;Note: Translate first in case of an error otherewise we
                                ;will get stuck on bad sector
        CALL    IDEwaitnotbusy          ;make sure drive is ready
        JNB     L_22
        JMP     SHOWerrors

L_22:   MOV     DH,COMMANDwrite
        MOV     DL,REGcommand
        CALL    IDEwr8D                 ;tell drive to write a sector
        CALL    IDEwaitdrq              ;wait unit it wants the data
        JNB     L_23
        JMP     SHOWerrors

L_23:   MOV     BP, RAM_DMA             ;Get Current DMA Address
        MOV     AX,[BP]
        MOV     BP,AX
        MOV     CH,0                    ;256X2 bytes

        MOV     AL,WRITEcfg8255
        OUT     IDECtrlPort,AL

WRSEC1_IDE:
        MOV     AL,[BP]
        INC     BP
        OUT     IDEportA,AL             ;Write the lower byte first
        MOV     AL,[BP]
        INC     BP
        OUT     IDEportB,AL             ;THEN High byte on B

        MOV     AL,REGdata
        PUSH    AX
        OUT     IDEportC,AL             ;Send write command
        OR      AL,IDEwrline            ;Send WR pulse
        OUT     IDEportC,AL
        POP     AX
        OUT     IDEportC,AL             ;Send write command
        DEC     CH
        JNZ     WRSEC1_IDE

        MOV     AL,READcfg8255          ;Set 8255 back to read mode
        OUT     IDECtrlPort,AL

        MOV     DL,REGstatus
        CALL    IDErd8D
        MOV     AL,DH
        AND     AL,1H
        JZ      L_24
        CALL    SHOWerrors              ;If error display status
L_24:   RET
```

```
WR_LBA:                         ;Write the logical block address to the drive's registers
                                ;Note we do not need to set the upper nibble of the LBA
                                ;It will always be 0 for these small CPM drives (so no High Cylinder
                                ;numbers etc).
        MOV     BP,RAM_SEC
        MOV     AX,[BP]         ;LBA mode, Low sectors go directly
        INC     AX              ;Sectors are numbered 1 -- MAXSEC (even in LBA mode)
        MOV     BP,RAM_DRIVE_SEC
        MOV     [BP],AL         ;For Diagnostic Diaplay Only
        MOV     DH,AL
        MOV     DL,REGsector    ;Send info to drive
        CALL    IDEwr8D         ;Write to 8255 A Register
                                ;Note: For drive we will have 0 - MAXSEC sectors only

        MOV     BP,RAM_TRK
        MOV     AX,[BP]
        MOV     BP,RAM_DRIVE_TRK
        MOV     [BP],AL
        MOV     DH,AL           ;Send Low TRK#
        MOV     DL,REGcylinderLSB
        CALL    IDEwr8D         ;Write to 8255 A Register

        MOV     BP,RAM_DRIVE_TRK+1
        MOV     [BP],AH
        MOV     DH,AH           ;Send High TRK#
        MOV     DL,REGcylinderMSB
        CALL    IDEwr8D         ;Send High TRK# (in DH) to IDE Drive
        CALL    IDEwr8D_X       ;Special write to 8255 B Register (Not A) to update LED HEX Display
                                ;High 8 bits ignored by IDE drive

        MOV     DH,1            ;For CPM, one sector at a time
        MOV     DL,REGseccnt
        CALL    IDEwr8D         ;Write to 8255 A Register
        RET

                                        ;Special version for MS-DOS system BIOS (see IBM BIOS Section)
DOS_WR_LBA:                             ;This will display Head, Cylinder and Sector on the LED HEX display
                                        ;instead of LBA sector numbers.
        MOV     DH,[CURRENT_HEAD]       ;OR in head info to lower 4 bits
        AND     DH,0FH                  ;Just in case
        OR      DH,10100000B            ;Set to >>>>> NON-LBA mode <<<<<
        MOV     DL,REGshd               ;Send "Head #" (in DH) to IDE drive
        CALL    IDEwr8D

        MOV     DH,[CURRENT_TRACK_HIGH]         ;Send High TRK#
        MOV     DL,REGcylinderMSB
        CALL    IDEwr8D                 ;Send High TRK# (in DH) to IDE Drive

        MOV     DH,[CURRENT_HEAD]       ;Get head info to lower 8 bits of the special
        AND     DH,0FH                  ;top two LED HEX displays.
        SHL     DH,1                    ;These 8 (high) data lines are ignored by the IDE drive
        SHL     DH,1
        SHL     DH,1
        SHL     DH,1
        OR      DH,[CURRENT_TRACK_HIGH]         ;Will display  the Head in top nibble and the two bits of the HIGH
bits
        MOV     DL,REGcylinderMSB       ;of the high cylinder in the low nibble.
        CALL    IDEwr8D_X               ;Special output to 8255 B Register (Not A) to update LED HEX Display ONLY

        MOV     DH,[CURRENT_TRACK]      ;Get low Track #
        MOV     DL,REGcylinderLSB       ;Send Low TRK# (in DH)
        CALL    IDEwr8D                 ;Special write to 8255 B Register (Not A)

        MOV     DH,[CURRENT_SECTOR]     ;Bits 0-5 only  (currently 1-17)
        MOV     DL,REGsector            ;Send "Sector#"
        CALL    IDEwr8D                 ;Write to 8255 A Register
```

```
        MOV     DH,[SECTORS_TO_DO]          ;# of CONTIGOUS sectors to send
        MOV     DL,REGseccnt
        CALL    IDEwr8D                     ;Write to 8255 A Register
        RET



IDEwaitnotbusy:                             ;Drive READY if 01000000
        MOV     CH,0FFH
        MOV     AH,0FFH             ;Delay, must be above 80H for 4MHz Z80. Leave longer for slower drives
        PUSH    BX                  ;AH is not changed in IDErd8D below
MoreWait:
        MOV     DL,REGstatus        ;wait for RDY bit to be set
        CALL    IDErd8D             ;Note AH or CH are unchanged
        MOV     AL,DH
        AND     AL,11000000B
        XOR     AL,01000000B
        JZ      DONE_NOT_BUSY
        DEC     CH
        JNZ     MoreWait
        DEC     AH
        JNZ     MoreWait
        MOV     CL,'$'
        CALL    CO
        CALL    CI
        JMP     yyy
        STC                         ;Set carry to indicate an error
        POP     BX
        RET
DONE_NOT_BUSY:
        OR      AL,AL               ;Clear carry it indicate no error
        POP     BX
        RET

yyy:    POP     BX
        JMP     IDEwaitnotbusy

                                    ;Wait for the drive to be ready to transfer data.
IDEwaitdrq:                         ;Returns the drive's status in Acc
        MOV     CH,0FFH
        MOV     AL,0FFH             ;Delay, must be above 80H for 4MHz Z80. Leave longer for slower drives
        PUSH    BX
MoreDRQ:
        MOV     DL,REGstatus        ;wait for DRQ bit to be set
        CALL    IDErd8D             ;Note AH or CH are unchanged
        MOV     AL,DH
        AND     AL,10001000B
        CMP     AL,00001000B
        JZ      DoneDRQ
        DEC     CH
        JNZ     MoreDRQ
        DEC     AH
        JNZ     MoreDRQ
        MOV     CL,'#'
        CALL    CO
        CALL    CI
        JMP     xxx
        STC                         ;Set carry to indicate error
        POP     BX
        RET
DoneDRQ:
        OR      AL,AL               ;Clear carry
        POP     BX
        RET

xxx:    POP     BX
        JMP     IDEwaitdrq
```

```
;-----------------------------------------------------------------
; Low Level 8 bit R/W to the drive controller.  These are the routines that talk
; directly to the drive controller registers, via the 8255 chip.
; Note the 16 bit Sector I/O to the drive is done directly
; in the routines READSECTOR & WRITESECTOR for speed reasons.

IDErd8D:                            ;READ 8 bits from IDE register @ [DL], return info in [DH]
        MOV     AL,DL               ;select IDE register
        OUT     IDEportC,AL         ;drive address onto control lines

        OR      AL,IDErdline        ;RD pulse pin (40H)
        OUT     IDEportC,AL         ;Assert read pin

        IN      AL,IDEportA
        MOV     DH,AL               ;return with data in [DH]

        MOV     AL,DL               ;<---Ken Robbins suggestion
        OUT     IDEportC,AL         ;Drive address onto control lines

        XOR     AL,AL
        OUT     IDEportC,AL         ;Zero all port C lines
        RET


IDEwr8D:                            ;WRITE Data in [DH] to IDE register @ [DL]
        MOV     AL,WRITEcfg8255             ;Set 8255 to write mode
        OUT     IDECtrlPort,AL

        MOV     AL,DH               ;Get data put it in 8255 A port
        OUT     IDEportA,AL

        MOV     AL,DL               ;select IDE register
        OUT     IDEportC,AL

        OR      AL,IDEwrline        ;lower WR line
        OUT     IDEportC,AL

        MOV     AL,DL               ;<-- Ken Robbins suggestion, raise WR line
        OUT     IDEportC,AL         ;deassert RD pin

        XOR     AL,AL               ;Deselect all lines including WR line
        OUT     IDEportC,AL

        MOV     AL,READcfg8255      ;Config 8255 chip, read mode on return
        OUT     IDECtrlPort,AL
        RET


IDEwr8D_X:                          ;WRITE Data in [DH] to IDE register @ [DL]
        MOV     AL,WRITEcfg8255             ;Set 8255 to write mode
        OUT     IDECtrlPort,AL

        MOV     AL,DH               ;Get data and put it in 8255 >>>> Port B <<<<
        OUT     IDEportB,AL

        MOV     AL,DL               ;select IDE register
        OUT     IDEportC,AL

        OR      AL,IDEwrline        ;lower WR line
        OUT     IDEportC,AL

        MOV     AL,DL               ;<-- Ken Robbins suggestion, raise WR line
        OUT     IDEportC,AL         ;Deassert RD pin

        XOR     AL,AL               ;Deselect all lines including WR line
        OUT     IDEportC,AL

        MOV     AL,READcfg8255      ;Config 8255 chip, read mode on return
        OUT     IDECtrlPort,AL
```

```
        RET



;*****************************************************************************************************
;
;       "BIOS" section to allow MS-DOS 2.1 to run on non-IBM hardware.
;       8086 assembly language for the CP/M-86 assembler. This is a highly
;       modified version of a BIOS first written by LogiCom Inc back in 1985.
;
;       It enables a standard IBM-PC PC-DOS V2.1 to run with S100Computers/N8VEM Boards.
;
;*****************************************************************************************************
;
;      The normal interrupts for the IBM, and their entry points
;      in this code are as follows:
;
;  Int   Name               BIOS entry
;    0   Divide by zero     DUMMY_RETURN
;    1   Single Step        DUMMY_RETURN
;    2   Non-maskable       NMIINT
;    3   Breakpoint         DUMMY_RETURN
;    4   Overflow           DUMMY_RETURN
;    5   Print Screen       DUMMY_RETURN
;    6   Reserved           DUMMY_RETURN
;    7   Reserved           DUMMY_RETURN
;    8   Timer Tic          TIMER        \
;    9   Keypressed         KEYHND        \
;    A   Reserved           DUMMY_RETURN   \
;    B   Comm Hardware      DUMMY_RETURN    \ Normal location for
;    C   Comm Hardware      DUMMY_RETURN    / IBM hardware interrupts
;    D   Disk Hardware      DUMMY_RETURN   /
;    E   Diskette Hardware  DUMMY_RETURN  /
;    F   Printer Hardware   DUMMY_RETURN /

;   10   Video Output       CONOUT        (10 through 1F are
;   11   Equipment check    EQUIP           software interrupts)
;   12   Memory Size        MEMSIZ
;   13   Disk I/O           DISKIO        <------ ALL DISK IO (Floppy & HDISK)
;   14   Comm I/O           COMMIO
;   15   Cassette I/O       DUMMY_RETURN
;   16   Keyboard I/O       CONIN
;   17   Printer I/O        LSTOUT
;   18   Basic              DUMMY_RETURN
;   19   Bootstrap          BOOT_DOS_INT
;   1A   Time of Day        TIME_OF_DAY
;   1B   Keyboard Break     DUMMY_RETURN
;   1C   User timer tic     DUMMY_RETURN
;   1D   Video Init.        VIDEO_PARM
;   1E   Diskette Parms     DISK_BASE (Pointer only)
;   1F   Graphics Char      0
;
;   40   Copy of Disk/IO    DISKIO (for systems with a HDISK)


IBM_BIOS:
        cli                             ;No interrupts yet please
        MOV     BX,IBM_SIGNON_MSG       ;Announce we are here
        CALL    PRINT_STRING            ;Note PRINT_STRING always uses the CS: override for the BX pointer

        push     DS
        XOR     AX,AX                   ;Set DS to data area for ROM usage in low RAM @ 400H....)
        MOV     DS,AX
        mov     byte [DEBUG_FLAG],0     ;Debug mode normally off
        POP     DS

        CALL    SETUP_IBM_BIOS          ;Initilize RAM and hardware
```

```
 IBM_LOOP:
       CALL    CRLF
       MOV     BX,IBM_MENU1          ;Enter start of menu
       CALL    PRINT_STRING

        XOR    AX,AX                 ;Set DS to data area for ROM usage in low RAM @ 400H....)
        MOV    DS,AX
       CMP     byte [DEBUG_FLAG],0H  ;Debug mode (normally off)
       POP     DS

       JNZ     MENU_ON
       MOV     BX,IBM_MENU_OFF              ;Enter "OFF"
       CALL    PRINT_STRING
       JMP     IBM_LOOP1
MENU_ON:
       MOV     BX,IBM_MENU_ON        ;Enter "ON"
       CALL    PRINT_STRING
IBM_LOOP1:
       MOV     BX,IBM_MENU2          ;Enter the rest of the menu
       CALL    PRINT_STRING

       MOV     AX,CS
       MOV     DS,AX                 ;Just to be safe for below

       call    CICO                  ;Get a command from Console
       mov     ah,0
       CMP     AL,ESC                ;Abort if ESC
       JNZ     NOT_ESC_IBM
       JMP     INIT                  ;Back to start of Monitor
NOT_ESC_IBM:
       cmp     al,'A'                ;Find meuu option from table
       jb      IBM_LOOP                     ;must be A to Z
       cmp     al,'Z'
       jg      IBM_LOOP
       sub     al,'A'                ;calculate offset
       shl     al,1                  ;X 2
       add     ax,IBM_TABLE          ;Note DS:=CS: in this monitor by default
       mov     bx,ax
       CALL    CRLF
       mov     ax,[cs:bx]            ;get location of routine CS:[BX]
       call    ax                    ;<----------- This is the PC-BIOS Menu CMD call
       jmp     IBM_LOOP              ;finished



;----------- Initilize RAM and hardware to look like an IBM-PC setup

SETUP_IBM_BIOS:
        mov     ax,cs
        mov     ds,ax                ;DS is this ROM CS

        sub     ax,ax
        mov     es,ax                ;ES: is 0H in RAM for STOW's below
       CLD                           ;Default to direction up

       mov     cx,256                ;Fill all 8086 interrupts initially with a default error trapping pointer
        sub     di,di                ;clear destination register
iloop:  mov     ax,dummy_return             ;set to just return from interrupt offsets
        stosw                        ;(ES: used for final location)
        mov     ax,cs                ;Interrupt segment pointer to here.
        stosw                        ;<-- Note the default segment will be this CS for all ints below
        loop    iloop

        mov     di,NMIint            ;First setup NMI vector in low RAM (at 8H)
        mov     ax,NMI_hnd           ;Have it point to Dummy return in this monitor
        stosw                        ;(ES: used for final location)

        mov     cx,8                 ;Set all 8 hardware interrupts for 8259A (at I/O port address 20H)
```

```
        mov     si,vec_tbl_8258A      ;Move the pointers in vec_tbl-8259A to low RAM starting at 20H
        mov     di,Start8259A_Ints    ;Note DS:(=CS:) is source,   ES: is destination
iloop1: movsw
        inc     di                    ;Skip over the segment pointer (already done above), to next vector offset
        inc     di
        loop    iloop1

        mov     cx,16                 ;Set all 16 software interrupts
        mov     si,vec_tbl_soft_ints
        mov     di,CRTINT             ;Start location in low RAM
iloop2: movsw                         ;Note DS: (=CS:) is source,    ES: is destination
        inc     di                    ;Skip over the segment pointer (already done above), to next vector offset
        inc     di
        loop    iloop2


                                      ;Now a few special case situations...
        MOV    SI,OLD_DISKIO          ;We need to handle software Int 40H (The relocated old INT 13H PC Bios Floppy
I/O)
        mov    DI,OLD_DISK_VEC
        movsw                         ;Note DS:(=CS:) is source,    ES: is destination  (The segment pointer is already
done above)

        MOV    SI,FDISK_3PARM_TBL     ;We need to move the boot diskette paramater table to Int 1EH*4 area. (Use 1.44M
3" Floppy)
        mov    DI,FDISK_PARMS
        movsw                         ;Note DS:(=CS:) is source,    ES: is destination

        MOV    SI, HDISK_PARM_TBL     ;Setup the default HARD DISK #1, table POINTER offset
        mov    DI,HDISK_PARMS         ;41H*4, (104H)
        movsw                         ;;Note DS: is source,    ES: is destination

        MOV    SI, HDISK_PARM_TBL     ;Setup the default HARD DISK #2, table POINTER offset
        mov    DI,HDISK2_PARMS                ;46H*4
        movsw                         ;Note DS:(=CS:) is source,    ES: is destination


                                              ;Now set up the memory variables
        XOR     AX,AX                         ;Now set DS: (=0) to data area for ROM usage in low RAM @400H
        MOV    DS,AX
        mov     word [expram],msize-64               ;show expansion ram size
        mov     word [memrsz],msize           ;and total memory size (640K)
        mov     word [eqflag],0100001001101101B      ;set equipment flag so IBM is happy

                                              ;bit  0     disk drives present
                                              ;bit  1     8087 Present
                                              ;bits 2,3   > 64K ram
                                              ;bits 4,5   default to colour card
                                              ;bits 6,7   floppy drives -1 (if bit 0 =1)
                                              ;bit  8     DMA support installed (PCjr, Tandy)
                                              ;bits 9,10,11 no of serial ports
                                              ;bit  12    no game adaptor
                                              ;bit  13    serial printer attachd (PCjr)
                                              ;bits 14,15 no of printers

        mov     ax,keybuff            ;keyboard interrupt pointers
        mov     [bufhd],ax
        mov     [buftl],ax
        mov     byte [chrcnt],0

        mov     byte [VERIFY_FLAG],0  ;Initially set for sector reads (rather than sector verifys)

                                      ;Initilize hardware to emmulate IBM-PC settings
       mov     bx,PIC_INIT_MSG                ;Send a signon about initilizing the 8259A
       call    PRINT_STRING

       mov     al,MasterICW1          ;Initilize the 8259A PIC Controller
       out     MASTER_PIC_PORT,al
       mov     al,MasterICW2          ;Ints starts at 20H in RAM
```

```
        out     MASTER_PIC_PORT+1,al
        mov     al,MasterICW4           ;No slaves above, so 8259 does not expect ICW3
        out     MASTER_PIC_PORT+1,al

        mov     al,11111111b            ;No V0 & V1 for now
        out     MASTER_PIC_PORT+1,al

                                        ;Next move the current time into the system tick bytes in low RAM
                                        ;Remember DS: is already set to data area for ROM usage in low RAM (400H)

        mov     word [timlow],0                 ;Next setup timer/RTC default values
        mov      word [timhi],0
        mov      word [timofl],0                ;Set clock tick info to 0 in low ram

READ_CMOS:
        MOV      AL,02                  ;Point to CMOS_SECONDS
        OUT     RTCSEL,AL
        IN      AL,RTCDATA

        CMP      AL,059H                        ;BCD 0-59 sec only
        JBE     SEC_OK
        JMP      TOD_ERROR

SEC_OK: CALL     CVT_BINARY
        MOV      BL,COUNTS_SEC          ;Timer needs sec X (COUNTS_SEC/sec)
        MUL      BL                     ;AX x BL
        MOV      CX,AX                  ;Store in CX

        MOV      AL,03H                 ;Point to CMOS_MINUTES
        OUT     RTCSEL,AL
        IN      AL,RTCDATA

        CMP      AL,059H                        ;BCD 0-59 minutes only
        JBE     MIN_OK
        JMP      TOD_ERROR

MIN_OK: CALL     CVT_BINARY
        MOV      BX,COUNTS_MIN          ;Timer needs mins X (COUNTS_MIN/sec)
        MUL      BX                     ;AX x BX
        ADD      AX,CX                  ;Add in seconds from above
        MOV      CX,AX                  ;Store in CX

        MOV      AL,04H                 ;Point to CMOS_HOURS
        OUT     RTCSEL,AL
        IN      AL,RTCDATA

        CMP      AL,023H
        JBE     HOUR_OK
        JMP       TOD_ERROR             ;BCD 0-24 Hours only

HOUR_OK:CALL     CVT_BINARY
        MOV      DX,AX

        MOV      BL,COUNTS_HOUR                 ;Timer needs hours X (COUNTS_MIN/sec)
        MUL      BL                     ;AL X BL
        ADD      AX,CX                  ;Add in seconds from above
        ADC      DX,0000H

        MOV      [timhi],DX
        MOV      [timlow],AX

        CMP     byte [DEBUG_FLAG],2     ;Is Detailed Debug mode on
        JGE     DEBUG_CMOS
        JMP     DONE_READ_CMOS          ;If not skip

DEBUG_CMOS:
        PUSH    AX                      ;Show first 8 bytes of sector data on serial output (for debugging)
        PUSH    BX
        PUSH    CX
```

```
        PUSH    DX
        MOV     BX,CMOS_DATA0_MSG      ;"CMOS DATA:Mins (BCD)/Hex = "
        CALL    SERIAL_PRINT_STRING
         MOV     AL,03H                 ;Point to CMOS_MINUTES
        OUT     RTCSEL,AL
        IN      AL,RTCDATA
        PUSH    AX
        MOV     CL,AL
        CALL    SERIAL_AL_HEXOUT
        MOV     CL,'/'
        CALL    SERIAL_OUT
        POP     AX
         CALL    CVT_BINARY
         MOV     CL,AL
        CALL    SERIAL_AL_HEXOUT

        MOV     BX,CMOS_DATA1_MSG      ;"CMOS DATA:Hours (BCD)/Hex = "
        CALL    SERIAL_PRINT_STRING
         MOV     AL,04H                 ;Point to CMOS_HOURS
        OUT     RTCSEL,AL
        IN      AL,RTCDATA
        PUSH    AX
        MOV     CL,AL
        CALL    SERIAL_AL_HEXOUT
        MOV     CL,'/'
        CALL    SERIAL_OUT
        POP     AX
         CALL    CVT_BINARY
         MOV     CL,AL
        CALL    SERIAL_AL_HEXOUT
        POP     DX
        POP     CX
        POP     BX
        POP     AX

DONE_READ_CMOS:
        MOV     AH,0H                  ;Initilize Serial Port (Used for debugging display if requited)
        MOV     AL,80H                 ;This sets for 9600 Baud. (However we will run at 38,400, see INT 14H)
        MOV     DX,0
         int     14H                    ;Serial out Handler   (Software Interrupt 14H)

                                       ;Next, check if the is an extra ROM's/Software on board. This follows the IBM
                                       ;format by looking at C8000H-F4000H (on 2K pages) for 55H,AAH and (length/512)
                                       ;in the 3rd byte.  You may wish to ignore the Checksum testing if you are testing
code in RAM.
                                       ;If a valid "ROM" then we do a JMPF to byte 3 in that ROM/Code. It assumes that
the code there
EXTRA_ROMS_CHECK:                      ;will finish with a far return. Note there may be more than one ROM module.
        MOV     DX,0C800H              ;Will do exactly as IBM ROM does it!
ROM_SCAN_1:
        MOV     DS,DX
        SUB     BX,BX
        MOV     AX,[BX]
        CMP     AX,0AA55H              ;Is the indicator flag there
        JNZ     NEXT_ROM
        CALL    ROM_CHECK
        JMP     WE_ARE_DONE

NEXT_ROM:
        ADD     DX,0080H               ;next 2K page
WE_ARE_DONE:
        CMP     DX,0F600H              ;At F6000H yet?
        JL      ROM_SCAN_1
        RET                            ;Finished SETUP_IBM_BIOS


TOD_ERROR:
        MOV     BX,CMOS_CLOCK_MSG      ;Error reading CMOS Clock chip
        CALL    PRINT_STRING
```

```
        JMP     DONE_READ_CMOS

ROM_CHECK:
        MOV     AX,40H                  ;Set ES=DSEG
        MOV     ES,AX
        SUB     AH,AH                   ;Zero out AH
        MOV     AL,[BX+2]               ;Byte 3 = length/512
        MOV     CL,09H                  ;X512
        SHL     AX,CL
        MOV     CX,AX
        PUSH    CX
        MOV     CL,4
        SHR     AX,CL
        ADD     DX,AX                   ;Point to next module
        POP     CX
        CALL    ROS_CHECKSUM_CNT        ;we may skip for testing see below
        JZ      ROM_CHECK1
        PUSH    AX
        MOV     BX,ROM_ERR_MSG          ;Report checksum error
        CALL    PRINT_STRING
        POP     AX
        CALL    AL_HEXOUT
        MOV     BX,H_MSG_CRLF
        CALL    PRINT_STRING
        RET

ROM_CHECK1:
        PUSH    DX                              ;Call is as IBM did!
        MOV     word [ES:IO_ROM_INIT],0003H     ;Load offset
        MOV     [ES:IO_ROM_SEG],DS              ;Load segment (normally it will not be this CS)
        CALL    [ES:IO_ROM_INIT]                ;Call the code in that particular ROM
        POP     DX
        RET

ROS_CHECKSUM_CNT:                       ;Do a checksum count on a ROM
        XOR     Al,AL
CHK26:  ADD     AL,[DS:BX]
        INC     BX
        LOOP    CHK26
        OR      AL,AL
;       XOR     AL,AL                   ;If we wish to ignore the checksum
        RET

CVT_BINARY:                             ;Convert BCD in [AL] to Binary in [AL]
        PUSH    CX
        PUSH    AX
        AND     AX,0FH
        MOV     CX,AX                   ;Save low digit
        POP     AX
        PUSH    CX                      ;On Stack
        MOV     CL,4
        ROR     AX,CL
        AND     AL,0FH

        MOV     CL,10
        MUL     CL

        POP     CX
        ADD     AX,CX                   ;Add in low digit
        POP     CX
        RET

                                        ;Clear the whole IBM-AT "extra store area" to 0's
                                        ;It seems MSDOS V4.01 counts on at least the diskette area being 0's
                                        ;It hangs on a boot otherewise!
ZERO_AT_RAM_AREA:
        PUSH    DS
        XOR     AX,AX                   ;Set DS to data area for ROM usage in low RAM @ 400H....)
        MOV     DS,AX
```

```
        mov     cx,(496H-490H)              ;Count of bytes in the area
        mov     di,DSK_STATE
        MOV    AX,0
iloop3: MOV    [DS:DI],AX           ;DS: (40H) is destination
        inc     di                  ;Skip over the segment pointer (already done above), to next vector offset
        inc     di
        loop    iloop3
        POP    DS
        RET



;----------- Menu CMD to Boot MS-DOS from a Floppy Disk using this BIOS

 MMENU_FBOOT_DOS:                   ;Come here from main menu. Debug mode ALWAYS off
       push    DS
       PUSH    AX
        XOR    AX,AX                ;Set DS to data area for ROM usage in low RAM @ 400H....)
        MOV    DS,AX
       mov     [DEBUG_FLAG],AX             ;Debug mode normally off (AX=0)
       POP     AX
       POP     DS

 MENU_FBOOT_DOS:                    ;Come here from IBM BIOS menu (Debug mode MAY be on)
       MOV     BX,FBOOT_DOS_MSG     ;Booting MS-DOS
       CALL    PRINT_STRING
       MOV     DL,0                 ;Make sure bit 7 is 0 for Floppy
       PUSH    DX                   ;Save value in DX (DL=0 for Floppy Boot)


COMMON_BOOT_DOS:                    ;Common BOOT MS-DOS/FreeDOS entry point
        CALL   SETUP_IBM_BIOS       ;Initilize RAM and hardware
       CALL    ZERO_AT_RAM_AREA     ;This area needs to be 0's for MS-DOS 4.01

       MOV     AL,11H               ;Point to Interrupt Control Register of MM581657A on PIC/RT board
       OUT     RTCSEL,AL
       MOV     AL,00000010B         ;Set 0.1 Second interrupt bit
       OUT     RTCDATA,AL           ;For this to come out on the S-100 V0 jumper pin 1 of P39 to pin 1 of P37 on the
RTC-PIC board
                                    ;Pin 13 of the MM581657A will pulse every 0.1 sec (accurately).

       MOV     AL,10H               ;Clear Interrupt Status Register
       OUT     RTCSEL,AL            ;This seems to be necessary to get the first int going!
       IN      AL,RTCDATA

       mov     al,11111100b         ;Allow S-100 bus ints V0 & V1 (only) now
       out     MASTER_PIC_PORT+1,al

       sti                          ;Enable hardware interrupts
       POP     DX                   ;Get back Floppy/HDISK info

        int     19H                 ;<<<<<<< Boot PC-DOS with software int 19H

                                    ;Should never return here IF no problem
       JMP     word  0F000H:INIT    ;Far Jump to F000H:INIT (Start of this monitor)


;----------- Menu CMD to Boot MS-DOS from a HARD Disk using this BIOS

MMENU_HBOOT_DOS:
       push    DS
       PUSH    AX
        XOR    AX,AX                ;Set DS to data area for ROM usage in low RAM @ 400H....)
        MOV    DS,AX
       mov     [DEBUG_FLAG],AX             ;Debug mode normally off (AX=0)
       POP     AX
       POP     DS

 MENU_HBOOT_DOS:
```

```
        MOV     BX,HBOOT_DOS_MSG        ;Booting MS-DOS
        CALL    PRINT_STRING
        MOV     DX,0080H                ;Make sure bit 7 is 1 for HDisk
        PUSH    DX                      ;Save value in DX (DL=80H for HDisk Boot)
        JMP     COMMON_BOOT_DOS



;----------- Menu CMD to test 8259A Interrupt driven Keypress code using this BIOS

MENU_KEY_TEST:
        MOV     BX,KEY_TEST_MSG                 ;Keyboard test
        CALL    PRINT_STRING

        mov     al,11111101b            ;Allow V1 on 8259A now
        out     MASTER_PIC_PORT+1,al

        sti                             ;Enable hardware interrupts

Next_Key:
        MOV     BX,IN_CHAR_MSG          ;Input character =
        CALL    PRINT_STRING

Next_Key1:
        MOV     AH,01H                  ;Check if anything there
         int     16H                    ;Get Keyboard status. Console Input Handler   (Software Interrupt 16H)
         JZ     Next_Key1

        MOV     AH,0H                   ;Get actual character from buffer
         int     16H                    ;Get Character. Console Input Handler   (Software Interrupt 16H)

         CMP    AL,ESC
         JZ     Key_Done
        MOV     CL,AL
        CALL    CO                      ;Send direct to Console
        MOV     BX,GOT_CHAR_MSG                 ;Recieved character =
        CALL    PRINT_STRING
         CALL   CRLF
        JMP     Next_Key
Key_Done:
        mov     al,11111111b            ;Do not Allow V1 on 8259A again
        out     MASTER_PIC_PORT+1,al
        cli                             ;Turn hardware int's back off
        JMP     IBM_BIOS



;----------- Menu CMD to test Console out code using this BIOS

MENU_CO_TEST:
        MOV     BX,CO_TEST_MSG          ;Keyboard test
        CALL    PRINT_STRING

Next_CO:MOV     BX,IN_CHAR_MSG          ;Input character =
        CALL    PRINT_STRING
        CALL    CI                      ;Return the char in AL
         CMP    AL,ESC
         JZ     CO_Done
         PUSH   AX
         MOV    CL,AL
         CALL   CO
        MOV     BX,OUT_CHAR_MSG                 ;"<---- Character recieved",CR,LF, Char displayed via Int 10H ="
        CALL    PRINT_STRING
         POP    AX

        MOV     AH,0EH                  ;AH=0EH = TTY output, char in AL

         int     10H                    ;Console out Handler   (Software Interrupt 10H)
```

```
        CALL    CRLF
        JMP     Next_CO
CO_Done:
        cli                             ;Turn hardware int's back off
        JMP     IBM_BIOS



;------------ Menu CMD to test combined key-in / video out using this BIOS

MENU_BUFF_IO:
        MOV     BX,BUFF_TEST_MSG        ;Keyboard buffer test
        CALL    PRINT_STRING

        mov     al,11111101b            ;Allow V1 on 8259A now
        out     MASTER_PIC_PORT+1,al

        sti                             ;Enable hardware interrupts

Next_CI:
        MOV     AH,01H                  ;Check if anything there
        int      16H                    ;Get Keyboard status. Console Input Handler   (Software Interrupt 16H)
        JZ      Next_CI

        MOV     AH,0H                   ;Get actual character from buffer to AL
        int      16H                    ;Get Character. Console Input Handler   (Software Interrupt 16H)

        CMP     AL,ESC
        JZ      CO_Done

        MOV     AH,0EH                  ;AH=0EH = TTY output, char in AL
        int      10H                    ;Console out Handler   (Software Interrupt 10H)
        JMP     Next_CI


;----------- Menu CMD to test Serial Port character output using this BIOS to a serial terminal
;           Make sure you have the Baud rate is the same on both ends. (We will leave it at 38,4000 Baud)

MENU_SIO_TEST:
        MOV     BX,SIO_TEST_MSG         ;Output to Serial port test
        CALL    PRINT_STRING

        MOV     AH,0H                   ;AH=0 Initilize Port
        MOV     AL,80H                  ;This sets for 9600 Baud. However we will run at 38,400 (see INT 14H)
        MOV     DX,0

        int      14H                    ;Serial out Handler   (Software Interrupt 14H)

        OR      AH,AH                   ;Any errors
        JZ      Next_SIO

        PUSH    AX
        MOV     BX,SIO_INIT_ERR                 ;Error initilizing Serial port
        CALL    PRINT_STRING
        POP     AX
        MOV     AL,AH
        CALL    AL_HEXOUT
        MOV     BX,H_MSG_CRLF
        CALL    PRINT_STRING
        JMP     IBM_BIOS

Next_SIO:
        CALL    CI                      ;Return the char in AL
        CMP     AL,ESC
        JZ      SIO_DoneTest

        PUSH    AX
        MOV     CL,AL
```

```
        CALL    CO
        POP     AX

        MOV     AH,01H                  ;AH=char output, char in AL
        MOV     DX,0

        int      14H                    ;Serial out Handler   (Software Interrupt 14H)

        OR      AH,AH                   ;Any errors
        JZ      Next_SIO
        PUSH    AX
        MOV     BX,SIO_ERR      ;Error sending to Serial port
        CALL    PRINT_STRING
        POP     AX
        MOV     AL,AH
        CALL    AL_HEXOUT
        MOV     BX,H_MSG_CRLF
        CALL    PRINT_STRING
        JMP     IBM_BIOS

SIO_DoneTest:
        JMP     IBM_BIOS



;----------- Menu CMD to test Timer code using this BIOS

MENU_TIMER_TEST:
        MOV     BX,TIMER_TEST_MSG       ;Timer test
        CALL    PRINT_STRING

        MOV     AL,11H                  ;Point to Interrupt Control Register of MM581657A on PIC/RT board
        OUT     RTCSEL,AL
        MOV     AL,00000010B            ;Set 0.1 Second interrupt bit
        OUT     RTCDATA,AL              ;For this to come out on the S-100 V0 jumper pin 1 of P39 to pin 1 of P37 on the
RTC-PIC board
                                        ;Pin 13 of the MM581657A will pulse every 0.1 sec (accurately).

        MOV     AL,10H                  ;Clear Interrupt Status Register
        OUT     RTCSEL,AL               ;This seems to be necessary to get the first INT going!
        IN      AL,RTCDATA

        mov     al,11111110b            ;Allow V0 on 8259A now
        out     MASTER_PIC_PORT+1,al

        sti                             ;Enable hardware interrupts

Next_Timer:
        MOV     BX,TIMER_DATA_MSG       ;Get Timer values
        CALL    PRINT_STRING

        CALL    CI                      ;Return the char in AL
        CMP     AL,ESC
        JZ      Timer_Done

        MOV     BX,TIMER_LOW_MSG        ;timlow =
        CALL    PRINT_STRING
        PUSH    DS
        XOR     AX,AX                   ;Set DS to data area for ROM usage in low RAM @ 400H....)
        MOV     DS,AX
        mov      AX,[timlow]
        CALL    AX_HEXOUT

        MOV     BX,TIMER_HIGH_MSG       ;timhi =
        CALL    PRINT_STRING
        mov      AX,[timhi]
        CALL    AX_HEXOUT

        MOV     BX,TIMER_OFLOW_MSG      ;timofl =
```

```
        CALL    PRINT_STRING
        mov     AX,[timofl]
        CALL    AX_HEXOUT
        MOV     BX,H_Msg                ;"H$"
        CALL    PRINT_STRING
        POP     DS
        JMP     Next_Timer
Timer_Done:
        mov     al,11111111b            ;Do not Allow V0 on 8259A again
        out     MASTER_PIC_PORT+1,al
        cli                             ;Turn hardware int's back off
        JMP     IBM_BIOS


;----------- Menu CMD to test Floppy Disk (5") sequential sector reads using this BIOS
;           Will read sequentially up to 9 X 512 byte sectors from 5" DDDS 360K floppy (9 Sec/Track)
;            into RAM using the ZFDC controller board. (IBM says never more than
;           9 sectors at a time for this type of disk,  actully never changes the track #, but the
;           ZFDC can handle this if it did anyway!)
;            Will always read into RAM starting at 500H using the ZFDC controller board

FSEQ_5RD_TEST:
        PUSH    DS                      ;Save Monitor current DS
        XOR     AX,AX                   ;Set DS to data area for ROM usage in low RAM @ 400H....)
        MOV     DS,AX

        MOV     BX,SEC_5RD_MSG          ;Say Reading sectors
        CALL    PRINT_STRING

        MOV     AL,0H                   ;Flag to indicate ZFDC board is NOT Initilized
        MOV     [ZFDC_INIT_FLAG],AL     ;DS is already set for low RAM area

        CALL    INIT_ZFDC                       ;Initilize the ZFDC board hardware

        CMP     byte [ZFDC_INIT_FLAG],0FFH      ;Is Board initilized correctly
        JZ      ZFDC_5OK1

        MOV     BX,ZFDC_FAIL_MSG
        CALL    PRINT_STRING
        POP     DS                      ;Balance up Monitor stack
        JMP     IBM_BIOS

ZFDC_5OK1:
        mov     dl,01H                  ;Drive 1, side A
        mov     ah,0H

        int     13h                     ;AH=0, reset floppy disk system

        JNC     RESET_5OK1
        MOV     BX,RESET_FAIL_MSG
        CALL    PRINT_STRING
        POP     DS                      ;Balance up stack
        JMP     IBM_BIOS


RESET_5OK1:
        MOV     BX,SIDE_REQUEST_MSG
        CALL    PRINT_STRING
        call    CICO                    ;Get a command from Console
        PUSH    AX
        MOV     BX,CRLFMSG              ;"CR,LF"
        CALL    PRINT_STRING
        POP     AX
        CMP     AL,'B'
        JZ      B5_SIDE
        MOV     BX,SIDE_A_SET_MSG
        CALL    PRINT_STRING
        MOV     DX,0001H                ;Side A (DL bit 7 = 0 so Floppy disk)
        JMP     OVER5_SIDE
```

```
B5_SIDE:MOV    BX,SIDE_B_SET_MSG
        CALL   PRINT_STRING
        MOV    DX,0101H              ;Side B, Disk 1 (ZFDC #2)

OVER5_SIDE:
        PUSH   DX                    ;Save side info for below

        MOV    BX,ENTERRAM_FTRKL     ;"Track number,(xxH)"
        CALL   PRINT_STRING
        CALL   GET2DIGITS            ;Get 8 bit value (2 digits) to AL. (BX, CX & DX Unchanged)
        MOV    CH,AL
        PUSH   CX                    ;Save for below
        MOV    BX,H_MSG_CRLF         ;"H  CR,LF"
        CALL   PRINT_STRING

        MOV    BX,ENTERRAM_SECL      ;"Starting sector number,(xxH) = "
        CALL   PRINT_STRING
        CALL   GET2DIGITS            ;Get 8 bit value (2 digits) to AL. (BX, CX & DX Unchanged)
        POP    CX
        MOV    CL,AL
        PUSH   CX                    ;Save Track & Sec for below
        MOV    BX,H_MSG_CRLF         ;"H  CR,LF"
        CALL   PRINT_STRING

SEQ_5OK4:
        MOV    BX,ENTER_COUNT        ;Enter # of sectors
        CALL   PRINT_STRING
        CALL   GET2DIGITS            ;Get 8 bit value (2 digits) to AL. (BX, CX & DX Unchanged)
        CMP    AL,09
        JBE    S5OK3
S5OK5:  MOV    BX,OVER_COUNT_10
        CALL   PRINT_STRING
        JMP    SEQ_5OK4              ;Try again
S5OK3:  OR     AL,AL
        JZ     S5OK5

        PUSH   AX                    ;Save sector count (already in AL)
        MOV    BX,H_MSG_CRLF         ;"H  CR,LF"
        CALL   PRINT_STRING

        SUB    AX,AX
        MOV    ES,AX
        MOV    BX,500H               ;Will always dump data to 0000:500H
        POP    AX
        POP    CX                    ;Track, Sec
        POP    DX                    ;Side & Drive 0
        mov     ah,02h               ;Read x sectors  (IBM has a max of 15 sectors/call fot IBM-AT)
                                     ;on a 1.2M Floppy disk in their IBM PC-AT Bios. I assume 18 for 1.44 Disk)
                                     ;(This is where MS-DOS loads MSDOS.SYS from on disk)
        int     13H                  ;AH=2, CX=0001, read 6 byte sectors -- as in early MSDOS systems!

        JNC    SEQ_5OK1              ;If NC then no errors
        MOV    BX,SQRD5FAILMSG
        CALL   PRINT_STRING
        POP    DS                    ;Balance up Monitor stack
        JMP    IBM_BIOS              ;Will return back up to start of Monitor

SEQ_5OK1:
        MOV    BX,SQRD5OKMSG
        CALL   PRINT_STRING

        MOV    CX,16                 ;Display the first 16 bytes at ES:BX in RAM
        SUB    AX,AX
        MOV    ES,AX
        MOV    BX,500H               ;Will always dump data to 0000:500H

        CALL   SIMPLE_SECTOR_DUMP    ;Dump first CX bytes of sector data at ES:BX on CRT
        POP    DS                    ;Balance up stack
        JMP    IBM_BIOS              ;All done
```

```
;----------- Menu CMD to test Floppy Disk (3") sequential sector reads using this BIOS
;            Will read sequentially 18 X 512 byte sectors from 3" DDDS 1.44M floppy (18 Sec/Track)
;             into RAM  using the ZFDC controller board. (IBM says never more than
;            18 sectors at a time for this type of disk)
;             Will always read into RAM starting at 500H using the ZFDC controller board

FSEQ_3RD_TEST:
        PUSH    DS                      ;Save Monitor current DS
         XOR    AX,AX                   ;Set DS to data area for ROM usage in low RAM @ 400H....)
         MOV    DS,AX

        MOV     BX,SEC_3RD_MSG          ;Say Reading sectors
        CALL    PRINT_STRING

        MOV     AL,0H                   ;Flag to indicate ZFDC board is NOT Initilized
        MOV     [ZFDC_INIT_FLAG],AL     ;DS is already set for low RAM area

        CALL    INIT_ZFDC                       ;Initilize the ZFDC board hardware

        CMP     byte [ZFDC_INIT_FLAG],0FFH      ;Is Board initilized correctly
        JZ      ZFDC_3OK1

        MOV     BX,ZFDC_FAIL_MSG
        CALL    PRINT_STRING
        POP     DS                      ;Balance up stack
        JMP     IBM_BIOS

ZFDC_3OK1:
         mov     dl,0H                  ;Drive 0, side A
        mov     ah,0H

         int     13H                    ;AH=0, reset floppy disk system

        JNC     RESET_3OK1
        MOV     BX,RESET_FAIL_MSG
        CALL    PRINT_STRING
        POP     DS                      ;Balance up stack
        JMP     IBM_BIOS


RESET_3OK1:
        MOV     BX,SIDE_REQUEST_MSG
        CALL    PRINT_STRING
        call    CICO                    ;Get a command from Console
        PUSH    AX
        MOV     BX,CRLFMSG               ;"CR,LF"
        CALL    PRINT_STRING
        POP     AX
        CMP     AL,'B'
        JZ      B3_SIDE
        MOV     BX,SIDE_A_SET_MSG
        CALL    PRINT_STRING
        MOV     DX,0000H                ;Side A (DL bit 7 = 0 so Floppy disk)
        JMP     OVER3_SIDE
B3_SIDE:MOV     BX,SIDE_B_SET_MSG
        CALL    PRINT_STRING
        MOV     DX,0100H                ;Side B, Disk 0

OVER3_SIDE:
        PUSH    DX                      ;Save side for below

        MOV     BX,ENTERRAM_FTRKL       ;"Track number,(xxH)"
        CALL    PRINT_STRING
        CALL    GET2DIGITS              ;Get 8 bit value (2 digits) to AL. (BX, CX & DX Unchanged)
        MOV     CH,AL
        PUSH    CX                      ;Save for below
        MOV     BX,H_MSG_CRLF           ;"H  CR,LF"
```

```
        CALL    PRINT_STRING

        MOV     BX,ENTERRAM_SECL        ;"Starting sector number,(xxH) = "
        CALL    PRINT_STRING
        CALL    GET2DIGITS              ;Get 8 bit value (2 digits) to AL. (BX, CX & DX Unchanged)
        POP     CX
        MOV     CL,AL
        PUSH    CX                      ;Save Track & Sec for below
        MOV     BX,H_MSG_CRLF           ;"H  CR,LF"
        CALL    PRINT_STRING

SEQ_3OK4:
        MOV     BX,ENTER_COUNT          ;Enter # of sectors
        CALL    PRINT_STRING
        CALL    GET2DIGITS              ;Get 8 bit value (2 digits) to AL. (BX, CX & DX Unchanged)
        CMP     AL,18
        JBE     S3OK3
S3OK5:  MOV     BX,OVER_COUNT_19
        CALL    PRINT_STRING
        JMP     SEQ_3OK4                ;Try again
S3OK3:  OR      AL,AL
        JZ      S3OK5

        PUSH    AX                      ;Save sector count (already in AL)
        MOV     BX,H_MSG_CRLF           ;"H  CR,LF"
        CALL    PRINT_STRING

        SUB     AX,AX
        MOV     ES,AX
        MOV     BX,500H                 ;Will always dump data to 0000:500H
        POP     AX
        POP     CX                      ;Track, Sec
        POP     DX                      ;Side & Drive 0
        mov     ah,02h                  ;Read x sectors  (IBM has a max of 15 sectors/call fot IBM-AT)
                                        ;on a 1.2M Floppy disk in their IBM PC-AT Bios. I assume 18 for 1.44 Disk)
                                        ;(This is where MS-DOS loads MSDOS.SYS from on disk)
        int     13H                     ;AH=2, CX=0001, read 6 byte sectors -- as in early MSDOS systems!

        JNC     SEQ_3OK1                ;If NC then no errors
        MOV     BX,SQRD5FAILMSG
        CALL    PRINT_STRING
        POP     DS                      ;Balance up stack
        JMP     IBM_BIOS                ;Will return back up to start of Monitor

SEQ_3OK1:
        MOV     BX,SQRD3OKMSG           ;Read sectors from 3" 1.44M Floppy disk OK
        CALL    PRINT_STRING

        MOV     CX,16                   ;Display the first 16 bytes at ES:BX in RAM
        SUB     AX,AX
        MOV     ES,AX
        MOV     BX,500H                 ;Will always dump data to 0000:500H

        CALL    SIMPLE_SECTOR_DUMP      ;Dump first CX bytes of sector data at ES:BX on CRT
        POP     DS                      ;Balance up stack
        JMP     IBM_BIOS                ;All done



;----------- Menu CMD to test HDISK sequential sector READ's using this BIOS
;           Will read 512 byte sectors from 2nd IDE CF-Card
;            into RAM starting at 500H using the IDE controller board
HSEQ_RD_TEST:
        PUSH    DS                      ;Save Monitor current DS
        XOR     AX,AX                   ;Set DS to data area for ROM usage in low RAM @ 400H....)
        MOV     DS,AX

        MOV     BX,SEC_HDRD_MSG                 ;Say Reading sectors
        CALL    PRINT_STRING
```

```
        MOV     BX,ONE_MOMENT_MSG       ;One moment while IDE disk is being initilized
        CALL    PRINT_STRING


        CALL    SET_DRIVE_B             ;Select the second Drive/CF card
        CALL    IDEinit                 ;Initialize drive 1. If there is no drive abort
        JZ      HSEQ_RD1

        MOV     BX,DRIVE2_ERR           ;Warn second IDE drive did not initilize
        CALL    PRINT_STRING
        POP     DS                      ;From above at start
        JMP     IBM_BIOS                ;Will return back up to start of Monitor

HSEQ_RD1:
        mov      dl,80H                 ;Hard Disk
        mov     ah,0H
        int      13h                    ;AH=0, reset floppy disk system

        JNC     HRESET_OK1
        MOV     BX,HRESET_FAIL_MSG      ;"Reset of HDisk Failed"
        CALL    PRINT_STRING
        POP     DS                      ;From above at start
        JMP     IBM_BIOS                ;Will return back up to start of Monitor

HRESET_OK1:
        XOR     AX,AX                   ;Set DS to data area for ROM usage in low RAM @ 400H....)
        MOV     DS,AX

        MOV     BX,HRESET_OK_MSG        ;"Reset of HDisk OK"
        CALL    PRINT_STRING

AGAIN:  MOV     BX,ENTERRAM_HEAD        ;"Starting HEAD number,(xxH) = "
        CALL    PRINT_STRING
        CALL    GET2DIGITS              ;Get 8 bit value (2 digits) to AL. (BX, CX & DX Unchanged)
        AND     AL,0FH
        MOV     [CURRENT_HEAD],AL
        MOV     BX,H_MSG_CRLF           ;"H  CR,LF"
        CALL    PRINT_STRING

        MOV     BX,ENTERRAM_FTRKL       ;"Track number,(xxH)"
        CALL    PRINT_STRING
        CALL    GET2DIGITS              ;Get 8 bit value (2 digits) to AL. (BX, CX & DX Unchanged)
        MOV     [CURRENT_TRACK],AL
        MOV     BX,H_MSG_CRLF           ;"H  CR,LF"
        CALL    PRINT_STRING

        MOV     BX,ENTERRAM_SECL        ;"Starting sector number,(xxH) = "
        CALL    PRINT_STRING
        CALL    GET2DIGITS              ;Get 8 bit value (2 digits) to AL. (BX, CX & DX Unchanged)
        AND     AL,00111111B
        MOV     [CURRENT_SECTOR],AL
        MOV     BX,H_MSG_CRLF           ;"H  CR,LF"
        CALL    PRINT_STRING

        MOV     BX,ENTER_COUNT          ;Enter # of sectors
        CALL    PRINT_STRING
        CALL    GET2DIGITS              ;Get 8 bit value (2 digits) to AL. (BX, CX & DX Unchanged)
        MOV     [SECTORS_TO_DO],AL
        MOV     BX,H_MSG_CRLF           ;"H  CR,LF"
        CALL    PRINT_STRING

        MOV     BX,LOOP_ESC_MSG                 ;"Will continously loop until ESC to abort "
        CALL    PRINT_STRING
        CALL    CI                      ;Wait for CR to start
        CMP     AL,CR
        JZ      XSEQ_5OK4
        POP     DS                      ;From above at start
        JMP     IBM_BIOS                ;Will return back up to start of Monitor
```

```
XSEQ_5OK4:
        SUB     AX,AX
        MOV     ES,AX
        MOV     DS,AX
        MOV     BX,500H                 ;Will always dump data to 0000:500H

        MOV     AH,02                   ;Read sector(s)
        MOV     AL,[SECTORS_TO_DO]
        MOV     CH,[CURRENT_TRACK]
        MOV     CL,[CURRENT_SECTOR]
        MOV     DH,[CURRENT_HEAD]
        MOV     DL,80H

        INT     13H                     ;Disk I/O Int

        JNC     READ_OK
        JMP     RD_ERROR
READ_OK:
        MOV     BX,SEC_READ_OK          ;Sector(s) read OK
        CALL    PRINT_STRING

        MOV     CX,16                   ;Display the first 16 bytes at ES:BX in RAM
        SUB     AX,AX
        MOV     ES,AX
        MOV     DS,AX                   ;Just to be safe below also
        MOV     BX,500H                 ;Will always dump data to 0000:500H

        CALL    SIMPLE_SECTOR_DUMP      ;Dump first CX bytes of sector data at ES:BX on CRT

        CALL    CSTS                    ;Any keyboard character will stop display
        JZ      HSEC_R7
        CALL    CI
        MOV     BX,CONTINUE_MSG
        CALL    PRINT_STRING
        CALL    CI
        CMP     AL,ESC
        JZ      IBM_BIOS1
HSEC_R7:
        CALL    CRLF
        MOV     CL,[CURRENT_SECTOR]
        INC     CL
        CMP     CL,DOS_MAXSEC           ;1-63 Sectors for custom Drive
        JLE     R_SAME_HEAD
        MOV     DH,[CURRENT_HEAD]
        INC     DH
        CMP     DH,DOS_MAXHEADS-1       ;(0...15), 16 heads Total for custom Drive
        JLE     R_SAME_TRACK
        MOV     byte [CURRENT_SECTOR],1       ;Back to sector 1
        MOV     byte [CURRENT_HEAD],0  ;back to head 0
        MOV     CH,[CURRENT_TRACK]      ;Next track
        INC     CH
        MOV     [CURRENT_TRACK],CH
        JMP     XSEQ_5OK4               ;Do next sector block

R_SAME_TRACK:
        MOV     byte [CURRENT_SECTOR],1       ;Back to sector 1
        MOV     [CURRENT_HEAD],DH       ;Next head
        JMP     XSEQ_5OK4               ;Do next sector block

R_SAME_HEAD:
        MOV     [CURRENT_SECTOR],CL
        JMP     XSEQ_5OK4               ;Do next sector block

IBM_BIOS1:
        POP     DS
        JMP     IBM_BIOS

RD_ERROR:
        MOV     BX,RD_ERR_MSG           ;"Read Error Sector Head ="
```

```
        CALL    PRINT_STRING
        MOV     AL,[CURRENT_HEAD]
        CALL    AL_HEXOUT
        MOV     BX,TRACK_MSG            ;"H  Track ="
        CALL    PRINT_STRING
        MOV     AL,[CURRENT_TRACK]
        CALL    AL_HEXOUT
        MOV     BX,SEC_MSG              ;"H  Sector ="
        CALL    PRINT_STRING
        MOV     AL,[CURRENT_SECTOR]
        CALL    AL_HEXOUT
        MOV     BX,H_MSG_CRLF           ;"H  CR,LF"
        CALL    PRINT_STRING
        POP     DS                      ;Balance up stack
        JMP     IBM_BIOS



;----------- IBM Menu CMD to check Sector R/W functions on IDE Board using INT 13H.

HSEC_RW_TEST:
        PUSH    DS
        MOV     BX,HRW_TEST_MSG                 ;Test Sector INT 13H Reade Write on Drive #2
        CALL    PRINT_STRING

        CALL    CICO
        CMP     AL,'Y'
        JZ      HSEC_RW0
        POP     DS                      ;Balance up stack
        JMP     IBM_BIOS

HSEC_RW0:
        MOV     BX,ONE_MOMENT_MSG       ;One moment while IDE disk is being initilized
        CALL    PRINT_STRING

        CALL    SET_DRIVE_B             ;Select the second Drive/CF card
        CALL    IDEinit                 ;Initialize drive 1. If there is no drive abort
        JZ      HSEC_RW1

        MOV     BX,DRIVE2_ERR           ;Warn second IDE drive did not initilize
        CALL    PRINT_STRING
        POP     DS                      ;Balance up stack
        JMP     IBM_BIOS

HSEC_RW1:
         mov     dl,80H                 ;Reset Hard Disk
        mov     ah,0H
         int     13h                    ;AH=0, reset floppy disk system

        JNC     HSEC_RW2
        MOV     BX,HRESET_FAIL_MSG      ;"Reset of HDisk Failed"
        CALL    PRINT_STRING
        POP     DS                      ;From above at start
        JMP     IBM_BIOS                ;Will return back up to start of Monitor

HSEC_RW2:
         XOR    AX,AX                   ;Set DS to data area for ROM usage in low RAM @ 400H....)
         MOV    DS,AX

        MOV     BX,HRESET_OK_MSG        ;"Reset of HDisk OK"
        CALL    PRINT_STRING

HSEC_RW3:
        MOV     BX,ENTERRAM_HEAD        ;"Starting HEAD number,(xxH) = "
        CALL    PRINT_STRING
        CALL    GET2DIGITS              ;Get 8 bit value (2 digits) to AL. (BX, CX & DX Unchanged)
        AND     AL,0FH
        MOV     [CURRENT_HEAD],AL
```

```
        MOV     BX,H_MSG_CRLF           ;"H  CR,LF"
        CALL    PRINT_STRING

        MOV     BX,ENTERRAM_FTRKL       ;"Track number,(xxH)"
        CALL    PRINT_STRING
        CALL    GET2DIGITS              ;Get 8 bit value (2 digits) to AL. (BX, CX & DX Unchanged)
        MOV     [CURRENT_TRACK],AL
        MOV     BX,H_MSG_CRLF           ;"H  CR,LF"
        CALL    PRINT_STRING

        MOV     BX,ENTERRAM_SECL        ;"Starting sector number,(xxH) = "
        CALL    PRINT_STRING
        CALL    GET2DIGITS              ;Get 8 bit value (2 digits) to AL. (BX, CX & DX Unchanged)
        AND     AL,00111111B
        MOV     [CURRENT_SECTOR],AL
        MOV     BX,H_MSG_CRLF           ;"H  CR,LF"
        CALL    PRINT_STRING

        MOV     BX,ENTER_COUNT          ;Enter # of sectors
        CALL    PRINT_STRING
        CALL    GET2DIGITS              ;Get 8 bit value (2 digits) to AL. (BX, CX & DX Unchanged)
        MOV     [SECTORS_TO_DO],AL
        MOV     BX,H_MSG_CRLF           ;"H  CR,LF"
        CALL    PRINT_STRING

        MOV     BX,LOOP_ESC_MSG                 ;"Will continously loop until ESC to abort "
        CALL    PRINT_STRING
        CALL    CI                      ;Wait for CR to start
        CMP     AL,CR
        JZ      HSEC_RW4
        POP     DS                      ;From above at start
        JMP     IBM_BIOS                ;Will return back up to start of Monitor

HSEC_RW4:
        SUB     AX,AX
        MOV     ES,AX
        MOV     DS,AX                   ;just in case
        MOV     BX,500H                 ;Will always dump data to 0000:500H

        MOV     AH,02                   ;Read sector(s)
        MOV     AL,[SECTORS_TO_DO]
        MOV     CH,[CURRENT_TRACK]
        MOV     CL,[CURRENT_SECTOR]
        MOV     DH,[CURRENT_HEAD]
        MOV     DL,80H

        INT     13H                     ;Disk I/O Int

        JNC     HSEC_RW5
        JMP     RD_ERROR

HSEC_RW5:
        MOV     CX,16                   ;Display the first 16 bytes at ES:BX in RAM
        SUB     AX,AX
        MOV     ES,AX
        MOV     DS,AX                   ;Just in case
        MOV     BX,500H                 ;Will always dump data to 0000:500H

        CALL    SIMPLE_SECTOR_DUMP      ;Dump first CX bytes of sector data at ES:BX on CRT

        SUB     AX,AX                   ;Now WRITE the secti=or back
        MOV     ES,AX
        MOV     DS,AX                   ;just in case
        MOV     BX,500H                 ;Will always dump data to 0000:500H

        MOV     AH,03                   ;Write sector(s)
        MOV     AL,[SECTORS_TO_DO]
        MOV     CH,[CURRENT_TRACK]
        MOV     CL,[CURRENT_SECTOR]
```

```
        MOV     DH,[CURRENT_HEAD]
        MOV     DL,80H

        INT     13H                     ;Write sector(s)

        JNC     HSEC_RW6
        JMP     WR_ERROR

HSEC_RW6:
        MOV     BX,SEC_BACK_OK          ;Sector(s) written BACK OK
        CALL    PRINT_STRING

        CALL    CSTS                    ;Any keyboard character will stop display
        JZ      HSEC_RW7
        CALL    CI
        MOV     BX,CONTINUE_MSG
        CALL    PRINT_STRING
        CALL    CI
        CMP     AL,ESC
        JZ      IBM_BIOS2
HSEC_RW7:
        CALL    CRLF
        MOV     CL,[CURRENT_SECTOR]
        INC     CL
        CMP     CL,DOS_MAXSEC           ;1-63 Sectors for custom Drive
        JLE     WR_SAME_HEAD
        MOV     DH,[CURRENT_HEAD]
        INC     DH
        CMP     DH,DOS_MAXHEADS-1       ;(0...15), 16 heads Total for custom Drive
        JLE     WR_SAME_TRACK
        MOV     byte [CURRENT_SECTOR],1       ;Back to sector 1
        MOV     byte [CURRENT_HEAD],0 ;back to head 0
        MOV     CH,[CURRENT_TRACK]      ;Next track
        INC     CH
        MOV     [CURRENT_TRACK],CH
        JMP     HSEC_RW4                ;Do next sector block

WR_SAME_TRACK:
        MOV     byte [CURRENT_SECTOR],1       ;Back to sector 1
        MOV     [CURRENT_HEAD],DH       ;Next head
        JMP     HSEC_RW4                ;Do next sector block

WR_SAME_HEAD:
        MOV     [CURRENT_SECTOR],CL
        JMP     HSEC_RW4                ;Do next sector block

IBM_BIOS2:
        POP     DS
        JMP     IBM_BIOS


WR_ERROR:
        MOV     BX,WR_ERR_MSG           ;"Write Error Sector Head ="
        CALL    PRINT_STRING
        MOV     AL,[CURRENT_HEAD]
        CALL    AL_HEXOUT
        MOV     BX,TRACK_MSG            ;"H  Track ="
        CALL    PRINT_STRING
        MOV     AL,[CURRENT_TRACK]
        CALL    AL_HEXOUT
        MOV     BX,SEC_MSG              ;"H  Sector ="
        CALL    PRINT_STRING
        MOV     AL,[CURRENT_SECTOR]
        CALL    AL_HEXOUT
        MOV     BX,H_MSG_CRLF           ;"H  CR,LF"
        CALL    PRINT_STRING
        POP     DS                      ;Balance up stack
        JMP     IBM_BIOS
```

```
;----------- IBM Menu CMD to check HEX display / LBA selection on IDE Board.
;           Should show High Cylinder, Low Cylinder and Sector # in Hex Display on IDE Board

LBA_DISPLAY_TEST:
       MOV    BX,LBA_TEST_MSG              ;Test LBA on Drive #2
       CALL   PRINT_STRING

       CALL   SET_DRIVE_B          ;Select the second Drive/CF card
       CALL   IDEinit              ;Initialize drive 1. If there is no drive abort
       JZ     LBA_002

       MOV    BX,DRIVE2_ERR        ;Warn second IDE drive did not initilize
       CALL   PRINT_STRING
       POP    DS
       JMP    IBM_BIOS

LBA_002:
       CALL   CRLF
       MOV    DH,11100000B         ;<<<< Set to LBA mode, head 0
       MOV    DL,REGshd            ;Send "Head #" (in DH)
       CALL   IDEwr8D              ;Write to 8255 A Register

       MOV    BX,TRKH_NUM          ;Enter High byte track number
       CALL   PRINT_STRING
       CALL   GET2DIGITS           ;Get 8 bit value (2 digits) to AL. (BX, CX & DX Unchanged)
       MOV    DH,AL
       MOV    DL,REGcylinderMSB
       CALL   IDEwr8D              ;Send High TRK# (in DH)
       CALL   IDEwr8D_X            ;Special write to 8255 B Register (Not A) to update LED HEX Display
                                   ;High 8 bits ignored by IDE drive

       MOV    BX,TRKL_NUM          ;"Low Track number,(xxH)"
       CALL   PRINT_STRING
       CALL   GET2DIGITS           ;Get 8 bit value (2 digits) to AL. (BX, CX & DX Unchanged)
       MOV    DH,AL                ;Get low Track #
       MOV    DL,REGcylinderLSB    ;Send Low TRK# (in DH)
       CALL   IDEwr8D              ;Special write to 8255 A

       MOV    BX,SECTOR_NUM        ;"Sector number,(xxH) = "
       CALL   PRINT_STRING
       CALL   GET2DIGITS           ;Get 8 bit value (2 digits) to AL. (BX, CX & DX Unchanged)
       MOV    DH,AL                ;Sector 1, Bits 0-5 only  (currently 1-17)
       MOV    DL,REGsector         ;Send "Sector#"
       CALL   IDEwr8D              ;Write to 8255 A Register

       MOV    AL,READcfg8255       ;Set 8255 back to read mode
       OUT    IDECtrlPort,AL
       MOV    BX,CHECK_DISPLAY_MSG ;Check display
       CALL   PRINT_STRING
       RET                         ;We arive here from IDE Menu, return


;----------- IBM Menu CMD to check HEX display CHS selection on IDE Board.
;           Should show Cylinder, Head and Sector # in Hex Display on IDE Board


CHS_DISPLAY_TEST:
       MOV    BX,CHS_TEST_MSG              ;Test CHS on Drive #2
       CALL   PRINT_STRING

       CALL   SET_DRIVE_B          ;Select the second Drive/CF card
       CALL   IDEinit              ;Initialize drive 1. If there is no drive abort
       JZ     CHS_002

       MOV    BX,DRIVE2_ERR        ;Warn second IDE drive did not initilize
       CALL   PRINT_STRING
       POP    DS
       JMP    IBM_BIOS
```

```
CHS_002:
        CALL    CRLF
        OR      DH,10100000B            ;Set to >>>>> NON-LBA mode <<<<<
        MOV     DL,REGshd               ;Send "Head #" (in DH)
        CALL    IDEwr8D                 ;Write to 8255 A Register

        MOV     BX,TRKH_NUM             ;"Cylinder number High,(xxH)
        CALL    PRINT_STRING
        CALL    GET2DIGITS              ;Get 8 bit value (2 digits) to AL. (BX, CX & DX Unchanged)
        AND     AL,00000011B            ;Only 2 bits accepted
        MOV     DH,AL
        PUSH    AX                      ;Save for below
        MOV     DL,REGcylinderMSB
        CALL    IDEwr8D                 ;Send High TRK# (in DH) to IDE Drive

        MOV     BX,HEAD_NUM             ;Enter Head number (0-FH)
        CALL    PRINT_STRING
        CALL    GET2DIGITS              ;Get 8 bit value (2 digits) to AL. (BX, CX & DX Unchanged)
        MOV     DH,AL
        AND     DH,0FH                  ;top two LED HEX displays.
        SHL     DH,1                    ;These 8 (high) data lines are ignored by the IDE drive
        SHL     DH,1
        SHL     DH,1
        SHL     DH,1
        POP     AX                      ;Get the tow bits of the high cylinder
        OR      DH,AL
        MOV     DL,REGcylinderMSB       ;of the high cylinder in the low nibble.
        CALL    IDEwr8D_X               ;Special output to 8255 B Register (Not A) to update LED HEX Display ONLY

        MOV     BX,TRKL_NUM             ;"Low Cylinder number,(xxH)"
        CALL    PRINT_STRING
        CALL    GET2DIGITS              ;Get 8 bit value (2 digits) to AL. (BX, CX & DX Unchanged)
        MOV     DH,AL                   ;Get low Track #
        MOV     DL,REGcylinderLSB       ;Send Low TRK# (in DH)
        CALL    IDEwr8D                 ;Special write to 8255 A

        MOV     BX,SECTOR_NUM           ;"Sector number,(xxH) = "
        CALL    PRINT_STRING
        CALL    GET2DIGITS              ;Get 8 bit value (2 digits) to AL. (BX, CX & DX Unchanged)
        MOV     DH,AL                   ;Sector 1, Bits 0-5 only  (currently 1-17)
        MOV     DL,REGsector            ;Send "Sector#"
        CALL    IDEwr8D                 ;Write to 8255 A Register

        MOV     AL,READcfg8255          ;Set 8255 back to read mode
        OUT     IDECtrlPort,AL
        MOV     BX,CHECK_DISPLAY_MSG    ;Check display
        CALL    PRINT_STRING
        JMP     IBM_BIOS                ;Will return back up to start IBM Menu



;--------------  Menu command to dump a floppy BOOT sector on the CRT
;               Note must have a functional INT 13H routine for this section to work

DUMP_B_SEC:
        PUSH    DS                      ;Save Monitor current DS
         XOR    AX,AX                   ;Set DS to data area for ROM usage in low RAM (400H))
        MOV     DS,AX

        MOV     BX,BOOT_3RD_MSG                 ;Say Reading Boot sector
        CALL    PRINT_STRING

        MOV     AL,0H                           ;Flag to indicate ZFDC board is NOT Initilized
        MOV     [ZFDC_INIT_FLAG],AL             ;DS is already set for low RAM area

        CALL    INIT_ZFDC                       ;Initilize the ZFDC board hardware
```

```
        CMP     byte [ZFDC_INIT_FLAG],0FFH    ;Is Board initilized correctly
        JZ      BS_ZFDC_3OK1

        MOV     BX,ZFDC_FAIL_MSG
        CALL    PRINT_STRING
        pop     ds                    ;Balance up Monitor stack
        JMP     IBM_BIOS

BS_ZFDC_3OK1:
        MOV     BX,DRIVE_SELECT_MSG   ;Floppy disk A: or B:
        CALL    PRINT_STRING

        call    CICO                  ;Get a command from Console
        PUSH    AX
        MOV     BX,CRLFMSG            ;"CR,LF"
        CALL    PRINT_STRING
        POP     AX
        CMP     AL,'B'
        JZ      B_DRIVE_SEL
        MOV     DX,0000H              ;Side A, Disk 0
        JMP     OVER_DRIVE_SEL
B_DRIVE_SEL:
        MOV     DX,00001H             ;Side A, Disk 1
OVER_DRIVE_SEL:
        PUSH    DX                    ;Save side for below

        mov     ah,0H
        int     13h                   ;AH=0, reset floppy disk system

        JNC     BS_RESET_3OK1
        MOV     BX,RESET_FAIL_MSG
        CALL    PRINT_STRING
        pop     dx
        pop     ds                    ;Balance up stack
        JMP     IBM_BIOS


BS_RESET_3OK1:
        SUB     AX,AX
        MOV     ES,AX
        MOV     BX,500H               ;Will always dump data to 0000:500H

        POP     DX                    ;Side & Drive 0
        MOV     CX,0001               ;1st sector on track 0
        MOV     AL,1                  ;1 sector
        mov     ah,02h                ;Read 1 sector

        int     13H

        JNC     BS_SEQ_3OK1           ;If NC then no errors
        MOV     BX,BOOT_INFO_FAIL_MSG
        pop     ds                    ;Balance up Monitor stack
        JMP     IBM_BIOS              ;Will return back up to start of Monitor

BS_SEQ_3OK1:
        MOV     BX,BOOT_INFOOKMSG
        CALL    PRINT_STRING

        SUB     AX,AX
        MOV     DS,AX
        MOV     SI,500H               ;Will always dump data to 0000:500H

        LODSB                         ;WRITE 1 BYTE BYTE, DS:[SI++] -> AL
        CALL    AL_HEXOUT
        LODSB
        CALL    AL_HEXOUT
        LODSB
        CALL    AL_HEXOUT
        MOV     BX,JMP_MSG            ;"  BOOT JUMP VECTOR"
```

```
        CALL    PRINT_STRING

        MOV     DL,8
BS_1:   LODSB                           ;Get a byte from RAM, DS:[SI++] -> AL
        MOV     CL,AL
        CALL    CO
        DEC     DL
        JNZ     BS_1
        MOV     BX,NAME_MSG             ;"   OEM NAME"
        CALL    PRINT_STRING

        LODSW
        CALL    AX_HEXOUT
        MOV     BX,BYTES_MSG            ;"   Bytes/Sec"
        CALL    PRINT_STRING
        LODSB
        CALL    AL_HEXOUT
        MOV     BX,CLUSTER_MSG          ;"   Sec/Cluster"
        CALL    PRINT_STRING
        LODSW
        CALL    AX_HEXOUT
        MOV     BX,RES_MSG              ;"   Reserved Sectors"
        CALL    PRINT_STRING
        LODSB
        CALL    AL_HEXOUT
        MOV     BX,FATS_MSG             ;"   FATS"
        CALL    PRINT_STRING
        LODSW
        CALL    AX_HEXOUT
        MOV     BX,ROOT_MSG             ;"   Root Dir Entries"
        CALL    PRINT_STRING
        LODSW
        CALL    AX_HEXOUT
        MOV     BX,SECTORS_MSG          ;"   Sectors"
        CALL    PRINT_STRING
        LODSB
        CALL    AL_HEXOUT
        MOV     BX,MEDIA_MSG            ;"   Media Byte"
        CALL    PRINT_STRING
        LODSW
        CALL    AX_HEXOUT
        MOV     BX,FAT_SEC_MSG          ;"   FAT Sectors"
        CALL    PRINT_STRING
        LODSW
        CALL    AX_HEXOUT
        MOV     BX,SEC_TRK_MSG          ;"   Sectors/Track"
        CALL    PRINT_STRING
        LODSW
        CALL    AX_HEXOUT
        MOV     BX,HEADS_MSG            ;"   Heads"
        CALL    PRINT_STRING
        LODSW
        CALL    AX_HEXOUT
        LODSW
        CALL    AX_HEXOUT
        MOV     BX,HIDDEN_MSG           ;"   Hidden Sectors"
        CALL    PRINT_STRING
        LODSW
        CALL    AX_HEXOUT
        LODSW
        CALL    AX_HEXOUT
        MOV     BX,HUGE_MSG             ;"   Huge Sectors"
        CALL    PRINT_STRING
        LODSB
        CALL    AL_HEXOUT
        MOV     BX,DRIVE_NO_MSG             ;"   Drive #"
        CALL    PRINT_STRING
        LODSB
        CALL    AL_HEXOUT
```

```
        MOV     BX,RESERVED_MSG              ;"   Reserved"
        CALL    PRINT_STRING
        LODSB
        CALL    AL_HEXOUT
        MOV     BX,BOOT_SIG_MSG              ;"   Boot Signature"
        CALL    PRINT_STRING
        LODSW
        CALL    AX_HEXOUT
        LODSW
        CALL    AX_HEXOUT
        MOV     BX,VOL_ID_MSG          ;"   Volunme ID"
        CALL    PRINT_STRING

        MOV     DL,11
BS_2:   LODSB                          ;Get a byte from RAM, DS:[SI++] -> AL
        MOV     CL,AL
        CALL    CO
        DEC     DL
        JNZ     BS_2
        MOV     BX,VOLUME_MSG          ;"   Volume Label"
        CALL    PRINT_STRING

        MOV     DL,8
BS_3:   LODSB                          ;Get a byte from RAM, DS:[SI++] -> AL
        CALL    AL_HEXOUT
        DEC     DL
        JNZ     BS_3
        MOV     BX,SYS_TYPE_MSG              ;"   File Sys Type"
        CALL    PRINT_STRING

        pop     ds                     ;Balance up Monitor DS from stack
        JMP     IBM_BIOS               ;All done


;
;--------------  Menu command to dump the Hard Disk MBR (Master Boot Record) Info on the CRT
;               Note must have a functional INT 13H routine for this section to work

DUMP_MBR:
        PUSH    DS                     ;Save Monitor current DS
         XOR    AX,AX                  ;Set DS to data area for ROM usage in low RAM (400H))
        MOV     DS,AX

        MOV     BX,BOOT_MBR_MSG               ;Say Reading MBR sector
        CALL    PRINT_STRING

        CALL    SET_DRIVE_B            ;Select the second Drive/CF card
        CALL    IDEinit                ;Initialize drive 1. If there is no drive abort
        JZ      MBR_002

        MOV     BX,DRIVE2_ERR          ;Warn second IDE drive did not initilize
        CALL    PRINT_STRING
        POP     DS
        JMP     IBM_BIOS

MBR_002:
        SUB     AX,AX
        MOV     ES,AX
        MOV     BX,500H                ;Will always dump data to 0000:500H

        MOV     DX,0080H               ;Head 0, HDidk 0
        MOV     CX,0001                ;1st sector on track 0
        MOV     AL,1                   ;read 1 sector
        mov      ah,02h                ;Read 1 sector

         int     13H

        JNC     MBR_003                ;If NC then no errors
        MOV     BX,BOOT_MBR_FAIL_MSG
```

```
        pop     ds                      ;Balance up Monitor stack
        JMP     IBM_BIOS                ;Will return back up to start of Monitor

MBR_003:
        MOV     BX,MBR_INFOOKMSG
        CALL    PRINT_STRING

        SUB     AX,AX
        MOV     DS,AX
        MOV     SI,500H + 1B8H          ;Will always dump data to 0000:500H

        LODSB                           ;WRITE 1 BYTE BYTE, DS:[SI++] -> AL
        CALL    AL_HEXOUT
        LODSB
        CALL    AL_HEXOUT
        LODSB
        CALL    AL_HEXOUT
        LODSB
        CALL    AL_HEXOUT
        MOV     BX,DISK_SIG_MSG                 ;"  Disk Signature (Optional)"
        CALL    PRINT_STRING

        LODSB
        CALL    AL_HEXOUT
        LODSB
        CALL    AL_HEXOUT
        MOV     BX,NULS_MSG             ;"  Usually Nulls (Optional)"
        CALL    PRINT_STRING

        LODSB                           ;0
        CALL    AL_HEXOUT
        MOV     BX,STATUS_MSG           ;"            Status Byte"
        CALL    PRINT_STRING
        MOV     BX,PT1_MSG              ;"First Partition Table "
        CALL    PRINT_STRING
        CALL    DUMP_PTBL

        LODSB                           ;0
        CALL    AL_HEXOUT
        MOV     BX,STATUS_MSG           ;"            Status Byte"
        CALL    PRINT_STRING
        MOV     BX,PT2_MSG              ;"Second Partition Table "
        CALL    PRINT_STRING
        CALL    DUMP_PTBL

        LODSB                           ;0
        CALL    AL_HEXOUT
        MOV     BX,STATUS_MSG           ;"            Status Byte"
        CALL    PRINT_STRING
        MOV     BX,PT3_MSG              ;"Third Partition Table "
        CALL    PRINT_STRING
        CALL    DUMP_PTBL

        LODSB                           ;0
        CALL    AL_HEXOUT
        MOV     BX,STATUS_MSG           ;"            Status Byte"
        CALL    PRINT_STRING
        MOV     BX,PT4_MSG              ;"Forth Partition Table "
        CALL    PRINT_STRING
        CALL    DUMP_PTBL

        LODSW
        CALL    AX_HEXOUT
        MOV     BX,SIGNATURE_MSG        ;"  LBR Signature Word "
        CALL    PRINT_STRING

        pop     ds                      ;Balance up Monitor DS from stack
        JMP     IBM_BIOS                ;All done
```

```
DUMP_PTBL:
        LODSB                           ;1-3
        CALL    AL_HEXOUT
        LODSB
        CALL    AL_HEXOUT
        LODSB
        CALL    AL_HEXOUT
        MOV     BX,STLBA_MSG            ;"            Start CHS Address"
        CALL    PRINT_STRING

        LODSB                           ;4
        CALL    AL_HEXOUT
        MOV     BX,PAR_TYPE_MSG                 ;"            Partition Type"
        CALL    PRINT_STRING

        LODSB                           ;5-7
        CALL    AL_HEXOUT
        LODSB
        CALL    AL_HEXOUT
        LODSB
        CALL    AL_HEXOUT
        MOV     BX,ECHS_MSG             ;"            End CHS Address"
        CALL    PRINT_STRING

        LODSW                           ;8-B
        CALL    AX_HEXOUT
        LODSW
        CALL    AX_HEXOUT
        MOV     BX,SLB_MSG              ;"            Start LBA Address"
        CALL    PRINT_STRING

        LODSW                           ;C-F
        CALL    AX_HEXOUT
        LODSW
        CALL    AX_HEXOUT
        MOV     BX,ELBA_MSG             ;"            End LBA Address"
        CALL    PRINT_STRING
        RET




;------------ Menu CMD to turn on/off sector dump info for reads/writes using this BIOS
;       DEBUG_FLAG = 0 if no debugging info sent to serial terminal
;       DEBUG_FLAG = 1 send just INT's info
;       DEBUG_FLAG = 2 send more detailed information

DEBUG_ON_OFF:
        push    ds
         XOR    AX,AX                   ;Set DS to data area for ROM usage in low RAM @ 400H....)
         MOV    DS,AX

        MOV     BX,DEBUG_SET_MSG
        CALL    PRINT_STRING
        call    CICO                    ;Look for 0,1 2 (only)
        CMP     AL,'1'
        MOV     byte [DEBUG_FLAG],01H
        MOV     BX,DUMP_ON1_MSG
        JZ      DUMP_DONE

        CMP     AL,'2'
        MOV     byte [DEBUG_FLAG],02H
        MOV     BX,DUMP_ON2_MSG
        JZ      DUMP_DONE

        CMP     AL,'3'
        MOV     byte [DEBUG_FLAG],03H
```

```
        MOV     BX,DUMP_ON3_MSG
        JZ      DUMP_DONE

        MOV     byte [DEBUG_FLAG],0
        MOV     BX,DUMP_OFF_MSG

DUMP_DONE:
        CALL    PRINT_STRING
        POP     DS
        RET


;******************************************************************************
;
;       Bootstrap Handler       (IBM-PC Software Interrupt 19H)
;
;               SYSTEM - BOOTSTRAP LOADER
;
;       For a floppy the BIOS will try to read sector 1, head 0, track 0 from drive A:
;       to 0000h:7C00h.  If this fails we will just abort.

;       For the IDE/CF Cards the BIOS will try to read sector 1, head 0, track 0 from
;       drive #2 of the IDE Board to 0000h:7C00h.  If this fails we will just abort.

;
;       For a hard disk, the BIOS will read sector 1, head 0, track 0 of the 2nd CF-Card
;       on the Dual IDE board. This sector should contain a master bootstrap loader and
;       a partition table (see http://www.ctyme.com/intr/rb-2270.htm#Table650).
;
;       After loading the master boot sector at 0000h:7C00h,
;       the master bootstrap loader is given control with:-
;
;                       CS:IP = 0000h:7C00h.
;                       DH = access bits 7-6,4-0: Don't care
;                       bit 5:=0 device supported by INT 13.
;                       DL = boot drive
;                               00h first floppy
;                               80h first hard disk
;
;       True IBM PCs and most clones issue an INT 18 (cassette) if neither floppy nor hard
;       disk have a valid boot sector. We will just abort.
;
;       To accomplish a warm boot equivalent to Ctrl-Alt-Del, store 1234h in
;       0040h:0072h and jump to FFFFh:0000h.  For a cold boot equivalent to
;       a reset, store 0000h at 0040h:0072h before jumping..
;
;       BUG: When loading the remainder of the DOS system files fails, various versions
;       of IBMBIO.COM/IO.SYS incorrectly restore INT 1E before calling INT 19, assuming
;       that the boot sector had stored the contents of INT 1E at DS:SI instead of on
;       the stack as it actually does


;******************************************************************************

BOOT_DOS_INT:
        STI                             ;Bootstrap Handler (Interrupt 19H)
        TEST    DL,80H                  ;Floppy or HD?
        JNZ     BOOT_HDISK


                                        ;<<< BOOT FLOPPY.  ZFDC Board MUST be active (IDE may be offline)

        PUSH    DS                      ;Save current DS on stack
        XOR     AX,AX                   ;Set DS to data area for ROM usage in low RAM @ 400H....)
        MOV     DS,AX

        MOV     AL,0H                           ;Flag to indicate ZFDC board is NOT Initilized
        MOV     [ZFDC_INIT_FLAG],AL
```

```
        CALL    INIT_ZFDC                       ;Initilize the ZFDC board hardware (360K & 1.44M disks)

        CMP     byte [ZFDC_INIT_FLAG],0FFH      ;Is Board initilized correctly
        POP     DS                              ;Balance up stack
        JZ      ZFDC_OK

        MOV     BX,ZFDC_FAIL_MSG
        CALL    PRINT_STRING
        JMP     IBM_BIOS                        ;Return will drop back to IBM_BIOS location

ZFDC_OK:
        CALL    SET_DRIVE_B             ;Select the second Drive/CF card
        CALL    IDEinit                 ;Initialize drive 1. If there is no drive abort
        JZ      FH_RESET_OK

        MOV     BX,DRIVE2_ERR           ;Warn second IDE drive did not initilize
        CALL    PRINT_STRING            ;Continue anyway with ZFDC/Floppy

FH_RESET_OK:
        sub     ax,ax
        mov     ds,ax                   ;DS -> 0
        mov     dx,0080H                ;DL = 80L  will always boot from IDE #2 disk for now.

        mov     ah,0
        int     13h                     ;AH=0, reset floppy disk system

        JNC     F_RESET_OK
        MOV     BX,RESET_FAIL_MSG
        CALL    PRINT_STRING
        JMP     IBM_BIOS                ;Will return back up to IBM_BIOS location

F_RESET_OK:
        XOR     AX,AX
        mov     DS,AX
        mov     ES,AX                   ;DS = ES = 0000H
        mov     ax,201h                         ;read one sector
        mov     bx,DOS_BOOT_LOC                 ;set ES:BX to data destination 7C00H (BB,00,7c)
        mov     cx,0001H                ;Track 0, sec 01
        mov     dx,0000H                ;side A,  (DL bit 7 = 0) drive 0,

        int     13H                     ;AH=2, CX=1, read 1 (the boot), sector

        JNC     F_BOOT_OK               ;If NC, then no errors
        MOV     BX,BOOT_FAIL_MSG
        CALL    PRINT_STRING
        JMP     IBM_BIOS                ;Will return back up to IBM_BIOS location

F_BOOT_OK:
        XOR     AX,AX
        MOV     DS,AX
        CMP     word [DOS_BOOT_SIGNATURE],0AA55H     ;Check we have a valid MBL signature
        JZ      F_BOOT_OK1

        MOV     BX,NO_MBL_MSG           ;No Floppy Boot Loader Signature detected
        CALL    PRINT_STRING
        JMP     IBM_BIOS                ;Will return back up to IBM_BIOS location

F_BOOT_OK1:
        MOV     BX,BOOT_OK_MSG
        CALL    PRINT_STRING

;       Call    CI                      ;Wait for CRT input for boot debugging  (info at 7C00H)

        MOV     DX,0                    ;Required see above
        JMP     word  0000H:DOS_BOOT_LOC  ;Far Jump, execute the boot code @0:7C00H


                                        ;<<< BOOT HDISK .  IDE  Board MUST be active (ZFDC board may be offline)
BOOT_HDISK:                             ;Boot MSDOS (or FreeDOS) from IDE/CF Card
```

```
        CALL    SET_DRIVE_B           ;Select the second Drive/CF card
        CALL    IDEinit               ;Initialize drive 1. If there is no drive abort
        JZ      BOOT_RESET_OK

        MOV     BX,DRIVE2_ERR         ;Warn second IDE drive did not initilize
        CALL    PRINT_STRING
        JMP     IBM_BIOS              ;Will return back up to start of Monitor

BOOT_RESET_OK:
        PUSH    DS
         XOR    AX,AX                 ;Set DS to data area for ROM usage in low RAM @ 400H....)
         MOV    DS,AX

        MOV     AL,0H                         ;Flag to indicate ZFDC board is NOT Initilized
        MOV     [ZFDC_INIT_FLAG],AL

        CALL    INIT_ZFDC                     ;Initilize the ZFDC board hardware (360K & 1.44M disks)

        CMP     byte [ZFDC_INIT_FLAG],0FFH    ;Is Board initilized correctly
        POP     DS                            ;Balance up stack
        JZ      BOOT_ZFDC_OK

        MOV     BX,ZFDC_FAIL_MSG
        CALL    PRINT_STRING                  ;Continue anyway with IDE Drive

BOOT_ZFDC_OK:
        sub     ax,ax
         mov     ds,ax                ;DS -> 0
        mov     dx,0080H              ;DL = 80H  will always boot from HDisk #2

         mov     ah,0
         int     13H                  ;AH=0, reset Hard Disk system

        JNC     HBOOT_RESET_OK
        MOV     BX,RESET_FAIL_MSG
        CALL    PRINT_STRING
        JMP     IBM_BIOS              ;Will return back up to IBM_BIOS location

HBOOT_RESET_OK:
        XOR     AX,AX
         mov     DS,AX
        mov     ES,AX                 ;DS = ES = 0000H
         mov     ax,201h                      ;read one sector
         mov     bx,DOS_BOOT_LOC              ;set ES:BX to data destination 7C00H (BB,00,7c)
         mov     cx,0001H             ;Track 0, sec 01 for MBL >>> Boot on Sector 12H <<<
         mov    dx,0080H              ;head 0, HDisk 0, (DL bit 7 = 1)

         int     13H                  ;AH=2, CX=1, read 1 (the boot), sector

        JNC     HDOS_BOOT_OK          ;If NC, then no errors
        MOV     BX,BOOT_FAIL_MSG
        CALL    PRINT_STRING
        JMP     IBM_BIOS              ;Will return back up to IBM_BIOS location

HDOS_BOOT_OK:
        MOV     BX,BOOT_OK_MSG
        CALL    PRINT_STRING

;       Call    CI                    ;Wait for CRT input for boot debugging
                                      ;(Can reset and look at BOOT sector)

        MOV     DX,0080H              ;Required see above
        JMP      word  0000H:DOS_BOOT_LOC  ;Far Jump, execute the boot code @0:7C00H


;*********************************************************************************************
;
;       Disk I/O Handler        (Software Interrupt 13H & 40H)
;
```

```
;Input: AH = 00h       DISK - RESET DISK SYSTEM
;       DL = drive (if bit 7 is set, both hard disks and floppy disks are reset)
;Return:AH = status (see below)
;       CF clear if successful (returned AH=00h)
;       CF set on error
;
;
;Input: AH = 01h       DISK - GET STATUS OF LAST OPERATION
;       DL = drive (bit 7 set for hard disk)
;Return:CF clear if successful (returned status 00h)
;       CF set on error
;       AH = status of previous operation (see below)
;
;
;Input: AH = 02h       READ  03H,WRITE SECTOR DATA
;       AL = number of sectors to read (must be nonzero)
;       CH = low eight bits of cylinder number
;       CL = sector number 1-63 (bits 0-5, high two bits of cylinder (bits 6-7, hard disk only)
;       DH = head number
;       DL = drive number (bit 7 set, for hard disk)
;       ES:BX -> data buffer
;
;Return:CF set on error
;       if AH = 11h (corrected ECC error), AL = burst length
;       CF clear if successful
;       AH = status (see below)
;       AL = number of sectors transferred (only valid if CF set for some BIOSes)
;
;
;Input: AH = 04h       DISK - VERIFY DISK SECTOR(S)
;       AL = number of sectors to verify (must be nonzero)
;       CH = low eight bits of cylinder number
;       CL = sector number 1-63 (bits 0-5) high two bits of cylinder (bits 6-7, hard disk only)
;       DH = head number
;       DL = drive number (bit 7 set, for hard disk)
;       ES:BX -> data buffer (PC,XT,AT with BIOS prior to 1985/11/15)
;
;Return:CF set on error
;       CF clear if successful
;       AH = status (see below)
;       AL = number of sectors verified
;
;
;Input: AH = 05h       FLOPPY - FORMAT TRACK
;       AL = number of sectors to format
;       CH = track number
;       DH = head number
;       DL = drive number
;       ES:BX -> address field buffer:-
;                     00h    BYTE    track number
;                     01h    BYTE    head number (0-based)
;                     02h    BYTE    sector number
;                     03h    BYTE    sector size (00h=128 bytes, 01h=256 bytes, 02h=512, 03h=1024)
;Return:CF set on error
;       CF clear if successful
;       AH = status (see below)
;                     Note: On AT or higher, call AH=17h first. The number of sectors per track is read from the
diskette
;                     parameter table pointed at by INT 1E
;
;Input: AH = 08h       RETURN DRIVE PARAMATERS
;       DL = drive number (bit 7 set, for hard disk)
;       ES:DI = 0000H:0000H
;                     Note: For systems predating the IBM AT, this call is only valid for hard disks, as it is
implemented
;                     by the hard disk BIOS rather than the ROM BIOS. The IBM ROM-BIOS returns the total number of
hard disks
;                     attached to the system regardless of whether DL >= 80h on entry
```

```
;Return:CF set on error
;       CF clear if successful
;       AH = status (see below)
;       AL = 00h on at least some BIOSes
;       BL = drive type (AT/PS2 floppies only)
;                       Values for diskette drive type:
;                       01h    360K
;                       02h    1.2M
;                       03h    720K
;                       04h    1.44M
;       CH = low eight bits of maximum cylinder number
;       CL = maximum sector number (bits 5-0)
;            high two bits of maximum cylinder number (bits 7-6)
;       DH = maximum head number
;       DL = number of drives
;       ES:DI -> drive parameter table (floppies only)

;Input: AH = 15h       GET DISK TYPE
;       DL = drive number (bit 7 set, for hard disk)

;Return:CF set on error
;       CF clear if successful
;       AH = type code (see below)
;               00h no such drive
;               01h floppy without change-line support
;               02h floppy (or other removable drive) with change-line support
;               03h hard disk
;                   CX:DX = number of 512-byte sectors

;RETURNED ERROR CODES IN AH:-
;       00h    successful completion
;       01h    invalid function in AH or invalid parameter
;       02h    address mark not found
;       03h    disk write-protected
;       04h    sector not found/read error
;       05h    reset failed (hard disk)
;       05h    data did not verify correctly (TI Professional PC)
;       06h    disk changed (floppy)
;       07h    drive parameter activity failed (hard disk)
;       08h    DMA overrun
;       09h    data boundary error (attempted DMA across 64K boundary or >80h sectors)
;       0Ah    bad sector detected (hard disk)
;       0Bh    bad track detected (hard disk)
;       0Ch    unsupported track or invalid media
;       0Dh    invalid number of sectors on format (PS/2 hard disk)
;       0Eh    control data address mark detected (hard disk)
;       0Fh    DMA arbitration level out of range (hard disk)
;       10h    uncorrectable CRC or ECC error on read
;       11h    data ECC corrected (hard disk)
;       20h    controller failure
;       31h    no media in drive (IBM/MS INT 13 extensions)
;       32h    incorrect drive type stored in CMOS (Compaq)
;       40h    seek failed
;       80h    timeout (not ready)
;       AAh    drive not ready (hard disk)
;       B0h    volume not locked in drive (INT 13 extensions)
;       B1h    volume locked in drive (INT 13 extensions)
;       B2h    volume not removable (INT 13 extensions)
;       B3h    volume in use (INT 13 extensions)
;       B4h    lock count exceeded (INT 13 extensions)
;       B5h    valid eject request failed (INT 13 extensions)
;       B6h    volume present but read protected (INT 13 extensions)
;       BBh    undefined error (hard disk)
;       CCh    write fault (hard disk)
;       E0h    status register error (hard disk)
;       FFh    sense operation failed (hard disk)

;*********************************************************************************************
```

```
OLD_DISKIO:                             ;Come here via INT 40H, (rearly) for the old Floppy Disk relocated INTs
       STI                              ;Normal INT 13H Entry point
       PUSH    DS                       ;For all commands use variables in low RAM if needed
       PUSH    AX
       XOR     AX,AX                    ;Set DS to data area for ROM usage in low RAM @ 400H....)
        MOV    DS,AX
       POP     AX

       CMP     byte [DEBUG_FLAG],0      ;Is Floppy Debug mode on
       JZ      COMMON_DISK_COMMANDS     ;If not skip to "normal" FDisk routines

       PUSH    AX
       PUSH    BX
       PUSH    CX
       MOV     BX,INT_40F_MSG           ;"Int 40H (<--Floppy) AX="
       CALL    SERIAL_PRINT_STRING
       POP     CX
       POP     BX
       POP     AX
       CALL    SERIAL_DISPLAY_REGISTERS ;Display Registers on serial port display (All registers retained)
       JMP     COMMON_DISK_COMMANDS     ;Go to "normal" Disk routines

DISKIO:                                 ;Normal INT 13H Entry point
       STI
       PUSH    DS                       ;For all commands use variables in low RAM if needed
       PUSH    AX
        XOR    AX,AX                    ;Set DS to data area for ROM usage in low RAM @ 400H....)
        MOV    DS,AX
       POP     AX

       CMP     byte [DEBUG_FLAG],0      ;Is Debug mode on
       JZ      COMMON_DISK_COMMANDS     ;If not skip

       PUSH    AX
       PUSH    BX
       PUSH    CX
       TEST    DL,80H                   ;Floppy or HDisk
       JNZ     DISKIO1
       MOV     BX,INT_13F_MSG           ;"Int 13H (Floppy) AX="
       JMP     DISKIO2
DISKIO1:MOV    BX,INT_13H_MSG           ;"Int 13H (** HDisk **) AX="
DISKIO2:CALL   SERIAL_PRINT_STRING
       POP     CX
       POP     BX
       POP     AX
       CALL    SERIAL_DISPLAY_REGISTERS ;Display Registers on serial port display (All registers retained)
                                        ;Fall through to COMMON_DISK_COMMANDS
COMMON_DISK_COMMANDS:
       TEST    DL,80H                   ;HDisk or Floppy CMD
        JZ     FD_COMMANDS              ;For Floppy disk commands
       JMP     HD_COMMANDS              ;For HDISK Commands


;----------------------- Floppy Disk Commands -----------------------------------------

FD_COMMANDS:
       TEST    AH,AH                    ;Is it a FDisk reset
       JNZ     N_FDISK_RESET
       JMP     FDISK_RESET
N_FDISK_RESET:
       CMP     AH,1                     ;Is it a FDisk status request
       JNZ     N_FDISK_STATUS
       JMP     FDISK_STATUS
N_FDISK_STATUS:
       CMP     AH,2                     ;Is it a FDisk read request
       JNZ     N_FDISK_READ
       MOV     byte [VERIFY_FLAG],0H    ;Turn off verify flag
       JMP     FDISK_READ
N_FDISK_READ:
```

```
        CMP     AH,3                            ;Is it a FDisk write request
        JNZ     N_FDISK_WRITE
        JMP     FDISK_WRITE
N_FDISK_WRITE:
        CMP     AH,4                            ;Is it a FDisk Verify request
        JNZ     N_FDISK_VERIFY
        MOV     byte [VERIFY_FLAG],0ffH             ;Turn on verify flag
        JMP     FDISK_READ                      ;Modified read
N_FDISK_VERIFY:
        CMP     AH,5                            ;Is it a FDisk format request
        JNZ     N_FDISK_FORMAT
        JMP     FDISK_FORMAT
N_FDISK_FORMAT:
        CMP     AH,8                            ;Is it a FDisk paramaters request
        JNZ     N_FDISK_PARAMS
        JMP     FDISK_PARAMS
N_FDISK_PARAMS:
        CMP     AH,15H                          ;GET DISK TYPE (XT 1986/1/10 or later,XT286,AT,PS)
        JNZ     N_FDISK_DASB
        JMP     FDISK_DASB
N_FDISK_DASB:
        CMP     AH,16H                          ;FDisk media change check request
        JNZ     NOT_VALID_DISK
        JMP     FDISK_MEDIA_CHANGE

NOT_VALID_DISK:                                     ;Thats all for now
        PUSH    AX
        MOV     BX,INVALID_AH_FMSG
        CALL    PRINT_STRING
        POP     AX
        MOV     AL,AH
        CALL    AL_HEXOUT
        MOV     BX,H_MSG_CRLF
        CALL    PRINT_STRING
        mov     byte [IBM_DISK_STATUS],cmderr ;Show bad command
                                            ;Fall through to DONE_DISK

DONE_DISK:                                  ;Most (but not all), floppy commands come back here before returning to
DOS
        mov     ah,[IBM_DISK_STATUS]
        OR      AH,AH                       ;Was there an error
        JZ      ALL_OK
        STC                                 ;Set carry to indicate an error
 ALL_OK:
        POP     ds                          ;Get back the origional saved DS at start
        RETF    2                           ;Remove the origional status flags on return (remember we got here via an
INT)


;---------------------- Floppy Disk Routines ---------------------------------------

FDISK_RESET:                                ;Home the disk head etc.
        PUSH    BX                          ;Save ALL
        PUSH    CX
        PUSH    DX
        MOV     [CURRENT_DRIVE],DL

        MOV     CL,CMD_SET_DRIVE            ;Set Drive Drive, ZFDC will just return if current drive
        CALL    S100OUT
        MOV     CL,[CURRENT_DRIVE]
        OR      CL,CL                       ;DL = 0 --> ZFDC Drive #3.  DL = 1 -->ZFDC Drive #2
        MOV     CL,3                        ;Default to Drive #3
        JZ      R_FLOPPY
        MOV     CL,2                        ;Drive #2
R_FLOPPY:
        CALL    S100OUT
        CALL    WAIT_FOR_ACK                ;Return Z (or NZ with error # in [AH])
        JNZ     FDISK_RESET_ERROR
```

```
        MOV     CL,CMD_SET_HOME                     ;Home the heads of the current drive
        CALL    S100OUT
        CALL    WAIT_FOR_ACK                ;Return Z (or NZ with error # in [AH])
        JNZ     FDISK_RESET_ERROR
        mov     byte [SEEK_STATUS],0        ;show good seek status
        mov     byte [IBM_DISK_STATUS],0    ;and good disk status
        POP     DX
        POP     CX
        POP     BX
        JMP     DONE_DISK                   ;and return

FDISK_RESET_ERROR:
        MOV     BX,HOME_ERR_MSG
        CALL    PRINT_STRING
        mov     byte [IBM_DISK_STATUS],seekerr      ;show seek error
        mov     byte [VERIFY_FLAG],0        ;Initially sector reads (rather than sctor verifys)
        POP     DX
        POP     CX
        POP     BX
        JMP     DONE_DISK                   ;and return (with error)



FDISK_STATUS:                               ;AH = 0
        mov     al,[IBM_DISK_STATUS]        ;Return past floppy status
        mov     byte [IBM_DISK_STATUS],0    ;reset status in low RAM for next time
        JMP     DONE_DISK                   ;and return



FDISK_PARAMS:                               ;AH = 8
        CMP     DL,00H                      ;DL=1 from INT call if drive B: (ZFDC controller drive #2, 5" disk)
        JZ      IS_144M_DISK
        MOV     DI,FDISK_5PARM_TBL          ;Return with drive paramater table in ES:DI
        MOV     AX,CS
        MOV     ES,AX                       ;And segment int ES:
        XOR     AX,AX                       ;Disk paramaters for 360K 5" Drive
        MOV     BH,0                        ;Always
        MOV     BL,01H                      ;0=Unknown, 1=360K, 2=1.2M, 3=720K, 4=1.44M
        MOV     CH,27H                      ;Max Track 39
        MOV     CL,9                        ;Max sector
        MOV     DH,1                        ;Max value of head
        MOV     DL,2                        ;Number of floppy disks
        mov     byte [IBM_DISK_STATUS],0    ;Show OK
        JMP     DONE_DISK                   ;and return

IS_144M_DISK:
        MOV     DI,FDISK_3PARM_TBL    ;Return with drive paramater table in ES:DI
        MOV     AX,CS
        MOV     ES,AX                       ;And segment int ES:
        XOR     AX,AX                       ;Disk paramaters for 1.44M 3" Drive
        MOV     BH,0                        ;Always
        MOV     BL,04H                      ;0=Unknown, 1=360K, 2=1.2M, 3=720K, 4=1.44M
        MOV     CH,4FH                      ;Max Track 79
        MOV     CL,18                       ;Max sector
        MOV     DH,1                        ;Max value of head
        MOV     DL,2                        ;Number of floppy disks!
        mov     byte [IBM_DISK_STATUS],0    ;Show OK
        JMP     DONE_DISK                   ;and return



FDISK_DASB:                                 ;AH = 15H, GET DISK TYPE (XT 1986/1/10 or later,XT286,AT,PS)
        XOR     AX,AX
        MOV     AH,01                       ;For now return flag "no disk change line support implemnented"
        mov     byte [IBM_DISK_STATUS],0    ;Show OK
        CLC                                 ;Clear CF
        JMP     ALL_OK                      ;Do not check status, just return
```

```
FDISK_MEDIA_CHANGE:                     ;AH = 16H
        XOR     AX,AX
        MOV     AX,06                   ;change line not support implemnented"
        mov     byte [IBM_DISK_STATUS],0    ;Show OK
        CLC                             ;Clear CF
        JMP     ALL_OK                  ;Do not check status, just return


;-------------------- READ FLOPPY DISK SECTORS ----------------------------------

FDISK_READ:                             ;AH=2, Read disk sector(s)
        PUSH    BX                      ;Save everything, DS already on stack
        PUSH    CX
        PUSH    DX
        PUSH    ES
        PUSH    DI                      ;Used in LES below and DMA_ADJUST

        MOV     [SECTORS_TO_DO],AL      ;save everything first
        MOV     byte [SECTORS_DONE],0
        MOV     [CURRENT_TRACK],CH
        MOV     [CURRENT_SECTOR],CL
        MOV     [CURRENT_HEAD],DH
        MOV     [CURRENT_DRIVE],DL
        MOV     [DMA_SEGMENT],ES        ;Save for below
        MOV     [DMA_OFFSET],BX

        CALL    DMA_ADJUST              ;Some DMA controllers cannot cross seg boundries, adjust

READ_COMMON:
        MOV     CL,CMD_SET_DRIVE        ;Set Drive Drive, ZFDC will just return if current drive
        CALL    S100OUT
        MOV     CL,[CURRENT_DRIVE]      ;DL from INT call
        OR      CL,CL                   ;DL = 0 --> ZFDC Drive #3.  DL = 1 -->ZFDC Drive #2
        MOV     CL,3                    ;Default to Drive #3
        JZ      RDD_FLOPPY
        MOV     CL,2                    ;Drive #2
RDD_FLOPPY:
        CALL    S100OUT
        CALL    WAIT_FOR_ACK            ;Return Z (or NZ with error # in [AH])
        JZ      READ_1
        mov     byte [IBM_DISK_STATUS],seekerr  ;show seek error
        JMP     ZFDC_READ_ERROR
READ_1:
        MOV     CL,CMD_SET_TRACK        ;Set Track
        CALL    S100OUT
        MOV     CL,[CURRENT_TRACK]
        CALL    S100OUT                 ;Send Selected track number
        CALL    WAIT_FOR_ACK            ;Return Z (or NZ with error # in [AH])
        JZ      READ_2
        mov     byte [IBM_DISK_STATUS],seekerr  ;show seek error
        JMP     ZFDC_READ_ERROR

READ_2:
        MOV     CL,CMD_SET_SIDE                 ;Set Drive Side/Head
        CALL    S100OUT
        MOV     CL,[CURRENT_HEAD]       ;Set side (Head 0,1)
        CALL    S100OUT
        CALL    WAIT_FOR_ACK            ;Return Z (or NZ with error # in [AH])
        JZ      READ_3
        mov     byte [IBM_DISK_STATUS],seekerr  ;show seek error
        JMP     ZFDC_READ_ERROR

READ_3:
        MOV     CL,CMD_DOS_SET_SECTOR   ;Set MS_DOS Sector (Note not CMD_SET_SECTOR for CPM)
        CALL    S100OUT
        MOV     CL,[CURRENT_SECTOR]
        CALL    S100OUT                 ;Send Selected track number
        CALL    WAIT_FOR_ACK            ;Return Z (or NZ with error # in [AH])
```

```
        JZ      READ_4
        mov     byte [IBM_DISK_STATUS],seekerr  ;show seek error
        JMP     ZFDC_READ_ERROR

READ_4: MOV     CL,CMD_SEEEK_TRACK                  ;Seek to that track (if not already there)
        CALL    S100OUT
        CALL    WAIT_FOR_ACK                ;Return Z (or NZ with error # in [AH])
        JZ      READ_5
        mov     byte [IBM_DISK_STATUS],seekerr  ;show seek error
        JMP     ZFDC_READ_ERROR

READ_5: MOV     CL,CMD_DOS_RD_MULTI_SEC             ;Routine assumes required Drive Table,Drive,(Side),Track, and
sector are already sent to board
        CALL    S100OUT
        MOV     CL,[SECTORS_TO_DO]          ;Count of sectors to be done (IBM says it will be (1---9 Max)
        CMP     CL,18                       ;IBM says it will always be <= 18 for 3" (9 for 5")
        JLE     READ_5a
        mov     byte [IBM_DISK_STATUS],dmaerr ;show as DMA error
        JMP     ZFDC_READ_ERROR

READ_5a:CALL    S100OUT                     ;Send sector count
        CALL    WAIT_FOR_ACK                ;Return Z (or NZ with error # in [AH])
        JZ      READ_6
        mov     byte [IBM_DISK_STATUS],rnferr ;show RNF error
        JMP     ZFDC_READ_ERROR

READ_6: LES     DI,[DMA_OFFSET]                     ;Point to initial DMA address  (ES:DI)

RD_LOOP:
        CMP     byte [DEBUG_FLAG],3         ;Is Detailed Floppy Debug mode on
        JNZ     RD_LOOP1                    ;If not skip
        CALL    DUMP_TRACK_PARAMS           ;Dump the Track,Head,Cylinder data to serial debug terminal

RD_LOOP1:                                   ;<<<<<<<<<<<< CORE DOS FLOPPY READ SECTOR LOOP >>>>>>>>>>>>>>>
        MOV     CX,512                      ;Assume 512 byte sectors always
        CMP     byte [VERIFY_FLAG],0ffH             ;Is it just a sector verify
        JZ      VERIFY_SECTOR

RDSEC:  MOV     BX,SECTOR_TIMEOUT           ;Put in a timeout count (Loop for status reads at most 256X4 times)
RDSEC1: DEC     BX                          ;Dec BC
        JNZ     RDSEC2                      ;Will wait 400H times before timing out
        MOV     AH,TIMEOUT_ERROR            ;Send Timeout error
        mov     byte [IBM_DISK_STATUS],timerr ;show as timeout error
        JMP     ZFDC_READ_ERROR

RDSEC2: IN      AL,S100STATUSB              ;Send data to ZFDC output
        TEST    AL,80H                      ;Is ZFDC in INPUT mode, if not wait
        JZ      RDSEC1
        TEST    AL,01H                      ;Has previous (if any) character been read.
        JZ      RDSEC1                      ;Z if not yet ready

        IN      AL,S100DATAB                ;Get data
        STOSB                               ;READ 1 BYTE BYTE, AL->ES:[DI++]
        LOOP    RDSEC

RDSEC5: mov     al,[SECTORS_DONE]           ;We have done one sector, are there more
        INC     al
        mov     [SECTORS_DONE],al           ;Store it
        CMP     [SECTORS_TO_DO],al          ;Have we done all yet
        JNZ     RD_LOOP

        CALL    WAIT_FOR_ACK                ;Return Z (or NZ with error # in [AH])
        JNZ     RD_SEC_ERR

        mov     byte [IBM_DISK_STATUS],0    ;Show good operation
        POP     DI                          ;Get back all origional registers
        POP     ES
        POP     DX
        POP     CX
```

```
        POP     BX
        mov     AL, [SECTORS_DONE]          ;Return # of sectors done
        JMP     DONE_DISK                   ;and return

RD_SEC_ERR:
        mov     byte [IBM_DISK_STATUS],crcerr ;Show CRC error
                                            ;Fall through to ZFDC_READ_ERROR
ZFDC_READ_ERROR:                            ;General read sector error reporting routine
        PUSH    AX
        MOV     BX,READ_ERR_MSG
        CALL    PRINT_STRING
        POP     AX
        MOV     AL,AH
        CALL    AL_HEXOUT
        MOV     BX,H_MSG_CRLF
        CALL    PRINT_STRING
        POP     DI                          ;Get back all origional registers
        POP     ES
        POP     DX
        POP     CX
        POP     BX
        mov     AL, [SECTORS_DONE]          ;Return # of sectors done
        JMP     DONE_DISK                   ;and return

VERIFY_SECTOR:                              ;Special case where we just check sector for CRC errors/verify
        MOV     BX,SECTOR_TIMEOUT           ;Put in a timeout count (Loop for status reads at most 256X4 times)
VRDSEC1:DEC     BX                          ;Dec BC
        JNZ     VRDSEC2                     ;Will wait 400H times before timing out
        MOV     AH,TIMEOUT_ERROR            ;Send Timeout error
        mov     byte [IBM_DISK_STATUS],timerr ;show as timeout error
        JMP     ZFDC_READ_ERROR

VRDSEC2:IN      AL,S100STATUSB              ;Send data to ZFDC output
        TEST    AL,80H                      ;Is ZFDC in INPUT mode, if not wait
        JZ      VRDSEC1
        TEST    AL,01H                      ;Has previous (if any) character been read.
        JZ      VRDSEC1                     ;Z if not yet ready

        IN      AL,S100DATAB                ;Get data
        LOOP    VERIFY_SECTOR
        JMP     RDSEC5                      ;Are there more sectors to verify



;-------------------- WRITE FLOPPY DISK SECTORS ----------------------------------

FDISK_WRITE:                                ;AH=3, Write disk
        PUSH    BX                          ;Save everything, DS already on stack
        PUSH    CX
        PUSH    DX
        PUSH    ES
        PUSH    DI                          ;Used in DMA_ADJUST
        PUSH    SI                          ;Need for LDS below

        MOV     [SECTORS_TO_DO],AL          ;save everything first
        MOV     byte [SECTORS_DONE],0
        MOV     [CURRENT_TRACK],CH
        MOV     [CURRENT_SECTOR],CL
        MOV     [CURRENT_HEAD],DH
        MOV     [CURRENT_DRIVE],DL
        MOV     [DMA_SEGMENT],ES            ;Save for below
        MOV     [DMA_OFFSET],BX

        CALL    DMA_ADJUST                  ;Some DMA controllers cannot cross seg boundries, adjust

WRITE_COMMON:
        MOV     CL,CMD_SET_DRIVE            ;Set Drive Drive, ZFDC will just return if current drive
        CALL    S100OUT
        MOV     CL,[CURRENT_DRIVE]          ;DL from INT call
```

```
        OR      CL,CL                           ;DL = 0 --> ZFDC Drive #3.  DL = 1 -->ZFDC Drive #2
        MOV     CL,3                            ;Default to Drive #3
        JZ      WDD_FLOPPY
        MOV     CL,2                            ;Drive #2
WDD_FLOPPY:
        CALL    S100OUT
        CALL    WAIT_FOR_ACK                    ;Return Z (or NZ with error # in [AH])
        JZ      WRITE_1
        mov     byte [IBM_DISK_STATUS],seekerr  ;Show seek error
        JMP     ZFDC_WRITE_ERROR
WRITE_1:
        MOV     CL,CMD_SET_TRACK                ;<<< Set Track
        CALL    S100OUT
        MOV     CL,[CURRENT_TRACK]
        CALL    S100OUT                         ;Send Selected track number
        CALL    WAIT_FOR_ACK                    ;Return Z (or NZ with error # in [AH])
        JZ      WRITE_2
        mov     byte [IBM_DISK_STATUS],seekerr  ;show seek error
        JMP     ZFDC_WRITE_ERROR
WRITE_2:
        MOV     CL,CMD_SET_SIDE                         ;<<< Set Drive Side/Head
        CALL    S100OUT
        MOV     CL,[CURRENT_HEAD]               ;Set side (Head 0,1)
        CALL    S100OUT
        CALL    WAIT_FOR_ACK                    ;Return Z (or NZ with error # in [AH])
        JZ      WRITE_3
        mov     byte [IBM_DISK_STATUS],seekerr  ;show seek error
        JMP     ZFDC_WRITE_ERROR

WRITE_3:
        MOV     CL,CMD_DOS_SET_SECTOR           ;Set MS_DOS Sector (Note not CMD_SET_SECTOR for CPM)
        CALL    S100OUT
        MOV     CL,[CURRENT_SECTOR]
        CALL    S100OUT                         ;Send Selected track number
        CALL    WAIT_FOR_ACK                    ;Return Z (or NZ with error # in [AH])
        JZ      WRITE_4
        mov     byte [IBM_DISK_STATUS],seekerr   ;show seek error
        JMP     ZFDC_WRITE_ERROR

WRITE_4:
        MOV     CL,CMD_SEEEK_TRACK                      ;<<< Seek to that track (if not already there)
        CALL    S100OUT
        CALL    WAIT_FOR_ACK                    ;Return Z (or NZ with error # in [AH])
        JZ      WRITE_5
        mov     byte [IBM_DISK_STATUS],seekerr  ;show seek error
        JMP     ZFDC_WRITE_ERROR

WRITE_5:MOV     CL,CMD_DOS_WR_MULTI_SEC             ;Routine assumes required Drive Table,Drive,(Side),Track, and
sector are already sent to board
        CALL    S100OUT
        MOV     CL,[SECTORS_TO_DO]              ;Count of sectors to be done (IBM says it will be (1---9 Max)
        CMP     CL,18                           ;IBM says it will always be <= 18 for 3" (9 for 5")
        JLE     WRITE_5a
        mov     byte [IBM_DISK_STATUS], dmaerr       ;show as DMA error
        JMP     ZFDC_WRITE_ERROR

WRITE_5a:CALL   S100OUT                         ;Send sector count
        CALL    WAIT_FOR_ACK                    ;Return Z (or NZ with error # in [AH])
        JZ      WRITE_6
        mov     byte [IBM_DISK_STATUS],rnferr ;show RNF error
        JMP     ZFDC_WRITE_ERROR

WRITE_6:PUSH    DS                              ;Remember low RAM DS pointer is no lonver valid
        MOV     AX,DS
        MOV     ES,AX                           ;ES will now have low RAM pointer
        LDS     SI,[DMA_OFFSET]
```

```
WR_LOOP:MOV    CX,512                          ;<<<<<<<<<<<< CORE DOS FLOPPY WRITE SECTOR LOOP >>>>>>>>>>>>>>>

WRSEC:  MOV    BX,SECTOR_TIMEOUT               ;Put in a timeout count (Loop for status reads at most 256X4 times)
WRSEC1: DEC    BX                              ;Dec BC
        JNZ    WRSEC2                          ;Will wait 400H times before timing out
        MOV    AH,TIMEOUT_ERROR                        ;Send Timeout error
        POP    DS                              ;get back origional DS
        mov     byte [IBM_DISK_STATUS],timerr ;show as timeout error
        JMP    ZFDC_WRITE_ERROR

WRSEC2: IN     AL,S100STATUSB                  ;Send data to ZFDC output
        TEST   AL,80H                          ;Is ZFDC in INPUT mode, if not wait
        JNZ    WRSEC1
        TEST   AL,02H                          ;Has previous (if any) character been written.
        JZ     WRSEC1                          ;Z if not yet ready

        LODSB                                  ;WRITE 1 BYTE BYTE, DS:[SI++] -> AL
        OUT    S100DATAB,AL                    ;Send it
        LOOP   WRSEC

        mov     al,[ES:SECTORS_DONE]           ;We have done one sector, are there more
        INC    al
        mov     [ES:SECTORS_DONE],al           ;Store it
        CMP     [ES:SECTORS_TO_DO],al          ;Have we done all yet
        JNZ    WR_LOOP                         ;read next 512 bytes

        POP    DS                              ;Balance up stack
        CALL   WAIT_FOR_ACK                    ;Return Z (or NZ with error # in [AH])
        JNZ    WR_SEC_ERR

        mov     byte [IBM_DISK_STATUS],0       ;Show good operation
        POP    SI
        POP    DI                              ;Get back all origional registers
        POP    ES
        POP    DX
        POP    CX
        POP    BX
        mov     AL, [SECTORS_DONE]             ;Return # of sectors done
         JMP   DONE_DISK                       ;and return

WR_SEC_ERR:
        mov     byte [IBM_DISK_STATUS],crcerr ;Show CRC error
                                               ;Fall through to ZFDC_WRITE_ERROR
ZFDC_WRITE_ERROR:                              ;General write sector error reporting routine
        CMP    AH,DISK_WP_ERR                  ;Special case for Write Protected Disk error
        JZ     F_DISK_WP_ERROR
        PUSH   AX
        MOV    BX,WRITE_ERR_MSG
        CALL   PRINT_STRING
        POP    AX
        MOV    AL,AH
        CALL   AL_HEXOUT
        MOV    BX,H_MSG_CRLF
        CALL   PRINT_STRING
WP_DONE:POP    SI
        POP    DI                              ;Get back all origional registers
        POP    ES
        POP    DX
        POP    CX
        POP    BX
        mov     AL, [SECTORS_DONE]             ;Return # of sectors done
        JMP    DONE_DISK                       ;and return

F_DISK_WP_ERROR:
         mov     byte [IBM_DISK_STATUS],wpterr       ;Write protected disk
         JMP   WP_DONE


;-------------------- FORMAT FLOPPY DISK ----------------------------------------------------------------
```

```
                                        ;Code not tested/complete yet!
FDISK_FORMAT:                           ;Format the current disk using teh ZFDC format track command
        PUSH    BX
        PUSH    CX
        PUSH    DX

        MOV     [CURRENT_DRIVE],DL
        MOV     [CURRENT_TRACK],CH

        CMP     DL,0                    ;If first track home heads
        JNZ     FORMAT_F1
        MOV     CL,CMD_SET_HOME                 ;Note this is a restore with NO verify. (disk my not be formatted)
        CALL    S100OUT
        CALL    WAIT_FOR_ACK            ;Return Z (or NZ with error # in [AH])
        JZ      FORMAT_F1
        mov     byte [IBM_DISK_STATUS],seekerr  ;show seek error
        JMP     ZFDC_FORMAT_ERROR

FORMAT_F1:
        MOV     CL,CMD_FORMAT_TRACK     ;Format a complete track on ZFDC controller
        CALL    S100OUT

        MOV     CL,[CURRENT_TRACK]      ;Send the track number
        CALL    S100OUT

        MOV     CL,CONFIRM_FORMAT       ;Now send SPECIAL OK to FORMAT Disk flag
        CALL    S100OUT
                                        ;<<< Now wait until track is formatted >>>
WAIT_F: CALL    S100STAT                ;Wait until ZFDC Board is ready
        JNZ     TRACK_DONE              ;NZ, something there!
        MOV     AH,1
        int     16H                     ;KEYBOARD - CHECK FOR KEYSTROKE
        JZ      WAIT_F                  ;Nothing, then wait some more
        MOV     AH,0                    ;Get character
        int     16H
        CMP     AL,ESC                  ;Was an ESC character eneterd
        JZ      ZFDC_FORMAT_ERROR
        JMP     WAIT_F

TRACK_DONE:
        CALL    S100IN                  ;Get returned Error # (Note this releases the SEND_DATA routine on the
ZFDC board)
        CMP     AL,NO_ERRORS_FLAG       ;Was SEND_OK/NO_ERRORS_FLAG sent back from ZFDC Board
        JNZ     ZFDC_FORMAT_ERROR
         mov     byte [IBM_DISK_STATUS],0     ;and good disk status
         POP    DX
         POP    CX
         POP    BX
         JMP    DONE_DISK               ;and return

ZFDC_FORMAT_ERROR:
        mov     byte [IBM_DISK_STATUS],cmderr ;Show as CMD error
        PUSH    AX
        MOV     BX,FORMAT_ERR_MSG
        CALL    PRINT_STRING
        POP     AX
        MOV     AL,AH
        CALL    AL_HEXOUT
        MOV     BX,H_MSG_CRLF
        CALL    PRINT_STRING
        POP     DX
        POP     CX
        POP     BX
        mov     AL, [SECTORS_DONE]      ;Return # of sectors done
        JMP     DONE_DISK               ;and return
```

```
;---------------- HARD DISK Routines ------------------------------------------------------------------

;We will use for MS-DOS Drive C: the second IDE Drive.
;Leaving the first for CPM86 (or, later the second MS-DOS hard disk)

HD_COMMANDS:
        TEST    AH,AH                           ;Is it a FDisk reset
        JNZ     N_HDISK_RESET
        JMP     HDISK_RESET
N_HDISK_RESET:
        CMP     AH,1                            ;Is it a HDISK status request
        JNZ     N_HDISK_STATUS
        JMP     HDISK_STATUS
N_HDISK_STATUS:
        CMP     AH,2                            ;Is it a HDISK read request
        JNZ     N_HDISK_READ
        MOV     byte [VERIFY_FLAG],0H           ;Turn off verify flag
        JMP     HDISK_READ
N_HDISK_READ:
        CMP     AH,3                            ;Is it a HDISK write request
        JNZ     N_HDISK_WRITE
        JMP     HDISK_WRITE
N_HDISK_WRITE:
        CMP     AH,4                            ;Is it a HDISK Verify request
        JNZ     N_HDISK_VERIFY
        MOV     byte [VERIFY_FLAG],0FFH                 ;Turn on verify flag
        JMP     HDISK_READ                      ;Modified read
N_HDISK_VERIFY:
        CMP     AH,5                            ;Is it a HDisk format request
        JNZ     N_HDISK_FORMAT
        JMP     HDISK_FORMAT
N_HDISK_FORMAT:
        CMP     AH,8                            ;Is it a HDisk paramaters request
        JNZ     N_HDISK_PARAMS
        JMP     HDISK_PARAMS
N_HDISK_PARAMS:
        CMP     AH,9                            ;Is it a HDisk Controller Initilize request
        JNZ     N_HDISK_INIT_REQ
        JMP     HDISK_INIT_REQ
N_HDISK_INIT_REQ:
        CMP     AH,0DH                          ;Is it a HDisk Reset request (Alternative)
        JNZ     N_HDISK_RESET2
        JMP     HDISK_RESET
N_HDISK_RESET2:
        CMP     AH,10H                          ;Is it a HDisk Ready check request
        JNZ     N_HDISK_READY_CHK
        JMP     HDISK_READY_CHK
N_HDISK_READY_CHK:
        CMP     AH,15H                          ;Is it a HDISK read DASB request
        JNZ     N_NOT_VALID_DISK
        JMP     HDISK_DASB
N_NOT_VALID_DISK:
        JMP     NOT_VALID_DISK                  ;Go to common/floppy error return




HDISK_RESET:                                    ;AH = 0, Home the disk head etc.
        CALL    SET_DRIVE_B                     ;Select the second Drive/CF card as MS-DOS Drive C:
        CALL    IDEinit                         ;Initialize drive 2. If there is no drive abort
        JNZ     HDISK_RESET_ERROR
        mov     byte [SEEK_STATUS],0            ;show good seek status
        mov     byte [IBM_DISK_STATUS],0        ;and good disk status
        JMP     DONE_DISK                       ;and return

HDISK_RESET_ERROR:
        MOV     BX,HOME_ERR_MSG
```

```
        CALL    PRINT_STRING
        mov     byte [IBM_DISK_STATUS],seekerr  ;show seek error
        JMP     DONE_DISK                       ;and return (with error)

HDISK_STATUS:                                   ;AH = 1
        mov     al,[IBM_DISK_STATUS]            ;Return past disk status
        mov     byte [IBM_DISK_STATUS],0        ;reset status in low RAM for next time
        JMP     DONE_DISK                       ;and return

HDISK_PARAMS:                                   ;AH = 8H Get Hard Drive Parms (We will assume one hard disk only, Custom
type)
        MOV     AH,0
        MOV     AL,DOS_MAXSEC                   ;Do NOT change ES or BX
        MOV     CH,DOS_MAXCYL_L-1               ;0FEH, low eight bits of maximum cylinder number
        MOV     CL,DOS_MAXSEC_CYL               ;3FH, maximum sector number (bits 5-0)+ two Cyl High Bits (Sectors
numbered 1....x)
                                                ;high two bits of maximum cylinder number (bits 7-6)
        MOV     DH,DOS_MAXHEADS-1               ;0FH, (0...15) 16 Heads
        MOV     DL,1                            ;Number of Hard Disks
        mov     byte [IBM_DISK_STATUS],0        ;Show OK
        JMP     DONE_DISK                       ;and return. This will put AH=0 & Clear CF


HDISK_INIT_REQ:                                         ;AH = 9H,   INITIALIZE CONTROLLER WITH DRIVE PARAMETERS (AT,PS)
HDISK_READY_CHK:                                ;AH = 10H,  HARD DISK - CHECK IF DRIVE READY
        mov     byte [IBM_DISK_STATUS],0        ;Since we have only one HDisk just return for now
        JMP     DONE_DISK

HDISK_DASB:                                     ;AH = 15H,   GET DISK TYPE (XT 1986/1/10 or later,XT286,AT,PS)
        MOV     AX,0310H                        ;AH, Indicates a Hard Disk
        MOV     CX,000FH
        MOV     DX,0BC10H                       ;This is what the AMI Bios returns for our cystom drive (CX:DX = Total
sectors)
        mov     byte [IBM_DISK_STATUS],0        ;Show OK
        CLC                                     ;Clear CF
        JMP     ALL_OK                          ;Do not check status, just return


HDISK_FORMAT:                                   ;AH = 05H,   Format disk - Return immediately with status ok
        mov     byte [IBM_DISK_STATUS],0        ;show good operation no matter what
        JMP     DONE_DISK                       ;and return



;-------------------- READ HARD DISK DISK SECTORS ----------------------------------

HDISK_READ:
        PUSH    BX                              ;Save everything, DS already on stack
        PUSH    CX
        PUSH    DX
        PUSH    ES
        PUSH    DI                              ;Used in LES below and DMA_ADJUST

        MOV     [SECTORS_TO_DO],AL              ;save everything first
        MOV     byte [SECTORS_DONE],0
        MOV     AL,CL                           ;Store Sector
        AND     CL,00111111B                    ;Strip High 2 track bits
        MOV     [CURRENT_SECTOR],CL
        MOV     AH,0                            ;Shift the top 2 bits of AL into AH
        SHL     AX,1
        SHL     AX,1
        MOV     [CURRENT_TRACK_HIGH],AH
        MOV     [CURRENT_TRACK],CH              ;Store low track#
        MOV     [CURRENT_HEAD],DH               ;Store Head#
        MOV     [CURRENT_DRIVE],DL              ;Actully for now always drive #2 on IDE Board
        MOV     [DMA_SEGMENT],ES                ;Save for below
        MOV     [DMA_OFFSET],BX

        CALL    DMA_ADJUST                      ;Some DMA controllers cannot cross seg boundries, adjust
```

```
        CMP     byte [DEBUG_FLAG],2           ;Is Detailed Hdisk/Floppy Debug mode on
        JL      HREAD_COMMON                  ;If not skip
        CALL    DUMP_TRACK_PARAMS             ;Dump the Track,Head,Cylinder data to serial debug terminal

HREAD_COMMON:
        CALL    DOS_WR_LBA                    ;Setup Drive, Track, Sector for MS-DOS formatted disk.
        CALL    IDEwaitnotbusy                ;Make sure drive is ready
        JNB     HL_19                         ;Carry flag set if problem
        CALL    SHOWerrors                    ;Show error data on CRT
        mov     byte [IBM_DISK_STATUS],seekerr       ;show seek error
        JMP     H_READ_ERROR                  ;General read HDisk sector error reporting routine
                                              ;and return (with error)

HL_19:  MOV     DH,COMMANDread
        MOV     DL,REGcommand
        CALL    IDEwr8D                       ;Send Sec read command to drive.
        CALL    IDEwaitdrq                    ;Wait until it's got the data
        JNB     HL_20                         ;Carry flag set if problem
        CALL    SHOWerrors                    ;Show error data on CRT
        mov     byte [IBM_DISK_STATUS],crcerr ;Show as CRC error
        JMP     H_READ_ERROR                  ;General read HDisk sector error reporting routine
                                              ;and return (with error)

HL_20:  LES     DI,[DMA_OFFSET]                       ;Point to initial DMA address (ES & DI already saved above)
HRD_LOOP:
        MOV     CX,256                        ;ALWAYS read 512 bytes to [CX] (256X2 bytes)

HRD_LOOP_BYTES:
        MOV     AL,REGdata                    ;REG regsiter address
        OUT     IDEportC,AL

        OR      AL,IDErdline                  ;08H+40H, Pulse RD line
        OUT     IDEportC,AL

        IN      AL,IDEportA                   ;Read the LOWER byte first
        STOSB                                 ;READ 1 BYTE BYTE, AL->ES:[DI++]
        IN      AL,IDEportB                   ;THEN read the UPPER byte
        STOSB                                 ;READ 1 BYTE BYTE, AL->ES:[DI++]

        MOV     AL,REGdata                    ;Deassert RD line
        OUT     IDEportC,AL

        LOOP    HRD_LOOP_BYTES                ;256 words, for 512 bytes

        CMP     byte [DEBUG_FLAG],2           ;Is Detailed HDISK/Floppy Debug mode on
        JL      HRDSEC4                       ;If not skip
        CALL    SERIAL_DUMP_RD_SECTOR_DATA    ;Dump first 16 bytes of data

HRDSEC4:
        MOV     CX,0FFFFH                     ;Need to wait until the IDE drive is ready
HRDSEC5:                                      ;with status data after potentially a long
        MOV     DL,REGstatus                  ;series of sector reads.
        CALL    IDErd8D                       ;Returned data in DH
        AND     DH,80H                        ;Is IDE Drive still busy (bit 7 low)
        JZ      HRDSEC6                       ;No, then check returned status
        LOOP    HRDSEC5                       ;wait until ready
HRDSEC6:
        MOV     AL,DH                         ;Was previous command completed without errors
        AND     AL,1H                         ;Ret AL=0 for all OK
        JZ      HNEXT_SECTOR_RD

        CALL    SHOWerrors                    ;Show error data on CRT
        mov     byte [IBM_DISK_STATUS],crcerr ;Show as CRC error
        JMP     H_READ_ERROR                  ;General write HDisk sector error reporting routine
                                              ;and return (with error)

                                              ;We have done one sector, are there more
                                              ;On hard disks (with XT and AT BIOSes), a multi-sector read
```

```
                                        ;continues on the next higher head of the same cylinder and if
HNEXT_SECTOR_RD:                        ;necessary, advances to the next higher cylinder on the first head.
        mov     al,[SECTORS_DONE]
        INC     al
        mov     [SECTORS_DONE],al       ;Store it
        CMP     [SECTORS_TO_DO],al      ;Have we done all yet
        JNE     HRD_LOOP

        mov     byte [IBM_DISK_STATUS],0    ;Show good operation
HRD_DONE:
        POP     DI                      ;Get back all origional registers
        POP     ES
        POP     DX
        POP     CX
        POP     BX
        mov     AL, [SECTORS_DONE]      ;Return # of sectors done
        JMP     DONE_DISK               ;and return

H_READ_ERROR:
        PUSH    AX
        MOV     BX,HREAD_ERR_MSG
        CALL    PRINT_STRING
        CALL    H_PRINT_CHS             ;Print current Cyl, Head, Sector (DS: points to low RAM data stores)
        POP     AX
        JMP     HRD_DONE




HDISK_WRITE:                            ;Arrive here with DS: pointing to low RAM data area
        PUSH    BX                      ;Save everything, DS already on stack
        PUSH    CX
        PUSH    DX
        PUSH    ES
        PUSH    DI                      ;Used in LES below and DMA_ADJUST
        PUSH    SI                      ;Need for LDS below

        MOV     [SECTORS_TO_DO],AL      ;save everything first
        MOV     byte [SECTORS_DONE],0
        MOV     AL,CL                   ;Store Sector
        AND     CL,00111111B            ;Strip High 2 track bits
        MOV     [CURRENT_SECTOR],CL
        MOV     AH,0                    ;Shift the top 2 bits of AL into AH
        SHL     AX,1
        SHL     AX,1
        MOV     [CURRENT_TRACK_HIGH],AH
        MOV     [CURRENT_TRACK],CH      ;Store low track#
        MOV     [CURRENT_HEAD],DH       ;Store Head#
        MOV     [CURRENT_DRIVE],DL      ;Actully for now always drive #2 on IDE Board
        MOV     [DMA_SEGMENT],ES        ;Save for below
        MOV     [DMA_OFFSET],BX

        CALL    DMA_ADJUST              ;Some DMA controllers cannot cross seg boundries, adjust

        CMP     byte [DEBUG_FLAG],2     ;Is Detailed Hdisk/Floppy Debug mode on
        JL      HWRITE_COMMON           ;If not skip
        CALL    DUMP_TRACK_PARAMS       ;Dump the Track,Head,Cylinder data to serial debug terminal

HWRITE_COMMON:
        CALL    DOS_WR_LBA              ;Setup Drive, Track, Sector for MS-DOS formatted disk.
        CALL    IDEwaitnotbusy          ;Make sure drive is ready
        JNB     HW_19                   ;Carry flag set if problem
        CALL    SHOWerrors              ;Show error data on CRT
        mov     byte [IBM_DISK_STATUS],seekerr      ;show seek error
        JMP     H_WRITE_ERROR           ;General write HDisk sector error reporting routine
                                        ;and return (with error)

HW_19:  MOV     DH,COMMANDwrite
        MOV     DL,REGcommand
```

```
        CALL    IDEwr8D                         ;Send Sec write command to drive.
        CALL    IDEwaitdrq                      ;Wait until it's got the data
        JNB     HW_20                           ;Carry flag set if problem
        CALL    SHOWerrors                      ;Show error data on CRT
        mov     byte [IBM_DISK_STATUS],crcerr ;Show as CRC error
        JMP     H_WRITE_ERROR                   ;General write HDisk sector error reporting routine
                                                ;and return (with error)

HW_20:  PUSH    DS                              ;Remember from now on, low RAM DS pointer is no longer valid
        MOV     AX,DS
        XOR     AX,AX
        MOV     ES,AX                           ;ES will now temporly have the low RAM pointer
        LDS     SI,[DMA_OFFSET]


HWR_LOOP:
        MOV     AL,WRITEcfg8255                         ;8255 to write mode
        OUT     IDECtrlPort,AL
        MOV     CX,256                          ;ALWAYS read 512 bytes to [CH] (256X2 bytes)

HWR_LOOP_BYTES:
        LODSB                                   ;WRITE 1 BYTE, DS:[SI++] -> AL
        OUT     IDEportA,AL                     ;Write the LOWER byte first
        LODSB                                   ;WRITE 1 BYTE, DS:[SI++] -> AL
        OUT     IDEportB,AL                     ;THEN UPPER byte on B

        MOV     AL,REGdata
        PUSH    AX
        OUT     IDEportC,AL                     ;Send write command
        OR      AL,IDEwrline                    ;Send WR pulse
        OUT     IDEportC,AL
        POP     AX
        OUT     IDEportC,AL                     ;Send write command
        LOOP    HWR_LOOP_BYTES                  ;One sector done

        MOV     AL,READcfg8255                  ;Set 8255 back to read mode
        OUT     IDECtrlPort,AL

        MOV     CX,0FFFFH                       ;Need to wait until the IDE drive is ready
HW_21:                                          ;with status data after potentially a long
        MOV     DL,REGstatus                    ;Series of sector writes
        CALL    IDErd8D                         ;Returned data in DH
        AND     DH,80H                          ;Is IDE Drive still busy
        JZ      HW_22                           ;No, then check returned staus
        LOOP    HW_21
HW_22:
        MOV     AL,DH                           ;Was previous command completed without errors
        AND     AL,1H                           ;Ret AL=0 for all OK
        JZ      HNEXT_SECTOR_WR

        POP     DS                              ;Get back DS for above
        CALL    SHOWerrors                      ;Show error data on CRT
        mov     byte [IBM_DISK_STATUS],crcerr ;Show as CRC error
        JMP     H_WRITE_ERROR                   ;General write HDisk sector error reporting routine

                                                ;We have done one sector, are there more
                                                ;On hard disks (with XT and AT BIOSes), a multi-sector read
                                                ;continues on the next higher head of the same cylinder and if
HNEXT_SECTOR_WR:                                ;necessary, advances to the next higher cylinder on the first head.
        mov     al,[ES:SECTORS_DONE]
        INC     al
        mov     [ES:SECTORS_DONE],al            ;Store it
        CMP     [ES:SECTORS_TO_DO],al           ;Have we done all yet
        JNZ     HWR_LOOP

        POP     DS                              ;Get back DS for above.
        mov     byte [IBM_DISK_STATUS],0        ;Show good operation
HWR_DONE:
        POP     SI
```

```
        POP     DI                          ;Get back all origional registers
        POP     ES
        POP     DX
        POP     CX
        POP     BX
        mov      AL, [SECTORS_DONE]         ;Return # of sectors done
         JMP    DONE_DISK                  ;and return

H_WRITE_ERROR:
        MOV     AL,READcfg8255              ;Set 8255 back to read mode
        OUT     IDECtrlPort,AL
        PUSH    AX
        MOV     BX,HWRITE_ERR_MSG           ;"HDisk Sector Write Error"
        CALL    PRINT_STRING
        CALL    H_PRINT_CHS                 ;Print current Cyl, Head, Sector (DS: points to low RAM data stores)
        POP     AX
        JMP     HWR_DONE

H_PRINT_CHS:                                ;DS: points to low RAM data stores
        MOV     BX,HD_MSG                   ;"  Head = "
        CALL    PRINT_STRING
        MOV     AL,[CURRENT_HEAD]
        CALL    AL_HEXOUT

        MOV     BX,CYL_MSG                  ;"H  Cyl = "
        CALL    PRINT_STRING
        MOV     AH,[CURRENT_TRACK_HIGH]
        MOV     AL,[CURRENT_TRACK]
        CALL    AX_HEXOUT

        MOV     BX,SEC_MSG                  ;"H  Sec = "
        CALL    PRINT_STRING
        MOV     AL,[CURRENT_SECTOR]
        CALL    AL_HEXOUT

        MOV     BX,BRAC1_MSG                ;"H  ("
        CALL    PRINT_STRING
        MOV     AL,[SECTORS_DONE]
        CALL    AL_HEXOUT

        MOV     BX,OF_MSG                   ;"H of "
        CALL    PRINT_STRING
        MOV     AL,[SECTORS_TO_DO]
        CALL    AL_HEXOUT

        MOV     BX,BRAC2_MSG                ;H") "
        CALL    PRINT_STRING
        RET


;----------------- SUPPORT ROUTINES FOR ZFDC BOARD FOR MSDOS/FREEDOS ----------------------------------------

INIT_ZFDC:                          ;Return 0FFH in [ZFDC_INIT_FLAG] and Z flag set if all OK
        OUT     RESETZFDCPORT,AL    ;Do a hardware reset. Does not matter what is in [AL]

        MOV     AX,5                ;~0.5 second at 10 MHz
        MOV     CX,0                ;Delay to allow board to setup hardware
WAITD:  LOOP    WAITD               ;Delay for ~0.5 seconds
        DEC     AX
        JNZ      WAITD

        IN      AL,S100DATAB        ;Check the board is there
        CMP     AL,CMD_HANDSHAKE    ;Make sure we get HANDSHAKE byte back
        MOV     AH,ZFDC_ABSENT      ;If not then no ZFDC board present
        JNZ     BADZFDC             ;If not there, just abort

        MOV     AL,CMD_HANDSHAKE    ;Send another byte just to be sure.
        OUT     S100DATAB,AL        ;This clears up ints on ZFDC board
        CALL    WAIT_FOR_ACK        ;Return Z (or NZ with error # in [AH])
```

```
        OR      AL,AL
        MOV     AH,ZFDC_INIT_ERROR      ;If not then no ZFDC board present
        JNZ     BADZFDC                 ;just abort

                                        ;Leave drives 0,1 UNFORMATTED/UNINITILIZED for now

        MOV     CL,CMD_SET_FORMAT       ;Send Set Disk Format to Drive CMD for drive #3 (1.44M 3" disk)
        CALL    S100OUT
        MOV     CL,3                    ;Floppy Drive 3, (ZFDC Board expects a 0H, 1H, 2H or 3H)
        CALL    S100OUT
        MOV     CL,IBM144               ;1.4M (For MSDOS) DDDS, 18 X 512 Byte Sectors, 80 Tracks. (See ZFDC Board Code
for more info)
        CALL    S100OUT
        CALL    WAIT_FOR_ACK            ;Return Z (or NZ with error # in [AH])
        JNZ     BADZFDC

        MOV     CL,CMD_SET_FORMAT       ;Send Set Disk Format to Drive CMD for drive #2 (360K 5" disk)
        CALL    S100OUT
        MOV     CL,2                    ;Floppy Drive 2, (ZFDC Board expects a 0H, 1H, 2H or 3H)
        CALL    S100OUT
        MOV     CL,MSDOS2               ;5", IBM PC, MSDOS 2.x, 512 byte, DDDS, 9 sector format (See ZFDC Board Code for
more info)
        CALL    S100OUT
        CALL    WAIT_FOR_ACK            ;Return Z (or NZ with error # in [AH])
        JNZ     BADZFDC

        MOV     CL,CMD_SET_DRIVE        ;<<< Set Drive Drive DOS A: ZFDC will just return if current drive
        CALL    S100OUT
        MOV     CL,3                    ;Set drive #3 as the current drive
        CALL    S100OUT
        CALL    WAIT_FOR_ACK            ;Return Z (or NZ with error # in [AH])
        JNZ     BADZFDC                 ;just abort

        PUSH    BX                      ;Return BX unaltered
        MOV     AL,0FFH                 ;Flag to indicate ZFDC board is setup OK
        MOV     [ZFDC_INIT_FLAG],AL     ;Note DS is already set for ROM usage in low RAM (400H)
        MOV     BX,ZFDC_OK_MSG          ;Announce success
        CALL    PRINT_STRING
        POP     BX
        XOR     AL,AL
        RET                             ;Return Z for all OK

BADZFDC:
        PUSH    BX                      ;Return BX unaltered
        MOV     AL,0H                   ;Flag to indicate ZFDC board is NOT OK
        MOV     [ZFDC_INIT_FLAG],AL     ;Note DS is already set for ROM usage in low RAM (400H)
        MOV     BX,ZFDC_FAIL_MSG        ;Announce failure
        CALL    PRINT_STRING
        POP     BX
        XOR     AL,AL
        DEC     AL
        RET                             ;Return NZ WITH ERROR IN AH


DUMP_TRACK_PARAMS:                      ;Dump the Track,Head,Cylinder data to serial debug terminal
        MOV     CL,CR
        CALL    SERIAL_OUT
        MOV     CL,LF
        CALL    SERIAL_OUT

        MOV     CL,'h'
        CALL    SERIAL_OUT
        MOV     AL,[CURRENT_HEAD]
        CALL    SERIAL_AL_HEXOUT

        MOV     CL,'.'
        CALL    SERIAL_OUT
        MOV     CL,'t'
```

```
        CALL    SERIAL_OUT
        MOV     AL,[CURRENT_TRACK]     ;Note DS is already set for ROM usage in low RAM (400H)
        CALL    SERIAL_AL_HEXOUT


        MOV     CL,'.'
        CALL    SERIAL_OUT
        MOV     CL,'s'
        CALL    SERIAL_OUT
        MOV     AL,[CURRENT_SECTOR]
        CALL    SERIAL_AL_HEXOUT

        MOV     CL,' '
        CALL    SERIAL_OUT
        MOV     CL,'#'
        CALL    SERIAL_OUT
        MOV     AL,[SECTORS_TO_DO]
        CALL    SERIAL_AL_HEXOUT
        MOV     CL,','
        CALL    SERIAL_OUT
        MOV     AL,[SECTORS_DONE]
        CALL    SERIAL_AL_HEXOUT
        MOV     CL,' '
        CALL    SERIAL_OUT
        MOV     CL,' '
        CALL    SERIAL_OUT
        RET


SERIAL_DUMP_RD_SECTOR_DATA:            ;Note this is only for sector reads. ES: is invalid for Writes
        PUSH    AX                     ;Show first 8 bytes of sector data on serial output (for debugging)
        PUSH    BX
        PUSH    CX
        PUSH    DI
        MOV     CX,16                  ;Show first 16 characters
DUMPS1: MOV     AL,[ES:DI]             ;DI will have the current address
        CALL    SERIAL_AL_HEXOUT
        INC     DI
        LOOP    DUMPS1
        MOV     BX,CR_TAB_MSG          ;CR to next like then tab in 18 spacs (for multisector reads)
        CALL    SERIAL_PRINT_STRING
        POP     DI
        POP     CX
        POP     BX
        POP     AX
        RET


SIMPLE_SECTOR_DUMP:                    ;Dump first CX bytes of sector data at ES:BX on CRT
        PUSH    DS

        PUSH    BX
        PUSH    CX

        PUSH    BX
        PUSH    CX

         XOR    AX,AX                  ;Set DS to data area for ROM usage in low RAM @ 400H....)
         MOV    DS,AX

        MOV     BX,SEQAT500            ;"First [CX] bytes of loaded Sector (@ES:BX) Head ="
        CALL    PRINT_STRING
        MOV     AL,[CURRENT_HEAD]
        CALL    AL_HEXOUT

        MOV     BX,TRACK_MSG           ;"H  Track ="
        CALL    PRINT_STRING
        MOV     AL,[CURRENT_TRACK]
        CALL    AL_HEXOUT
```

```
        MOV     BX,SEC_MSG              ;"H  Sector ="
        CALL    PRINT_STRING
        MOV     AL,[CURRENT_SECTOR]
        CALL    AL_HEXOUT
        MOV     BX,START_DATA_MSG       ;"H  Start of Data =  CR,LF"
        CALL    PRINT_STRING

        POP     CX                      ;From above
        POP     BX
ONE_LINE_SECTOR1:
        MOV     AX,[ES:BX]              ;High byte/low byte order
        PUSH    AX
        CALL    AL_HEXOUT
        POP     AX
        MOV     AL,AH
        CALL    AL_HEXOUT
        INC     BX
        INC     BX
        LOOP    ONE_LINE_SECTOR1

        CALL    CRLF
        POP     CX                      ;Again from above
        POP     BX

SECTOR_DUMP1:
        PUSH    CX
        MOV     CL,[ES:BX]              ;High byte/low byte order
        and     cl,7fh
        cmp     cl,' '                  ;filter out control characters
        jnc     xloop3
xloop4: mov     cl,'.'
xloop3: cmp     cl,'~'
        jnc     xloop4
        CALL    CO
        INC     BX                      ;Next character
        POP     CX
        LOOP    SECTOR_DUMP1
        CALL    CRLF
        POP     DS                      ;Balance up stack
        RET


S100STAT:                               ;Check if ZFDC has any data for S-100 system
        IN      AL,S100STATUSB
        TEST    AL,01H                  ;Anything there ?
        JZ      S100ST1                 ;Return 0 if nothing
        XOR     AL,AL
        DEC     AL                      ;Return NZ, & 0FFH in AL if something there
S100ST1:RET



S100IN: IN      AL,S100STATUSB          ;Check if ZFDC has any data for S-100 system
        TEST    AL,80H                  ;Is ZFDC in input mode, if not, wait
        JZ      S100IN                  ;If low then ZFDC board is still in input mode, wait
        TEST    AL,01H
        JZ      S100IN
        IN      AL,S100DATAA            ;return with character in AL
        RET


S100OUT:IN      AL,S100STATUSB          ;Send data to ZFDC output (arrive with character to be sent in C)
        TEST    AL,80H                  ;Is ZFDC in output mode, if not wait
        JNZ     S100OUT
        TEST    AL,02H                  ;Has previous (if any) character been read.
        JZ       S100OUT                        ;Z if not yet ready
        MOV     AL,CL
        OUT     S100DATAB,AL
```

```
        RET


WAIT_FOR_ACK:                       ;Delay to wait for ZFDC to return data. There is a timeout of about 2 sec.
        PUSH    BX                  ;This can be increased if you are displaying debugging info on the ZFDC
        PUSH    DX                  ;HEX LED display.
        MOV     BX,0
        MOV     DL,STATUSDELAY      ;Timeout, (about 2 seconds)
WAIT1:  IN      AL,S100STATUSB      ;Check if ZFDC has any data for S-100 system
        TEST    AL,80H              ;Is ZFDC in input mode
        JZ      WAIT2               ;if low then ZFDC is still in input mode
        CALL    S100STAT            ;Wait until ZFDC Board sends something
        JZ      WAIT2
        CALL    S100IN              ;Get returned Error # (Note this releases the SENDDATA routine on the ZFDC board)
        MOV     AH,AL               ;<<< Store Error Code (if any) in AH
        CMP     AL,NO_ERRORS_FLAG   ;Was SENDOK/NOERRORSFLAG sent back from ZFDC Board
        POP     DX                  ;Balance up stack
        POP     BX
        RET                         ;Return NZ if problem, Z if no problem

WAIT2:  DEC     BH
        JNZ     WAIT1               ;Try for ~2 seconds
        DEC     BH
        DEC     BL
        JNZ     WAIT1
        DEC     BH
        DEC     BL
        DEC     DL
        JNZ     WAIT1
        XOR     AL,AL
        DEC     AL
        MOV     AH,3FH              ;Flag as local Time out error
        POP     DX                  ;Balance up stack
        POP     BX
        RET                         ;Return NZ flag set if timeout & 0FFH in [AL]
                                    ;Error code in AH


;       Adjust DMASEG:DMAOFF via [ES:DI] so that the in DI is the
;       smallest possible. This process is called normalization.
;       Registers:   Only ES and DI altered

DMA_ADJUST:
        MOV     ES,[DMA_SEGMENT]
        MOV     DI,[DMA_OFFSET]

        PUSH    AX
        PUSH    DI
        SHR     DI,1                ; Get paragraph to low 12 bits
        SHR     DI,1                ; Shift 0's in at hi 4 bits
        SHR     DI,1
        SHR     DI,1
        MOV     AX,ES               ; Get segment to Bx
        ADD     AX,DI               ; Add in segment skew
        MOV     ES,AX               ; Restore dma segment
        POP     DI                  ; Get back original offset
        AND     DI,0FH              ; Only need within paragraph

        MOV     [DMA_SEGMENT],ES
        MOV     [DMA_OFFSET],DI             ;<<< Later use LES (or for Sec Write LDS)
        POP     AX
        RET



;*******************************
;
;       Non Maskable Interrupt Handler (for IBM-PC is int #2, or 08H in RAM)
;
```

```
;*********************************

NMI_hnd:                            ;Non Maskable Interrupt Handler (Note uses current stack!)
        PUSHF                       ;Should not get here. If so send warning and continue
        PUSH    AX
        PUSH    BX
        PUSH    CX
        MOV     BX,NMI_MSG          ;Announce we got an NMI Interrupt
        CALL    PRINT_STRING        ;Note PRINT_STRING always uses the CS: override for the BX pointer
        POP     CX
        POP     BX
        POP     AX
        POPF                        ;Note NMI does not push the flags on to the stack
        IRET


;*********************************
;
;       Keypressed Handler        (for IBM-PC is int #9 via 8259A to 24H in RAM)
;
;       IRQ1 - KEYBOARD DATA READY
;              This interrupt is generated when data is received from the keyboard. This is normally
;              a scan code (from either a keypress OR a key release), but may also be an ACK or NAK
;              of a command on AT-class keyboards.
;
;              Note: This IRQ may be masked by setting bit 1 on the 8259A I/O port 21h.
;
;              If the BIOS supports an enhanced (101/102-key)keyboard, it calls INT 15/AH=4Fh after reading the
;              scan code from the keyboard and before further processing all further processing uses the scan code
;              returned from INT 15/AH=4Fh.
;
;              The default interrupt handler is at F000h:E987h in 100%-compatible BIOSes. The interrupt handler
performs
;              the following actions for certain special keystrokes:-
;
;              Ctrl-Break clear keyboard buffer, place word 0000h in buffer, invokes INT 1B, and sets flag at
0040h:0071h
;              SysReq invokes INT 15/AH=85h (SysReq is often labeled SysRq)
;              Ctrl-Numlock place system in a tight wait loop until next INT 09
;              Ctrl-Alt-Del jump to BIOS startup code (either F000h:FFF0h or the destination of the jump at that
address)
;              Shift-PrtSc invokes INT 05
;              <<<<<< None of thes are yet implemented
;
;*********************************

keyhnd: pushf                       ;This interrupt can strike any time, so save all
        push    ax
        push    ds
        push    bx
        XOR     AX,AX               ;Set DS to data area for ROM usage in low RAM @ 400H....)
        MOV     DS,AX

        in      al,KEYIN              ;get data
        and     al,7fh                ;strip parity bit (if any)
        mov     bl,[chrcnt]         ;get current character count
        cmp     bl,chrmax             ;is the buffer full?
        jge     keyxt                 ;ignore if buffer full
        inc     bl
        mov     [chrcnt],bl         ;store new character count
        mov     bx,[buftl]          ;get destination address
        mov     [bx],al                     ;store the character
        inc     bx                    ;bump buffer address
        cmp     bx,keybuff+32       ;at end of buffer?
        jl      keyhn1                ;skip if not
        mov     bx,keybuff          ;reset to start of buffer
keyhn1: mov     [buftl],bx          ;store adr for next character
keyxt:  pop     bx
        mov     al,NS_EOI
```

```
        OUT     MASTER_PIC_PORT,al
        pop     ds
        pop     ax
        popf
        iret



;*******************************
;
;       Timer Handler         (for IBM-PC is int #8 via 8259A to 20H in RAM)
;       IRQ0 - SYSTEM TIMER
;              On a PC this is generated 18.2 times per second by channel 0 of the 8254 system timer,
;              this interrupt is used to keep the time-of-day clock updated. It can strike any time in a program!
;
;*******************************
                                ;Note:  The IBM PC clock interrupts at
                                ;1193180/65536 counts/sec (Approx 18.2 per second).
                                ;Our clock interrupts at ~60 hz so adjust to approximate
                                ;the IBM clock, the time constants in this routine must be
                                ;adjusted accordingly if accurate time is to be kept by PC-DOS.

timer:                          ;This interrupt can strike any time, so save all (flags are already saved)
        push    ax
        push    ds
        XOR     AX,AX           ;Set DS to data area for ROM usage in low RAM @ 400H....)
        MOV     DS,AX

        MOV     AL,10H          ;Point to Interrupt Status Register of MM581657A on PIC/RT board
        OUT     RTCSEL,AL
        IN      AL,RTCDATA      ;This resets the interrupt for the next 0.1 sec INT on the chip
        AND     AL,00000010B    ;Check its the 0.1 Second interrupt bit
        JZ      timer2          ;If not just skip

        inc     word [timlow]   ;Bump count
        jnz     timer1
        inc     word [timhi]     ;Bump high part of count
timer1: cmp     word [timhi],18h ;End of a day?
        jnz     timer2            ;24 hours at 3600 sec/hr
        cmp     word [timlow],0b0h ;and 1193180/65536 tics/sec
        jnz     timer2            ;= 1573040 tics (1800B0 hex)
        sub     ax,ax           ;0 to AX
        mov     [timlow],ax
        mov     [timhi],ax
        mov     byte [timofl],1

timer2: nop                     ;No need for disk motor timeout check with ZFDC board

        int     1CH             ;Go to user timer int at 1CH, IRET when done <<<<

        sti
        mov     al,NS_EOI       ;End with Send_EOI
        OUT     MASTER_PIC_PORT,al
        pop     ds
        pop     ax
        iret                    ;IRET will return flags



Send_EOI:                       ;General routine to send EOI to 85293A
        PUSH    AX
        mov     al,NS_EOI
        OUT     MASTER_PIC_PORT,al
        POP     AX
dummy_return:
        iret                    ;Remember IRET will pop the flags

;*******************************
```

```
;
;       Time of Day Handler     (For IBM-PC Software Interrupt 1AH)
;
;Input  AH = 00h       TIME - GET SYSTEM TIME
;
;Return:CX:DX = number of clock ticks since midnight
;       AL = midnight flag, nonzero if midnight passed since time last read
;
;Input  AH = 01h       TIME - SET SYSTEM TIME
;       CX:DX = number of clock ticks since midnight
;Return:Nothing
;
;Input: AH = 02h       TIME - GET REAL-TIME CLOCK TIME (AT, XT286, PS)
;       CF clear to avoid a bug
;Return:CF clear if successful
;       CH = hour (BCD)
;       CL = minutes (BCD)
;       DH = seconds (BCD)
;       DL = daylight savings flag (00h standard time, 01h daylight time)
;       CF set on error (i.e. clock not running or in middle of update)
;
;Input: AH = 03h       TIME - SET REAL-TIME CLOCK TIME (AT, XT286, PS)
;       CH = hour (BCD)
;       CL = minutes (BCD)
;       DH = seconds (BCD)
;       DL = daylight savings flag (00h standard time, 01h daylight time)
;Return:Nothing
;
;Input: AH = 04h       TIME - GET REAL-TIME CLOCK DATE (AT, XT286, PS)
;       CF clear to avoid bug (see below)
;
;Return:CF clear if successful
;       CH = century (BCD)
;       CL = year (BCD)
;       DH = month (BCD)
;       DL = day (BCD)
;       CF set on error
;
;Input: AH = 05h       TIME - SET REAL-TIME CLOCK DATE (AT, XT286, PS)
;       CH = century (BCD)
;       CL = year (BCD)
;       DH = month (BCD)
;       DL = day (BCD)
;Return:Nothing
;
;*********************************

time_of_day:
        sti
        push    ds
        PUSH    AX
        XOR     AX,AX                   ;Set DS to data area for ROM usage in low RAM @ 400H....)
        MOV     DS,AX
        POP     AX

        CMP     byte [DEBUG_FLAG],0     ;Is Debug mode on
        JZ      Xtime_of_day
        CMP     AH,00H                  ;Skip simple Get System Time
        JZ      Xtime_of_day
        PUSH    AX
        PUSH    BX
        PUSH    CX
        MOV     BX,INT_1AH_MSG          ;"Int 1AH (Time) AX="
        CALL    SERIAL_PRINT_STRING
        POP     CX
        POP     BX
        POP     AX
        CALL    SERIAL_DISPLAY_REGISTERS      ;Display Registers on serial port display (All registers retained)
```

```
Xtime_of_day:
        TEST   AH,AH                ;AH=0 read system tick time?
        JZ     READ_TICKS           ;go do it if so
        CMP    AH,1                 ;AH = 1set tick time?
        JZ     SET_TICKS
        CMP    AH,4                 ;AH = 4 Get Date?
        JZ     READ_DATE
        JMP    TIME_DONE
                                    ;Set the time
SET_TICKS:
        cli                         ;no interrupts while we set it
        mov     [timlow],dx
        mov     [timhi],cx
        mov     byte [timofl],0
        sti                         ;interrupts ok now
TIME_DONE:
       pop     ds
        iret

READ_TICKS:
       cli                          ;Read the time
        mov     al,[timofl]
        mov     byte [timofl],0
        mov     cx,[timhi]
        mov     dx,[timlow]
        sti
        pop     ds
        iret

READ_DATE:                          ;AH = 4H, Read CMOS RTC
       MOV     CX,2011H             ;We will force century to 2011 for now (must be a valid century)

       MOV     AL,07H               ;Point to Months register
       OUT     RTCSEL,AL
       IN      AL,RTCDATA
       MOV     DH,AL                ;Store BCD month in DH

       MOV     AL,06H               ;Point to Days of the month register
       OUT     RTCSEL,AL
       IN      AL,RTCDATA
       MOV     DL,AL                ;Store BCD day in DL

       XOR     AH,AH
       POP     ds                   ;Get back the origional saved DS at start
       retf    2                    ;Remove the origional status flags on return (remember we got here via an INT)


;-----------  Routines to test Video Board Int 10H Functions out using this BIOS
;            The value of AH, CX, etc are used to control the positioning of characters on then CRT see below

XY_VIDEO:
;;      MOV     BX,VIDIO_TEST_MSG     ;Video Board XY positioning etc tests. Enter AX Value
;       CALL    PRINT_STRING
;       CALL    GET4DIGITS
        MOV     DI,0200H
        PUSH    DI                    ;AX value in DI

;       MOV     BX,ENTER_BX_MSG              ;Enter BX Value
;       CALL    PRINT_STRING
;       CALL    GET4DIGITS
        MOV     DI,0
        PUSH    DI                    ;BX value in DI

;       MOV     BX,ENTER_CX_MSG              ;Enter CX Value
;       CALL    PRINT_STRING
;       CALL    GET4DIGITS
        MOV     DI,0
        PUSH    DI                    ;CX value in DI
```

```
        MOV     BX,ENTER_DX_MSG              ;Enter DX Value
        CALL    PRINT_STRING
        CALL    GET4DIGITS
        PUSH    DI                      ;DX value in DI

        MOV     AX,DI
        CALL    AX_HEXOUT

        MOV     BX,ACTIVATE_INT_MSG     ;Will now activate the Int 10H command
        CALL    PRINT_STRING

        CALL    CICO
        CMP     AL,ESC
        JNZ     XY_VIDEO1
        JMP     IBM_BIOS

XY_VIDEO1:
        POP     DX
        POP     CX
        POP     BX
        POP     AX

        INT     10H                     ;Carry out the Int 10H interrupt

        MOV     CL,'#'
        CALL    CO

        CALL    CICO
        CMP     AL,ESC
        JNZ     XY_VIDEO
        JMP     IBM_BIOS



;*********************************
;
;       Video Output Handler    (Software Int# 10H)
;       Will recognize the following settings:-
;
;Input: AH = 00h       VIDEO - SET VIDEO PARAMATERS
;       AL = Display Mode
;

;Input: AH = 01h       VIDEO - SET TEXT-MODE CURSOR SHAPE
;       CH = cursor start and options (see below)
;       CL = bottom scan line containing cursor (bits 0-4)
;               Bitfields for cursor start and options:
;               7       should be zero
;               6,5     cursor blink.
;                       (00=normal, 01=invisible, 10=erratic, 11=slow).
;                       (00=normal, other=invisible on EGA/VGA)
;               4-0     topmost scan line containing cursor
;Return Nothing
;
;Input: AH = 02h       VIDEO - SET CURSOR POSITION
;       BH = page number (0-3 in modes 2&3. 0-7 in modes 0&1. 0 in graphics modes)
;       DH = row (00h is top)
;       DL = column (00h is left)
;Return:Nothing
;
;Input: AH = 03h       VIDEO - GET CURSOR POSITION AND SIZE
;       BH = page number (0-3 in modes 2&3. 0-7 in modes 0&1. 0 in graphics modes)
;Return:AX = 0000h     (Phoenix BIOS - only)
;       CH = start scan line of cursor
;       CL = end scan line of cursor
;       DH = row (00h is top)
;       DL = column (00h is left)
;
```

```
;Input: AH = 05h       VIDEO - SET PAGE
;       BH = page number (0-3 in modes 2&3. 0-7 in modes 0&1. 0 in graphics modes)
;
;Input: AH = 06h       VIDEO - SCROLL UP WINDOW
;       AL = number of lines by which to scroll up (00h = clear entire window)
;       BH = attribute used to write blank lines at bottom of window
;       CH,CL = row,column of window's upper left corner
;       DH,DL = row,column of window's lower right corner
;Return:Nothing
;
;Input: AH = 07h       VIDEO - SCROLL DOWN WINDOW
;       AL = number of lines by which to scroll down (00h=clear entire window)
;       BH = attribute used to write blank lines at top of window
;       CH,CL = row,column of window's upper left corner
;       DH,DL = row,column of window's lower right corner
;Return:Nothing
;
;Input:AH = 08h       VIDEO - READ CHARACTER AND ATTRIBUTE AT CURSOR POSITION
;       BH = page number (00h to number of pages - 1) (see #00010)
;Return:       AH = character's attribute (text mode only)
;       AL = character
;
;Input: AH = 09h       VIDEO - WRITE CHARACTER AND ATTRIBUTE AT CURSOR POSITION
;       AL = character to display
;       BH = page number (00h to number of pages - 1)
;       BL = attribute (text mode) or color (graphics mode)
;              if bit 7 set in <256-color graphics mode, character is XOR'ed onto screen
;       CX = number of times to write character
;Return:Nothing

;Input: AH = 0Ah       VIDEO - WRITE CHARACTER ONLY AT CURSOR POSITION
;       AL = character to display
;       BH = page number (00h to number of pages - 1)
;       BL = attribute color (graphics mode)
;              if bit 7 set in <256-color graphics mode, character is XOR'ed onto screen
;       CX = number of times to write character
;Return:Nothing
;
;
;
;Input: AH = 0Eh       VIDEO - TELETYPE OUTPUT
;       AL = character to write
;       BH = page number
;       BL = foreground color (graphics modes only)
;Return:Nothing
;
;
;Input: AH = 0Fh       VIDEO - GET VIDEO PARAMATERS
;Return:
;       AH = Number of CRT Columns
;       AL = Display Mode
;       BH = Current page
;
;*********************************

CONOUT: STI                         ;For now just dump character on Propeller Console IO board
        PUSH   DS
        PUSH   AX                   ;<<< Save character (in AL) on stack >>>

        PUSH   AX                   ;Need for Debugging output below
        XOR    AX,AX                ;Set DS to data area for ROM usage in low RAM @ 400H....)
        MOV    DS,AX
        POP    AX

        CMP    byte [DEBUG_FLAG],0  ;Is Debug mode on
        JZ     VIDEO_LOOKUP
        CMP    AH,0EH               ;Skip simple TTY Out debugging
        JZ     VIDEO_LOOKUP
```

```
        PUSH    BX
        MOV     BX,INT_10H_MSG          ;"Int 1AH (VIDEO) AX="
        CALL    SERIAL_PRINT_STRING
        POP     BX
        CALL    SERIAL_DISPLAY_REGISTERS  ;Display Registers on serial port display (All registers retained)


VIDEO_LOOKUP:                           ;Use a lookup table to locate the correct AH option
        CMP     AH,13H                  ;First check range
        JLE     VIDEO_AH_OK

        PUSH    BX
        MOV     BX,VID_PARM_MSG                 ;"Invalid Int 1AH (VIDEO)"
        CALL    SERIAL_PRINT_STRING
        POP     BX
        POP     AX
        CALL    SERIAL_DISPLAY_REGISTERS  ;Display Registers on serial port display (All registers retained)
        POP     DS
        IRET


VIDEO_AH_OK:
        PUSH    BX
        MOV     BL,AH
        MOV     BH,0
        SHL     BX,1                    ;X 2
        ADD     BX,VIDEO_TABLE
        mov     AX,[cs:bx]              ;;Get location of routine CS:[BX]
        POP     BX

        JMP     AX                      ;<------- This is the Video Routine AH call (Note AX on Stack)


VIDEO_TABLE:
        DW      SET_VIDEO_PARMS                 ;<--0   Set Mode
        DW      VIDEO_TBD               ;1      Set Cursor Type
        DW      SET_CURSOR_POS          ;<--2   Set Cursor Positrion
        DW      GET_CURSOR_POS          ;<--3   Get Cursor Position
        DW      VIDEO_TBD               ;4      Read Light Pen
        DW      SET_PAGE                ;<--5   Set page
        DW      SCROOL_UP               ;<--6   Scoll up [AL] lines
        DW      SCROOL_DOWN             ;<--7   Scroll down [AL] lines
        DW      READ_CHAR_ATT           ;<--8   Read Char & Attribute at cursor position
        DW      WRITE_AT_CURSOR_ATT     ;<--9   Write character & attribute at current cursor position
        DW      WRITE_AT_CURSOR                 ;<--0AH Write character at current cursor position
        DW      VIDEO_TBD               ;0BH    Set Color
        DW      VIDEO_TBD               ;0cH    Write Dot
        DW      VIDEO_TBD               ;0dH    Read Dot
        DW      VIDEO_TTY               ;<--0EH ****** Simple TTY mode *****
        DW      GET_VIDEO_PARMS                 ;<--0FH Get Video state
        DW      VIDEO_TBD               ;10H    reserved
        DW      VIDEO_TBD               ;11H    reserved
        DW      VIDEO_TBD               ;12H    reserved
        DW      WRITE_STRING            ;13H    Write String


SET_VIDEO_PARMS:                        ;AH = 0h, AL= Mode     VIDEO - GET VIDEO PRAMATERS
        POP     AX                      ;Balance up stack
        MOV     AX,0003H
        POP     DS
        IRET

                                        ;AH = 02h       VIDEO - SET CURSOR POSITION
SET_CURSOR_POS:                                 ;DH = row (00h is top), DL = column (00h is left)
        POP     AX                      ;Balance up stack
        CALL    VIDEO_SET_CURSOR        ;Set Cursor at [DX]
        POP     DS
        IRET
```

```
                                ;AH = 03h      VIDEO - GET CURSOR POSITION AND SIZE
GET_CURSOR_POS:                         ;DH = row (00h is top), DL = column (00h is left)
        POP     AX              ;Balance up stack
        JNZ     VIDEO_NOT_FINISHED      ;<<<< Ignore for now
                                ;CH = start scan line for cursor
        MOV     CH,06           ;CL = end scan line for cursor
        MOV     CL,07           ;DH = row (00h is top)
        MOV     DH,0            ;DL = column (00h is left)
        MOV     DL,0
        POP     DS
        IRET


                                ;AH = 05h      VIDEO - SET PAGE
SET_PAGE:                       ;BH = page number (0-3 in modes 2&3. 0-7 in modes 0&1. 0 in graphics modes)
        POP     AX              ;We will just return for now
        MOV     BH,0            ;Always
        POP     DS
        IRET


                                ;AH = 06h      VIDEO - SCROLL UP WINDOW
SCROOL_UP:                      ;AL = number of lines by which to scroll up (00h = clear entire window)
        POP     AX              ;BH = attribute used to write blank lines at bottom of window
        OR      CX,CX           ;Start 0,0?, (CH,CL = row,column start, DH,DL = row,column end)
        JZ      SCROLL_UP_0
        PUSH    AX
        PUSH    DX
        MOV     DX,CX           ;At least we will reposition cursor
        CALL    VIDEO_SET_CURSOR        ;Set Cursor at [DX]
        POP     DX
        POP     AX
        JMP     VIDEO_NOT_FINISHED      ;Will ignore DX for now

SCROLL_UP_0:
        OR      AL,AL           ;AL has number of lines to scroll
        JNZ     SCROLL_UP_1
        MOV     AL,40           ;0 for current 40 line CRT
SCROLL_UP_1:
        PUSH    AX
        MOV     AH,ESC
        CALL    FAST_CONOUT
        MOV     AH,'D'
        CALL    FAST_CONOUT
        POP     AX
        DEC     AL
        JNZ     SCROLL_UP_1
        POP     DS
        IRET


                                ;AH = 07h      VIDEO - SCROLL DOWN WINDOW
SCROOL_DOWN:                    ;AL = number of lines by which to scroll up (00h = clear entire window)
        POP     AX              ;BH = attribute used to write blank lines at bottom of window
        OR      CX,CX           ;Start 0,0?, (CH,CL = row,column start, DH,DL = row,column end)
        JZ      SCROLL_DOWN_0
        PUSH    AX
        PUSH    DX
        MOV     DX,CX           ;At least we will reposition cursor
        CALL    VIDEO_SET_CURSOR        ;Set Cursor at [DX]
        POP     DX
        POP     AX
        JMP     VIDEO_NOT_FINISHED      ;Will ignore DX for now

SCROLL_DOWN_0:
        CMP     AL,0            ;AL has number of lines to scroll
        JNZ     SCROLL_DOWN_1
        MOV     AL,40           ;0 for current 40 line CRT
```

```
SCROLL_DOWN_1:
       PUSH    AX
       MOV     AH,ESC
       CALL    FAST_CONOUT
       MOV     AH,'M'
       CALL    FAST_CONOUT
       POP     AX
       DEC     AL
       JNZ     SCROLL_DOWN_1
       POP     DS
       IRET


READ_CHAR_ATT:                      ;AH = 08h      VIDEO - READ CHARACTER AND ATTRIBUTE AT CURSOR POSITION
       POP     AX                   ;AL = character to display
       OR      BH,BH                ;BH = page number (00h to number of pages - 1)
       JNZ     VIDEO_NOT_FINISHED   ;BL = attribute (text mode) or color (graphics mode)

       MOV     AH,07                ;Return: AH = character's attribute (text mode only)
       MOV     AL,0                 ;        AL = character (Not implemented)
       POP     DS
       IRET



WRITE_AT_CURSOR_ATT:                ;AH = 09h      VIDEO - WRITE CHARACTER AND ATTRIBUTE AT CURSOR POSITION
WRITE_AT_CURSOR:                    ;AH = 0Ah      VIDEO - WRITE CHARACTER ONLY AT CURSOR POSITION
       POP     AX                   ;AL = character to display
       OR      BH,BH                ;BH = page number (00h to number of pages - 1)
       JNZ     VIDEO_NOT_FINISHED   ;BL = attribute (text mode) or color (graphics mode)
                                    ;CX = number of times to write character
       MOV     AH,AL                ;Store Char in AH
       PUSH    CX                   ;Save CX
AT_CURSOR1:
       CALL    FAST_CONOUT
       LOOP    AT_CURSOR1           ;Repeat CX times
       POP     CX
       POP     DS
       IRET


VIDEO_TTY:                          ;AH = 0EH     Simple send character to Console
       IN      AL,KEYSTAT           ;Default Propeller or SD SYSTEMS VIDIO BOARD PORT
       AND     AL,4H                ;Is board ready for character
       JZ      VIDEO_TTY
       POP     AX                   ;<-- Get character from AX on stack above
       OUT     KEYOUT,AL
       POP     DS
       IRET


                                    ;AH = 0Fh      VIDEO - GET VIDEO PARAMATERS
GET_VIDEO_PARMS:                    ;Return:
       POP     AX                   ;AH = Number of CRT Columns, AL = Display Mode
       MOV     AX,5003H
       MOV     BX,0                 ;BH = Current page
       POP     DS
       IRET


WRITE_STRING:                       ;AH = 13H      VIDEO - WRITE CHARACTER AND ATTRIBUTE AT CURSOR POSITION
       POP     AX                   ;ES:BP = string pointer
       OR      BH,BH                ;CX = string length
       JNZ     VIDEO_NOT_FINISHED   ;DX = cursor position
                                    ;BH = page number (00h to number of pages - 1)
       OR      AL,AL                ;AL = 0, do not move cursor, AL = 1,  update cursor. AL=3 or 4 add attriute also
       JNZ     W0                   ;Currently only AL=0 & 1 mode implemented
       CALL    VIDEO_SET_CURSOR     ;Set Cursor at [DX]
W0:    OR      CX,CX                ;For zero length string, just return
```

```
        JNZ     W1
        POP     DS
        IRET

W1:     PUSH    BP                      ;Save BP
        PUSH    CX                      ;Save CX
W2:     MOV     AH,[ES:BP]              ;Send string to console
        CALL    FAST_CONOUT
        INC     BP
        LOOP    W2
        POP     CX
        POP     BP
        POP     DS
        IRET



;---------------------- VIDEO SUPPORT ROUTINES

FAST_CONOUT:                            ;Fast send Character (in AH) to Propeller board
        IN      AL,KEYSTAT              ;Propeller or SD SYSTEMS VIDIO BOARD PORT
        AND     AL,4H                   ;Is board ready for character
        JZ      FAST_CONOUT
        MOV     AL,AH
        OUT     KEYOUT,AL
        RET

VIDEO_SET_CURSOR:                       ;Set cursor location to DH & DL
        MOV     AH,ESC
        CALL    FAST_CONOUT             ;Send is VT100 Format "ESC [ row ; column H"
        MOV     AH,'['
        CALL    FAST_CONOUT

        MOV     AL,DH                   ;DH = row (00h is top)
        CALL    HEX_TO_BCD              ;(DX unaltered for below)
        PUSH    AX
        MOV     AH,AL
        ROR     AH,1
        ROR     AH,1
        ROR     AH,1
        ROR     AH,1
        AND     AH,0FH
        ADD     AH,30H                  ;Convert to ASCII
        CALL    FAST_CONOUT             ;Send ROW 10's digit
        POP     AX
        MOV     AH,AL
        AND     AH,0FH                  ;Low nibble
        ADD     AH,30H                  ;Convert to ASCII
        CALL    FAST_CONOUT             ;Send ROW 1's digit

        MOV     AH,';'
        CALL    FAST_CONOUT

        MOV     AL,DL                   ;DL = Column (00h is left)
        CALL    HEX_TO_BCD

        PUSH    AX
        MOV     AH,AL
        ROR     AH,1
        ROR     AH,1
        ROR     AH,1
        ROR     AH,1
        AND     AH,0FH
        ADD     AH,30H                  ;Convert to ASCII
        CALL    FAST_CONOUT             ;Send ROW 10's digit
        POP     AX
        MOV     AH,AL
        AND     AH,0FH                  ;Low nibble
        ADD     AH,30H                  ;Convert to ASCII
```

```
        CALL    FAST_CONOUT             ;Send ROW 1's digit
        MOV     AL,AH

        MOV     AH,'H'
        CALL    FAST_CONOUT
        RET


VIDEO_NOT_FINISHED:
        PUSH    AX
        PUSH    BX
        MOV     BX,VID_PARM_TBD1_MSG    ;"Int 10H Video paramater routine not fully implemented"
        JMP     VIDEO_TBD1

VIDEO_TBD:
        POP     AX                      ;Get origional AH value
        PUSH    AX
        PUSH    BX
        MOV     BX,VID_PARM_TBD_MSG     ;"Int 10H Video paramater not yet implemented"
VIDEO_TBD1:
        CALL    SERIAL_PRINT_STRING
        POP     BX
        POP     AX
        CALL    SERIAL_DISPLAY_REGISTERS  ;Display Registers on serial port display (All registers retained)
        POP     DS
        IRET


; Input: AL = input number   Output: AL = BCD
HEX_TO_BCD:
        pushf                           ; Save flags register
        push cx                         ; Save general-purpose regs
        push dx
        push ax

        sub ah, ah                      ; We don't want a high-order byte so we don't have a divide overflow
        mov dl, 0Ah                     ; Divide by 10
        div dl                          ; Unsigned divide. Quotient in al,remainder in ah.
        mov dl, ah                      ; Save remainder
        mov ah, al                      ; Move quotient (multiple of 10)
        mov cl, 4                       ; and shift into high nibble of al
        shr ax, cl                      ; (8086 imposes stupid restrictions on shr operands)
        or al, dl                       ; Set low nibble of al to remainder
        pop dx                          ; Recover ah (pulling its value into dx first)
        mov ah,dh                       ; restore cx, dx and flags
        pop dx
        pop cx
        popf
        ret                             ; All done.




;*********************************
;
;       Console Input Handler   (Software Interrupt 16H)
;       Return with keyboard buffer character in AL
;
;Input: AH = 00h       KEYBOARD - GET KEYSTROKE
;
;Return:AH = BIOS scan code
;       AL = ASCII character
;                     Note:   On extended keyboards, this function discards any extended keystrokes,
;                             returning only when a non-extended keystroke is available. The BIOS
;                             scan code is usually, but not always, the same as the hardware scan
;                             code processed by INT 09. It is the same for ASCII keystrokes and most
;                             unshifted special keys (F-keys, arrow keys, etc.), but differs for shifted
;                             special keys. Some (older) clone BIOSes do not discard extended keystrokes
```

```
;                           and manage function AH=00h and AH=10h the same.
;
;Input: AH = 01h       KEYBOARD - CHECK FOR KEYSTROKE
;
;Return:ZF set if no keystroke available
;       ZF clear if keystroke available
;       AH = BIOS scan code
;       AL = ASCII character
;                           Note: If a keystroke is present, it is not removed from the keyboard buffer;
;                           however, any extended keystrokes which are not compatible with 83/84- key keyboards
;                           are removed by IBM and most fully-compatible BIOSes in the process of checking
;                           whether a non-extended keystroke is available. Some (older) clone BIOSes do not
;                           discard extended keystrokes and manage function AH=00h and AH=10h the same.
;
;Input: AH = 02h       KEYBOARD - GET SHIFT FLAGS
;
;Return:AL = shift flags (see below)
;       AH destroyed by many BIOSes
;                     Bitfields for keyboard shift flags:-
;                     7      Insert active
;                     6      CapsLock active
;                     5      NumLock active
;                     4      ScrollLock active
;                     3      Alt key pressed (either Alt on 101/102-key keyboards)
;                     2      Ctrl key pressed (either Ctrl on 101/102-key keyboards)
;                     1      left shift key pressed
;                     0      right shift key pressed


;*********************************

CONIN:  sti
        push    ds
        push    bx
        XOR    BX,BX                ;Set DS to data area for ROM usage in low RAM @ 400H....)
        MOV    DS,BX

Xconi0: or      ah,ah                  ;read keyboard?
        jnz     coni1                  ;skip if not

coni0:  mov     al,[chrcnt]          ;any data in buffer?
        test    al,al
        je      coni0                  ;wait for a key
        mov     bx,[bufhd]           ;get buffer address
        mov     al,[bx]                ;character to al
        mov     ah,0                   ;scan code always zero
        inc     bx
        cmp     bx,keybuff+32        ;at end of buffer?
        jl      coni00
        mov     bx,keybuff           ;reset buffer address if so
coni00: mov     [bufhd],bx
        cli                            ;turn off interrupts
        dec     byte [chrcnt]        ;while we adjust count
        sti
        pop     bx
        pop     ds
        iret                           ;return char in AL, AH=0

coni1:  cmp     ah,1                   ;read status?
        jne     coni2                  ;skip if not
        mov     al,[chrcnt]          ;get character count
        test    al,al                  ;Z-flag = availability
        mov     bx,[bufhd]
        mov     al,[bx]                ;character to al
        mov     ah,0                   ;scan code = 0
conirt: pop     bx
        pop     ds
        retf    2                      ;throw away flags
```

```
coni2:  cmp     ah,3                    ;read shift status
        jne     coni3
        mov     al,0                    ;set status to zero
coni3:  pop     bx
        pop     ds
        iret



;*********************************
;
;       Printer Output Handler     (Software Interrupt 17H)
;
;Input: AH = 00h       PRINTER - WRITE CHARACTER
;       AL = character to write
;       DX = printer number (00h-02h)
;
;Return:AH = printer status
;              Bitfields set for printer status:
;              7      not busy
;              6      acknowledge
;              5      out of paper
;              4      selected
;              3      I/O error
;              2-1    unused
;              0      timeout
;
;Input: AH = 01h       PRINTER - INITIALIZE PORT
;       DX = printer number (00h-02h)
;
;Return:AH = printer status (same as above)
;
;Input: AH = 02h       PRINTER - GET STATUS
;       DX = printer number (00h-02h)
;
;Return:AH = printer status (see above)
;
;*********************************

LST_OUT:PUSH   AX                      ;Note we will assume only one printer
        PUSH    CX
        CMP     AH,0
        JNZ     NOT_PLIST              ;AH = 1 or 2;
        CALL    LIST_OUT1              ;AH = 0; Print a character (in AL) on printer
LDONE:  POP     CX
        POP     AX
        XOR     AH,AH                  ;Retur Z  set (and AH = 0 ) if all OK
        IRET                           ;<- Note IRET

NOT_PLIST:                             ;Get List Status
        CALL    LIST_STATUS            ;Must be initilize or a status check. Same ending
        JZ      LDONE                  ;If it matches xxxx0110B we are OK
PSTAT:  TEST    AL,00001000B           ;Test for paper out
        JNZ     PAPER_OUT
        POP     CX                     ;Else just return busy signal
        POP     AX                     ;Just in case return with character in AL
        MOV     AH,00000001B           ;return with timeout bit set
        IRET
PAPER_OUT:
        POP     CX
        POP     AX                     ;Just in case return with character in AL
        MOV     AH,00100000B           ;Flag for paper out
        IRET



LIST_OUT1:                             ;Remember can be called by IBM BIOS section or the monitor section
        MOV     CL,AL                  ;For BIOS interrupt printing character is in AL
LIST_OUT:                              ;Within this monitor character is in CL
```

```
        MOV     CH,0FFH                 ;Check status up to 255 times
LO2:    CALL    LIST_STATUS             ;XXXX0110 if ready
        JZ      LIST_OK
        DEC     CH
        JNZ     LO2
LIST_OK:MOV     AL,0FFH                 ;Setup strobe high to low then high
        OUT     PRINTER_STROBE,AL
        MOV     AL,CL
        OUT     PRINTER_OUT,AL          ;Now Data
        XOR     AL,AL                   ;STROBE FOR CENTRONICS
        OUT     PRINTER_STROBE,AL
        MOV     AL,0FFH                 ;Raise strobe again
        OUT     PRINTER_STROBE,AL
        RET

LIST_STATUS:                            ;Remember can be called by IBM BIOS section or the monitor section
        IN      AL,PRINTER_STATUS
CENSTAT:AND     AL,00001111B            ;XXXX0110 IS READY (BIT 3=PAPER BIT 2=FAULT
        CMP     AL,00000110B            ;BIT 1=SELECT  BIT 0=BUSY
        RET




;*******************************
;
;       BASIC Handler      (Software Interrupt 18h)
;
;*******************************

basic:  PUSH    AX
        PUSH    BX
        PUSH    CX
        MOV     BX,NO_BASIC_MSG         ;Announce we got an BASIC Interrupt
         jmp     NO_INT_SUPPORT


;*******************************
;
;       Equipment Check Handler              (Software Interrupt 11H)
;
;*******************************

equip:  push    ds                        ;save data segment
        XOR     AX,AX                   ;Set DS to data area for ROM usage in low RAM @ 400H....)
        MOV     DS,AX
        mov     ax,[eqflag]
        pop     ds
        iret



;*******************************
;
;       Memory Size Handler     (Software Interrupt 12H)
;       BIOS - GET MEMORY SIZE
;       Return:AX = kilobytes of contiguous memory starting at absolute address 00000h
;                             Note: This call returns the contents of the word at 0040h:0013h;
;                                   in PC and XT, this value is set from the switches on the motherboard
;*******************************

memsiz: push    ds
        XOR     AX,AX                   ;Set DS to data area for ROM usage in low RAM @ 400H....)
        MOV     DS,AX
        mov     ax,[memrsz]
        pop     ds
        iret
```

```
;*********************************
;
;       Interrupt 1Bh Keyboard Break
;
;*********************************

kbd_break:
        PUSH    AX
        PUSH    BX
        PUSH    CX
        MOV     BX,NO_BREAK_MSG               ;Announce we got an BREAK Interrupt
         jmp     NO_INT_SUPPORT



;*********************************
;
;       Interrupt 1Ch (28 Decimal)  User Timer Tic
;
;*********************************

user_timer:
        IRET                          ;Just return




;*********************************
;
;       Comm I/O Handler              (Software Interrupt 14H)
;
;       Note: We will leave it at 19,200 Baud (faster than on origional PC
;
;Input: AH = 00h       SERIAL - INITIALIZE PORT
;       AL = port parameters
;               Paramater Bit Description
;               7-5    data rate (110,150,300,600,1200,2400,4800,9600 bps)
;               4-3    parity (00 or 10 = none, 01 = odd, 11 = even)
;               2      stop bits (set = 2, clear = 1)
;               1-0    data bits (00 = 5, 01 = 6, 10 = 7, 11 = 8)
;       DX = port number (00h-03h)
;Return:AH = line status
;               Bit(s)  Description
;               7      carrier detect
;               6      ring indicator
;               5      data set ready
;               4      clear to send
;               3      delta carrier detect
;               2      trailing edge of ring indicator
;               1      delta data set ready
;               0      delta clear to send
;
;Input: AH = 01h       SERIAL - WRITE CHARACTER TO PORT
;       AL = character to write
;       DX = port number (00h-03h)
;Return:AH bit 7 clear if successful
;       AH bit 7 set on error
;       AH bits 6-0 = port status
;
;Input: AH = 02h       SERIAL - READ CHARACTER FROM PORT
;       AL = 00h (ArtiCom)
;       DX = port number (00h-03h
;       Return:AH = line status
;       AL = received character if AH bit 7 clear


;S100Computers Serial I/O Board Initilization
;Note only SSC A of the Zilog SCC serial ports will be set used (and set to 38,400 Baud initially).
```

```
;Will leave SSC B at 38,400 for speech synthizer (untouched). So DX will be ignored

commio: PUSH    AX                      ;Note we will assume only two serial ports
        PUSH    BX                      ;so DX = 0 or 1
        PUSH    CX
        CMP     AH,0
        JZ      INIT_SIO                ;Initilize serial port
        CMP     AH,1
        JZ      WR_SIO                  ;Write to serial port
        JMP     RD_SIO                  ;Must be AH=2, read from serial port

INIT_SIO:                               ;Program Channel A
        MOV     AH,AL                   ;Store Baud etc in AH
        CMP     DX,0
        JNZ     SIO_DONE                ;Skip serial ports 1,2 & 3

        MOV     AL,04H                  ;Point to WR4
        OUT     ACTL,AL
        MOV     AL,44H                  ;X16 clock,1 Stop,NP
        OUT     ACTL,AL

        MOV     AL,03H                  ;Point to WR3
        OUT     ACTL,AL
        MOV     AL,0C1H                 ;Enable reciever, Auto Enable, Recieve 8 bits
;       MOV     AL,0E1H                 ;Enable reciever, No Auto Enable, Recieve 8 bits (for CTS bit)
        OUT     ACTL,AL

        MOV     AL,05H                  ;Point to WR5
        OUT     ACTL,AL
        MOV     AL,0EAH                 ;Enable, Transmit 8 bits
        OUT     ACTL,AL                 ;Set RTS,DTR, Enable

        MOV     AL,0BH                  ;Point to WR11
        OUT     ACTL,AL
        MOV     AL,56H                  ;Recieve/transmit clock = BRG
        OUT     ACTL,AL

        MOV     AL,0CH                  ;Point to WR12
        OUT     ACTL,AL
;       MOV     AL,40H                  ;Low Byte 2400 Baud (Note can expand later, AH has Baud rate bits)
;       MOV     AL,1EH                  ;Low Byte 4800 Baud
;       MOV     AL,0EH                  ;Low Byte 9600 Baud
;       MOV     AL,06H                  ;Low byte 19,200 Baud
        MOV     AL,02H                  ;Low byte 38,400 Baud <<<<<<<<<<<<<<<
        OUT     ACTL,AL

        MOV     AL,0DH                  ;Point to WR13
        OUT     ACTL,AL
        MOV     AL,00H                  ;High byte for Baud
        OUT     ACTL,AL

        MOV     AL,0EH                  ;Point to WR14
        OUT     ACTL,AL
        MOV     AL,01H                  ;Use 4.9152 MHz Clock.
        OUT     ACTL,AL

        MOV     AL,0FH                  ;Point to WR15
        OUT     ACTL,AL
        MOV     AL,00H                  ;Generate Int with CTS going high
        OUT     ACTL,AL
SIO_DONE:
        POP     CX                      ;We will assume no problem, always!
        POP     BX
        POP     AX
        XOR     AH,AH
        IRET                            ;Note IRET not RET

WR_SIO:                                 ;Write a character to SSC Channel A
        MOV     AH,AL                   ;Store char in AH
```

```
        MOV     CX,256                  ;Will try 256 times, then timeout
WR_SIO1:IN      AL,ACTL                 ;(A0), Is SCC TX Buffer empty
        AND     AL,04H
        JNZ     SENDSER                 ;NZ if ready to recieve character
        LOOP    WR_SIO1
BAD_SER:POP     CX
        POP     BX
        POP     AX
        XOR     AH,AH
        OR      AH,80H                  ;Flag we have a problem
        IRET                            ;Note IRET not RET

SENDSER:MOV     AL,AH
        OUT     ADTA,AL                 ;(A2), Send it
        JMP     SIO_DONE

RD_SIO:                                 ;Read a character from SSC Channel A
        MOV     CX,256                  ;Will try 256 times, then timeout
RD_SIO1:IN      AL,ACTL                 ;(A0), Is SCC TX Buffer empty
        AND     AL,01H
        JNZ     GETSER                  ;NZ if something there
        LOOP    RD_SIO1
        JMP     BAD_SER
GETSER: POP     CX                      ;Get back everything
        POP     BX
        POP     AX
        XOR     AH,AH
        IN      AL,ADTA                 ;(A2), return with data
        IRET                            ;Note IRET not RET



SERIAL_OUT:                             ;Simple write a character to SSC Channel#1 on S100Computers Serial IO Board
        MOV     AH,CL                   ;Store char in AH
        PUSH    CX
        MOV     CX,256                  ;Will try 256 times, then timeout
SERIAL_OUT1:
        IN      AL,ACTL                 ;(A0), Is SCC TX Buffer empty
        AND     AL,04H
        JNZ     SERIAL_OUT2             ;NZ if ready to recieve character
        LOOP    SERIAL_OUT1
        POP     CX
        XOR     AH,AH
        OR      AH,80H                  ;Flag we have a problem
        RET                             ;Note RET not IRET
SERIAL_OUT2:
        MOV     AL,AH
        OUT     ADTA,AL                 ;(A2), Send it
        POP     CX                      ;We will assume no problem, always!
        XOR     AH,AH                   ;Z for no problem
        RET                             ;Note RET not IRET



;*********************************
;
;       Old Cassette Handler          (Software Interrupt 15H)
;       We will use this as a staging point for a far Jump if an extra
;       ROM is discovered duting the BIOS initilization sequence
;       Things like SCSI adaptors etc.
;
;*********************************

CASSETTE:
        push    DS
        PUSH    AX
        XOR     AX,AX                   ;Set DS to data area for ROM usage in low RAM @ 400H....)
        MOV     DS,AX
        CMP     byte [DEBUG_FLAG],0           ;Is Debug mode on
```

```
        POP     AX
        POP     DS
        JZ      Cassette1                       ;If not skip

        PUSH    AX
        PUSH    BX
        PUSH    CX
        MOV     BX,INT_15_MSG           ;"Int 15H (Cassette) AX="
        CALL    SERIAL_PRINT_STRING
        POP     CX
        POP     BX
        POP     AX
        CALL    SERIAL_DISPLAY_REGISTERS        ;Display Registers on serial port display (All registers retained)

Cassette1:
        CMP     AH,41H                  ;Extenal Wait event (Unused)
        JZ      EXT_WAIT

        CMP     AH,0C0H
        JZ      GET_DESCRIPTION_TABLE

        CMP     AH,0C1H                 ;RETURN EXTENDED-BIOS DATA-AREA SEGMENT ADDRESS (PS)
        JZ      EXT_BIOS_DATA

        CMP     AH,88H
        JZ      HIGH_RAM_CHECK

        PUSH    AX
        PUSH    BX
        PUSH    CX
        PUSH    AX
        MOV     BX,CASSETTE_MSG                 ;Announce we got an EXTRA Interrupt
        CALL    PRINT_STRING            ;Send msg pointed to by CS:BX
        POP     AX
        MOV     AL,AH
        CALL    AL_HEXOUT
        MOV     BX,H_MSG_CRLF           ;"H",CR,LF
        CALL    PRINT_STRING            ;Send msg pointed to by CS:BX
        POP     CX
        POP     BX
        POP     AX
        STC                             ;Set carry to indicate INT is not supported
        retf    2                       ;Remove the origional status flags on return (remember we got here via an INT)

 GET_DESCRIPTION_TABLE:                         ;AH=C0H
        MOV     AX,CS                   ;Return pointer with ES:BX
        MOV     ES,AX
        MOV     BX,SYS_TABLE
        XOR     AX,AX
        CLC                             ;Clear carry
        retf    2                       ;Remove the origional status flags on return

 HIGH_RAM_CHECK:                        ;AH=88H
        MOV     AX,0h                   ;Using 8086, so 0H RAM above 1M
        CLC                             ;Set carry
        retf    2                       ;Remove the origional status flags on return

EXT_WAIT:                               ;AH=41H
        STC                             ;Set carry
        retf    2                       ;Remove the origional status flags on return


EXT_BIOS_DATA:                          ;AH= C1H, Extended BIOS Data Area Segment in ES
        STC                             ;Set carry  (Used on PS/2, not needed here)
        retf    2                       ;Remove the origional status flags on return


;----------------- SUPPORT ROUTINES FOR IBM-PC BIOS ---------------------------------------
```

```
dumpreg:                            ;Dump all 8086 registers to screen
        CALL    PRINT_8086_REGISTERS
        CALL    PRINT_SEG_REGISTERS
        RET


SERIAL_DISPLAY_REGISTERS:           ;For Debugging only, Send to serial port Register values of registers with INTs
        PUSH    AX                  ;Save everything
        PUSH    BX
        PUSH    CX
        PUSH    DX

        PUSH    DX                  ;we will display in this order
        PUSH    CX
        PUSH    BX
        PUSH    AX

        MOV     BX,INT_AX_MSG       ;"AX="
        CALL    SERIAL_PRINT_STRING
        POP     AX
        CALL    SERIAL_AX_HEXOUT    ;Get AX

        MOV     BX,INT_BX_MSG       ;"H BX="
        CALL    SERIAL_PRINT_STRING
        POP     AX                  ;Get BX
        CALL    SERIAL_AX_HEXOUT

        MOV     BX,INT_CX_MSG       ;"H CX="
        CALL    SERIAL_PRINT_STRING
        POP     AX                  ;Get CX
        CALL    SERIAL_AX_HEXOUT

        MOV     BX,INT_DX_MSG       ;"H DX="
        CALL    SERIAL_PRINT_STRING
        POP     AX                  ;Get DX
        CALL    SERIAL_AX_HEXOUT

        MOV     BX,H_Msg            ;"H"
        CALL    SERIAL_PRINT_STRING

        POP     DX                  ;Restore everything
        POP     CX
        POP     BX
        POP     AX
        RET



;*****************************************************************
;
;       Data contained in BIOS (Does not get modified, rommable)
;
;       Interrupt vector table for 8259A

vec_tbl_8258A:                                          ;Pointer to 8259A Hardware interrupts used here
        dw      timer           ;Interrupt Base + 0 ;Will use timer
        dw      keyhnd          ;Interrupt Base + 1 ;Will use for keyboard press
        dw      Send_EOI        ;Interrupt Base + 2
        dw      Send_EOI        ;Interrupt Base + 3
        dw      Send_EOI        ;Interrupt Base + 4
        dw      Send_EOI        ;Interrupt Base + 5
        dw      Send_EOI        ;Interrupt Base + 6
        dw      Send_EOI        ;Interrupt Base + 7

vec_tbl_soft_ints:                                  ;Pointer to software interrupts used here
        dw      CONOUT          ;interrupt 10
        dw      equip           ;interrupt 11
        dw      memsiz          ;interrupt 12
        dw      DISKIO          ;interrupt 13
```

```
        dw      commio          ;interrupt 14
        dw      CASSETTE        ;interrupt 15
        dw      CONIN           ;interrupt 16
        dw      LST_OUT         ;interrupt 17
        dw      basic           ;interrupt 18
        dw      BOOT_DOS_INT    ;interrupt 19
        dw      time_of_day     ;interrupt 1A
        dw      kbd_break       ;interrupt 1B
        dw      user_timer      ;interrupt 1C
        dw      video_parm      ;interrupt 1D
        dw      FDISK_3PARM_TBL ;interrupt 1E             ;Default to 5" 360K Disk
        dw      0               ;interrupt 1F
;
;       Miscellaneous Data Area
;
video_parm      dw      0
;
;       Default Floppy Disk Parameters Tables
;       Most are unique to the NEC 765 controller used in the IBM-PC.
;       I do not use them in this BIOS

FDISK_5PARM_TBL db      0DFH        ;For 5" 360K Disks
                db      2
                db      25          ;Time delay for motor
                db      2             ;512 byte sectors
                db      09H           ;sectors per track!
                db      02ah        ;GAP length
                db      0ffh        ;DTL
                db      050h        ;GAP length for format
                db      0f6h        ;Fill byte for format
                db      25          ;Head settle time
                db      4           ;Motor stat time
               db      11           ;length of Table

FDISK_3PARM_TBL db      0AFH        ;For 3" 1.44M Disks
                db      2
                db      25          ;Time delay for motor
                db      2             ;512 byte sectors
                db      12H           ;18 sectors per track
                db      1BH         ;GAP length
                db      0FFH        ;DTL
                db      6CH         ;GAP length for format
                db      0F6H        ;Fill byte for format
                db      0FH         ;Head settle time
                db      8           ;Motor stat time
               db      11           ;length of Table



;       Default Hard Disk Parameters Table:-
;       Custom HDISK:  1024 Cylinders, 15 heads, 63 sectors,  512MB Total

HDISK_PARM_TBL  DW      DOS_MAXCYL  ;0, Max Cylinders
               DB      DOS_MAXHEADS ;2, Max heads (15)
               DW      0000H        ;3, Not used on AT
               DW      0FFFFH       ;5, Start Write Precomp (not used)
               DB      0H           ;7, ECC burst length (not used)
               DB      08H          ;8, "Control Byte" (Bit 7 = disable retrys)
               DB      0H,0H,0H     ;9, Timeouts no used on AT
               DW      0400H        ;A, Landing zone
               DB      DOS_MAXSEC   ;B, Sec/track
               DB      0H,0H,0H,0H  ;C, Reserved

SYS_TABLE       DW      8H          ;Called by INT 15H, AH=C0H called by MSDOS V3+
                DB      0FCH        ;Machine ID Byte
                DB      0           ;Sub model
                DB      0           ;BIOS version
                DB      10H         ;Keyboard Int
                DB      0,0,0
```

```
;       Interrupt messages for checkout
;
msg10   db      13,10,'Int 10h',0
msg11   db      13,10,'Int 11h',0
msg12   db      13,10,'Int 12h',0
msg13   db      13,10,'Int 13h',0
msg14   db      13,10,'Int 14h',0
msg15   db      13,10,'Int 15h',0
msg16   db      13,10,'Int 16h',0
msg17   db      13,10,'Int 17h',0
msg18   db      13,10,'Int 18h',0
msg19   db      13,10,'Int 19h',0
msg1a   db      13,10,'Int 1Ah',0
msg1b   db      13,10,'Int 1Bh',0
msg1c   db      13,10,'Int 1Ch',0
msg1d   db      13,10,'Int 1Dh',0
msg1e   db      13,10,'Int 1Eh',0
msg1f   db      13,10,'Int 1Fh',0
;
xtmsg   db      13,10,'   Exit',0
;
;       end of the IBM_PC BIOS code
;




;=============CORE  SUPPORT ROUTINES ========================================

;       Calculate length difference between DS:SI(end) and ES:DI(start)

CLENGTH:
        MOV     AX,DS                   ;DS has segment of final value
        MOV     CX,ES                   ;ES has segment of start value
        SUB     AX,CX                   ;Check if finish is the next segment up
        JZ      SAME_SEGMENT
        CMP     AX,1000H                ;Max length must be < 64K
        JG      BAD_RANGE
        MOV     AX,0FFFFH
        SUB     AX,DI                   ;Calculate start up to end of segment
        ADD     AX,SI                   ;Add in the part from the next segment up.
        INC     AX                      ;Count = difference +1
        MOV     CX,AX                   ;Return value in CX
        RET

SAME_SEGMENT:
        MOV     CX,SI
        sub     CX,DI
        CMP     CX,0FFFEH
        JZ      BAD_RANGE
        inc     cx                      ;count = difference +1
        ret

BAD_RANGE:
        PUSH    BX
        PUSH    CX
        MOV     BX,RangeErrMsg          ;Range error
        CALL    PRINT_STRING
        jmp     ToMonitor               ;Note this will clean up the stack




;  Send to console the address ES+DI  ;CX Unchanged

SHOW_ADDRESS_ES:
```

```
        push    cx
        mov     ax,es
        mov     cl,12
        shr     ax,cl                   ;Get high nibble down to AL
        call    hexdigout
        MOV     BX,DI
        call    BX_HEXOUT               ;Then next 4 digits in BX
        call    BLANK
        pop     cx
        ret


;       BINARY OUTPUT                   ;Send what is in [al] in bits
AL_BINOUT:                              ;No registers altered (except AL)
        push    cx
        mov     cx,8
binout1: push   cx
        shl     al,1
        jb      bout1
        mov     cl,'0'
        push    ax
        call    CO
        pop     ax
        jmp     binend
bout1:  mov     cl,'1'
        push    ax
        call    CO
        pop     ax
binend: pop     cx
        loop    binout1
        pop     cx
        ret


;       HEXCHK                          ;check for a valid HEX DIGIT
HEX_check:
        sub     al,'0'                  ;convert to binary if ok set carry if problem
        jb      hret
        cmp     al,0ah
        cmc
        jnb     hret
        sub     al,7
        cmp     al,10
        jb      hret
        cmp     al,16
        cmc
hret:   ret



;  Send to console the address DS+SI  ;CX Unchanged

SHOW_ADDRESS_DS:
        push    cx                      ;Same but send upper nibble of ds reg
        mov     ax,ds
        mov     cl,12
        shr     ax,cl                   ;Get high nibble down to AL
        call    hexdigout
        MOV     BX,SI
        call    BX_HEXOUT               ;Then next 4 digits in BX
        call    BLANK
        pop     cx
        ret


;  Send to console the address SS+SI  ;Used (Only) by sector display routine. CX Unchanged

SHOW_ADDRESS_SS:
        push    cx                      ;Same but send upper nibble of ds reg
```

```
        mov     ax,ss
        mov     cl,12
        shr     ax,cl                   ;Get high nibble down to AL
        call    hexdigout
        MOV     BX,SI
        call    BX_HEXOUT               ;Then next 4 digits in BX
        call    BLANK
        pop     cx
        ret

;       Get 8 bit value (2 digits) to AL. (BX, CX & DX Unchanged), terminator in AH - normally 0

GET2DIGITS:
        PUSH    BX
        PUSH    CX
        mov     bx,0                    ;Default to 0H

        call    CICO                    ;1st Console input digit to AL
        cmp     al,'0'                  ;alphanumeric?
        jb      bexit2
        call    HEX_check               ;convert to binary and check it
        jb      err2
        add     bl,al                   ;Move into BX
        mov     cl,4
        shl     bx,cl                   ;shift in last addition to high nibble on BL

        push    BX                      ;Just in case
        call    CICO                    ;2nd Console input digit to AL
        pop     BX

        cmp     al,'0'                  ;alphanumeric?
        jb      bexit2
        call    HEX_check               ;convert to binary and check it
        jb      err2
        add     bl,al                   ;Move into BX
        MOV     AL,BL
        MOV     AH,0                    ;Ret 0 in AH if all OK
        POP     CX
        POP     BX
        ret

err2:   POP     CX                      ;Cleanup stack
        POP     BX
        JMP     ERR                     ;Then normal error exit

bexit2: cmp     al,' '                  ;save terminator, if SP,CR accept only 1 digit
        je      bgood2
        cmp     al,','
        je      bgood2
        cmp     al,CR
        je      bgood2
        cmp     al,ESC
        je      bgood2
        POP     CX                      ;Cleanup stack
        POP     BX
        JMP     ERR                     ;Then normal error exit

bgood2: mov     ah,al                   ;Save SP,"," or CR in AH
        MOV     BH,0
        mov     cl,4
        shr     bx,cl                   ;shift down last addition to low nibble on BL
        MOV     AL,BL
        POP     CX
        POP     BX
        ret


;       Get (up to) 16 bit value (4 digits) to DI. Termination byte in AH
```

```
GET4DIGITS:
       PUSH    BX
       PUSH    CX
       MOV     CX,5                ;4 characters mnaximum + CR
       mov     bx,0
loop4b: call   CICO                ;Console input to AL
       cmp     al,'0'              ;alphanumeric?
       jb      bexit
       push    cx
       mov     cl,4
       shl     bx,cl               ;shift in last addition
       pop     cx
       call    HEX_check           ;convert [AL] to binary and check it
       jb      AddressError
       add     bl,al
       loop    loop4b
       MOV     DI,BX
       POP     CX
       POP     BX
       ret                         ;Will return BX = xxxxH


;      Get (up to) 20 bit parameter. 16 bit value (4 digits) to DI.
;      If 5 digits, first digit entered to ES (BX,CX, DX unaltered)

GET5DIGITS:                        ;Will return ES=000xH, DI = xxxxH
       PUSH    BX
       PUSH    CX
       mov     cx,6                ;Max count of 5 characters + CR
       mov     bx,0                ;So initially ES=0, see below
loopb: call    CICO                ;Console input to AL
       cmp     al,'0'              ;alphanumeric?
       jb      bexit

       push    cx                  ;Save character count
       push    bx                  ;force the highest nibble to ds:
       and     bx,0f000h
       mov     es,bx
       pop     bx
       mov     cl,4
       shl     bx,cl               ;shift in last addition
       pop     cx
       call    HEX_check           ;convert to binary and check it
       jb      AddressError
       add     bl,al
       loop    loopb               ;Do up to 5 characters

bexit: MOV     DI,BX               ;Move data to DI
       cmp     al,' '              ;Terminate with a SP, "," or CR only
       je      bgood
       cmp     al,','
       je      bgood
       cmp     al,CR
       je      bgood
       jmp     ERR
bgood: mov     ah,al               ;Save terminator
       POP     CX                  ;Balance up stack
       POP     BX
       ret

AddressError:
       MOV     BX,AddressErrMsg    ;Range error
       CALL    PRINT_STRING
       jmp     ToMonitor           ;Note this will clean up the stack



;      For debugging display
```

```
DEBUG_AX:
        PUSH    AX
        PUSH    BX
        PUSH    CX
        CALL    AX_HEXOUT
        POP     CX
        POP     BX
        POP     AX
        RET


;   Display ALL 8086 registers

PRINT_8086_REGISTERS:               ;Print AX,BX,CX,DX,SI & DI Registers
        PUSHF                       ;Will print all on CRT on one line followed by a CRLF
        PUSH    AX
        PUSH    BX
        PUSH    CX
        PUSH    SI                  ;Will pop from stack from here
        PUSH    DI
        PUSH    DX
        PUSH    CX
        PUSH    BX
        PUSH    AX

        MOV     BX,AXMSG            ;[AX]=
        CALL    PRINT_STRING
        POP     AX
        CALL    AX_HEXOUT

        MOV     BX,BXMSG            ;[BX]=
        CALL    PRINT_STRING
        POP     AX
        CALL    AX_HEXOUT

        MOV     BX,CXMSG            ;[CX]=
        CALL    PRINT_STRING
        POP     AX
        CALL    AX_HEXOUT

        MOV     BX,DXMSG            ;[DX]=
        CALL    PRINT_STRING
        POP     AX
        CALL    AX_HEXOUT

        MOV     BX,DIMSG            ;[DI]=
        CALL    PRINT_STRING
        POP     AX
        CALL    AX_HEXOUT

        MOV     BX,SIMSG            ;[SI]=
        CALL    PRINT_STRING
        POP     AX
        CALL    AX_HEXOUT

        MOV     BX,H_MSG            ;Final H
        CALL    PRINT_STRING
        POP     CX
        POP     BX
        POP     AX
        POPF
        RET



;   Display 8086 Segment registers

PRINT_SEG_REGISTERS:                ;Print current RAM loction of the stack
        PUSHF                       ;Will print all on CRT on one line followed by a CRLF
```

```
        PUSH    AX
        PUSH    BX
        PUSH    CX
        MOV     BX,SSMSG              ;[SS]=
        CALL    PRINT_STRING
        MOV     AX,SS
        CALL    AX_HEXOUT

        MOV     BX,SPMSG              ;[SP]=
        CALL    PRINT_STRING
        MOV     AX,SP
        SUB     AX,10                 ;Adjust because we saved stuff first
        CALL    AX_HEXOUT

        MOV     BX,CSMSG               ;[CS]=
        CALL    PRINT_STRING
        MOV     AX,CS
        CALL    AX_HEXOUT

        MOV     BX,DSMSG               ;[DS]=
        CALL    PRINT_STRING
        MOV     AX,DS
        CALL    AX_HEXOUT

        MOV     BX,ESMSG               ;[ES]=
        CALL    PRINT_STRING
        MOV     AX,ES
        CALL    AX_HEXOUT

        MOV     BX,BPMSG               ;[BP]=
        CALL    PRINT_STRING
        MOV     AX,BP
        CALL    AX_HEXOUT

        MOV     BX,H_MSG              ;Final H
        CALL    PRINT_STRING
        POP     CX
        POP     BX
        POP     AX
        POPF
        RET

;       CHECK FOR ^S or ESC  AT CONSOL
CTRL_CHECK:
        call    CSTS
        cmp     al,0
        jz      ctlexit
        call    CICO
        cmp     al,'S'-40h            ;^S will pause
        jnz     ctlcchek              ;possibly ^C
xwait:  call    CSTS
        cmp     al,0
        jz      xwait
        ret
ctlcchek:
        cmp     al,ESC                ;ESC will abort
        jz      ERR
ctlexit:ret



;       SEND CRLF with an ESC at keyboard check
CRLF_CHECK:
        push    cx
        push    bx
        call    CTRL_CHECK            ;Will jump to err if ESC
        mov     cl,CR
        call    CO
        mov     cl,LF
```

```
        call    CO
        pop     bx
        pop     cx
        ret


;       SIMPLE SEND CRLF
CRLF:   push    cx
        push    bx
        mov     cl,CR
        call    CO
        mov     cl,LF
        call    CO
        pop     bx
        pop     cx
        ret


;       PRINT A BLANK SPACE
BLANK:  push    cx
        mov     cx,1
        call    TABS
        pop     cx
        ret


;       TABS                            ;[cx] = number of spaces
TABS:   push    cx
        mov     cl,' '
        call    CO
        pop     cx
        loop    TABS
        ret


;       ERROR ABORT ROUTINE
ERR:    MOV     BX,ERR_MSG              ;Invalid Command (or code not yet done)
        CALL    PRINT_STRING
        jmp     ToMonitor



;       BX_HEXOUT                       ;bx output as 4 hex digits
BX_HEXOUT:                              ;No registers altered
        push    ax
        mov     al,bh
        call    AL_HEXOUT
        mov     al,bl
        call    AL_HEXOUT
        pop     ax
        ret


;       AX_HEXOUT                       ;output the 4 hex digits in [AX]
AX_HEXOUT:                              ;No registers altered
        PUSH    AX
        MOV     AL,AH
        CALL    AL_HEXOUT
        POP     AX
        CALL    AL_HEXOUT
        RET



;       AL_HEXOUT                       ;output the 2 hex digits in [AL]
AL_HEXOUT:                              ;No registers altered (except AL)
        push    cx
        push    ax
        mov     cl,4                    ;first isolate low nibble
```

```
        shr     al,cl
        call    hexdigout
        pop     ax
        call    hexdigout               ;get upper nibble
        pop     cx
        ret

hexdigout:
        and     al,0fh                  ;convert nibble to ascii
        add     al,90h
        daa
        adc     al,40h
        daa
        mov     cl,al
        call    CO
        ret


;   ROUTINE TO PRINT A STRING   CS:BX = START OF STRING  $ or 0 = FINISH

PRINT_STRING:
        push    cx
print1: mov     al,[CS:bx]              ;Note this routine does NOT assume DS = CS here.
        inc     bx                      ;By using the CS over-ride we will always have
        cmp     al,'$'                  ;a valid pointer to messages at the end of this monitor
        jz      print2
        cmp     AL,0                    ;Also terminate with 0's
        JZ      print2
        mov     cl,al
        call    CO
        jmp     print1
print2: pop     cx
        ret

;   ROUTINE TO PRINT A STRING  TO S100Computers Serial Port #1  BX = START OF STRING  $ or 0 = FINISH
;   This routine is used mainly for Debugging the IBM BIOS section. No registers altered

SERIAL_PRINT_STRING:
        push    AX
        push    cx
sprint1:mov     al,[CS:bx]              ;Note this routine does NOT assume DS = CS here.
        inc     bx                      ;By using the CS over-ride we will always have
        cmp     al,'$'                  ;a valid pointer to messages at the end of this monitor
        jz      sprint2
        cmp     AL,0
        JZ      sprint2
        mov     cl,al
        call    SERIAL_OUT              ;Send to serial port #1
        jmp     sprint1
sprint2:pop     cx
        pop     AX
        ret


;       SERIAL_AX_HEXOUT                ;Output the 4 hex digits in [AX] to serial port (used for debugging)
SERIAL_AX_HEXOUT:                       ;No registers altered
        PUSH    AX
        MOV     AL,AH
        CALL    SERIAL_AL_HEXOUT
        POP     AX
        CALL    SERIAL_AL_HEXOUT
        RET


;       SERIAL_AL_HEXOUT                ;output the 2 hex digits in [AL]
SERIAL_AL_HEXOUT:                       ;No registers altered (except AL)
        push    cx
        push    ax
        mov     cl,4                    ;first isolate low nibble
```

```
        shr     al,cl
        call    SERIAL_hexdigout
        pop     ax
        call    SERIAL_hexdigout      ;get upper nibble
        pop     cx
        ret

SERIAL_hexdigout:
        and     al,0fh                ;convert nibble to ascii
        add     al,90h
        daa
        adc     al,40h
        daa
        MOV     AH,01H                ;AH=char output, char in AL
        MOV     DX,0
         int     14H                  ;Serial out Handler   (Software Interrupt 14H)
        ret


;<<<<<<<<<<<<<<<<<<<<<< MAIN CONSOL OUTPUT ROUTINE >>>>>>>>>>>>>>>>>>>>>>>>

CO:     IN      AL,KEYSTAT            ;SD SYSTEMS VIDIO BOARD PORT
        AND     AL,4H
        JZ      CO
        MOV     AL,CL
        CMP     AL,BELL               ;IS IT A BELL
        JZ      BELL1
        CMP     AL,0H                 ;SD BOARD CANNOT TAKE A NULL
        JNZ     LXX3
        RET
LXX3:   CMP     AL,ESC                ;Don't actully echo an ESC character
        JNZ     LX2
        RET
LX2:    OUT     KEYOUT,AL
        MOV     AL,CL                 ;MAKE SURE TO RETURN WITH [AL] CONTAINING CHAR
        RET

BELL1:  MOV     AL,06H                ;SEND A BELL
        OUT     KEYOUT,AL
        MOV     AL,1FH
        CALL    DELAY
        MOV     AL,CL
        OUT     KEYOUT,AL
        RET


DELAY:  DEC     AL                    ;GENERAL COUNT DOWN TIME DELAY
        JNZ     LX4
        RET                           ;LENGTH SET IN [A]
LX4:    PUSH    AX
        MOV     AL,05H
MORE:   DEC     AL
        PUSH    AX
        XOR     AL,AL
MORE2:  DEC     AL
        JNZ     MORE2
        POP     AX
        JNZ     MORE
        POP     AX
        JMP     DELAY


;<<<<<<<<<<<<<<<<<< MAIN CONSOL STATUS ROUTINE >>>>>>>>>>>>>>>>>>>>>>>>>>>>>

CSTS:   IN      AL,KEYSTAT
        TEST    AL,02H
        JZ      NONE
        XOR     AL,AL
        DEC     AL
```

```
        RET                         ;RETURN WITH 0FFH IN [A] IF SOMETHING

NONE:   XOR     AL,AL
        RET



;<<<<<<<<<<<<<<<<<<<< MAIN CONSOL INPUT ROUTINE >>>>>>>>>>>>>>>>>>>>

CI:     CALL    CSTS                ;Wait until something is there
        JZ      CI
        IN      AL,KEYIN
        AND     AL,7FH
        RET


;<<<<<<<<<<<<<<< CONSOLE INPUT WITH ECHO ON CONSOLE + LC->UC  <<<<<<<<<<<<

CICO:   CALL    CI                  ;Char -> AL
        AND     AL,7FH
        JZ      BAD_CHAR            ;No Nulls
        CMP     AL,','              ;Allow "," character
        JZ      CIC1
        CMP     AL,CR               ;ACCEPT ONLY CR,LF,SP
        JZ      CIC1
        CMP     AL,LF
        JZ      CIC1
        CMP     AL,SPACE
        JZ      CIC1
        CMP     AL,ESC              ;Also ESC
        JZ      CIC1

        CMP     AL,'0'
        JB      BAD_CHAR
        CMP     AL,':'              ;Allow 0-9
        JB      CIC1
        CMP     AL,'A'
        JB      BAD_CHAR            ;do not allow : to @
        CMP     AL,'['              ;Is upper case A to Z
        JB      CIC1
        CMP     AL,'a'
        JB      BAD_CHAR
        CMP     AL,'{'
        JB      UPPER_CASE
        JMP     BAD_CHAR
UPPER_CASE:
        AND     AL,5FH              ;THIS CONVERTS ALL LC->UC
CIC1:   PUSH    AX
        PUSH    CX
        MOV     CL,AL
        CALL    CO                  ;DISPLAY ON CONSOLE
        POP     CX
        POP     AX
        RET
;
BAD_CHAR:
        MOV     AL,BELL             ;SEND BELL TO INDICATE BAD DATA
        CALL    CIC1
        MOV     AL,'?'              ;SEND ? TO INDICATE BAD DATA
        JMP     CIC1


SPEAKOUT:
        MOV     AL,0H               ;Will try 256 times, then timeout
SOUT1:  PUSH    AX
        IN      AL,BCTL             ;Are we ready for a character
        AND     AL,04H
        JNZ     SENDS
        POP     AX
```

```
        DEC     AL
        JNZ     SOUT1
        RET
SENDS:  POP     AX
        MOV     AL,CL
        OUT     BDTA,AL                 ;Send it
        RET



;SPEAKTOMM THIS IS A ROUTINE TO SEND A STRING TO TALKER [BX] AT STRING
STOMM:  MOV     AL,[BX]
        CMP     AL,'$'                  ;Terminate with "$" or 0
        JZ      STOMM1
        OR      AL,AL
        JZ      STOMM1
        MOV     CL,AL
        CALL    SPEAKOUT
        INC     BX
        JMP     STOMM
STOMM1: MOV     CL,CR                   ;MUST END WITH A CR
        JMP     SPEAKOUT


POO:    RET                             ;NO PUNCH OUTPUT AT THE MOMENT
RI:     MOV     AL,1AH                  ;NO READER AT THE MOMENT
        RET


NOT_DONE_WARNING:
        mov     bx,TO_BE_DONE           ;Signon notice
        call    PRINT_STRING            ;Note up until now stack was not used
        RET


;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

                ;End of the bios code


;************************************ DATA SECTION ******************************************

                        ;MAIN MENU COMMAND BRANCH TABLE

ctable          dw      MAP             ;A              ;Display Memory Map
                dw      ERR             ;B
                DW      MMENU_FBOOT_DOS         ;C              ;LOAD MS-DOS from 5" Floppy ((No debugging))
                dw      DISPLAY_RAM     ;D              ;Display Memory contents
                dw      ERR             ;E
                dw      FILL            ;F              ;Fill memory contents
                dw      GOTO            ;G              ;Jump to a SEG:ADDRESS location
                dw      HEXMATH         ;H              ;Add & Subtract two Hex numbers
                dw      TIMEC           ;I              ;Get current time
                dw      TEST_RAM        ;J              ;Test RAM
                dw      KCMD            ;K              ;Display this menu
                dw      TEST_8259       ;L              ;Test 8259A hardware
                dw      MOVE            ;M              ;Move memory
                dw      MYIDE           ;N              ;Sub-menu to test/diagnose IDE Board
                dw      ERR             ;O
                dw      MMENU_HBOOT_DOS         ;P              ;LOAD MS-DOS from HDISK (No debugging)
                dw      QUERY           ;Q              ;Query In or Out to a port
                dw      INPORTS         ;R              ;Display all active 8086 INPUT ports
                dw      SUBSTITUTE      ;S              ;Substitute byte values in RAM
                dw      DISPLAY_ASCII   ;T              ;Display ASCII in RAM
                dw      UPDATE          ;U              ;Round out seconds to 00
                dw      VERIFY          ;V              ;Verify two memory regions are the same
                dw      JMP_500H        ;W              ;For CPM86/DOS Boot,  jump to exactly 500H in RAM
                dw      IBM_BIOS        ;X              ;IBM-PC Sub menu
                DW      PATCH           ;Y              ;Quick patch to move RAM 4000H-9000H to f4000h & JUMP to it
```

```
                dw      Z80             ;Z              ;Return back to Z80 master


                        ;IDE COMMAND BRANCH TABLE

IDE_TABLE       DW  SET_DRIVE_A                 ; "A"  Select Drive A
                DW  SET_DRIVE_B                 ; "B"  Select Drive B
                DW  ERR                 ; "C"  LOAD CPM (If present)
                DW  DISPLAY             ; "D"  Sector contents display:- ON/OFF
                DW  RAMCLEAR            ; "E"  Clear RAM buffer
                DW  FORMAT              ; "F"  Format current disk
                DW  ERR                 ; "G"  Restore backup
                DW  ERR                 ; "H"  Backup partition
                DW  NEXT_SECT           ; "I"  Next Sector
                DW  PREV_SECT           ; "J"  Previous sector
                DW  IDE_LOOP            ; "K"
                DW  SET_LBA             ; "L"  Set LBA value (Set Track,sector)
                DW  ERR                 ; "M"
                DW  SPINDOWN            ; "N"  Power down hard disk command
                DW  DRIVE_ID            ; "O"  Show current Drive ID
                DW  ERR                 ; "P"
                DW  LBA_DISPLAY_TEST    ; "Q"  Check the LBA mode HEX display on the IDE board is working correctly
                DW  READ_SEC            ; "R"  Read sector to data buffer
                DW  SEQ_SEC_RD          ; "S"  Sequental sec read and display contents
                DW  ERR                 ; "T"
                DW  SPINUP              ; "U"  Power up hard disk command
                DW  N_RD_SEC            ; "V"  Read N sectors
                DW  WRITE_SEC           ; "W"  Write data buffer to current sector
                DW  N_WR_SEC            ; "X"  Write N sectors
                DW  COPY_AB             ; "Y"  Copy Drive A to Drive B
                DW  VERIFY_AB           ; "Z"  Verify Drive A:= Drive B:

                        ;IBM_BIOS COMMAND BRANCH TABLE

IBM_TABLE       dw      MENU_TIMER_TEST ;A
                dw      ERR             ;B
                dw      MENU_FBOOT_DOS ;C  Boot MS-DOS from 5" floppy (Allow Debugging)
                dw      DEBUG_ON_OFF   ;D
                dw      MENU_KEY_TEST   ;E
                dw      MENU_CO_TEST    ;F
                dw      MENU_BUFF_IO   ;G
                dw      XY_VIDEO       ;H
                dw      ERR            ;I
                dw      ERR            ;J
                dw      ERR            ;K
                dw      ERR            ;L
                dw      ERR            ;M
                dw      ERR            ;N
                dw      MENU_SIO_TEST  ;O
                dw      MENU_HBOOT_DOS ;P   Boot MS-DOS from HDISK (Allow Debugging)
                dw       CHS_DISPLAY_TEST ;   "Q"  Check the CHS mode HEX display on the IDE board is working correctly
                dw      ERR            ;R
                dw      FSEQ_5RD_TEST  ;S
                dw      FSEQ_3RD_TEST  ;T
                dw      HSEQ_RD_TEST   ;U
                dw      ERR            ;V
                dw      HSEC_RW_TEST   ;W   Hard Disk Sector Read/Write test using INT 13H
                dw      ERR            ;X
                dw      DUMP_B_SEC     ;Y   Display Floppy Boot sector info
                dw      DUMP_MBR       ;Z   Display the Hard Disk MBR information


SIGNON          db      SCROLL,QUIT,BELL,CR,LF,LF,'8086 Monitor (At F000:8000H) V6.4 (John Monahan, 9/16/2011). Time=$'
CLEANUP         DB      CR,LF,BELL,'>$'
SMSG            DB      'THE 8086 ROM MONITOR VERSION 6.4 IS NOW ACTIVE$'
TO_BE_DONE      DB      CR,LF,'Code not done yet!',CR,LF,'$'
AXMSG           DB      '[AX]=$'
BXMSG           DB      'H  [BX]=$'
CXMSG           DB      'H  [CX]=$'
```

```
DXMSG           DB      'H  [DX]=$'
DIMSG           DB      'H  [DI]=$'
SIMSG           DB      'H  [SI]=$'
SSMSG           DB      '[SS]=$'
SPMSG           DB      'H  [SP]=$'
CSMSG           DB      'H  [CS]=$'
DSMSG           DB      'H  [DS]=$'
ESMSG           DB      'H  [ES]=$'
BPMSG           DB      'H  [BP]=$'
H_MSG           DB      'H$'
AddressErrMsg   DB      CR,LF,'Address paramater error.$'
RangeErrMsg     DB      CR,LF,'Paramater range error.$'

MAIN_MENU       DB      CR,LF
                DB      'A=Memmap  C=DOS(F)     D=Disp RAM   F=Fill RAM   G=Goto',CR,LF
                DB      'H=Math    I=Time       J=Test RAM   K=Menu       L=Test 8259',CR,LF
                DB      'M=Move    N=IDE Menu   Q=Ports      P=DOS(H)     R=In Ports',CR,LF
                DB      'S=Subs    T=Disp ASCII U=Adj Time   V=Verify     W=JMP to 500H',CR,LF
                DB      'X=PC-BIOS Z=Z80        Y=PATCH',CR,LF,'$'
DIFF_Header_Msg         DB      CR,LF,'Source  Value     Destination Value     Difference$'
MATCHES_OK      DB      CR,LF,'Both RAM locations match$'
PORTS8_MSG      DB      CR,LF,'Input Ports (0-256 Ports,8 bits)',CR,LF,'$'
PORTS16_MSG     DB      CR,LF,LF,'Input Ports (0-64K Ports,16 bits)',CR,LF,'$'
MORE_MSG        DB      CR,LF,'Continue ? (Y/N) $'
MSG30           DB      CR,LF,'Adj :- $'
MSG12T          DB      '    $'
MSG16T          DB      '/20$'
JMSG            DB      CR,LF,'Continous RAM test. Enter start and ending address.',CR,LF,'$'
STARTJMSG       DB      CR,LF,'Starting RAM test. Hit ESC to abort',CR,LF,'$'
RAM_Test_Count  DB      CR,'RAM test loop count = $'
TMMSG           DB      CR,LF,'Time:- $'
GET_SEG_MSG     DB      CR,LF,'Enter Segment (xxxxH)->$'
GET_OFFS_MSG    DB      CR,LF,'Enter Offset (xxxxH)->$'
MATH_MSG        DB      CR,LF,'Hex Math. Enter xxxxH,xxxxH:- $'
MATH_HEADER     DB      CR,LF,'Sum   Difference',CR,LF,'$'
PIC_SIGNON      DB      '8086 test of Interrupts on PIC/RTC board',CR,LF
                DB      'Press any key to start...$'
CRLFMSG         DB      CR,LF,'$'
TrapIntMSG      DB      'Trap int. detected at a non-assigned location.$'
TrapFFIntMSG    DB      'Trap int. detected at 0FFH in RAM.$'
DebugTrapMSG    DB      'Trap int. detected Software Debug INT at 0CH in RAM.$'
Int0MSG         DB      'V0 $'
Int1MSG         DB      'V1 $'
Int2MSG         DB      'V2 $'
Int3MSG         DB      'V3 $'
Int4MSG         DB      'V4 $'
Int5MSG         DB      'V5 $'
Int6MSG         DB      'V6 $'
Int7MSG         DB      'V7 $'



IDE_SIGNON0     DB      CR,LF,LF,'IDE Disk Drive Test Menu Routines.$'
IDE_SIGNON4     DB      '(L) Set LBA value       (R) Read Sector to Buffer (W) Write Buffer to Sector',CR,LF
                DB      '(D) Set Sec Display $'
IDE_SIGNON1     DB      'ON  $'
IDE_SIGNON2     DB      'OFF $'
IDE_SIGNON3     DB      '(S) Sequential Sec Read   (F) Format Disk',CR,LF
                DB      '(V) Read N Sectors      (X) Write N Sectors       (I) Next Sector',CR,LF
                DB      '(J) Previous Sector     (U) Power Up              (N) Power Down',CR,LF
                DB      '(A) Select Drive A      (B) Select Drive B        (E) Clear Sector Buffer',CR,LF
                DB      '(Y) Copy A: to B:       (Z) Verify A: = B:        (Q) IDE Board LBA Display Test',CR,LF
                DB      '(ESC) Back to Main Menu',CR,LF
                DB      LF,'Current settings:- $'
IDE_MENU        DB      'Enter a Command:- $'

IDE_HARDWARE    DB      CR,LF,'Initilizing IDE Drive hardware.$'
INIT_DR_OK      DB      CR,LF,'IDE Drive Initilized OK.',CR,LF,LF,'$'
INIT_ERROR      DB      CR,LF,'Initilizing Drive Error.',CR,LF,'$'
```

```
ID_ERROR       DB      'Error obtaining Drive ID.',CR,LF,'$'
DRIVE2_ERR     DB      CR,LF,'Second IDE drive was not initilized.$'
msgmdl         DB      CR,LF,'Drive/CF Card Information:-',CR,LF
               DB      'Model: $'
msgsn          DB      'S/N:   $'
msgrev         DB      'Rev:   $'
msgcy          DB      'Cylinders: $'
msghd          DB      ', Heads: $'
msgsc          DB      ', Sectors: $'
msgCPMTRK      DB      'CPM TRK = $'
msgCPMSEC      DB      ' CPM SEC = $'
msgLBA         DB      '  (LBA = 00$'
MSGBracket     DB      ')$'
H_Msg          DB      'H$'
H_MSG_CRLF     DB      'H',CR,LF,'$'
NotDoneYet     DB      CR,LF,'Command Not Done Yet$'
CONFIRM_WR_MSG DB      CR,LF,LF,BELL,'Will erase data on the current drive, '
               DB      'are you sure? (Y/N)...$'
msgrd          DB      'Sector Read OK',CR,LF,'$'
msgwr          DB      'Sector Write OK',CR,LF,'$'
SET_LBA_MSG    DB      'Enter CPM style TRK & SEC values (in hex).',CR,LF,'$'
SEC_RW_ERROR   DB      'Drive Error, Status Register = $'
ERR_REG_DATA   DB      'Drive Error, Error Register = $'
ENTERRAM_SECL  DB      'Starting sector number,(xxH) = $'
ENTERRAM_HEAD  DB      'Starting HEAD number,(xxH) = $'
ENTERRAM_FTRKL DB      'Enter Starting Track number,(xxH) = $'
ENTERRAM_TRKL  DB      'Track number (LOW byte, xxH) = $'
ENTERRAM_TRKH  DB      'Track number (HIGH byte, xxH) = $'
ENTER_HEAD     DB      'Head number (01-0f) = $'
ENTER_COUNT    DB      'Number of sectors to R/W (xxH) = $'
ENTERRAM_DMA   DB      'Enter DMA Adress (Up to 5 digits, xxxxxH) = $'
OVER_COUNT_10  DB      CR,LF,'1 & 9 sectors. Only!',CR,LF,'$'
OVER_COUNT_19  DB      CR,LF,'1 & 18 sectors. Only!',CR,LF,'$'
DRIVE_BUSY     DB      'Drive Busy (bit 7) stuck high.   Status = $'
DRIVE_NOT_READY DB     'Drive Ready (bit 6) stuck low.  Status = $'
DRIVE_WR_FAULT DB      'Drive write fault.    Status = $'
UNKNOWN_ERROR  DB      'Unknown error in status register.   Status = $'
BAD_BLOCK      DB      'Bad Sector ID.    Error Register = $'
UNRECOVER_ERR  DB      'Uncorrectable data error.  Error Register = $'
READ_ID_ERROR  DB      'Error setting up to read Drive ID',CR,LF,'$'
SEC_NOT_FOUND  DB      'Sector not found. Error Register = $'
INVALID_CMD    DB      'Invalid Command. Error Register = $'
TRK0_ERR       DB      'Track Zero not found. Error Register = $'
UNKNOWN_ERROR1 DB      'Unknown Error. Error Register = $'
CONTINUE_MSG   DB      CR,LF,'To Abort enter ESC. Any other key to continue. $'
FORMAT_MSG_A   DB      'Fill disk sectors of Disk [A] with 0E5H$'
FORMAT_MSG_B   DB      'Fill disk sectors of Disk [B] with 0E5H$'
ATHOME_MSG     DB      CR,LF,BELL,'Already on Track 0, Sector 0$'
AT_START_MSG   DB      CR,LF,BELL,'Already at start of disk!$'
AT_END_MSG     DB      CR,LF,BELL,'At end of Disk!$'
FILL_MSG       DB      CR,LF,'Sector buffer area cleared to 0000....$'
READN_MSG      DB      CR,LF,'Read N sectors from disk.$'
WRITEN_MSG     DB      CR,LF,'Write N sectors to disk.$'
DiskCopyMsg    DB      CR,LF,'Copy CPM Partition on Drive A to Drive B (Y/N)? $'
DiskVerifyMsg  DB      CR,LF,'Will verify CPM Partition on Drive A to Drive B.$'
CopyDone       DB      CR,LF,'Disk Copy Done.$'
VERIFY_ERR     DB      CR,LF,BELL,'Verify Error. $'
VerifyDone     DB      CR,LF,'Disk Verify Done.$'
CR_To_Continue DB      CR,LF,'Hit any key to continue.$'
OK_CR_MSG      DB      ' OK',CR,LF,'$'
COPY_ERR       DB      CR,LF,BELL,'Sector Copy Error.$'
CURRENT_MSG_A  DB      '   Current Drive = [A]',CR,LF,'$'
CURRENT_MSG_B  DB      '   Current Drive = [B]',CR,LF,'$'
FORMAT_ERR     DB      CR,LF,BELL,'Sector Format Error$'
ERR_MSG        DB      CR,LF,BELL,'Invalid Command (or code not yet done)',CR,LF,'$'


IBM_SIGNON_MSG DB      CR,LF,LF,'IBM PC BIOS Initilizing$'
IBM_MENU1      DB      CR,LF,LF,'IBM-PC BIOS Test Menu.     (Debug Flag = $'
```

```
IBM_MENU_ON     DB      'ON)',CR,LF,'$'
IBM_MENU_OFF    DB      'OFF)',CR,LF,'$'
IBM_MENU2       DB      'A=Timer Test             C=MS-DOS Boot (Floppy)  D=Toggle Debug Flag',CR,LF
                DB      'E=Key Press Test         F=Consol Out Test       G=Buffered Consol I/O Test',CR,LF
                DB      'O=Out to Serial Port     P=MS-DOS Boot (HDISK)   S=5" Floppy Sector RD Test',CR,LF
                DB      'T=3" Floppy Sec RD Test  U=HDisk Sec RD Test     W=HDisk Sector R/W Test',CR,LF
                DB      'Q=CHS LED Display Test   Y=Floppy Boot Sec Info  Z=Hard Disk MBR Info',CR,LF
                DB      'H=Int 10H Video Board   (ESC) Back to Main Menu',CR,LF,LF,'>$'

NMI_MSG         DB      CR,LF,BELL,'Recieved an NMI Interrupt.',CR,LF,'$'
ZFDC_FAIL_MSG   DB      CR,LF,BELL,'ZFDC Board failed to initilize',CR,LF,'$'
ZFDC_OK_MSG     DB      CR,LF,'ZFDC Board Initilize OK',CR,LF,'$'
PIC_INIT_MSG    DB      CR,LF,'Initilizing 8259A PIC (Port 20H, Ints 0 & 1 only)$'
RESET_FAIL_MSG DB       CR,LF,BELL,'Reset of floppy drive failed.',CR,LF,'$'
HRESET_FAIL_MSG         DB      CR,LF,BELL,'Reset of Hard Disk drive failed.',CR,LF,'$'
BOOT_FAIL_MSG   DB      CR,LF,BELL,'Boot sector read on floppy drive failed.',CR,LF,'$'
BOOT_OK_MSG     DB      CR,LF,'Boot Sector Loader Signature Valid (AA55H).',CR,LF,'Now doing Far JMP to
0000:7C00H',CR,LF,LF,'$'
READ_ERR_MSG    DB      CR,LF,BELL,'Floppy Sector Read Error. Error returned = $'
WRITE_ERR_MSG   DB      CR,LF,BELL,'Floppy Sector Write Error. Error returned = $'
HREAD_ERR_MSG   DB      CR,LF,BELL,'HDisk Multi-Sector Read Error.$'
HWRITE_ERR_MSG DB       CR,LF,BELL,'HDisk Multi-Sector Write Error.$'
HOME_ERR_MSG    DB      CR,LF,BELL,'Disk reset error.',CR,LF,'$'
NO_BASIC_MSG    DB      CR,LF,BELL,'BASIC Handler error.',CR,LF,'$'
NO_BREAK_MSG    DB      CR,LF,BELL,'Keyboard Break Handler error.',CR,LF,'$'
NO_COMM_MSG     DB      CR,LF,BELL,'Serial Communications Handler error.',CR,LF,'$'
CASSETTE_MSG    DB      CR,LF,BELL,'Cassette Handler error. AH=$'
FBOOT_DOS_MSG   DB      CR,LF,'Booting MS-DOS from 5" Floppy Disk$'
HBOOT_DOS_MSG   DB      CR,LF,'Booting MS-DOS from HARD Disk$'
KEY_TEST_MSG    DB      CR,LF,'Software Interrupt driven Keyboard Input test',CR,LF,'$'
IN_CHAR_MSG     DB      CR,LF,'Type one character: $'
GOT_CHAR_MSG    DB      '<--- Character via software Int 16H.$'
CO_TEST_MSG     DB      CR,LF,'Software Interrupt driven Console/Video out test',CR,LF,'$'
OUT_CHAR_MSG    DB      '<--- Character Recieved$'
TIMER_TEST_MSG DB       CR,LF,'8259A Interrupt driven Timer Test$'
TIMER_DATA_MSG DB       CR,LF,'Enter any key to read timer data. (ESC to Abort)$'
TIMER_LOW_MSG   DB      CR,LF,'Timer Low Value = $'
TIMER_HIGH_MSG DB       'H',CR,LF,'Timer High Value = $'
TIMER_OFLOW_MSG         DB      'H',CR,LF,'Timer Overflow Value = $'
BUFF_TEST_MSG   DB      CR,LF,'Type keyboard characters as fast as you can!',CR,LF,'$'
SQRDHFAILMSG    DB      CR,LF,BELL,'Error reading sectors from HARD disk',CR,LF,'$'
SQRD5FAILMSG    DB      CR,LF,BELL,'Error reading sectors from 5" Floppy disk',CR,LF,'$'
SQRD3FAILMSG    DB      CR,LF,BELL,'Error reading sectors from 3" Floppy disk',CR,LF,'$'
SQRDHOKMSG      DB      CR,LF,'Read sectors from HARD disk OK!',CR,LF,'$'
SQRD5OKMSG      DB      CR,LF,'Read sectors from 5" 360K Floppy disk OK!$'
SQRD3OKMSG      DB      CR,LF,'Read sectors from 3" 1.44M Floppy disk OK!',CR,LF,'$'
DEBUG_SET_MSG   DB      CR,LF,'Set Debug level (0 = OFF, 1 = INTs only, 2 = +HDisk Info, 3 = +Floppy Info) $'
DUMP_ON1_MSG    DB      CR,LF,'Debug flag ON (Level 1)',CR,LF,'$'
DUMP_ON2_MSG    DB      CR,LF,'Debug flag ON (Level 2)',CR,LF,'$'
DUMP_ON3_MSG    DB      CR,LF,'Debug flag ON (Level 3)',CR,LF,'$'
DUMP_OFF_MSG    DB      CR,LF,'Debug flag OFF',CR,LF,'$'
SEC_5RD_MSG     DB      CR,LF,'Sequentially read sectors from 5" Floppy disk',CR,LF,'$'
SEC_3RD_MSG     DB      CR,LF,'Sequentially read sectors from 3" Floppy disk',CR,LF,'$'
SEC_HDRD_MSG    DB      CR,LF,'Read sector from HARD Disk test using Int 13H',CR,LF,'$'
ROM_ERR_MSG     DB      CR,LF,BELL,'Checksum error found in ROM (C8000H-F4000H). Got AL= $'
NOT_DONE_MSG    DB      CR,LF,BELL,'Code Not done yet',CR,LF,'$'
INVALID_AH_FMSG         DB      CR,LF,BELL,'Invalid AH paramater in Floppy Handler. AH=$'
INVALID_AH_HMSG         DB      CR,LF,BELL,'Invalid AH paramater in HDisk Handler. AH=$'
SIO_TEST_MSG    DB      CR,LF,'Serial Port (A3H) Test.'
                DB      CR,LF,'Enter any text. (ESC to stop).',CR,LF,'>','$'
SIO_INIT_ERR    DB      CR,LF,'Serial Port Initilization Error. AH=$'
SIO_ERR         DB      CR,LF,'Error sending character to Serial Port. AH=$'
INT_13F_MSG     DB      CR,LF,'Int 13H (Floppy)$'
INT_40F_MSG     DB      CR,LF,'Int 40H (<--Floppy)$'
INT_13H_MSG     DB      CR,LF,'Int 13H (*HDisk*)$'
INT_AX_MSG      DB      ' AX=$'
INT_BX_MSG      DB      'H BX=$'
INT_CX_MSG      DB      'H CX=$'
INT_DX_MSG      DB      'H DX=$'
```

```
INT_1AH_MSG    DB      CR,LF,'Int 1AH (Time)$'
INT_10H_MSG    DB      CR,LF,'Int 10H (VIDEO)$'
INT_15_MSG     DB      CR,LF,'Int 15H (Cassette)$'
SIDE_REQUEST_MSG DB    CR,LF,'Read from Side A or Side B (A/B) $'
SIDE_A_SET_MSG DB      CR,LF,'Will read from Side A',CR,LF,'$'
SIDE_B_SET_MSG DB      CR,LF,'Will read from Side B',CR,LF,'$'
FORMAT_ERR_MSG DB      CR,LF,'ZFDC Track Format error $'
CMOS_CLOCK_MSG DB      CR,LF,BELL,'CMOS RTC Error',CR,LF,'$'
CMOS_DATA0_MSG DB      CR,LF,'CMOS DATA:Mins (BCD)/Hex = $'
CMOS_DATA1_MSG DB      ' Hours (BCD)/Hex = $'
PATCH_MSG      DB      CR,LF,'Moving RAM 2100H-6FFFH to F2000H. JMPF to F2000H',CR,LF,'$'
SECTOR_NUM_MSG DB      CR,LF,'Starting requested Sector = $'
HRESET_OK_MSG  DB      CR,LF,'Reset of Hard Disk drive OK.',CR,LF,'$'
RD_ERR_MSG     DB      CR,LF,BELL,'Sector READ Error Returned.'
               DB      CR,LF,'Head = $'
TRACK_MSG      DB      'H   Track = $'
SEC_MSG        DB      'H   Sector = $'
WR_ERR_MSG     DB      CR,LF,BELL,'Sector WRITE Error Returned.'
               DB      CR,LF,'Head = $'
ESC_END_MSG    DB      CR,LF,'Press ESC to Abort. Any other key to continue $'
SEQAT500       DB      CR,LF,LF,'Sector(s) loaded @ 0000:500H.'
               DB      CR,LF,'Head = $'
CR_TAB_MSG     DB      CR,LF,'                    $'
LBA_TEST_MSG   DB      CR,LF,'Test for LBA on IDE drive #2 (using LBA mode)$'
CHS_TEST_MSG   DB      CR,LF,'Test for CHS on IDE drive #2 (using non-LBA mode)$'
TRKL_NUM       DB      CR,LF,'Enter TRACK/Cylinder number (LOW byte, xxH) = $'
TRKH_NUM       DB      CR,LF,'Enter TRACK/Cylinder number (HIGH byte, xxH) = $'
HEAD_NUM       DB      CR,LF,'Enter HEAD number,(0-FH, 0xH) = $'
SECTOR_NUM     DB      CR,LF,'Enter SECTOR number (xxH) = $'
CHECK_DISPLAY_MSG DB   CR,LF,'Check the IDE Board HEX display.$'
BOOT_3RD_MSG   DB      CR,LF,'Display Floppy Boot Sector Information.',CR,LF,'$'
DRIVE_SELECT_MSG DB    CR,LF,'Please select floppy disk (A or B) $'
BOOT_INFO_FAIL_MSG DB  CR,LF,'Error reading Boot disk sector.$',CR,LF
BOOT_INFOOKMSG DB      CR,LF,'Floppy Boot Sector Information:-',CR,LF,LF,'$'

JMP_MSG        DB       '     Boot JMP Vector',CR,LF,'$'
NAME_MSG       DB       '   OEM Name',CR,LF,'$'
BYTES_MSG      DB       '       Bytes/Sec',CR,LF,'$'
CLUSTER_MSG    DB       '         Sec/Cluster',CR,LF,'$'
RES_MSG        DB       '       Reserved Sectors',CR,LF,'$'
FATS_MSG       DB       '         FATS',CR,LF,'$'
ROOT_MSG       DB       '       Root Dir Entries',CR,LF,'$'
SECTORS_MSG    DB       '       Sectors',CR,LF,'$'
MEDIA_MSG      DB       '         Media Byte',CR,LF,'$'
FAT_SEC_MSG    DB       '       FAT Sectors',CR,LF,'$'
SEC_TRK_MSG    DB       '       Sectors/Track',CR,LF,'$'
HEADS_MSG      DB       '       Heads',CR,LF,'$'
HIDDEN_MSG     DB       '   Hidden Sectors',CR,LF,'$'
HUGE_MSG       DB       '   Huge Sectors',CR,LF,'$'
DRIVE_NO_MSG   DB       '         Drive #',CR,LF,'$'
RESERVED_MSG   DB       '         Reserved',CR,LF,'$'
BOOT_SIG_MSG   DB       '         Boot Signature',CR,LF,'$'
VOL_ID_MSG     DB       '          Volume ID',CR,LF,'$'
VOLUME_MSG     DB       '       Volume Label',CR,LF,'$'
SYS_TYPE_MSG   DB       '  File Sys Type',CR,LF,LF,'$'
NO_MBL_MSG     DB      CR,LF,BELL,'Invalid Floppy Boot Loader Signature detected',CR,LF,'$'
BOOT_MBR_MSG   DB      CR,LF,'Reading Hard Disk MBR sector, (C=0, H=0, S=1)',CR,LF,'$'
BOOT_MBR_FAIL_MSG DB   CR,LF,BELL,'Error reading Hard Disk MBR sector',CR,LF,'0'
MBR_INFOOKMSG  DB      'Hard Disk Master Boot Record:-',CR,LF,'$'

DISK_SIG_MSG   DB      '   Hard Disk Signature',CR,LF,'$'
NULS_MSG       DB      '       Usually Nulls (Optional)',CR,LF,LF,'$'
PT1_MSG        DB      ' First Partition Table',CR,LF,'$'
PT2_MSG        DB      ' Second Partition Table',CR,LF,'$'
PT3_MSG        DB      ' Third Partition Table',CR,LF,'$'
PT4_MSG        DB      ' Forth Partition Table',CR,LF,'$'
SIGNATURE_MSG  DB      '      LBR Signature Word',CR,LF,'$'
STATUS_MSG     DB      ' Status Byte, $'
STLBA_MSG      DB      ' Start CHS Address, $'
```

```
PAR_TYPE_MSG   DB      ' Partition Type, $'
ECHS_MSG       DB      ' End CHS Address',CR,LF,'$'
SLB_MSG        DB      ' Start LBA Address, $'
ELBA_MSG       DB      ' End LBA Address',CR,LF,LF,'$'
CYL_MSG        DB      'H Cyl=$'
HD_MSG         DB      '   Head=$'
BRAC1_MSG      DB      'H ($'
OF_MSG         DB      'H of $'
BRAC2_MSG      DB      'H)',CR,LF,'$'
DRIVE1_MSG     DB      '  on Drive A',CR,LF,'$'
DRIVE2_MSG     DB      '  on Drive B',CR,LF,'$'
HRW_TEST_MSG   DB      CR,LF,'Hard Disk Sector R/W test using INT 13H',CR,LF
               DB      CR,LF,'>>> WARNING <<< Data on Disk will be overwritten. Continue...(Y/N) $'
ONE_MOMENT_MSG DB      CR,LF,'One moment while IDE Drive is being initilized',CR,LF,'$'
ASK_WR_MSG     DB      CR,LF,'Write data back to Hard Disk...(Y/N)$'
START_DATA_MSG DB      'H  Start of Data =',CR,LF,'$'
SEC_READ_OK    DB      CR,LF,'Sector(s) read OK',CR,LF,'$'
SEC_BACK_OK    DB      CR,LF,'Sector(s) written back OK',CR,LF,'$'
LOOP_ESC_MSG   DB      CR,LF,'Will now continously R/W sectors until ESC is entered. CR to start',CR,LF,'$'

VIDIO_TEST_MSG DB      CR,LF,'Int 10H tests for control of Video Board I/O',CR,LF
               DB      CR,LF,'Enter value of [AX], (xxxxH) $'
ENTER_BX_MSG   DB      CR,LF,'Enter value of [BX], (xxxxH) $'
ENTER_CX_MSG   DB      CR,LF,'Enter value of [CX], (xxxxH) $'
ENTER_DX_MSG   DB      CR,LF,'Enter value of [DX], (xxxxH) $'
ACTIVATE_INT_MSG DB    CR,LF,'Enter CR to implement the INT 10H interrupt, (ESC to Abort) $'

VID_PARM_TBD_MSG DB    CR,LF,'Int 10H Video paramater not yet implemented'
               DB      CR,LF,'               ','$'
VID_PARM_TBD1_MSG DB   CR,LF,'Int 10H Video paramater not fully completed'
               DB      CR,LF,'               ','$'
VID_PARM_MSG   DB      CR,LF,'Invalid Int 10H Video paramater',CR,LF,'$'

       DB      '<------ END OF 8086 Monitor V6.4 (John Monahan. Sept 16,2011)  '
       DB      '                                              '


       %if     MONITOR_ROM

       TIMES 0FFF0H-($-$$) DB 0

       JMP     word  0F000H:INIT     ;Far Jump to F000H:INIT (Start of this monitor)

       TIMES 9         DB 0

       DB      0FCH                  ;"Model Number" IBM PC/AT (At FFFEH)
       DB      0H                    ;Skip Checksum
       %endif


;---------------------- LOW RAM VARIABLES (Used mainly by PC-BIOS section) ----------------------

       absolute       2H*4
NMIint:        resw      2           ;Non-maskable interrupt location (8H)

       absolute       5H*4
PrintScreen:   resw      2           ;Not yet implemented

       absolute       8H*4          ;Location for our hardware interrupts (20H, Same as IBM-PC hardware)
Start8259A_Ints        resw      2           ;    8   Timer Tic          TIMER        \
                                     ;    9   Keypressed         KEYHND        \
                                     ;    A   Reserved           DUMMY_RETURN   \
                                     ;    B   Comm Hardware      DUMMY_RETURN    \Normal location for
                                     ;    C   Comm Hardware      DUMMY_RETURN    /IBM hardware interrupts
                                     ;    D   Disk Hardware      DUMMY_RETURN   /
                                     ;    E   Diskette Hardware  DUMMY_RETURN  /
                                     ;    F   Printer Hardware   DUMMY_RETURN /
```

```
        absolute        10H*4
CRTINT          resw    2               ;Software interrupt used in this BIOS (and IBM-PC/AT)

        absolute        13H*4
MAIN_DISK_VEC   resw    2               ;Disk (Hard & Floppy) software interrupt
        absolute        (13H*4)+2
MAIN_DISK_SEG   resw    2               ;Disk software interrupt segment (Normally this CS)


        absolute        1EH*4           ;Pointer to Floppy Disk paramaters table
FDISK_PARMS     resw    2               ;On MSDOS Boot, this points to the boot Floppy Disk Variables Table
        absolute        (1EH*4)+2
FDISK_PARMS_SEG         resw    2               ;Disk Variables Table segment (Normally this CS)


        absolute        40H*4
OLD_DISK_VEC    resw    2               ;Pointer to the origional PC Floppy Disk Int Vector (relocated because of HDISK)
        absolute        (40H*4)+2
OLD_DISK_SEG    resw    2               ;New Floppy Disk software interrupt segment (Normally this CS)

        absolute        41H*4
HDISK_PARMS     resw    2               ;Pointer to HARD DISK #1 paramater table
        absolute      (41H*4)+2
HDISK_PARMS_SEG         resw    2               ;HARD DISK paramater table segment (Normally this CS)

        absolute        46H*4
HDISK2_PARMS    resw    2               ;Pointer to HARD DISK #2 paramater table
        absolute      (46H*4)+2
HDISK2_PARMS_SEG resw   2               ;HARD DISK paramater table segment (Normally this CS)



         absolute       400H            ;Low RAM data area (set the same as for IBM-PC BIOS)

RS232_BASE      resw    4               ;Addresses for RS232 Adaptors (if any)
PRINTER_BASE    resw    4               ;Address of Printers (if any)

eqflag          resw    1               ;equipment flag (two bytes)
MFG_TST         resb    1               ;MFG initilization flag (not used)
memrsz          resw    1               ;memory size
expram          resw    1               ;expansion ram size

keyboard_CTL1   resb    1
keyboard_CTL2   resb    1
chrcnt          resb    1               ;characters in buffer (Alt_Keypad on PC)

bufhd           resw    1               ;keyboard buffer head  (40:1AH)
buftl           resw    1               ;keyboard buffer tail  (40:1CH)
keybuff         resw    16              ;keyboard data buffer  (40:1EH
kbend           resw    1               ;end of buffer
chrmax  equ      32                     ;buffer length
                                        ;\
                                        ; \
                                        ;  Keyboard buffer area
                                        ; /
                                        ;/
        absolute        43EH

SEEK_STATUS     resb    1               ;Seek status  (40:3EH)
CURRENT_HEAD    resb    1               ;On IBM PC, motor status (40:3FH)
CURRENT_DRIVE   resb    1               ;On IBM PC, motor count (40:40H)

IBM_DISK_STATUS         resb    1               ;returned disk status  (40:41H)
                                        ;
DMA_OFFSET      resw    1               ;DMA offset address for controller  (On PC this area is used by FDC)
DMA_SEGMENT     resw    1               ;DMA segmant address for controller
CURRENT_SECTOR resb     1
CURRENT_TRACK   resb    1
CURRENT_TRACK_HIGH resb         1
```

```
       absolute       449H           ;Video board paramaters

CRT_MODE       resb   1              ;Video Display Mode (40:49H)
CRT_COLS       resw   1
CRT_LEN        resw   1
CRT_START      resw   1
CURSOR_POSN    resw   1              ;IBM has 8! (we will use area for disk info)(Page 1, @ 40:50H)

ZFDC_ERR_CODE  resb   1              ;Error code returned by ZFDC controller in AH for error

               resw   4              ;Spare (to align variables below or extra pages if needed above)

DEBUG_FLAG     resb   1              ;If not Zero display track, side, sector etc info during disk R/W
ZFDC_INIT_FLAG         resb   1              ;Flag to indicate ZFDC board has been initilised
SECTORS_TO_DO  resb    1             ;Number of sectors to transfer in current operation
SECTORS_DONE   resb    1             ;Number actually transferred
VERIFY_FLAG    resb   1              ;0 for normal sector reads, NZ, if just secor verifys required

CURSOR_TYPE    resw   1              ;Cursor Type (40:60H)
ACTIVE_PAGE    resb   1
ADDR_6845      resw   1
CRT_MODE_SET   resb   1
CRT_PALLETTE   resb   1

IO_ROM_INIT    resw   1              ;Anchor location to implement extra ROMS
IO_ROM_SEG     resw   1
INTR_FLAG      resb   1

timlow         resw    1             ;timer low count  (40:6CH) for timer
timhi          resw    1             ;timer high count
timofl         resb    1             ;timer overflow flag

BIOS_BREAK     resb   1              ;Bit 7 = 1 if break key pressed  (40:71H)
RESET_FLAG     resw   1              ;1234H if KB reset underway (40:72H)


                                     ;Additional data stores on IBM-AT
DISK_STATUS1   resb   1              ;(40:74H) Hard Disk Data Areas
HF_NUM         resb   1
CONTROL_BYTE   resb   1
PORT_OFF       resb   1

PRINT_TIM_OUT  resw   2              ;Printer & RS232 Time-out variables
RS232_TIM_OUT  resw   2

BUFFER_START   resw   1              ;Additional Keyboard data area (on IBM-AT)
BUFFER_END     resw   1

       absolute       48BH           ;;To keep same a AT

LAST_RATE      resb   1              ;(40:88H) Addditional Floppy data

HF_STATUS      resb   1              ;(40:8CH)  Additional HDisk data area
HF_ERROR       resb   1
HF_INT_FLAG    resb   1
HF_CNTRL       resb   1

DSK_STATE      resw   2              ;(40:90H) Additional Diskette Area (must be 0 for MS-DOS 4.01)
DSK_TRK        resb   3

KB_FLAG_2      resb   1              ;Additional keyboard LED flag

                                     ;(40:98H)  RTC additional data area
USER_FLAG      resw   1              ;offset address of user wait flag
USER_FLAG_SEG  resw   1              ;segment       "      "       "
RTC_LOW        resw   1              ;user RTC low word
```

```
RTC_HIGH       resw    1               ;user RTC high word
RTC_WAIT_FLAG  resb    1               ;user wait active

       absolute        0B800H          ;IBM-PC Color Video Board RAM area

Video_Ram      resw    16384 *2        ;Video RAM area (IF, Lomas or other S100 PC Video bopard is used)

                                       ;High RAM data area used (only) for the IDE Board diagnostic functions
                                       ;These variables will normally be accessed as SS:[BP]
                                       ;This is used by the IDE drive diagnostic commands ONLY. We need an area
                                       ;to store buffers in RAM. We cannot assume 1MG of RAM (i.e. just below this
                                       ;monitor at F000:8000H. At initilization the monitor will determine the top
                                       ;segment of available RAM. It will place the IDE RAM buffers/variable at
                                       ;xxxx:7000H using SS:BP to access them. In a full 1MG system they will
       %if     MONITOR_ROM             ;be at F000:7000H, 1K below ROM
               absolute 7000H
       %else
               absolute 1000H          ;For test/debug system where this BIOS is at F000:2000H
       %endif


RAM_DRIVE_SEC  resw      1             ;This area will be in top of RAM well below stack (used by IDE Board sections)
RAM_DRIVE_TRK  resw      1
RAM_DRIVE_HEAD resw      1
RAM_DRIVE_COUNT        resw      1
RAM_SEC        resw      1
RAM_TRK        resw      1
DELAYStore     resw      1
RAM_DMA        resw      1
RAM_DMA_STORE  resw      1
SECTOR_COUNT   resw      1
CURRENT_IDE_DRIVE resw    1
DISPLAY_FLAG   resw      1

       %if     MONITOR_ROM             ;be at F000:7000H, 1K below ROM
               absolute 7200H
       %else
               absolute 1200H          ;For test/debug system where this BIOS is at F000:2000H
       %endif

IDE_Buffer     resb      200H          ;512 Byte buffer for IDE Sector R/W
IDE_Buffer2    resb     200H           ;512 Byte buffer for IDE Sector Verify

               absolute     7c00h      ;0000:7c00H

DOS_BOOT_LOC:  resw      1             ;<---MS-DOS/FreeDOS BOOT LOCATION

               absolute     7c00h+510  ;0000:7dfeH

DOS_BOOT_SIGNATURE:    resw      1              ;<---MS-DOS Valid Boot Signature Location (0AA55H)
```