```
;      SBC-MON.Z80  This is a stripped down version of the main MASTER-Z80 monitor program for my system.
;      It was modified from version 5.1 of that program (starting on 9/23/2015). Please read and see that
;      code located at (http://s100computers.com/Software%20Folder/Master/Master.htm) before changing this
;      monitor code.
;
;      Assemble and SLR's Z80ASM Assembler (Can also use the Cromemco Assembler)
;      Use:- Z80ASM SBC-MON FH
;
;      NOTE. This board utilizes an 8K 28C64 EEPROM (or 27C64 UV-ROM).
;
;      To assemble under windows...
;      Load Altair.EXE in Windows CMD box
;      do cpm3
;      I:
;      I:>Submit SBC-MON
;
;      SBC-MON.HEX  is written back to the same windows folder that the PC file "altair.exe" is in.

;      Programming an EEPROM for the SBC-Z80 Board with a PROM burner is fairly straightfoward.
;      Using a 28C64 EEPROM and a Wellon VP-280 or VP290 Programmer
;      For a monitor at F000H-FFFFH:-

;      Make sure the ORG is E000H (BASE_PORT).
;      Load the SBC-Z80.HEX file
;      Clear Buffer Options:- 00
;      Load Buffer Address :- 0000
;      From File address use:- E000H
;      File Size use:- 2000H
;
;      Recent History...
;
;      V1.0  9/23/2015   Started with the modified MASTER.Z80 V5.1 Monitor. Removed the page switching meny option.
;      V1.1  10/7/2015   Removed many of the MASTER.Z80 other S100 board routines. Added IDE/CF Card Diagnostics
;      V1.11 10/10/2015  Splice in IDE diagnostic routines from MYIDE.ASM
;      V1.12 10/11/2015      Switch to 8K size
;      V2.2  10/18/2015  Spliced in IDE.ASM routines, Seperate menu for IDE/CF-Card diagnostics
;
;
;      Bugs:-
;      None at this stage.
;      Be carefull where the DMA buffer is located. The default is high up at C000H in RAM. Multi-sector reads
;      can overwrite the stack and RAM variables at D1F00.  If you can, relocate to LOW RAM with the "D" IDE Menu command
;      Currently the BOOT CPM section is untested/incomplete.
;
;
```

```
FALSE       EQU   0
TRUE        EQU   NOT FALSE


BASE_PORT   EQU   30H         ;Base port set with SW101. (Note by using 30H we can use software already on CF-CARDS that
                              ;assume the presence of the dual S100 bus IDE/CF-Board in the system)


ROM_BASE    EQU   0E000H            ;Start or EPROM Location normally 0E000H. (Assume a 28C64)
                              ;Note can test running in low RAM. Assemble on PC. Download SBC-MON.COM with
                              ;the "X" XModem command. Load at 100H. Afterwards J100.

RAM_BASE    EQU   0C000H            ;Default location of RAM buffer area for IDE/CF card diagnostic routines
                              ;Can be changed with "D" IDE menu command


      ORG   ROM_BASE          ;<--------<<<<<< LOCATION OF START OF MONITOR (First part)



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).
NO_ENH      EQU 17H                   ;Turns off whatever is on
FAST  EQU   10H               ;High speed scrool
ESC   EQU   1BH
DELETE      EQU   7FH
BACKS EQU   08H
CLEAR EQU   1AH               ;TO CLEAR SCREEN
RST7  EQU   38H               ;RST 7 (LOCATION FOR TRAP)
NN    EQU   0H                ;[I] INITIAL VALUE
SOH   EQU   1                 ; For XModem etc.
EOT   EQU   4
ACK   EQU   6
NAK   EQU   15H
;
STARTCPM    EQU   100H        ;LOCATION WHERE CPM LOADER WILL BE PLACED FOR COLD BOOT
CPM_BOOT_COUNT    EQU   12          ;Allow up to 12 CPM sectors for CPMLDR
```

```
MAXSEC            EQU   3DH         ;Sectors per track for CF my Memory drive, Kingston CF 8G. (For CPM format, 0-3CH)
                                    ;This 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 CPM3 system because
                                    ;my Seagate drive has 3D sectors/track. Don't want different CPM3.SYS files around
                                    ;so this program as is will also work with a Seagate 6531 IDE drive



; BIT MAP OF IOBYTE BASE_PORT + 6H:- X X X X  X X X X           (if xx111100= CONFIG, will use onboard USB chip for
Console I/O)
;                                   | | | |  | | | |.............  1=CONSOLE IN DATA from Console IO board
;                                   | | | |  | | |...............  1=CONSOLE OUT DATA to Console IO board
;                                   | | | |  | |.................  0=CONSOLE OUT DATA also to Printer (unused)
;                                   | | | |  |........Unused
;                                   | | | |...........Unused
;                                   | | |.........................  1= IOBYTE not active/implemented (output to Console IO
Board)
;                                   | |
;                                   | |...............USB status, data available to recieve on this Computer
;                                   |.................USB Status, data CAN be written to PC
;
;
IOBYTE            EQU   BASE_PORT+6H      ;See above
USB_DATA    EQU   BASE_PORT+4H      ;PORT FOR DLP-USB Controller chip (Note different from chip on the S100 Serial I/O
Board).
USB_STATUS  EQU   BASE_PORT+6H      ;Status port for USB port ( bits 6,7 of IOBYTE Port)
USB_RXE           EQU   80H         ;RXF#, If Bit 7 = 0, data available to recieve on this Computer
USB_TXE           EQU   40H         ;TXE#  If Bit 6 = 0 data CAN be written to PC
RAM_BANK    EQU   BASE_PORT+6 ;Output to this board bit 0 to switch lower 32K of RAM


;*****************************************************************************************************
;
;            EQUATES FOR OTHER POSSIBLE BOARDS IN THE S100 BUS SYSTEM
;               (Note. If the board is not present the code will ignore the hardware)
;
;*****************************************************************************************************


;-------------- S100Computers PROPELLER CONSOLE_IO (OR SD SYSTEMS VIDIO BOARD) FOR CONSOLE INPUT & OUTPUT

S100_CONSOL_STATUS      EQU   0H          ;Note will utilize this board if IOBYTE bits 0 & 1 are ZERO (or bit 5 is 1).
S100_CONSOL_IN          EQU   1H
S100_CONSOL_OUT         EQU   1H
```

```
;-------------- S100Computers Parallel Ports I/O Board ------------------------------------------------------

ST8C4       EQU   TRUE        ;TRUE if S100_Parallel_IO Board.  False if IMSAI PIO Boad (or ignored).

IF    ST8C4                   ;If S100_Parallel_IO Board for Printer output
PRINTER_CTRL      EQU   0C2H        ;ST8C4 Control Port
PRINTER_STATUS    EQU   0C1H        ;ST8C4 Status port
PRINTER_OUT EQU   0C0H        ;ST8C4 Data port
PRINTER_ST_LOW    EQU   0DH         ;OUT STROBE LOW
PRINTER_ST_HIGH   EQU   0CH         ;OUT STROBE HIGH
ELSE
PRINTER_STATUS    EQU   5           ;IN, IMSAI PIO Board PARRELL PORT
PRINTER_OUT EQU   5           ;OUT
PRINTER_STROBE    EQU   4           ;OUT
ENDIF


;-------------- S100Computers Serial I/O Board ------------------------------------------------------------------

S100_BASE_PORT    EQU   0A1H        ;For XModem communication routines on S100 bus Serial board (if present)
MODEM_CTL_PORT    EQU   S100_BASE_PORT     ;A1H (Note modem I/O will be on CHANNEL A. Speaker on CHANNEL B
MODEM_DATA_PORT   EQU   S100_BASE_PORT+2 ;A3H

MODEM_SEND_MASK   EQU   4
SEND_READY  EQU   4           ;VALUE WHEN READY
MODEM_RECV_MASK   EQU   1
RECV_READY  EQU   1           ;BIT ON WHEN READY
MODEM_ERR_LIMIT   EQU   8           ;Max number of Modem serial port re-reads aborting
MODEM_RTS_DELAY   EQU   1           ;Time to check Modem RTS line (See XMODEM_LOAD & P_XMODEM_LOAD). Not critical.

RECVD_SECT_NO     EQU   0H          ;IX Offset for XModem Recieved Sector Number
SECTNO            EQU   1H          ;IX Offset for XModem CURRENT SECTOR NUMBER
ERRCT       EQU   2H          ;IX Offset for XModem ERROR COUNT


;-------------- S100Computers SMB Board PORT ASSIGNMENTS ----------------------------------------------------

S100_IOBYTE EQU   0EFH        ;"IOBYTE Port" on the SMB. Note this IOBYTE port is not currently used in this monitor
SW_TMAX           EQU   0EEH        ;OUTPUT BIT 0 HIGH FROM THIS PORT LOWERS DMA0* ON THE SMB_V2 (SWITCH IN THE 8086
FAMILY of boards)
                            ;OUTPUT BIT 1 HIGH FROM THIS PORT LOWERS DMA1* ON THE SMB_V2 (SWITCH IN THE 68000 CPU
Board)
                            ;OUTPUT BIT 2 HIGH FROM THIS PORT LOWERS DMA2* ON THE SMB_V2
```

```
                              ;OUTPUT BIT 3 HIGH FROM THIS PORT LOWERS DMA3* ON THE SMB_V2
SW_TMA0     EQU   0EDH              ;INPUT FROM THIS PORT LOWERS DMA0* (SWITCHES IN THE 8088,8086,80286 or 80386 boards)
DIAG_LEDS   EQU   05H         ;LED BAR on V2 SMB

;-------------- S100Computers MSDOS Support Board PORT ASSIGNMENTS -------------------------------------

CMOS_PORT   EQU   70H         ;Base Port for CMOS Clock/Date Chip on MSDOS Support Board
MASTER_PIC_PORT   EQU   20h         ;Hardware port the 8259A (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 HARD DISK CONTROLLER COMMANDS ETC. -------------------------------------
IDEPORTA    EQU   030H  ;Lower 8 bits of IDE interface (8255)
IDEPORTB    EQU   031H  ;Upper 8 bits of IDE interface
IDEPORTC    EQU   032H  ;Control lines for IDE interface
IDEPORTCTRL EQU   033H  ;8255 configuration port

READCFG8255 EQU   10010010b   ;Set 8255 IDEportC to output, IDEportA/B input
WRITECFG8255      EQU   10000000b   ;Set all three 8255 ports to output mode

;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 bits

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 insIDE 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

SEC$SIZE    EQU   512   ;Assume sector size as 512. (Not tested for other sizes)
MAXSEC            EQU   3DH   ;Sectors per track for CF my Memory drive, Kingston CF 8G. (For CPM format, 0-3CH)
                        ;This 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 CPM3 system because
                        ;my Seagate drive has 3D sectors/track. Don't want different CPM3.SYS files around
                        ;so this program as is will also work with a Seagate 6531 IDE drive

MAXTRK            EQU   0FFH  ;CPM3 allows up to 8MG so 0-256 "tracks"
BUFFER$ORG  EQU   3000H ;<----- Will place all sector data here

CPM$BOOT$COUNT    EQU   12    ;Allow up to 12 CPM sectors for CPMLDR
CPMLDR$ADDRESS    EQU   100H  ;Load the CPMLDR at 100H in RAM

DEBUG       EQU   TRUE  ;For a display of error codes returned from CF-Card drive



;-----------------------------------------------------------------------------------------------
```

```
;JUMP TABLE TO ROUTINES WITHIN THE MONITOR.Z80 :-

;PROGRAM CODE BEGINS HERE
;FIRST A JUMP TABLE FOR KEY JUMPS INTO THE SBC-Z80. NOTE THESE SHOULD NOT BE CHANGED BECAUSE
;SOME OF MY OLDER SOFTWARE JUMPS DIRECTLY TO THEM. (WHERE POSSIBLE, TDL's ZAPPLE SBC-Z80 FORMAT IS USED).

ZAPPLE:         JP    COLD              ;INITILIZATION ON RESET
ZCI:      JP    CI              ;CONSOL INPUT
ZCO:      JP    CO              ;CONSOL OUTPUT
ZLO:      JP    LO              ;LIST OUTPUT
ZCSTS:          JP    CSTS              ;CONSOL STATUS
ZMEMCK:         JP    MEMSIZ                  ;GET HIGHEST RAM RETURNS IT IN [HL]
ZSTART:         JP    START             ;JUMP TO MONITOR DO NOT RESET HARDWARE
ZDELAY:         JP    DELAY             ;SOFTWARE DELAY LENGTH IN [A]
ZLSTAT:         JP    LSTAT             ;LIST STATUS
ZPMSG0:         JP    TOM               ;DISPLAY STRING ON CONSOL [HL]=START ADD. [B]=LENGTH
ZPMSG$:         JP    PRINT_STRING            ;DISPLAY STRING ON CONSOL [HL]=START ADD. '$'=END
ZHLSP:          JP    HLSP              ;DISPLAY [HL] ON CONSOL THEN ONE SPACE
ZBITS:          JP    BITS1             ;DISPLAY 8 BITS OF [A] ON CONSOL
ZLBYTE:         JP    LBYTE             ;DISPLAY [A] ON CONSOL
ZHEXSP:         JP    HEXSP             ;PUT 16 BIT PARAMETERS ON STACK FROM CONSOL, [C]=PARAMETER #
ZCRLF:          JP    CRLF              ;SEND CRLF TO CONSOL
ZHILO:          JP    HILO              ;RANGE CHECK (INC [HL], IF HL=DE THEN SET CARRY)
ZPCHK:          JP    PCHK              ;INPUT FROM CONSOL & TEST FOR DELIMITERS RET {Z} IF
ZCCHK:          JP    CCHK              ;CHECK FOR ^S & ESC AT KEYBOARD
ZPSPACE         JP    PSPACE                  ;Print a space on Console
ZLADR     JP    LADR            ;DISPLAY [HL] ON CONSOLE (no space)
ZGETHL          JP    GETHL             ;Get 16 bit value in [HL]
;
;
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;
;     NOTE TABLE MUST BE WITHIN 0-FFH BOUNDRY (CHECK YOUR LISTING)
;
;COMMAND BRANCH TABLE

TBL:  DW  FLUSH               ; "@"  SEND FF to LaserJet printer
      DW  MEMMAP              ; "A"  DISPLAY A MAP OF MEMORY
      DW  BEGIN               ; "B"
      DW  BEGIN               ; "C"
      DW  DISP                ; "D"  DISPLAY MEMORY (IN HEX & ASCII)
      DW  ECHO                ; "E"  ECHO CHAR IN TO CHAR OUT
      DW  FILL                ; "F"  FILL MEMORY WITH A CONSTANT
      DW  GOTO                ; "G"  GO TO [ADDRESS]
      DW  BEGIN               ; "H"
```

```
      DW  IDE_MENU                  ; "I"  GOTO IDE MMENU
      DW  RAMTEST             ; "J"  NON-DESTRUCTIVE MEMORY TEST
      DW  KCMD                ; "K"  DISPLAY THE LIST OF SBC-Z80 COMMANDS
      DW  BEGIN               ; "L"
      DW  MOVE                ; "M"  MOVE BLOCK OF MEMORY (START,FINISH,DESTINATION)
      DW  BEGIN               ; "N"
      DW  BOOT_8086                 ; "O"  Boot up 8086 (if present)
      DW  CPMBOOT             ; "P"  Boot up CPM
      DW  QUERY               ; "Q"  QUERY PORT (IN OR OUT)
      DW  INPORTS             ; "R"  Read ALL Input Ports
      DW  SUBS                ; "S"  SUBSTITUTE &/OR EXAMINE MEMORY
      DW  TYPE                ; "T"  TYPE ASCII PRESENT IN MEMORY
      DW  BEGIN               ; "U"
      DW  VERIFY              ; "V"  COMPARE MEMORY
      DW  BEGIN               ; "W"
      DW  BOOT_XMODEM               ; "X"  Download a file over USB port to RAM
      DW  SWAP_RAM                  ; "Y"  Switch lower 23K RAM page
      DW  SIZE                ; "Z"  FIND HIGHEST R/W RAM
;
;
                              ; IDE MENU COMMAND BRANCH TABLE
IDE_TBL:DW  IDE_ERROR                 ; "A"
      DW  IDE_ERROR                   ; "B"  Backup partition
      DW  CPMBOOT             ; "C"  LOAD CPM (If present)
      DW  SET_DMA             ; "D"  SET BUFFER ADDRESS
      DW  IDE_ERROR                   ; "E"
      DW  IDE_ERROR                   ; "F"
      DW  IDE_ERROR                   ; "G"
      DW  IDE_ERROR                   ; "H"
      DW  IDE_ERROR                   ; "I"
      DW  IDE_ERROR                   ; "J"
      DW  IDE_ERROR                   ; "K"
      DW  SET$LBA             ; "L"  Set LBA value (Set Track,sector)
      DW  PREV$SEC                    ; "M"  Previous sector
      DW  NEXT$SECT                   ; "N"  Next Sector
      DW  IDE_ERROR                   ; "O"
      DW  CPMBOOT             ; "P"  LOAD CPM (If present)
      DW  IDE_ERROR                   ; "Q"
      DW  READ$SEC                    ; "R"  Read sector to data buffer
      DW  SEQ$RD              ; "S"  Sequental sec read and display contents
      DW  IDE_ERROR                   ; "T"
      DW  IDE_ERROR                   ; "U"
      DW  N$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  SHOW$ID             ; "Y"  CF Card Paramaters
      DW  IDE_ERROR                 ; "Z"

;
;--------------------------------------------------------------------------
;
COLD:
;     LD    A,'#'             ;For quick hardware diagnostic test
;     OUT   (S100_CONSOL_OUT),A     ;Force a "#" on the CRT if ROM access is active
;     OUT   (USB_DATA),A            ;Force a "#" on the CRT if ROM access is active

      CALL  INIT_LBA          ;Zero in LBA paramaters after a reset

BEGIN:                              ;Can use the next 3 lines initially to debug hardware
      DI                    ;No interrupts
      XOR   A               ;SET INTERUPT TO PAGE 0H
      LD    I,A             ;Z80 Interrupt page 0
      OUT   (SW_TMAX),A       ;Make sure TMA0*,TMA1*,TMA2* & TMA3* S100 lines are high on V2 SMB

SETUP_STACK:
      LD    SP,AHEAD-4        ;SETUP A FAKE STACK
      JP    MEMSZ1                ;RETURNS WITH TOP OF RAM IN [HL]
      DW    AHEAD             ;A Return opcode will pick up this address
AHEAD:      LD    SP,HL             ;[HL] CONTAINS TOP OF RAM - WORK AREA

      PUSH  HL
      POP   IX                ;Store stack pointer for below in [IX]

IF    ST8C4                   ;If S100_Parallel_IO Board for Printer output
      LD    A,08H             ;Initilize the ST8C4 PC-Printer Port IO
      OUT   (PRINTER_CTRL),A
ELSE
      LD    A,0FFH                  ;IMSAI PIO Board. Clear Printer strobe, comes up 0 on a reset
      OUT   (PRINTER_STROBE),A
ENDIF
                              ;Turn all LED's off as a diagnostic on parallel port board
      LD    A,00000000B       ;FLAG PROGRESS VISUALLY FOR DIAGNOSTIC (ALL LED' ON)
      OUT   (DIAG_LEDS),A           ;LED's will go off one at a time

      LD    A,10000000B       ;FLAG PROGRESS VISUALLY FOR DIAGNOSTIC (1 LED off)
      OUT   (DIAG_LEDS),A

                              ;We need to clear the 8259A otherewise the 8086 monitor sometimes hangs
      LD    A,MasterICW1            ;Initilize the 8259A PIC Controller (;EDGE triggered, 4 bytes, single Master,ICW4
needed)
```

```
      OUT   (MASTER_PIC_PORT),A
      LD    A,MasterICW2            ;Ints starts at 20H in RAM (IBM-PC uses 8X4 = 20H)
      OUT   (MASTER_PIC_PORT+1),A
      LD    A,MasterICW4            ;No slaves above, so 8259 does not expect ICW3
      out   (MASTER_PIC_PORT+1),A

      LD    A,11111111b       ;Allow no interrupts to the 8259A with Z80.
      out   (MASTER_PIC_PORT+1),A

      LD    A,11000000B       ;Flag progress
      OUT   (DIAG_LEDS),A

      LD    HL,SIGNON_MSG           ;Have a Stack, so now we can use CALL
      CALL  PRINT_STRING

      LD    A,11100000B       ;FLAG PROGRESS (Have a Stack with 3 LED's ON)
      OUT   (DIAG_LEDS),A
      CALL  CRLF

      LD    HL,SP_MSG         ;Print Current Stack Location
      CALL  PRINT_STRING

      LD    A,11110000B       ;FLAG PROGRESS (I/O board initilized, 4 LED's ON)
      OUT   (DIAG_LEDS),A

      PUSH  IX                ;SP is stored from above in [IX]
      POP   HL
      CALL  HLSP              ;Print HL/SP

      LD    HL,IOBYTE_MSG           ;Print Current IOBYTE value
      CALL  PRINT_STRING

      IN    A,(IOBYTE)        ;Show IOBYTE. If bit 2=0 (force printer output, but remember CMP/3 boot will hang if no
printer)
      CALL  ZBITS

      CALL  CRLF              ;Then CRLF
      CALL  CSTS              ;Check if garbage at keyboard
      CALL  NZ,CI             ;If so flush it

      LD    A,11111000B       ;FLAG PROGRESS (Ready to go, 5 LED's ON)
      OUT   (DIAG_LEDS),A

      LD    A,11111100B       ;FLAG PROGRESS (Initilization done, 6 LED's ON)
      OUT   (DIAG_LEDS),A
```

```
      CALL  CSTS              ;Flush CI status port


;-------THIS IS THE START ON THE MAIN SBC-Z80 LOOP--------------------------------

START:      LD    DE,START
      PUSH  DE                ;EXTRA UNBALANCED POP & [DE] WOULD END UP IN [PC]
      CALL  CRLF
      LD    C,BELL                  ;A BELL HERE WILL SIGNAL WHEN JOBS ARE DONE
      CALL  CO
      LD    C,'-'
      CALL  CO
      LD    C,'>'
      CALL  CO

STARO:      CALL  TI                ;Main loop. SBC-Z80 will stay here until cmd.
      AND   7FH
      JR    Z,STARO
      SUB   '@'               ;Commands @ to Z only
      RET   M
      CP    1BH               ;A-Z only
      RET   NC
      ADD   A,A
      LD    HL,TBL
      ADD   A,L
      LD    L,A
      LD    A,(HL)
      INC   HL
      LD    H,(HL)
      LD    L,A
      LD    C,02H
      JP    (HL)              ;JUMP TO COMMAND TO COMMAND (from TABLE)
;
;

;PRINT MAIN SBC-Z80 MENU ON CRT

KCMD: LD    HL,SIGNON_MSG           ;Menu Option "Y", List Menu Options
      CALL  PRINT_STRING
      LD    HL,MAIN_MENU_MSG        ;Then Menu Message
      JP    PRINT_STRING


SWAP_RAM:                     ;Swap lowest 32K of RAM
```

```
      LD    HL,SWAP_RAM_MSG
      CALL  PRINT_STRING
      CALL  ZPCHK             ;Get a character
      CP    A,'0'
      JR    Z,Page0
      CP    A,'1'
      JR    Z,Page1
      JP    INVALID_DATA

Page0:      LD    A,0
      OUT   (RAM_BANK),A            ;Output to this board bit 0 to switch lower 32K of RAM
      LD    HL,PAGE0_MSG
      CALL  PRINT_STRING
      JP    BEGIN
Page1:      LD    A,1
      OUT   (RAM_BANK),A            ;Output to this board bit 0 to switch lower 32K of RAM
      LD    HL,PAGE0_MSG            ;Note if this code is being tested in low RAM system will hang here.
      CALL  PRINT_STRING
      JP    BEGIN



;SEND MESSAGE TO CONSOL MESSAGE IN [HL],LENGTH IN [B]

TOM:  LD    C,(HL)
      INC   HL
      CALL  CO
      DJNZ  TOM
      RET

USB_PRINT_STRING:             ;Special print srring for Xmodem
      IN    A,(IOBYTE)
      AND   A,00000011B
      RET   NZ
PRINT_STRING:                       ;A ROUTINE TO PRINT OUT A STRING @ [HL]
      LD    A,(HL)                  ;UP TO THE FIRST '$'.
      INC   HL
      CP    '$'
      RET   Z
      LD    C,A
      CALL  ZCO
      JR    PRINT_STRING


;ABORT IF ESC AT CONSOL, PAUSE IF ^S AT CONSOL
```

```
CCHK: CALL  CSTS              ;FIRST IS THERE ANYTHING THERE
      RET   Z
      CALL  CI
      CP    'S'-40H
      JR    NZ,CCHK1
CCHK2:      CALL  CSTS              ;WAIT HERE UNTIL ANOTHER INPUT IS GIVEN
      JR    Z,CCHK2
CCHK1:      CP    ESC
      RET   NZ                ;RETURN EXECPT IF ESC

;RESTORE SYSTEM AFTER ERROR

ERROR:      CALL  MEMSIZ                  ;GET RAM AVAILABLE - WORKSPACE IN [HL]
      LD    SP,HL             ;SET STACK UP IN WORKSPACE AREA
      LD    C,'*'
      CALL  CO
      JP    START

;PRINT HIGHEST RAM MEMORY FROM BOTTOM

SIZE: CALL  MEMSIZ                  ;RETURNS WITH [HL]= RAM AVAILABLE-WORKSPACE

LFADR:      CALL  CRLF

;PRINT [HL] AND A SPACE

HLSP: PUSH  HL
      PUSH  BC
      CALL  LADR              ;Print [HL] with no space afterwards
      LD    C,SPACE
      CALL  CO
      POP   BC
      POP   HL
      RET

;PRINT A SPACE

PSPACE:     LD    C,SPACE
      JP    CO

;CONVERT HEX TO ASCII

CONV: AND   0FH
      ADD   A,90H
```

```
      DAA
      ADC   A,40H
      DAA
      LD    C,A
      call  ZCO
      RET

;;;;;;;

;GET TWO PARAMETERS AND PUT THEM IN [HL] & [DE] THEN CRLF

EXLF: CALL  HEXSP
      POP   DE
      POP   HL

;SEND TO CONSOL CR/LF

CRLF: PUSH  AF
      PUSH  BC
      LD    C,CR
      CALL  CO
      LD    C,LF
      CALL  CO
      POP   BC
      POP   AF
      RET

;PUT THREE PARAMETERS IN [BC] [DE] [HL] THEN CR/LF

EXPR3:      INC   C                 ;ALREADY HAD [C]=2 FROM START
      CALL  HEXSP
      CALL  CRLF
      POP   BC
      POP   DE
      POP   HL
      RET

;GET ONE PARAMETER

EXPR1:      LD    C,01H
HEXSP:      LD    HL,0000
EX0:  CALL  TI
EX1:  LD    B,A
      CALL  NIBBLE
      JR    C,EX2X
```

```
      ADD   HL,HL
      ADD   HL,HL
      ADD   HL,HL
      ADD   HL,HL
      OR    L
      LD    L,A
      JR    EX0
EX2X: EX    (SP),HL
      PUSH  HL
      LD    A,B
      CALL  QCHK
      JR    NC,SF560
      DEC   C
      RET   Z
SF560:      JP    NZ,ERROR
      DEC   C
      JR    NZ,HEXSP
      RET
EXF:  LD    C,01H
      LD    HL,0000H
      JR    EX1

;RANGE TEST ROUTINE CARRY SET = RANGE EXCEEDED

HILOX:      CALL  CCHK
      CALL  HILO
      RET   NC
      POP   DE                ;DROP ONE LEVEL BACK TO START
      RET
HILO: INC   HL                ;RANGE CHECK SET CARRY IF [DE]=[HL]
      LD    A,H
      OR    L
      SCF
      RET   Z
      LD    A,E
      SUB   L
      LD    A,D
      SBC   A,H
      RET

;PRINT [HL] ON CONSOL

LADR: LD    A,H
      CALL  LBYTE
      LD    A,L
```

```
LBYTE:      PUSH  AF
      RRCA
      RRCA
      RRCA
      RRCA
      CALL  SF598
      POP   AF
SF598:      CALL  CONV              ;Convert to ASCII AND print it
      RET

;THIS IS A CALLED ROUTINE USED TO CALCULATE TOP OF RAM IS USED BY
;THE ERROR ROUTINE TO RESET THE STACK.
;Returns top of RAM in [HL]

MEMSIZ:     PUSH  BC                ;SAVE [BC]
MEMSZ1:     LD    HL,0FFFFH         ;START FROM THE TOP DOWN
MEMSZ2:     LD    A,(HL)
      CPL
      LD    (HL),A
      CP    (HL)
      CPL                     ;PUT BACK WHAT WAS THERE
      LD    (HL),A
      JP    Z,GOTTOP
      DEC   H                 ;TRY 100H BYTES LOWER
      JR    MEMSZ2                  ;KEEP LOOKING FOR RAM
GOTTOP:     POP   BC                ;RESTORE [BC]
      RET

NIBBLE:     SUB   30H
      RET   C
      CP    17H
      CCF
      RET   C
      CP    LF
      CCF
      RET   NC
      SUB   07H
      CP    LF
      RET

COPCK:      LD    C,'-'
      CALL  CO

PCHK: CALL  TI
```

```
;TEST FOR DELIMITERS

QCHK: CP    SPACE
      RET   Z
      CP    ','
      RET   Z
      CP    CR
      SCF
      RET   Z
      CCF
      RET


;KEYBOARD HANDELING ROUTINE (WILL NOT ECHO CR/LF)
;IT CONVERTS LOWER CASE TO UPPER CASE FOR LOOKUP COMMANDS
;ALL OTHERE CHARACTERS ARE ECHOED ON CONSOL

TI:   CALL  CI
      CP    CR
      RET   Z
      PUSH  BC
      LD    C,A
      CALL  CO
      LD    A,C
      POP   BC
      CP    40H               ;LC->UC
      RET   C
      CP    7BH
      RET   NC
SF754:      AND   5FH
      RET

GETHL:      PUSH  BC                ;Return a HEX value in [HL]
      LD    C,1               ;1 paramater
      CALL  ZHEXSP
      POP   HL
      POP   BC
      RET


;DISPLAY 8 BITS OF [A] (No registers changed)

BITS1:      PUSH  DE
      PUSH  BC
      LD    E,A
```

```
      CALL  BITS
      POP   BC
      POP   DE
      RET

;DISPLAY 8 BITS OF [A] (B & C registers changed)

BITS: LD    B,08H
      CALL  PSPACE
SF76E:      SLA   E
      LD    A,18H
      ADC   A,A
      LD    C,A
      CALL  CO
      DJNZ  SF76E
      RET

;MEMORY MAP PROGRAM CF.DR.DOBBS VOL 31 P40.
;IT WILL SHOW ON CONSOL TOTAL MEMORY SUMMARY OF RAM,PROM, AND NO MEMORY

MEMMAP:
      CALL  ZCRLF
      LD    HL,0
      LD    B,1
MAP1: LD    E,'R'             ;PRINT R FOR RAM
      LD    A,(HL)
      CPL
      LD    (HL),A
      CP    (HL)
      CPL
      LD    (HL),A
      JR    NZ,MAP2
      CP    (HL)
      JR    Z,PRINT
MAP2: LD    E,'p'
MAP3: LD    A,0FFH
      CP    (HL)
      JR    NZ,PRINT
      INC   L
      XOR   A
      CP    L
      JR    NZ,MAP3
      LD    E,'.'
PRINT:      LD    L,0
      DEC   B
```

```
      JR    NZ,NLINE
      LD    B,16
      CALL  ZCRLF
      CALL  HXOT4
NLINE:      LD    A,SPACE
      CALL  OTA
      LD    A,E
      CALL  OTA
      INC   H
      JR    NZ,MAP1
      CALL  ZCRLF
      CALL  ZCRLF
      JP    ZSTART

;16 HEX OUTPUT ROUTINE

HXOT4:      LD    C,H
      CALL  HXO2
      LD    C,L
HXO2: LD    A,C
      RRA
      RRA
      RRA
      RRA
      CALL  HXO3
      LD    A,C
HXO3: AND   0FH
      CP    10
      JR    C,HADJ
      ADD   A,7
HADJ: ADD   A,30H
OTA:  PUSH  BC
      LD    C,A
      CALL  ZCO               ;SEND TO CONSOL
      POP   BC
      RET

;DISPLAY MEMORY IN HEX

DISP: CALL  EXLF              ;GET PARAMETERS IN [HL],[DE]
      LD    A,L               ;ROUND OFF ADDRESSES TO XX00H
      AND   0F0H
      LD    L,A
      LD    A,E               ;FINAL ADDRESS LOWER HALF
      AND   0F0H
```

```
      ADD   A,10H             ;FINISH TO END 0F LINE
SF172A:     CALL  LFADR
SF175A:     CALL  BLANK

      CALL  CSTS              ;For some reason the USB port requires this!
                              ;otherewise you have to press CR for each character
                              ;Requires further analysis!
      LD    A,(HL)
      CALL  ZLBYTE
      CALL  HILOX
      LD    A,L
      AND   0FH
      JR    NZ,SF175A
      LD    C,TAB             ;INSERT A TAB BETWEEN DATA
      CALL  ZCO
      LD    B,4H              ;ALSO 4 SPACES
TA11: LD    C,SPACE
      CALL  ZCO
      DJNZ  TA11
      LD    B,16              ;NOW PRINT ASCII (16 CHARACTERS)
      PUSH  DE                ;TEMPORLY SAVE [DE]
      LD    DE,0010H
      SBC   HL,DE
      POP   DE
T11:  LD    A,(HL)
      AND   7FH
      CP    ' '               ;FILTER OUT CONTROL CHARACTERS'
      JR    NC,T33
T22:  LD    A,'.'
T33:  CP    07CH
      JR    NC,T22
      LD    C,A               ;SET UP TO SEND
      CALL  ZCO
      INC   HL
      DJNZ  T11               ;REPEAT FOR WHOLE LINE
      JR    SF172A


;INSPECT AND / OR MODIFY MEMORY

SUBS: LD    C,1
      CALL  ZHEXSP
      POP   HL
SF2E3:      LD    A,(HL)
      CALL  ZLBYTE
```

```
      LD    C,'-'
      CALL  ZCO
      CALL  ZPCHK
      RET   C
      JR    Z,SF2FC
      CP    5FH
      JR    Z,SF305
      PUSH  HL
      CALL  EXF
      POP   DE
      POP   HL
      LD    (HL),E
      LD    A,B
      CP    CR
      RET   Z
SF2FC:      INC   HL
SF2FD:      LD    A,L
      AND   07H
      CALL  Z,LFADR
      JR    SF2E3
SF305:      DEC   HL
      JR    SF2FD


;Print a space on screen

BLANK:      PUSH  BC
      PUSH  HL
      LD    C,' '
      CALL  ZCO
      POP   HL
      POP   BC
      RET

;FILL A BLOCK OF MEMORY WITH A VALUE

FILL: CALL  EXPR3
SF1A5:      LD    (HL),C
      CALL  HILOX
      JR    NC,SF1A5
      POP   DE
      JP    ZSTART

;GO TO A RAM LOCATION
```

```
GOTO: LD    C,1               ;SIMPLE GOTO FIRST GET PARMS.
      CALL  HEXSP
      CALL  CRLF
      POP   HL                ;GET PARAMETER PUSHED BY EXF
      JP    (HL)

; GET OR OUTPUT TO A PORT

QUERY:      CALL  ZPCHK
      CP    'O'               ;OUTPUT TO PORT
      JR    Z,OUT_PORT
      CP    'I'               ;INPUT FROM PORT
      JP    Z,IN_PORT
      LD    C,'*'
      JP    ZCO               ;WILL ABORT IF NOT 'I' OR 'O'

IN_PORT:
      LD    C,1               ;IN Port
      CALL  ZHEXSP
      POP   BC
      IN    A,(C)
      JP    ZBITS
;
OUT_PORT:
      CALL  ZHEXSP                  ;OUT Port
      POP   DE
      POP   BC
      OUT   (C),E
      RET


; MEMORY TEST

RAMTEST:
      PUSH  HL
      PUSH  BC
      LD    HL,RAM_TEST_MSG
      CALL  PRINT_STRING
      POp   BC
      POP   HL
      CALL  EXLF
SF200:      LD    A,(HL)
      LD    B,A
      CPL
      LD    (HL),A
```

```
      XOR   (HL)
      JR    Z,SF215
      PUSH  DE
      LD    D,B
      LD    E,A               ;TEMP STORE BITS
      CALL  ZHLSP
      CALL  BLANK
      LD    A,E
      CALL  ZBITS
      CALL  ZCRLF
      LD    B,D
      POP   DE
SF215:      LD    (HL),B
      CALL  HILOX
      JR    SF200

;MOVE A BLOCK OF MEMORY TO ANOTHER LOCATION

MOVE: CALL  EXPR3
SF21E:      LD    A,(HL)
      LD    (BC),A
      INC   BC
      CALL  HILOX
      JR    SF21E

;VERIFY ONE BLOCK OF MEMORY WITH ANOTHER

VERIFY:     CALL  EXPR3
VERIO:      LD    A,(BC)
      CP    (HL)
      JR    Z,SF78E
      PUSH  BC
      CALL  CERR
      POP   BC
SF78E:      INC   BC
      CALL  HILOX
      JR    VERIO
      RET
;
CERR: LD    B,A
      CALL  ZHLSP
      LD    A,(HL)
      CALL  ZLBYTE
      CALL  BLANK
      LD    A,B
```

```
      CALL  ZLBYTE
      JP    ZCRLF

ECHO: LD    HL,ECHO_MSG
      CALL  PRINT_STRING
ECHO1:      CALL  CI                ;Routeen to check keyboard etc.
      CP    'C'-40H               ;Loop until  ^C
      RET   Z
      CP    'Z'-40H
      RET   Z
      LD    C,A
      CALL  CO
      JR    ECHO1



;READ ASCII FROM MEMORY

TYPE: CALL  EXLF
SF30B:      CALL  LFADR
      LD    B,56
SF310:      LD    A,(HL)
      AND   7FH
      CP    SPACE
      JR    NC,SF319
SF317:      LD    A,2EH
SF319:      CP    7CH
      JR    NC,SF317
      LD    C,A
      CALL  ZCO
      CALL  HILOX
      DJNZ  SF310
      JR    SF30B


;     Display all active IO input ports in the system
;
INPORTS:
      LD    HL,ALL_PORTS
      CALL  PRINT_STRING
      LD    B,0               ;Now loop through all ports (0-FF)
      LD    D,6               ;Display 6 ports across
      LD    E,0FFH                  ;Will contain port number
LOOPIO:     LD    C,E
      LD    A,E
```

```
      CP    A,SW_TMA0         ;Inputting here will switch out the Z80 to 8086/80286
      JR    Z,SKIP                  ;Skip because we don't want that right now
;
      IN    A,(C)             ;Remember  [ZASMB does not work with this opcode,SLR is OK]
      CP    A,0FFH                  ;No need for 0FF's
      JR    Z,SKIP
      LD    H,A               ;store port data in H for below
      LD    A,E               ;Need to print port # first
      CALL  LBYTE             ;Print port number
      LD    C,'-'
      CALL  ZCO
      LD    C,'>'
      CALL  ZCO
      LD    A,H               ;get back port data
      CALL  LBYTE             ;print it
      LD    C,TAB
      CALL  ZCO
      DEC   D                 ;6 ports per line
      JR    NZ,SKIP
      LD    D,6
      CALL  ZCRLF
SKIP: DEC   E                 ;Next Port
      DJNZ  LOOPIO
      CALL  ZCRLF
      RET


;
;<<<<<<<<<<<<<<<<<<<<<< MAIN CONSOL OUTPUT ROUTINE >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

CO:   IN    A,(IOBYTE)        ;Where is the data going to
      BIT   5,A
      JR    NZ,PROP_CO        ;No IOBYTE port active
      BIT   1,A
      JR    Z,PROP_CO         ;IF xxxxxxxx0 then output directly to Propeller board
CO1:  IN    A,(USB_STATUS)          ;Input must be from USB port on this SBC
      AND   A,USB_TXE
      JP    NZ,CO1                  ;HIGH, Output FIFO not ready yet to send character
      LD    A,C               ;LOW, OK to send
      CP    07H               ;IS IT A BELL
      RET   Z                 ;Do not send BELL to to TELNET Window (at least for now)!
      CP    CR                ;Send CR
      JR    Z,CO2
      CP    LF                ;Send LF
      JR    Z,CO2
```

```
      CP    TAB               ;Send TAB
      JR    Z,CO2
      CP    BACKS             ;Send backspace
      JR    Z,CO2
      CP    SPACE
      RET   C                 ;If less than ASCII space then ignore
CO2:  OUT   (USB_DATA),A
      RET

PROP_CO:
      IN    A,(S100_CONSOL_STATUS)  ;SD SYSTEMS VIDIO BOARD PORT
      AND   4H
      JR    Z,PROP_CO         ;Not yet ready, try both outputs
      LD    A,C
      CP    07H               ;IS IT A BELL
      JP    Z,BELL1                 ;Special case
      CP    0H                ;SD BOARD CANNOT TAKE A NULL!
      RET   Z
      OUT   (S100_CONSOL_OUT),A
      RET

LOX:  CALL  CO                ;OUTPUT TO BOTH PRINTER & CONSOLE
      CALL  LO
      RET

BELL1:      LD    A,06H             ;SEND A BELL
      OUT   (S100_CONSOL_OUT),A
      LD    A,0FH
      CALL  DELAY
      LD    A,07H
      OUT   (S100_CONSOL_OUT),A
      RET


DELAY:      DEC   A                 ;GENERAL COUNT DOWN TIME DELAY
      RET   Z                 ;LENGTH SET IN [A]
      PUSH  AF
      LD    A,05H
MORE: DEC   A
      PUSH  AF
      XOR   A
MORE2:      DEC   A
      JR    NZ,MORE2
      POP   AF
      JR    NZ,MORE
```

```
      POP   AF
      JR    DELAY


;<<<<<<<<<<<<<<<<<<< MAIN CONSOL STATUS ROUTINE >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

CSTS: IN    A,(IOBYTE)        ;Where is data comming from
      BIT   5,A
      JR    NZ,PROP_CSTS            ;No IOBYTE port active
      BIT   0,A
      JR    Z,PROP_CSTS       ;IF xxxxxxxx0 then input directly from Propeller board
      IN    A,(USB_STATUS)          ;Input must be from USB port on this SBC
      AND   USB_RXE
      JP    NZ,NO_CSTS        ;HIGH, No character yet in FIFO
      JP    GOT_CSTS          ;LOW, Character ready, return 0FFH

PROP_CSTS:
      IN    A,(S100_CONSOL_STATUS)
      AND   02H
      JP    Z,NO_CSTS         ;Zero if nothing
GOT_CSTS:
      XOR   A
      DEC   A                 ;RETURN WITH 0FFH IN [A] IF SOMETHING
      RET
NO_CSTS:
      XOR   A                 ;IF IOBYTE bit 1 then skip modem input
      RET                     ;RETURN WITH 0 IN A IF NOTHING THERE


;<<<<<<<<<<<<<<<<<<< MAIN CONSOL INPUT ROUTINE >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

CI:   IN    A,(IOBYTE)        ;Where is data comming from
      BIT   5,A
      JR    NZ,PROP_CI        ;No IOBYTE port active
      BIT   0,A
      JP    Z,PROP_CI         ;LOW IF xxxxxxxx0 then input directly from Propeller board
USB_CI:
      IN    A,(USB_STATUS)          ;Is there a character to recieve from the PC via the USB Port
      AND   USB_RXE
      JP    NZ,USB_CI         ;HIGH No Character yet
      IN    A,(USB_DATA)
      AND   7FH
      RET

PROP_CI:
```

```
      IN    A,(S100_CONSOL_STATUS)  ;NEED CONSTAT TO CLEAN UP SHIFT KEYS ETC
      AND   02H
      JR    Z,PROP_CI         ;Wait until something there
      IN    A,(S100_CONSOL_IN)
      AND   7FH
      RET

;>>>>>>>>>>>>>>>>>>>> MAIN PRINTER OUTPUT ROUTINE <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

LO:   LD    B,0FFH                  ;First make sure an actual printer is connected
LO2:  CALL  LSTAT
      JR    NZ,LO1
      DJNZ  LO2
      XOR   A
      LD    A,C
      RET                     ;RET Z if Printer problem (Not switched on)


IF    ST8C4                   ;If S100_Parallel_IO Board for Printer output
LO1:  LD    A,PRINTER_ST_HIGH ;Make sure strobe is high
      OUT   (PRINTER_CTRL),A
      LD    A,C               ;Send Data from [C]
      OUT   (PRINTER_OUT),A
      LD    A,PRINTER_ST_LOW  ;Now send strobe High->Low
      OUT   (PRINTER_CTRL),A
      LD    A,PRINTER_ST_HIGH ;Now returb strobe back high
      OUT   (PRINTER_CTRL),A
      OR    A,A
      RET                     ;Ret NZ if OK

ELSE                          ;IMSAI PIO Board PARRELL PORT

LO1:  LD    A,0FFH                  ;Setup strobe high to low then high
      OUT   (PRINTER_STROBE),A
      LD    A,C
      OUT   (PRINTER_OUT),A         ;Now Data
      XOR   A                 ;STROBE FOR CENTRONICS
      OUT   (PRINTER_STROBE),A
      LD    A,0FFH                  ;Raise strobe again
      OUT   (PRINTER_STROBE),A
      OR    A,A
      RET                     ;Ret NZ if OK
ENDIF
```

```
FLUSH:      LD    C,FF              ;Send a Form Feed to laserJet Printer
      CALL  LO                ;This forces a partial page to be printed
      RET

;>>>>>>>>>>>>>>>>>>>>>> PRINTER STATUS ROUTINE <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

IF    ST8C4                     ;If S100_Parallel_IO Board for Printer output
LSTAT:      IN    A,(PRINTER_STATUS)      ;0CH status port of ST8C4
      AND   11111111B               ;For now
      CP    11011111B               ;should see 11011111 if printer is selected, ready, with paper etc.
      JR    Z,LSTAT1
      XOR   A
      RET                     ;Return Z if not ready
LSTAT1:     XOR   A                 ;PUT 0FFH IN [A] IF READY & NO ZERO FLAG
      DEC   A
      RET
ELSE                          ;IMSAI PIO Board PARRELL PORT
LSTAT:      IN    A,(PRINTER_STATUS)
      AND   00001111B               ;XXXX0110 IS READY (BIT 3=PAPER BIT 2=FAULT
      CP    00000110B               ;BIT 1=SELECT  BIT 0=BUSY
      JR    Z,LSTAT1
      XOR   A
      RET
LSTAT1:     XOR   A                 ;PUT 0FFH IN [A] IF READY & NO ZERO FLAG
      DEC   A
      RET
ENDIF


;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> XMODEM ROUTINES <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

BOOT_XMODEM:
;This routine will download any XMODEM formatted data from a file on a PC over teh USB port.
;If the USB port is NOT also also used console I/O progress of the download will appear on the screen.
;If the USB port is ALSO used for console I/O no progress is shown until the download is complete.

      LD    HL,MODEM_SIGNON         ;Send Modem signon message
      CALL  PRINT_STRING

      LD    HL,0              ;SP to HL
      ADD   HL,SP
      LD    DE,40H                  ;Drop 40H bytes below to be safe
      SBC   HL,DE
      PUSH  HL
      POP   IX                ;Will store data  below in RAM 40H bytes below SP to be safe
```

```
      XOR   A,A               ;Initilize flags
      LD    (IX+RECVD_SECT_NO),A
      LD    (IX+SECTNO),A
      LD    (IX+ERRCT),A

      LD    HL,MODEM_RAM_LOC  ;Get RAM location for where to place code
      CALL  PRINT_STRING
      LD    C,1
      CALL  ZHEXSP                  ;Get 16 bit value, put on stack
      POP   IY                ;DMA Value now in IY
      CALL  ZCRLF

      LD    B,1               ;TIMEOUT DELAY
      CALL  RECV              ;GOBBLE UP GARBAGE CHARS FROM THE LINE



RECV_LOOP:                    ;<------- MAIN RECIEVE LOOP
      XOR   A,A               ;GET 0
      LD    (IX+ERRCT),A            ;INITIAL ERROR COUNT SET TO 0
RECV_HDR:
      LD    HL,RMSG
      CALL  USB_PRINT_STRING  ;Skip print if USB port is also console
      LD    A,(IX+SECTNO)           ;Get current sector number
      INC   A
      PUSH  AF                ;Save [A]
      IN    A,(IOBYTE)        ;Is USB port also console
      AND   00000011B
      JR    NZ,SKIP0
      POP   AF
      CALL  LBYTE             ;Show Sector Number on Console if USB port is NOT also console
      LD    HL,MODEM_RAM_MSG  ;"H.  IF OK, will write to RAM location"
      CALL  PRINT_STRING
      PUSH  IY
      POP   HL                ;IY to HL
      CALL  LADR              ;Show DMA Address if USB port is NOT also console
      CALL  ZCRLF
      LD    B,5               ;5 SEC TIMEOUT
      CALL  RECV
      JP    NC,RHNTO          ;IF ALL OK (NO TIMEOUT), THEN DROP DOWN TO RHNTO TO GET DATA
      JP    RECV_HDR_TIMEOUT

SKIP0:      POP   AF                ;If USB Port is also console skip status display
      LD    B,5               ;5 SEC TIMEOUT
```

```
      CALL  RECV
      JP    NC,RHNTO          ;IF ALL OK (NO TIMEOUT), THEN DROP DOWN TO RHNTO TO GET DATA
      JP    RECV_SECT_ERR

RECV_HDR_TIMEOUT:
      CALL  TOUT              ;PRINT TIMEOUT
RECV_SECT_ERR:                      ;PURGE THE LINE OF INPUT CHARS
      LD    B,1               ;1 SEC W/NO CHARS
      CALL  RECV
      JP    NC,RECV_SECT_ERR  ;LOOP UNTIL SENDER DONE
      LD    A,NAK
      CALL  SEND              ;SEND NAK
      LD    A,(IX+ERRCT)
      INC   A
      LD    (IX+ERRCT),A
      CP    A,MODEM_ERR_LIMIT
      JP    C,RECV_HDR
      CALL  CHECK_FOR_QUIT
      JP    Z,RECV_HDR
      LD    HL,BAD_HEADER_MSG
      CALL  PRINT_STRING
      JP    EXIT


RHNTO:      CP    A,SOH             ;GOT CHAR - MUST BE SOH
      JP    Z,GOT_SOH         ;Z IF OK
      OR    A,A               ;00 FROM SPEED CHECK?
      JP    Z,RECV_HDR
      CP    A,EOT
      JP    Z,GOT_EOT
                              ;DIDN'T GET SOH -
      PUSH  AF                ;Save [A]
      IN    A,(IOBYTE)        ;Is USB port also console
      AND   00000011B
      JR    NZ,SKIP1
      POP   AF
      CALL  LBYTE             ;Print [A] on console
      LD    HL,ERRSOH
      CALL  PRINT_STRING
      JR    RECV_SECT_ERR
SKIP1:      POP   AF
      JR    RECV_SECT_ERR

GOT_SOH:
      LD    B,1
```

```
      CALL  RECV
      JP    C,RECV_HDR_TIMEOUT
      LD    D,A               ;D=BLK #
      LD    B,1
      CALL  RECV              ;GET CMA'D SECT #
      JP    C,RECV_HDR_TIMEOUT
      CPL
      CP    A,D               ;GOOD SECTOR #?
      JP    Z,RECV_SECTOR

      LD    HL,ERR2                 ;GOT BAD SECTOR #
      CALL  USB_PRINT_STRING
      JP    RECV_SECT_ERR


RECV_SECTOR:                        ;Sector is OK, so read data and place in RAM
      LD    A,D               ;GET SECTOR #
      LD    (IX+RECVD_SECT_NO),A
      LD    C,0               ;INIT CKSUM
      LD    E,80H             ;Sector Byte Count
      PUSH  IY
      POP   HL                ;DMA address (IY) to HL

RECV_CHAR:
      LD    B,1               ;1 SEC TIMEOUT
      CALL  RECV              ;GET CHAR
      JP    C,RECV_HDR_TIMEOUT
      LD    (HL),A                  ;STORE CHAR
      INC   HL
      DEC   E                 ;Next sector byte
      JP    NZ,RECV_CHAR

                                            ;VERIFY CHECKSUM
      LD    D,C               ;SAVE CHECKSUM
      LD    B,1               ;TIMEOUT
      CALL  RECV              ;GET CHECKSUM
      JP    C,RECV_HDR_TIMEOUT
      CP    A,D               ;CHECK
      JP    NZ,RECV_CKSUM_ERR

                                            ;GOT A SECTOR, WRITE IF = 1+PREV SECTOR
      LD    A,(IX+RECVD_SECT_NO)
      LD    B,A               ;SAVE IT
      LD    A,(IX+SECTNO)           ;GET PREV
      INC   A                 ;CALC NEXT SECTOR #
```

```
      CP    B                 ;MATCH?
      JP    NZ,DO_ACK
      LD    (IX+SECTNO),A           ;UPDATE SECTOR #
DO_ACK:     LD    A,ACK
      CALL  SEND

      PUSH  HL                ;ALL OK SO SAVE DMA Address in IY
      POP   IY
      JP    RECV_LOOP         ;Back to Top recieve loop

RECV_CKSUM_ERR:
      LD    HL,ERR3
      CALL  USB_PRINT_STRING
      JP    RECV_SECT_ERR

GOT_EOT:
      LD    A,ACK             ;ACK THE EOT
      CALL  SEND
      JP    XFER_CPLT

;-----------------------------------------------------
;   XMODEM USB PORT GET CHARACTER ROUTINE
;-----------------------------------------------------

RECV: PUSH  DE                ;SAVE D,E
MSEC: LD    DE,0BBBBH         ;1 SEC DCR COUNT
MWTI: IN    A,(USB_STATUS)
      AND   A,USB_RXE
      JP    Z,MCHAR                 ;GOT CHAR
      DEC   E                 ;COUNT DOWN
      JP    NZ,MWTI                 ;FOR TIMEOUT
      DEC   D
      JP    NZ,MWTI
      DEC   B                 ;DCR # OF SECONDS
      JP    NZ,MSEC                 ;MODEM TIMED OUT RECEIVING
      POP   DE                ;RESTORE D,E
      SCF                     ;CARRY SHOWS TIMEOUT
      RET
                              ;GOT MODEM CHAR
MCHAR:      IN    A,(USB_DATA)
      POP   DE                ;RESTORE DE
      PUSH  AF                ;CALC CHECKSUM
      ADD   A,C
      LD    C,A
      POP   AF
```

```
      OR    A,A               ;TURN OFF CARRY TO SHOW NO TIMEOUT
      RET

;-----------------------------------------------------
;   XMODEM USB PORT SEND CHARACTER ROUTINE
;-----------------------------------------------------

SEND: PUSH  AF                ;CHECK IF MONITORING OUTPUT
      ADD   A,C               ;CALC CKSUM
      LD    C,A
SENDW:      IN    A,(USB_STATUS)          ;Don't worry PC is always fast enough!
      AND   A,USB_TXE
      JP    NZ,SENDW
      POP   AF                ;GET CHAR
      OUT   (USB_DATA),A
      RET

;---------------- SUPPORT ROUTINES -----------------------------

TOUT: IN    A,(IOBYTE)        ;Is USB port also console
      AND   00000011B
      RET   NZ
      LD    HL,TOUTM          ;PRINT TIMEOUT MESSAGE
      CALL  PRINT_STRING
      LD    A,(IX+ERRCT)
      CALL  LBYTE
      CALL  ZCRLF
      RET

CO_A: PUSH  AF                ;PRINT VALUE of [A] on CRT
      PUSH  BC
      LD    C,A
      CALL  CO
      POP   BC
      POP   AF
      RET

CHECK_FOR_QUIT:                     ;MULTIPLE ERRORS, ASK IF TIME TO QUIT
      XOR   A,A               ;GET 0
      LD    (IX+ERRCT),A            ;RESET ERROR COUNT
      LD    HL,QUITM
      CALL  USB_PRINT_STRING

CI3:  CALL  CSTS              ;NEED CONSTAT TO CLEAN UP SHIFT KEYS ETC
      JP    Z,CI3
```

```
      CALL  CI
      PUSH  AF                ;SAVE CHAR
      CALL  ZCRLF
      POP   AF
      CP    A,'R'
      RET   Z                 ;RETURN IF RETRY
      CP    A,'r'
      RET   Z
      CP    A,'Q'             ;QUIT?
      JP    NZ,LCQ
      OR    A,A               ;TURN OFF ZERO FLAG
      RET
LCQ:  CP    A,'q'
      JP    NZ,CHECK_FOR_QUIT
      OR    A,A               ;TURN OFF ZERO FLAG
      RET

XFER_CPLT:                    ;DONE - CLOSE UP SHOP
      LD    HL,MODEM_DONE_MSG
      CALL  PRINT_STRING
EXIT: JP    ZSTART


BOOT_8086:
      LD    HL,MSG_8086
      CALL  PRINT_STRING
      LD    A,01
      OUT   (SW_TMAX),A       ;Lower DMA0* on SMB V3 (if present)
      NOP
      NOP
      JP    BEGIN

INVALID_DATA:
      LD    HL,Invalid_Msg
      CALL  PRINT_STRING
      JP    BEGIN
;
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> IDE/CF Card ROUTINES <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<


IDE_MENU:
      JP    IDE_LOOP

      call  IDEinit                 ;initialize the board and drive. If there is no drive abort
      jp    z,IDE_LOOP        ;Setup for main menu commands
```

```
      ld    de,INIT$ERROR
      call  PSTRING
      call  SHOWerrors
      jp    ABORT

ABORT:      jp    BEGIN             ;Else jump to start of monitor


IDE_LOOP:                         ;A 1 line prompt
      ld    de,IDE_MENU_MSG         ;List command options
      call  PSTRING

      call  wrlba               ;Update LBA on drive
      call  DISPLAYposition         ;Display current Track,sector,head#

      ld    de,Prompt           ;'Please enter a command >'
      call  PSTRING

      call  GETCMD                  ;Simple UC character Input (Note, no fancy checking)
      cp    ESC                 ;ESC back to main menu
      jp    z,BEGIN
      call  upper
      call  ZCRLF

      sbc   '@'                 ;Adjust to 0,1AH
      add   a,A                 ;X2
      ld    hl,IDE_TBL          ;Get menu selection
      add   a,L
      ld    L,A
      ld    a,(hl)
      INC   HL
      ld    h,(hl)
      ld    L,A                 ;Jump to table pointer
      jp    (hl)                ;JMP (HL)



READ$SEC:                         ;Read Sector @ LBA to the RAM buffer
      ld    hl,(@DMA)           ;Point to buffer

      call  READSECTOR

      jp    z,main1b            ;Z means the sector read was OK
      call  ZCRLF
      jp    IDE_LOOP
```

```
main1b:     ld    de, msgrd         ;Sector read OK
      call  PSTRING

      ld    HL,(@DMA)         ;Point to buffer.
      call  HEXDUMP                 ;Show sector data
      jp    IDE_LOOP

WRITE$SEC:                    ;Write data in RAM buffer to sector @ LBA
      ld    de,msgsure        ;Are you sure?
      call  PSTRING
      call  ZCI
      call  upper
      LD    C,A               ;Print response
      CALL  ZCO
      CP    'Y'
      PUSH  AF
      CALL  CRLF
      POP   AF
      jp    nz,IDE_LOOP
      CALL  ZCRLF

      ld    HL,(@DMA)

      call  WRITESECTOR

      jp    z,IDE_LOOP        ;Z means the sector write was OK
      call  ZCRLF
      jp    IDE_LOOP
main2b:     ld    de, msgwr         ;Sector written OK
      call  PSTRING
      jp    IDE_LOOP


SET$LBA:                      ;Set the logical block address
      ld    de,GET$LBA
      call  PSTRING
      call  ghex32lba         ;Get new CPM style Track & Sector number and put them in RAM at @SEC & @TRK
      jp    c,main3b          ;Ret C set if abort/error
      call  wrlba             ;Update LBA on drive
main3b:     call  ZCRLF
      jp    IDE_LOOP

NEXT$SECT:
      ld    a,(@SEC)
      inc   A
```

```
      cp    MAXSEC-1
      jp    nc,RANGE$ERROR
      ld    (@SEC),a
      call  wrlba             ;Update LBA on drive
      call  ZCRLF
      jp    IDE_LOOP
RANGE$ERROR:
      ld    de,RANGE$MSG
      call  PSTRING
      jp    IDE_LOOP

PREV$SEC:
      ld    a,(@SEC)
      or    A
      jp    z,RANGE$ERROR
      dec   A
      ld    (@SEC),a
      call  wrlba             ;Update LBA on drive
      call  ZCRLF
      jp    IDE_LOOP



CPMBOOT:                      ;Boot CPM from IDE system tracks -- if present
      ld    de,BOOTCPM$MSG
      call  PSTRING
      call  IDEinit                 ;initialize the board and drive. If there is no drive abort
      ld    a,0               ;Load from track 0,sec 1, head 0 (Always)
      ld    (@SEC),a          ;Remember sectors are numbered +1
      xor   A
      ld    (@TRK+1),a
      ld    (@TRK),a

      ld    a,CPM$BOOT$COUNT  ;Count of CPMLDR sectors  (12)
      ld    (@SECTOR$COUNT),a
      ld    hl,CPMLDR$ADDRESS ;DMA address where the CPMLDR resides in RAM (100H)
      ld    (@DMA),hl

NextRCPM:
      call  wrlba             ;Update LBA on drive
      call  DISPLAYposition         ;Display current Track,sector,head#
      call  ZCRLF

      ld    hl,(@DMA)
      call  READSECTOR        ;read a sector
```

```
      ld    (@DMA),hl

      ld    a,(@SECTOR$COUNT)
      DEC     A
      ld    (@SECTOR$COUNT),a
      jp    z,LOAD$DONE

      ld    hl,(@SEC)
      inc   hl
      ld    (@SEC),hl         ;Note we assume we alway will stay on tarck 0 in this special case
      jp    NextRCPM

LOAD$DONE:
      ld    e,REGstatus       ;Check the R/W status when done
      call  IDErd8D
      BIT   0,D
      jp    nz,CPMLoadErr           ;Z if no errors
      ld    hl,CPMLDR$ADDRESS
      ld    a,(hl)
      cp    31H               ;EXPECT TO HAVE 31H @80H IE. LD SP,80H
      jp    nz,CPMLoadErr1          ;Z if no errors
      jp    100H              ;Now jump here where the code for the CPMLDR resides

CPMLoadErr1:
      ld    de,CPM$ERROR1           ;Drive data error
      call  PSTRING
      jp    IDE_LOOP

CPMLoadErr:
      ld    de,CPM$ERROR            ;Drive Read Error
      call  PSTRING
      jp    IDE_LOOP


N$RD$SEC:                     ;Read N sectors >>>> NOTE no check is made to not overwrite
      ld    de,ReadN$MSG            ;CPM etc. in high RAM
      call  PSTRING
      call  GETHEX
      jp    c,IDE_LOOP        ;Abort if ESC (C flag set)
      CALL  CRLF
      CALL  CRLF

      ld    (@SECTOR$COUNT),a ;store sector count
      ld    HL,(@DMA)         ;Point to buffer
```

```
NextRSec:
      ld    de,MultiRD_MSG
      call  PSTRING
      call  wrlba             ;Update LBA on drive
      call  DISPLAYposition         ;Display current Track,sector,head#

      ld    hl,(@DMA)
      call  READSECTOR
      ld    (@DMA),hl

      ld    a,(@SECTOR$COUNT)
      DEC     A
      ld    (@SECTOR$COUNT),a
      jp    z,IDE_LOOP

      ld    hl,(@SEC)
      inc   hl
      ld    (@SEC),hl
      ld    a,L               ;0 to 62 CPM Sectors
      cp    MAXSEC-1
      jp    nz,NextRSec

      ld    hl,0              ;Back to CPM sector 0
      ld    (@SEC),hl
      ld    hl,(@TRK)         ;Bump to next track
      inc   hl
      ld    (@TRK),hl
      ld    a,L               ;0-FFH tracks (only)
      jp    nz,NextRSec

      ld    de,AtEnd          ;Tell us we are at end of disk
      call  PSTRING
      jp    IDE_LOOP


N$WR$SEC:                     ;Write N sectors
      ld    de,WriteN$MSG           ;How many sectors
      call  PSTRING
      call  GETHEX
      jp    c,IDE_LOOP        ;Abort if ESC (C flag set)
      ld    (@SECTOR$COUNT),a ;store sector count

      ld    de,msgsure        ;Are you sure?
      call  PSTRING
      call  ZCI
```

```
      call  upper
      LD    C,A               ;Print response
      CALL  ZCO
      CP    'Y'
      PUSH  AF
      CALL  CRLF
      POP   AF
      jp    nz,IDE_LOOP
      CALL  ZCRLF

      ld    HL,(@DMA)         ;Point to current DMA buffer

NextWSec:
      ld    de,MultiWR_MSG
      call  PSTRING
      call  wrlba             ;Update LBA on drive
      call  DISPLAYposition         ;Display current Track,sector,head#

      ld    hl,(@DMA)
      call  WRITESECTOR       ;Actully, Sector/track values are already updated
      ld    (@DMA),hl         ;above in wrlba, but WRITESECTOR is used in multiple places.
                              ;A repeat does no harm -- speed is not an issue here
      ld    a,(@SECTOR$COUNT)
      DEC     A
      ld    (@SECTOR$COUNT),a
      jp    z,IDE_LOOP

      ld    hl,(@SEC)
      inc   hl
      ld    (@SEC),hl
      ld    a,L               ;0 to 62 CPM Sectors
      cp    MAXSEC-1
      jp    nz,NextWSec

      ld    hl,0              ;Back to CPM sector 0
      ld    (@SEC),hl
      ld    hl,(@TRK)         ;Bump to next track
      inc   hl
      ld    (@TRK),hl
      ld    a,L               ;0-FFH tracks (only)
      or    A
      jp    nz,NextWSec

      ld    de,AtEnd          ;Tell us we are at end of disk
      call  PSTRING
```

```
      jp    IDE_LOOP



IDE_ERROR:
      ld    de, msgErr        ;CMD error msg
      call  PSTRING
      jp    BEGIN


;---------------- Support Routines -----------------------------------------

driveid:call      IDEwaitnotbusy          ;Do the IDEntify drive command, and return with the buffer
                            ;filled with info about the drive
      ret   c                 ;If Busy return NZ
      ld    d,COMMANDid
      ld    e,REGcommand
      call  IDEwr8D                 ;issue the command

      call  IDEwaitdrq        ;Wait for Busy=0, DRQ=1
      jp    c,SHOWerrors

      ld    b,0               ;256 words
      ld    hl,IDbuffer       ;Store data here
      call  MoreRD16          ;Get 256 words of data from REGdata port to [HL]
      ret



SEQ$RD:
      call  IDEwaitnotbusy          ;Sequentially read sectors one at a time from current position
      jp    c,SHOWerrors
      call  ZCRLF
NEXTSEC:
      ld    HL,(@DMA)

      call  READSECTOR        ;If there are errors they will show up in READSECTOR

      jp    z,SEQOK
      ld    de,CONTINUE$MSG         ;To Abort enter ESC. Any other key to continue.
      call  PSTRING
      call  ZCI
      cp    ESC               ;Abort if ESC
```

```
      jp    IDE_LOOP

SEQOK:      CALL  CRLF
      call  DISPLAYposition         ;Display current Track,sector,head#

      ld    HL,(@DMA)         ;Point to buffer

      call  HEXDUMP                 ;Display sector contents
      call  ZCRLF
      call  ZCRLF
      call  ZCRLF

      ld    de,CONTINUE$MSG         ;To Abort enter ESC. Any other key to continue.
      call  PSTRING
      call  ZCI
      cp    ESC
      JP    Z,IDE_LOOP
      call  ZCRLF
NEXTSEC1:

      ld    hl,(@SEC)
      inc   hl
      ld    (@SEC),hl
      ld    a,L               ;0 to 62 CPM Sectors
      cp    MAXSEC-1
      jp    nz,NEXTSEC

      ld    hl,0              ;Back to CPM sector 0
      ld    (@SEC),hl
      ld    hl,(@TRK)         ;Bump to next track
      inc   hl
      ld    (@TRK),hl
      jp    NEXTSEC                 ;Note will go to last sec on disk unless stopped


DISPLAYposition:              ;Display current DMA, track,sector & head position
      LD    DE,msgDMA         ;Show current DMA Address
      CALL  PSTRING
      ld    a,(@DMA+1)        ;High DMA byte
      call  phex
      ld    a,(@DMA)          ;Low DMA byte
      call  phex

      ld    de,msgCPMTRK            ;Display in LBA format
      call  PSTRING                 ;---- CPM FORMAT ----
```

```
      ld    a,(@TRK+1)        ;High TRK byte
      call  phex
      ld    a,(@TRK)          ;Low TRK byte
      call  phex

      ld    de,msgCPMSEC
      call  PSTRING                 ;SEC = (16 bits)
      ld    a,(@SEC+1)        ;High Sec
      call  phex
      ld    a,(@SEC)          ;Low sec
      call  phex
                              ;---- LBA FORMAT ----
      ld    de, msgLBA
      call  PSTRING                 ;(LBA = 00 (<-- Old "Heads" = 0 for these drives).
      ld    a,(@DRIVE$TRK+1)  ;High "cylinder" byte
      call  phex
      ld    a,(@DRIVE$TRK)          ;Low "cylinder" byte
      call  phex
      ld    a,(@DRIVE$SEC)
      call  phex
      ld    de, MSGBracket          ;)$
      call  PSTRING
      ret


SHOW$ID:
      call  driveid                 ;Get the drive ID info. If there is no drive, abort
      jp    z,ID$OK1
      ld    de,ID$ERROR
      call  PSTRING
      call  SHOWerrors
      jp    ABORT

ID$OK1:                             ;print the drive's model number
      ld    de, msgmdl
      call  PSTRING
      ld    hl,IDbuffer + 54
      ld    b,10              ;character count in words
      call  printname         ;Print [HL], [B] X 2 characters
      call  ZCRLF
                              ;print the drive's serial number
      ld    de, msgsn
      call  PSTRING
      ld    hl,IDbuffer + 20
      ld    b, 5              ;Character count in words
```

```
      call  printname
      call  ZCRLF
                           ;Print the drive's firmware revision string
      ld    de, msgrev
      call  PSTRING
      ld    hl,IDbuffer + 46
      ld    b, 2
      call  printname      ;Character count in words
      call  ZCRLF
                           ;print the drive's cylinder, head, and sector specs
      ld    de, msgcy
      call  PSTRING
      ld    hl,IDbuffer + 2
      call  printparm
      ld    de,msghd
      call  PSTRING
      ld    hl,IDbuffer + 6
      call  printparm
      ld    de, msgsc
      call  PSTRING
      ld    hl,IDbuffer + 12
      call  printparm
      call  ZCRLF
                           ;Default position will be first block
      ld    hl,0
      ld    (@SEC),hl      ;Default to Track 0, Sec 0
      ld    (@TRK),hl
      ld    hl,buffer      ;Set DMA address to buffer
      ld    (@DMA),hl

      call  IDEinit              ;For some reason this need to be here after getting the drive ID.
      jp    IDE_LOOP       ;otherewise sector #'s are off by one!



printname:                 ;Send text up to [B]
      inc   hl             ;Text is low byte high byte format
      ld    c,(hl)
      call  ZCO
      dec   bc
      ld    c,(hl)
      call  ZCO
      inc   hl
      inc   hl
```

```
      DEC     B
      jp    nz,printname
      ret


;     Print a string in [DE] up to '$'

PSTRING:
      push  bc
      push  de
      push  hl
      ex    de,hl
PSTRX:      ld    a,(hl)
      cp    '$'
      jp    z,DONEP
      ld    c,A
      call  ZCO
      inc   hl
      jp    PSTRX
DONEP:      pop   hl
      pop   de
      pop   bc
      ret


SHOWerrors:
   IF NOT DEBUG
      or    A               ;Set NZ flag
      scf                   ;Set Carry Flag
      ret
   ELSE
      call  ZCRLF
      ld    e,REGstatus     ;Get status in status register
      call  IDErd8D
      ld    a,D
      and   1H
      jp    nz,MoreError          ;Go to  REGerr register for more info
                            ;All OK if 01000000
      push  af              ;save for return below
      and   80H
      jp    z,NOT7
      ld    de,DRIVE$BUSY         ;Drive Busy (bit 7) stuck high.   Status =
      call  PSTRING
      jp    DONEERR
NOT7: and   40H
```

```
      jp    nz,NOT6
      ld    de,DRIVE$NOT$READY      ;Drive Not Ready (bit 6) stuck low.  Status =
      call  PSTRING
      jp    DONEERR
NOT6: and   20H
      jp    nz,NOT5
      ld    de,DRIVE$WR$FAULT ;Drive write fault.    Status =
      call  PSTRING
      jp    DONEERR
NOT5  LD    DE,UNKNOWN$ERROR
      call  PSTRING
      jp    DONEERR

MoreError:                    ;Get here if bit 0 of the status register indicted a problem
      ld    e,REGerr          ;Get error code in REGerr
      call  IDErd8D
      ld    a,D
      push  af

      and   10H
      jp    z,NOTE4
      ld    de,SEC$NOT$FOUND
      call  PSTRING
      jp    DONEERR

NOTE4:      and   80H
      jp    z,NOTE7
      ld    de,BAD$BLOCK
      call  PSTRING
      jp    DONEERR
NOTE7:      and   40H
      jp    z,NOTE6
      ld    de,UNRECOVER$ERR
      call  PSTRING
      jp    DONEERR
NOTE6:      and   4H
      jp    z,NOTE2
      ld    de,INVALID$CMD
      call  PSTRING
      jp    DONEERR
NOTE2:      and   2H
      jp    z,NOTE1
      ld    de,TRK0$ERR
      call  PSTRING
      jp    DONEERR
```

```
NOTE1:      ld    de,UNKNOWN$ERROR1
      call  PSTRING
      jp    DONEERR

DONEERR:pop af
      push  af
      call  ZBITS
      call  ZCRLF
      pop   af
      or    A                ;Set Z flag
      scf                    ;Set Carry flag
      ret
   ENDIF


;---------------------------------------------------------------------
; Print a 16 bit number in RAM located @ [HL]
; (Note Special Low Byte First. Used only for Drive ID)

printparm:
      inc   hl               ;Index to high byte first
      ld    a,(hl)
      call  PHEX
      dec   bc               ;Now low byte
      ld    a,(hl)
      call  PHEX
      ret

; Print an 8 bit number, located in [A]
PHEX: push  af
      push  bc
      push  af
      rrca
      rrca
      rrca
      rrca
      call  ZCONV
      pop   af
      call  ZCONV
      pop   bc
      pop   af
      ret

ZCONV:      and   0FH              ;HEX to ASCII and print it
      add   a,90H
```

```
      daa
      adc   a,40H
      daa
      ld    c,A
      call  ZCO
      ret

INIT_LBA:                     ;Initilize LBA addresss & DNA Addresss
      LD    A,0
      ld    (@SEC),a
      ld    (@SEC+1),a
      ld    (@TRK),a
      ld    (@TRK+1),a
      LD    HL,buffer         ;set default DMA address to RAM_BASE
      LD    (@DMA),HL
      RET

ghex32lba:                    ;get CPM style Track# & Sector# data and convert to LBA format
      ld    de,ENTER$SECL           ;Enter sector number
      call  PSTRING
      call  GETHEX                  ;get 2 HEX digits
      ret   c
      ld    (@SEC),a          ;Note: no check data is < MAXSEC, sectors start 0,1,2,3....
      call  ZCRLF

      ld    de,ENTER$TRKL           ;Enter low byte track number
      call  PSTRING
      call  GETHEX                  ;get 2 more HEX digits
      ret   c
      ld    (@TRK),a
      call  ZCRLF

      ld    de,ENTER$TRKH           ;Enter high byte track number
      call  PSTRING
      call  GETHEX                  ;get 2 more HEX digits
      ret   c
      ld    (@TRK+1),a
      call  ZCRLF
      xor   A
      or    A                 ;To return NC
      ret


GETHEX:
      call  GETCMD                  ;Get a character from keyboard & ECHO
```

```
      cp    ESC
      jp    z,HEXABORT
      cp    '/'               ;check 0-9, A-F
      jp    c,HEXABORT
      cp    'F'+1
      jp    nc,HEXABORT
      call  ASBIN             ;Convert to binary
      rlca                    ;Shift to high nibble
      rlca
      rlca
      rlca
      ld    b,A               ;Store it
      call  GETCMD                  ;Get 2nd character from keyboard & ECHO
      cp    ESC
      jp    z,HEXABORT
      cp    '/'               ;check 0-9, A-F
      jp    c,HEXABORT
      cp    'F'+1
      jp    nc,HEXABORT
      call  ASBIN             ;Convert to binary
      or    B                 ;add in the first digit
      or    A                 ;To return NC
      ret
HEXABORT:
      scf                     ;Set Carry flag
      ret


GETCMD:     call  ZCI               ;GET A CHARACTER, convert to UC, ECHO it
      call  UPPER
      cp    ESC
      ret   z                 ;Don't echo an ESC
      push  af                ;Save it
      push  bc
      ld    c,A
      call  ZCO               ;Echo it
      pop   bc
      pop   af                ;get it back
      ret

                              ;Convert LC to UC
UPPER:      cp    'a'               ;must be >= lowercase a
      ret   c                 ; else go back...
      cp    'z'+1             ;must be <= lowercase z
      ret   nc                ; else go back...
```

```
      sub   'a'-'A'                 ;subtract lowercase bias
      ret

                              ;ASCII TO BINARY CONVERSION ROUTINE
ASBIN:      sub   30H
      cp    0AH
      ret   m
      sub   07H
      ret


HEXDUMP:                      ;Print a hexdump of the data in the 512 byte buffer @[HL]
      push  af                ;Save everything
      push  bc
      push  de
      push  hl

      call  ZCRLF             ;CR/LF first
      ld    d,32              ;Print 32 lines total
      ld    b,16              ;16 characters across
      ld    (@StartLineHex),hl      ;Save the buffer location for ASCII display below
      ld    hl,0
      ld    (@BYTE$COUNT),hl

SF172:      call  ZCRLF
      ld    hl,(@BYTE$COUNT)
      ld    a,H
      call  PHEX              ;Print byte count in sector
      ld    a,L
      call  PHEX
      push  de
      ld    de,16
      add   hl,de
      pop   de
      ld    (@BYTE$COUNT),hl  ;store for next time
      call  BLANK
      ld    hl,(@StartLineHex)
      ld    (@StartLineASCII),hl    ;Store for ASCII display below
SF175:      ld    a,(hl)
      call  LBYTE             ;Display [A] on CRT/LCD
      inc   hl
      DJNZ  SF175
      ld    (@StartLineHex),hl      ;Save for next line later
      call  ShowAscii         ;Now translate to ASCII and display
      ld    b,16              ;16 characters across for next line
```

```
      dec   D
      jp    nz,SF172          ;Have we done all 32 lines
;
      call  ZCRLF
      pop   hl                ;Get back origional registers
      pop   de
      pop   bc
      pop   af
      ret

ShowAscii:                    ;Now show as ascii info
      ld    hl,(@StartLineASCII)
      ld    b,16              ;16 ASCII characters across
XF172:      call  BLANK             ;send a space character
      call  BLANK
XF175:      ld    a,(hl)
      and   7FH
      cp    ' '               ;FILTER OUT CONTROL CHARACTERS
      jp    nc,XT33
XT22: ld    a,'.'
XT33: cp    07CH
      jp    nc,XT22
      ld    c,A               ;SET UP TO SEND
      push  bc
      call  ZCO
      pop   bc
      inc   hl                ;Next position in buffer
      DJNZ  XF175
      ret
;
;
;=============================================================================================
;
;      IDE Drive BIOS Routines written in a format that can be used directly with CPM3
;
;=============================================================================================
;
IDEinit:                      ;Initilze the 8255 and drive then do a hard reset on the drive,
      ld    a,READcfg8255           ;Config 8255 chip (10010010B), read mode on return
      out   (IDEportCtrl),a         ;Config 8255 chip, READ mode

                              ;Hard reset the disk drive
                              ;For some reason some CF cards need to the RESET line
                              ;pulsed very carefully. You may need to play around
      ld    a,IDErstline            ;with the pulse length. Symptoms are: incorrect data comming
```

```
      out   (IDEportC),a            ;back from a sector read (often due to the wrong sector being read)
                                    ;I have a (negative)pulse of 2.7uSec. (10Mz Z80, two IO wait states).
      ld    b,20H                   ;Which seem to work for the 5 different CF cards I have.
ResetDelay:
      DEC     B
      jp    nz,ResetDelay           ;Delay (reset pulse width)

      xor   A
      out   (IDEportC),a            ;No IDE control lines asserted (just bit 7 of port C)
      call  DELAY$32

      ld    d,11100000b             ;Data for IDE SDH reg (512bytes, LBA mode,single drive,head 0000)
                                    ;For Trk,Sec,head (non LBA) use 10100000
                                    ;Note. Cannot get LBA mode to work with an old Seagate Medalist 6531 drive.
                                    ;have to use teh non-LBA mode. (Common for old hard disks).

      ld    e,REGshd                ;00001110,(0EH) for CS0,A2,A1,
      call  IDEwr8D                 ;Write byte to select the MASTER device
;
      ld    b,0FFH                  ;<<< May need to adjust delay time for hard disks
WaitInit:
      ld    e,REGstatus             ;Get status after initilization
      call  IDErd8D                 ;Check Status (info in [D])
      ld    a,D
      and   80H
      jp    z,DoneInit              ;Return if ready bit is zero
      ld    a,2
      call  DELAYX                  ;Long delay, drive has to get up to speed
      DEC     B
      jp    nz,WaitInit
      call  SHOWerrors              ;Ret with NZ flag set if error (probably no drive)
      ret
DoneInit:
      xor   A
      ret


DELAYX:     ld    (@DELAYStore),a
      push  bc
      ld    bc,0FFFFH               ;<<< May need to adjust delay time to allow cold drive to
DELAY2:     ld    a,(@DELAYStore)         ;    get up to speed.
DELAY1:     DEC     A
      jp    nz,DELAY1
      dec   bc
      ld    a,C
```

```
      or    B
      jp    nz,DELAY2
      pop   bc
      ret


DELAY$32: ld      a,40              ;DELAY ~32 MS (DOES NOT SEEM TO BE CRITICAL)
DELAY3:     ld    b,0
M0:   DJNZ  M0
      DEC     A
      jp    nz,DELAY3
      ret


                          ;Read a sector, specified by the 3 bytes in LBA
                          ;Z on success, NZ call error routine if problem
READSECTOR:
      call  wrlba         ;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
      jp    c,SHOWerrors        ;Returned with NZ set if error

      ld    d,COMMANDread
      ld    e,REGcommand
      call  IDEwr8D             ;Send sec read command to drive.
      call  IDEwaitdrq    ;wait until it's got the data
      jp    c,SHOWerrors

      LD    HL,(@DMA)     ;DMA address
      ld    b,0           ;Read 512 bytes to [HL] (256X2 bytes)
MoreRD16:
      ld    a,REGdata     ;REG regsiter address
      out   (IDEportC),a

      or    IDErdline     ;08H+40H, Pulse RD line
      out   (IDEportC),a

      in    a,(IDEportA)        ;Read the lower byte first (Note very early versions had high byte then low byte
      ld    (hl),A              ;this made sector data incompatable with other controllers).
      inc   hl
      in    a,(IDEportB)        ;THEN read the upper byte
      ld    (hl),A
      inc   hl
```

```
      ld    a,REGdata         ;Deassert RD line
      out   (IDEportC),a
      DJNZ  MoreRD16

      ld    e,REGstatus
      call  IDErd8D
      ld    a,D
      and   1H
      call  nz,SHOWerrors           ;If error display status
      ret

                            ;Write a sector, specified by the 3 bytes in LBA
                            ;Z on success, NZ to error routine if problem
WRITESECTOR:
      call  wrlba           ;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
      jp    c,SHOWerrors

      ld    d,COMMANDwrite
      ld    e,REGcommand
      call  IDEwr8D                 ;tell drive to write a sector
      call  IDEwaitdrq        ;wait unit it wants the data
      jp    c,SHOWerrors

      ld    hl,(@DMA)
      ld    b,0               ;256X2 bytes

      ld    a,WRITEcfg8255
      out   (IDEportCtrl),a

WRSEC1:     ld    a,(hl)
      inc   hl
      out   (IDEportA),a            ;Write the lower byte first (Note early versions had high byte then low byte
      ld    a,(hl)                  ;this made sector data incompatable with other controllers).
      inc   hl
      out   (IDEportB),a            ;THEN High byte on B
      ld    a,REGdata
      push  af
      out   (IDEportC),a            ;Send write command
      or    IDEwrline         ;Send WR pulse
      out   (IDEportC),a
      pop   af
      out   (IDEportC),a
```

```
      DJNZ  WRSEC1

      ld    a,READcfg8255           ;Set 8255 back to read mode
      out   (IDEportCtrl),a

      ld    e,REGstatus
      call  IDErd8D
      ld    a,D
      and   1H
      call  nz,SHOWerrors           ;If error display status
      ret


wrlba:                              ;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 drives
      ld    a,(@SEC)          ;LBA mode Low sectors go directly
      inc   A                 ;Sectors are numbered 1 -- MAXSEC (even in LBA mode)
      ld    (@DRIVE$SEC),a          ;For Diagnostic Display Only
      ld    d,A
      ld    e,REGsector       ;Send info to drive
      call  IDEwr8D                 ;Note: For drive we will have 0 - MAXSEC sectors only

      ld    hl,(@TRK)
      ld    a,L
      ld    (@DRIVE$TRK),a
      ld    d,L               ;Send Low TRK#
      ld    e,REGcylinderLSB
      call  IDEwr8D

      ld    a,H
      ld    (@DRIVE$TRK+1),a
      ld    d,H               ;Send High TRK#
      ld    e,REGcylinderMSB
      call  IDEwr8D

      ld    d,1               ;For now, one sector at a time
      ld    e,REGseccnt
      call  IDEwr8D
      ret


IDEwaitnotbusy:                     ;ie Drive READY if 01000000
      ld    b,0FFH
      ld    a,0FFH                  ;Delay, must be above 80H for 4MHz Z80. Leave longer for slower drives
```

```
      ld    (@DELAYStore),a

MoreWait:
      ld    e,REGstatus       ;wait for RDY bit to be set
      call  IDErd8D
      ld    a,D
      and   11000000B
      xor   01000000B
      jp    z,DoneNotbusy
      DEC     B
      jp    nz,MoreWait
      ld    a,(@DELAYStore)         ;Check timeout delay
      DEC     A
      ld    (@DELAYStore),a
      jp    nz,MoreWait
      scf                     ;Set carry to indicqate an error
      ret
DoneNotBusy:
      or    A                 ;Clear carry it indicate no error
      ret

                              ;Wait for the drive to be ready to transfer data.
                              ;Returns the drive's status in Acc
IDEwaitdrq:
      ld    b,0FFH
      ld    a,0FFH                  ;Delay, must be above 80H for 4MHz Z80. Leave longer for slower drives
      ld    (@DELAYStore),a

MoreDRQ:
      ld    e,REGstatus       ;wait for DRQ bit to be set
      call  IDErd8D
      ld    a,D
      and   10001000B
      cp    00001000B
      jp    z,DoneDRQ
      DEC     B
      jp    nz,MoreDRQ
      ld    a,(@DELAYStore)         ;Check timeout delay
      DEC     A
      ld    (@DELAYStore),a
      jp    nz,MoreDRQ
      scf                     ;Set carry to indicate error
      ret
DoneDRQ:
      or    A                 ;Clear carry
```

```
      ret


SET_DMA:
      LD    HL,DMA_Loc_MSG2         ;Enter starting RAM buffer (DMA) location (xxxxH):$'
      CALL  PRINT_STRING
      CALL  ZGETHL                  ;Setup DMA location in [HL]
      LD    (@DMA),HL
      CALL  CRLF
      JP    IDE_LOOP

;---------------------------------------------------------------------
; 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 I/O to the drive (which is only for SEC R/W) is done directly
; in the routines READSECTOR & WRITESECTOR for speed reasons.
;
IDErd8D:                        ;READ 8 bits from IDE register in [E], return info in [D]
      ld    a,E
      out   (IDEportC),a              ;drive address onto control lines

      or    IDErdline           ;RD pulse pin (40H)
      out   (IDEportC),a              ;assert read pin

      in    a,(IDEportA)
      ld    d,A                 ;return with data in [D]

      ld    a,E                 ;<---Ken Robbins suggestion
      out   (IDEportC),a              ;deassert RD pin

      xor   A
      out   (IDEportC),a              ;Zero all port C lines
      ret


IDEwr8D:                        ;WRITE Data in [D] to IDE register in [E]
      ld    a,WRITEcfg8255            ;Set 8255 to write mode
      out   (IDEportCtrl),a

      ld    a,D                 ;Get data put it in 8255 A port
      out   (IDEportA),a

      ld    a,E                 ;select IDE register
      out   (IDEportC),a
```

```
      or    IDEwrline         ;lower WR line
      out   (IDEportC),a

      ld    a,E               ;<-- Ken Robbins suggestion, raise WR line
      out   (IDEportC),a            ;deassert RD pin

      xor   A                 ;Deselect all lines including WR line
      out   (IDEportC),a

      ld    a,READcfg8255           ;Config 8255 chip, read mode on return
      out   (IDEportCtrl),a
      ret


CPM$MOVE$CODE                       ;This code is written to reside at 0H. Will be relocated by this
      ld    hl,BUFFER         ;this program to move the boot CPMLDR to 100H in RAM (Overwriting this program)
      ld    de,100H
      ld    bc,(12*512)
      LDIR
      jp    100H
CPM$MOVE$CODE$END:

;-----------------------------------------------------------------------------------
;
;
;
SIGNON_MSG: DB SCROLL,QUIT,NO_ENH,FAST,BELL,CR,LF,LF
            DB 'SBC-Z80 ROM MONITOR @ E000H (V2.2 J.Monahan,10/14/2015)$'

MAIN_MENU_MSG:    DB    CR,LF,LF
            DB    'A=Memmap        D=Display RAM     E=Echo Text    F=Fill RAM '
            DB    CR,LF
            DB    'G=Goto Address  I=IDE Menu        J=Test RAM     K=Show Menu'
            DB    CR,LF
            DB    'M=Move RAM      O=Boot 8086       QI,O=Port      P=Boot CPM'
            DB    CR,LF
            DB    'R=Show Ports    S=Subsitute RAM   T=RAM Ascii    V=Verify RAM'
            DB    CR,LF
            DB      'X=XModem        Y=Swap RAM Page   Z=Top Of RAM'
            DB    CR,LF,LF,'$'

IDE_MENU_MSG:     DB    CR,LF,LF,'SBC-Z80 ROM MONITOR IDE MENU',CR,LF,LF
            DB    'L=Set LBA value     R=Read 1 Sec to Buffer     W=Write Buffer to 1 Sec',CR,LF
            DB    'S=Show Sec Data     V=Read N Secs to buffer    X=Write buffer to N Secs',CR,LF
            DB    'N=LBA Next Sector   M=LBA Previous Sector      P=Boot CPM',CR,LF
```

```
            DB    'D=Buffer Address    Y=CF Card Paramerers       (ESC) Back to main menu',CR,LF,LF,'$'

SP_MSG              DB    'SP=$'
IOBYTE_MSG  DB      ' IOBYTE=$'
MODEM_SIGNON:       DB    CR,LF,'Get a File from a PC',13,10,'$'
RMSG:       DB      'WAITING FOR SECTOR #$'
ERRSOH:             DB    'H RECEIVED, NOT SOH',0DH,0AH,'$'
ERR2:       DB      '++BAD SECTOR # IN HDR',0DH,0AH,'$'
ERR3:       DB      '++BAD CKSUM ON SECTOR',0DH,0AH,'$'
TOUTM:              DB    'TIMEOUT $'
QUITM:              DB    0DH,0AH,'MULTIPLE ERRORS.'
            DB      0DH,0AH,'TYPE Q TO QUIT, R TO RETRY:$'
MODEM_DONE_MSG:     DB    13,10,'TRANSFER COMPLETE$'
BAD_HEADER_MSG:     DB    CR,LF,'INVALID HEADER.',0DH,0AH,'$'
MODEM_RAM_MSG:      DB    'H. If OK will write to RAM at $'
MODEM_RAM_LOC:      DB    CR,LF,'Enter RAM location (xxxxH +CR): $'
MSG_8086    DB      CR,LF,'Switching to 8086 Board.'CR,LF,'$'
SWAP_RAM_MSG        DB    CR,LF,'Swap Lowest 32K of RAM. (Enter 0 or 1)$'
PAGE0_MSG   DB      CR,LF,'Page 0 RAM active',CR,LF,'$'
PAGE1_MSG   DB      CR,LF,'Page 1 RAM active',CR,LF,'$'
ALL_PORTS   DB      CR,LF,'Active I/O Ports detected:-',CR,LF,'$'
Invalid_Msg DB      CR,LF,BELL,'Invalid Data',CR,LF,'$'

INIT$ERROR: DB      'INITILIZING DRIVE ERROR.',CR,LF,'$'
ID$ERROR:   DB      'ERROR OBTAINING DRIVE ID.',CR,LF,'$'
msgmdl:             DB    'Model: $'
msgsn:              DB    'S/N:   $'
msgrev:             DB    'Rev:   $'
msgcy:              DB    'Cylinders: $'
msghd:              DB    ', Heads: $'
msgsc:              DB    ', Sectors: $'
msgDMA:             DB    'Buffer = $'
msgCPMTRK:  DB      'H,  CPM TRK = $'
msgCPMSEC:  DB      ',  CPM SEC = $'
msgLBA:             DB    ',  (LBA = 00$'
MSGBRACKET  DB      ')$'

Prompt:             DB    CR,LF,LF,'Please enter command >$'
msgsure:    DB      CR,LF,'Warning: this will change data on the drive, '
            DB      'are you sure? (Y/N)...$'
msgrd:              DB    CR,LF,'Sector Read OK',CR,LF,'$'
msgwr:              DB    CR,LF,'Sector Write OK',CR,LF,'$'
GET$LBA:    DB      'ENTER CPM STYLE TRK & SEC VALUES (IN HEX).',CR,LF,'$'
SEC$RW$ERROR        DB    'DRIVE ERROR, Status Register = $'
ERR$REG$DATA        DB    'DRIVE ERROR, Error Register = $'
```

```
ENTER$SECL  DB    'Starting sector number,(xxH) = $'
ENTER$TRKL  DB    'Track number (LOW byte, xxH) = $'
ENTER$TRKH  DB    'Track number (HIGH byte, xxH) = $'
ENTER$HEAD  DB    'Head number (01-0f) = $'
ENTER$COUNT DB    'Number of sectors to R/W = $'
DRIVE$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. $'
READN$MSG   DB    CR,LF,'Read multiple sectors from current disk/CF card to RAM buffer.'
            DB    CR,LF,'How many 512 byte sectores (xx HEX):$'
WRITEN$MSG  DB    CR,LF,'Write multiple sectors RAM buffer current disk/CF card.'
            DB    CR,LF,'How many 512 byte sectores (xx HEX):$'
MultiRD_MSG DB    CR,LF,'Reading Sec $'
MultiWR_MSG DB    CR,LF,'Writing Sec $'

MSGERR            DB    CR,LF,'Sorry, that was not a valid menu option!$'
ATEND       DB    CR,LF,'At end of disk partition!',CR,LF,'$'
H$MSG       DB    'H$'
RANGE$MSG   DB    CR,LF,'Sector value out of range.',CR,LF,'$'
CPM$ERROR   DB    CR,LF,'Error reading CPMLDR.',CR,LF,'$'
CPM$ERROR1  DB    CR,LF,'Data error reading CPMLDR.',CR,LF,'$'
DMA_Loc_MSG2      DB    CR,LF,'Enter starting RAM buffer (DMA) location (xxxxH):$'
BOOTCPM$MSG DB    CR,LF,'Booting CPM',CR,LF,'$'
ECHO_MSG    DB    CR,LF,'Test keyboard by typing characters. Type ^C to abort',CR,LF,'$'
RAM_TEST_MSG    DB      CR,LF,'Enter RAM range (xxxx,yyyy). Any stuck bytes will be displayed',CR,LF,'$'

; -------------------------- RAM usage ----------------------------------------

IDBUFFER    EQU   RAM_BASE+ 1000H              ;512 Bytes @ D000H for CF-Card paramaters
buffer            EQU   RAM_BASE               ;Default DMA buffer at D000H

@DMA        EQU   RAM_BASE + 1F00H        ;Remember Stack is at ~DFF0H
@DRIVE$SEC  EQU   RAM_BASE + 1F02H
@DRIVE$TRK  EQU   RAM_BASE + 1F04H
@SEC        EQU   RAM_BASE + 1F06H
@TRK        EQU   RAM_BASE + 1F08H
```

```
@STARTLINEHEX     EQU   RAM_BASE + 1F10H
@STARTLINEASCII   EQU   RAM_BASE + 1F12H
@BYTE$COUNT EQU   RAM_BASE + 1F14H
@SECTOR$COUNT     EQU   RAM_BASE + 1F16H
@DELAYSTORE EQU   RAM_BASE + 1F18H

;END
```