

```

; THIS IS A PROGRAM TO EXAMINE & FORMAT CPM DISKS USEING A VERSAFLOPPY II DISK
; CONTROLLER. NOTE IT IS SELF A CONTAINED PROGRAM (EXCEPT FOR CONSOLE I/O)
;
; It is a completely new Diagnostic program and utilizes a new set of core BIOS
; like functions that can easily be adapted for CPM+ etc. It does not utilize
; any of the old SD_Systems BIOS/Diagnostic code (which I found to be convoluted, difficult
; to modify and specific for a few disk formats). This program utilizes a disk parameter
; table driven approach for many common 8" & 5" formats. Others can be easily added.
; It works with both the 1791 & 1795 chips (Set the EQU's below).
;
; The program requires a Z80 CPU and utilizes the marvelous Z80 assembler (ZASMB) written
; by P.F Ridler in 1984. See (http://retrotechnology.com/herbs\_stuff/s\_sd.html#other)
; on Herbs site. The assembler here was setup to take long label names (see the docs).
; However most common Z80 assemblers should work, with minor changes to a few lines of code.
; ZASMB allows the use uses a few more logical OP codes that Zilog left out
; (eg CP A,20H rather than CP 20H). This assembler is extremely fast and generates
; direct .com code. It does however have limited ifdef etc options.
;
;
; JOHN MONAHAN (monahan@vitasoft.org) 27/4/2009 VERSION 1.0
;
; V0.1 Basic Sector ID field working
; V0.2 Seek test jumpy
; V0.3 Seek done, before read sector core stuff
; V0.5 Sector read OK for 8" SD
; V0.7 Sector Read blanked out menu, itself OK. Seek redone.
; V0.8 Reading 128 byte sectors with my READ_SECTOR routine OK
; V0.9 Block read for sectors
; V0.10 Start of Formatting OK
; V0.11 Filled in comments of formatting
; V0.12 Start of sector track write image display
; V0.14 Fixed abort etc. Sec formatting display working, ESC to stop display
; V0.15 Sec ID errors on display. general cleanup of abort stuff
; V0.16 Variable sec size display. Second side not working
; V0.17 Second side working before splitting UNIT into two sections (Hardware & Software)
; V0.18 New format using IO_BYTE and IX pointer to tables for disk/drive info.
; V0.19 Before simplifying moving IO_BYTE data to Disk paramater table
; V0.20 Start of HW_BYTE in disk paramater table. No format yet.
; V0.21 8" format done with new layout.
; V0.22 Before New Format Routine & Tables
; V0.24 Before switching to my memory track build routines
; V0.25 My track image not working yet
; V0.27 Software SD Track (no sides) working. No hardware yet
; V0.28 8" format (CPM cannot read it). Cannot access 5" drives
; V0.29 Straightened our 5". (DW in table). Still CPM read problem
; V0.30 Start of track dump SD 8" Format now working
; V0.31 Cleaned up seek no verify functions to one general function
; V0.33 Seems to be working now
; V0.34 Random sector read test working (write not done yet)
; V0.35 Start or read sec & seek error returns. Have seperate flag on CMD's
; V0.37 New error routines Seek done. Modeled after VF1 Bios
; V0.38 New error routines, centralized Home, Seek etc started.
; V0.39 New format for sec read with seek_V done
; V0.40 New side selection and redo of odds and ends
; V0.41 Split menu's for 5" & 8" drives. All OK on Sec reads
; V0.42 Dump track working again.
; V0.43 Format not working
; V0.44 Re-did disk format menus and selection. Started on Format stuff
; V0.45 Combine IY & IX reg flags into a single IX+HW_BYTE flag
; V0.46 Put track size info in tables and use it for track dumps.
; V0.48 Cleaned up things. END_ROUTINE put in.
; V0.50 Sector Read/write done (track format broken)
; V0.51 Format not working.
; V0.52 Format SSSD 8" seems to be OK now. Need READ_ID at start
; V0.53 Format was still not working. Changed chip and used Restore with V flag on.
; V0.54 SOLID FORMATTING of 8". Moved all 1791/5 CMD's to drive tables. (Need to up 5")
; V0.55 Formating of al 8" SS disks working
; V0.56 Combined all sec R/W tests into one core routine
; V0.58 Working on side selection, fixed some formatting issues including CPM86 disks

```

```

;      V0.59  Fixed side_sel to actually select the proper side (Note debug display present)
;      V0.60  Formatting 5' DDDS disk now working and compatible with CPM3
;      V0.61  Many formats added for 5" disks
;      V0.62  Before skew tables for formatting (Removed HP & KAYPRO)
;      V0.63  Skew table for formatting done DD & SD
;      V0.64  All formats done and working. Corrected (DRIVE)->IY problem
;      V0.65  Load CMD done
;      V0.66  Both sector dump and load working. repaired disk selection again.
;      V0.67  Split out all the sector R/W functions again into separate routines - easier to maintain.
;      V0.68  Moved 179x CMDS to start. Take care of 1795 side bits. Rearranged main menu
;      V0.69  Solve status port hang-up in invalid hardware.
;      V0.70  Started Disk copy
;      V0.71  Disk copying of 8" done. Slow with no sector skews
;      V0.72  Skew for disk copying inserted - still slow
;      V0.73  Went to multi-sec R/W but errors so far
;      V0.74  Installed Verify cmd as well. Copy now working for 128 byte sectors
;      V0.75  Improvements to M_SEC_RD/WR
;      V0.76  Multi sector R/W working.
;      V0.77  Step-in CMD implemented but copy still requires HOME on A:
;      V0.78  System copy done but needs work. Need to have global COPY,COMPARE etc.
;      V0.79  Copy disk generalized/global now working on DS disks
;      V0.80  Verify command now generalized and working
;      V0.81  Sys copy done, 1795/91 mods added
;      V0.82  Problems with 1795 seek & drive select.
;      V0.83  Complete rearrangement of hardware selection. Seek working up to before Format
;      V0.84  All cleaned up for 1791 chip
;      V0.85  Had to flip A & B side hardware I/O selections for 1795 chip
;      V0.86  Hardware side select finally straightened out. Before IY usage
;      V0.87  Initial IY usage - untested
;      V0.88  IY working throughout. Seems solid for 1791 & 1795 chips.
;      V1.01  Set only 1 system track for CPM 1K sector 8" disks (instead of instead of 2)
;      V1.02  Strip parity bit from direct keyboard input (if any).
;
;      To Do:-
;;      Add 5" dos sectors format initialization
;
;
;
;
FALSE EQU 0
TRUE EQU 1
;
CHIP_1795 EQU FALSE ;<---- RD & WR sector cmd's bit patterns are different for these chips!
CHIP_1791 EQU TRUE
;
NBYTES EQU 128 ;BYTES PER SECTOR UNIT (eg, *2 FOR 256 BYTE SECTORS)
LF EQU 0AH
CR EQU 0DH
BELL EQU 07H
CLEAR EQU 1AH ;SD Systems Video Board Clear Screen
TAB EQU 09H
ESC EQU 1BH
EOL EQU 1CH ;SD Systems Video Board Clear to end of line
PAGE_SIZE EQU 16 ;Number of lines at a time to display memory contents on CRT
STATUS_DELAY EQU 5 ;Time-out for waiting for status port to ho not busy. (~5 seconds @ 4MHz)
;
RO_FLAG EQU 01 ;Flag for sequential sectors R test (Read sectors)
WO_FLAG EQU 02 ;Flag for sequential sectors W test (Write to sectors)
RW_FLAG EQU 03 ;Flag for sequential sectors R/W test (Both)
RRW_FLAG EQU 04 ;Flag for random track and sector R/W test
TRK_RO_FLAG EQU 05 ;Flag for reading sectors from just one track continuously
CPM86_FLAG EQU 01 ;Flag to indicate after 5" disk formatting CPM86 first sector
;needs to be modified
TEST_FILL EQU 01 ;Character to write for sector write test.
;
SIMPLE EQU 00h ;Output information detail to CRT
COMPLEX EQU 0FFH
;
;Will place these values here for easy RAM analysis if a crash

```

```

;The rest go to the end of the program
TADDR EQU 40H ;STORE FOR DMA ADDRESS
@TRK EQU 43H ;NEW TRACK <---- DO not change the order of these locations
@SIDE EQU 44H ;NEW SIDE the IY register will ALWAYS point to TRK
@SCTR EQU 45H ;NEW SECTOR
DRIVE_1 EQU 46H ;CURRENT DRIVE SELECT BITS
DRIVE_2 EQU 47H ;SECOND DRIVE SELECT BITS (For copy, verify, sysgen etc)
IOBYTE EQU 48H ;Combined drive selection, density, side and size bits for hardware
; ;is inverted and sent to the VFII "SELECT" port
;
; PORTS & COMMANDS FOR FOR 1791 or 1795
;
RSET EQU 50H ;<----- VERSAFLOPPY BASE PORT ADDRESS (I use Port 50H, SD-Systems uses 60H)
SELECT EQU 53H ;DRIVE SELECT PORT
STATUS EQU 54H ;STATUS PORT
TRACK EQU 55H ;TRACK PORT
SECTOR EQU 56H ;SECTOR PORT
DATA EQU 57H ;DATA PORT
CMD EQU 54H ;COMMAND PORT
;
;Note: the 1791/5 chip for most commands used bits 0&1 to set the head
;motor stepping rate. 00 being 3ms (@2MH clock),11 being 15 ms. My
;Tandon 8" drives take the fastest rate. Older drives may not.
;
RSCMD EQU 00001100B ;(0CH),RESTORE CMD <----- (Some drives require
SKNCMD EQU 00011000B ;(18H),SEEK NO VERIFY CMD <-- (a slower stepping rate r1,r0)
SKCMD EQU 00011100B ;(1CH),SEEK WITH VERIFY CMD <---
RDACMD EQU 11000100B ;(C0H),READ TRACK/SECTOR ID CMD
STEPIN EQU 01011100B ; Step-in verify on dstination track

RDCMD91 EQU 10000000B ;(80H),READ SECTOR CMD 1791 chip
WRCMD91 EQU 10100000B ;(A0H),WRITE SECTOR CMD

RDCMD95 EQU 10001000B ;(88H),READ SECTOR CMD 1795 chip
WRCMD95 EQU 10101000B ;(A8H),WRITE SECTOR CMD

WRTCMD EQU 11110100B ;(F4H),Write a whole track command
RDTCMD EQU 11100100B ;(E4H),Read a whole track command
;
;
SEC_RETRY_MAX EQU 4 ;Number of times to try R/W a sector before returning an error
SEEK_RETRY_MAX EQU 2 ;Number of times to try R/W a sector before returning an error
HOME_ERR_MASK EQU 80H ;Error mask for Type I Home CMD
SIN_ERR_MASK EQU 90H ;Step head in one track command error bits
SK_ERR_MASK EQU 90H ;Track Seek error bits
ID_ERR_MASK EQU 9FH ;Sector ID read error mask
RS_ERR_MASK EQU 0BFH ;Read sector data error mask
MRS_ERR_MASK EQU 0AFH ;Multi-sector Read data error mask
WS_ERR_MASK EQU 0EFh ;Write sector data error mask
MWS_ERR_MASK EQU 0EFh ;Multi-sector Write data error mask
RT_ERR_MASK EQU 80H ;Read Track error mask
WT_ERR_MASK EQU 0E0H ;Write Track error mask
;
;
FBUFFER EQU 5000H ;Buffer to build track image OR display sectors reads
FBUFFER2 EQU 6000H ;2nd buffer here for sector R/W test
;
;The Index register IX is used throughout the program to point to the disk paramater
;table of the currently selected drive/disk format
;(IOBYTE) will point to (DRIVE) and will contain the current DRIVE
;hardware selection bits. Drive select 0,1,2,3 and side A or side B bits as well as
;other flags (see below)
;Remember the bits for disk selection, density and side selection are inverted on
;the actual Versafloppy II board hardware
;Register IX is not used for anything else in this program.
;
;The Index register IY is used throughout the program to point to the current/requested
;Track, Side & Sector being worked on. They are at memory locations @TRK... IY will
;always point to @TRK and assumes the others follow. Do not relocate or change the order.

```

```

;
;
; EQUATES FOR [IX] REGISTAR INTO DISK PARAMATER TABLE
;
NSCTRS EQU 0 ;Sectors/Track for disk
NTRKS EQU 1 ;Tracks/Side
HW_BYTE EQU 2 ;Will contain bit flags for:-
;Bits 0,1 are used for drive selection
;Bits 2 & 3 are currently unused
;Bit 4=0 For SS disk hardware, 1= DS disk
;Bit 5=0 8" disk, 1= 5" disk
;Bit 6=0 Single density, 1= Double density
;Bit 7=0 if CURRENT selected side is A
; 1 if CURRENT selected side is B
;Note: handle this byte with care it is
;central to many functions within the
;program.

HEADR EQU 3 ;For Formatting
GAP1 EQU 4 ; "
GAP2 EQU 5 ; "
GAP3 EQU 6 ; "
GAP4 EQU 7 ; "
GAP4R EQU 8 ; "
SIZE EQU 9 ;1=128 Byte sectors.....4=1024 Byte sectors
GAP_FILL_CHAR EQU 10 ;Byte used in disk formating
DATA_FILL_CHAR EQU 11 ; " " "
TRK_SIZE EQU 12 ;TWO bytes containing the track size of that disks format
SPECIAL_FLAG EQU 14 ;flag byte for cases where after formatting disk need to be initalized
;normally 0, CPM86_FLAG = 1

SKEW EQU 15 ;Low address of sector skew table
SKEW1 EQU 16 ;High address of sector skew table
FORMAT_NUM EQU 17 ;Each format will have a unique number.
SYS_TRKS EQU 18 ;How many tracks for system usually 2 for 8-inch disks
TITLE EQU 19 ;Text string describing the disk format
;
;
; EQUATES FOR [IY] REGISTAR FOR TRACK, SIDE, SECTOR locations
;
TRK EQU 0
SIDE EQU 1
SCTR EQU 2
;
;-----
ORG 100H
;
LD (SP_SAVE),SP
LD SP,STACK
JP START
;-----
;
; HARDWARE DEPENDENT STUFF
; The only other hardware links are through the Versafloppy II board.
; Remember to make sure it is set to the correct 1791/5 & base port mentioned above.
;
;
CONST: IN A,(0) ;console status for SD Systems 8024 Video board
AND A,02H ;anything there
RET Z ;return 0 if nothing
XOR A,A
DEC A ;return NZ, & 0FFH in A if something there
RET
;
CI: IN A,(0) ;console input
AND A,02H
JR Z,CI
IN A,(1) ;return with character in A
AND 7FH ;Strip parity (IF ANY).
RET
;

```

```

CO:    IN      A, (0)           ;console output (arrive with character in C)
      AND     A,04H           ;Note character is in C and A on return.
      JR      Z,CO
      LD      A,C
      OUT     (1),A
      RET

;-----
START: LD      A,SIMPLE
      LD      (CRTDISP),A     ;Start off with simple diagnostic display
OVER:  LD      HL,SIGNON
      CALL    PMSG
      LD      IY,@TRK        ;Always points here
      XOR     A,A            ;set everything to zero
      LD      (IY+TRK),A
      LD      (IY+SIDE),A
      INC     A
      LD      (IY+SCTR),A     ;Track 0, side A, sector 1

      CALL    SELECT_DR_1    ;Put current drive hardware selection in (DRIVE_1)
      JR      Z,GET_DR_TABLE ;Get the current disk format from user, point IX to table
      CP     A,OFFH
      JR      Z,OVER         ;Invalid drive, start over
      CALL    ZCRLF          ;Must be an abort ESC
      LD      SP,(SP_SAVE)
      JP     0H             ;Reboot CP/M if ESC

GET_DR_TABLE:
      CALL    SELECT_IX      ;Set IX to point to the current drive table
      JR      Z,LOOP
      CP     A,OFFH
      JR      Z,OVER         ;Invalid drive, start over
      CALL    ZCRLF          ;Must be an abort ESC
      LD      SP,(SP_SAVE)
      JP     0H             ;Reboot CP/M if ESC

;
LOOP:  LD      A,(CRTDISP)    ;Check if detailed display flag is on
      OR     A,A
      JR      NZ,MENU1
      LD      HL,MAIN_MENU0  ;Main Menu loop (Detailed Display OFF)
      JR      LOOP1
MENU1: LD      HL,MAIN_MENU1  ;Main Menu loop (Detailed Display ON)
LOOP1: CALL    PMSG
      CALL    START_DRIVE_1  ;Select for "the current drive" in hardware

      CALL    SHOW_HW_TITLE  ;Describe the currently selected drive
      CALL    SHOW_HW_BYTE   ;Print out HW_BYTE info

      LD      HL,FBUFFER     ;The DMA RAM address will be here for sector R/W's
      LD      (TADDR),HL

      CALL    HOME           ;Move head to track 0 with a RESTORE CMD
      CALL    NZ,SHOW_ERRORS ;If an error in restoring head to TRK 0 say so.

      BIT     0,A           ;If NZ, then timeout on Status port, bad hardware
      JR      NZ,TO_MENU    ;Try another drive or format disk

      CALL    SHOW_IX_TABLE  ;Show info about current disk parameters

      CALL    READ_ID        ;Get Actual track ID
      JR      Z,ID_FINE     ;Was there errors
      CALL    SHOW_ERRORS    ;If an error show bit flags
      JR      TO_MENU

ID_FINE:CALL ZCRLF
      CALL    SHOW_ID        ;Show a typical Track ID field on track 0

TO_MENU:
      LD      HL,MENU_OPTIONS
      CALL    PMSG

```

```

LD      A,(CRTDISP)      ;Two menu lists (Set using menu #D)
OR      A,A              ;A detailed display or a simple one.
JR      NZ,MENUA
LD      HL,MENU1_MSG
JR      MENUB
MENUA:  LD      HL,MENU2_MSG
MENUB:  CALL    PMSG

;-----MAIN MENU-----
CALL    GETCMD          ;Find out Menu Option in A
LD      (CMD_STORE),A
CP      A,ESC          ;Abort if ESC character
JR      Z,FIN
CP      A,'0'          ;Get a new current disk
JP      Z,OVER
CP      A,'1'
CALL    Z,SEEK_TEST     ;Seek test
CP      A,'2'
CALL    Z,SEC_READ_TEST ;Sequential Read sectors test.
CP      A,'3'
CALL    Z,SEC_WRITE_TEST;Sequential Write sectors test.
CP      A,'4'
CALL    Z,SEC_RW_TEST   ;Sequential sec R/W test
CP      A,'5'
CALL    Z,RAND_SEC_TEST ;Random Track/Sector read/write test
CP      A,'6'
CALL    Z,TRACK_TEST    ;Read continously sectors from a specific track
CP      A,'7'
CALL    Z,TRACK_DUMP    ;Load one whole track to RAM (at location 5000H)
CP      A,'8'
CALL    Z,LOAD_SECTORS ;Load sectors into RAM
CP      A,'9'
CALL    Z,DUMP_SECTORS ;Write sectors to disk
CP      A,'F'
CALL    Z,FORMAT_DISK   ;Go to disk formatting section
CP      A,'C'
CALL    Z,COPY_DISK     ;Copy disk A: to B:
CP      A,'V'
CALL    Z,VERIFY_DISK   ;Verify disk A:=B:
CP      A,'D'
CALL    Z,TOGGLE_CRT    ;Switch on/off detail info display option
CP      A,'S'
CALL    Z,COPY_CPM3_SYS ;Copy CPM system tracks to another disk.
CP      A,'I'
CALL    Z,IBIFORM_DISK  ;Quick format of blank 8" disk in B:
JP      LOOP
FIN:    CALL    ZCRLF
LD      SP,(SP_SAVE)
JP      Z,0H           ;Return to CPM

;
;
;----- TOGGLE ON/OFF DETAILED INFORMATION DISPLAY ON CRT/LCD FOR SOME COMMANDS
TOGGLE_CRT:
LD      A,(CRTDISP)
CPL
LD      (CRTDISP),A
LD      A,(CMD_STORE) ;So we dont pick up other menu items
RET     ;Back to main menu

;
;
;----- SEEK DIAGNOSTIC TEST -----
;
;Simply tests head movement control
SEEK_TEST:
LD      HL,SEKMSG
CALL    PMSG
XOR     A,A             ;Setup for the BIOS below
LD      (IY+TRK),A     ;Start with track 0
LD      (IY+SIDE),A    ;Start on Side A

```

```

INC      A
LD       (IY+SCTR),A      ;Start at sector 1
CALL    START_DRIVE_1    ;Select for "the current drive" in hardware, Side A
CALL    HOME              ;Restore head to track 0
CALL    NZ,SHOW_ERRORS   ;print out errors if any

SEKT:   CALL    ZCRLF
CALL    SEEK_TRACK_V     ;Test SEEK ability/timing of chip
;
CALL    STEP_IN_CMD      ;Can also test Step in head one track & verify
CALL    NZ,SHOW_ERRORS   ;If Error show error flags

CALL    SHOW_T_LOC       ;Print out current track #
CALL    READ_ID          ;Get current track ID
CALL    NZ,SHOW_ERRORS   ;print out the track ID errors
CALL    SHOW_ID          ;Show track ID

CALL    CHECKABORT       ;SP will halt, ESC will abort
JR      NZ,END_CMD       ;Return back to main menu

INC     (IY+TRK)         ;Need to bump track up one
LD      A,(IY+TRK)       ;Store here
CP      A,(IX+NTRKS)     ;Are we at the end yet
JR      NZ,SEKT
LD      (IY+TRK),0
CALL    ZCRLF            ;Extra CR/LF fro another loop
JR      Z,SEKT           ;start again

;
END_CMD:
XOR     A,A
LD      (IY+TRK),A       ;Always back to Track 0, Side A
LD      (IY+SIDE),A
CALL    HOME             ;Move head back to Track 0
LD      A,(CMD_STORE)   ;So we dont pick up other menu items IF we return from here
RET

; ----- READ SECTORS TEST -----
; This routine will sequentially read all sectors on a disk. Both sides
; if a 2 sided disk.
;
SEC_READ_TEST:
LD      HL,RDTST_MSG     ;Say read test
CALL    PMSG
CALL    ZCRLF
XOR     A,A              ;Setup for the BIOS below
LD      (IY+TRK),A       ;Start with track 0
LD      (IY+SIDE),A      ;Start on Side A
INC     A
LD      (IY+SCTR),A      ;Start at sector 1

CALL    START_DRIVE_1    ;Select for "the current drive" in hardware, Side A
CALL    HOME
CALL    NZ,SHOW_ERRORS   ;print out errors if any

READ_LOOP1:
CALL    ZCRLF
CALL    SEEK_TRACK_V     ;Seek with verify command to 1791/5
CALL    NZ,SHOW_ERRORS   ;If any show error flags

LD      HL,FBUFFER       ;Place sector data here
LD      (TADDR),HL      ;Setup DMA address for BIOS

CALL    READ_SECTOR      ;<<<<<<<< READ 1 SECTOR >>>>>>
CALL    NZ,SHOW_ERRORS   ;If any, show error flags

CALL    SHOW_TSS_LOC     ;Announce current Track, Sec, (head if 2 sided)
CALL    SEC_DISPLAY      ;See if detail list the sector contents is req

INC     (IY+SCTR)        ;Get next sector
LD      A,(IY+SCTR)      ;Store new sec # in A

```

```

DEC      A                      ;Because sectors are numbered 1,2,3...
CP      A, (IX+NSCTRS)         ;Have we done a complete track yet
JR      NZ, SECTRK_DONE1

CALL    SWAP_SIDES             ;Sides swap check
LD      A, 1                   ;Back to sector 1 no matter what
LD      (IY+SCTR), A           ;Store for loop test
JR      NZ, SECTRK_DONE1       ;If B side (NZ), same track, back to sec 1

INC     (IY+TRK)               ;Need to bump track up one
LD      A, (IY+TRK)            ;Store here
CP      A, (IX+NTRKS)          ;Are we at the end yet
JP      Z, END_CMD             ;Yes we are done yet
;
SECTRK_DONE1:
CALL    CHECKABORT             ;SP will halt, ESC will abort
JR      Z, READ_LOOP1          ;Loop until abort or all tracks done
LD      A, (CRTDISP)           ;If detailed CRT display is on then one ESC
OR      A, A                   ;will switch it off
JP      Z, END_CMD             ;If Z do not list the sector contents.
XOR     A, A                   ;Just turn it off. Next time a real abort.
LD      (CRTDISP), A
JR      READ_LOOP1
;
;
; ----- WRITE SECTORS TEST -----
; This routine will sequentially write to all sectors on a disk. Both sides
; if a 2 sided disk. It will write the character TEST_FILL in each sector
;
SEC_WRITE_TEST:
LD      HL, WRTST_MSG           ;Say write test
CALL    PMSG
CALL    ZCRLF                   ;CR, LF at start.
XOR     A, A                   ;Setup for the BIOS below
LD      (IY+TRK), A            ;Start with track 0
LD      (IY+SIDE), A           ;Start on Side A
INC     A
LD      (IY+SCTR), A           ;Start at sector 1
CALL    START_DRIVE_1          ;Select for "the current drive" in hardware, Side A
CALL    HOME
CALL    NZ, SHOW_ERRORS         ;print out errors if any

LD      HL, FBUFFER            ;Sector data to be written is here
LD      (TADDR), HL           ;Setup DMA address for BIOS

LD      C, TEST_FILL           ;Data fill character (01H)
CALL    FILL_BUFFER            ;Fill buffer with character to be written

CALL    CHECK_WP               ;See if disk is write protected
JP      NZ, END_CMD            ;Abort
;
WRITE_LOOP1:
CALL    ZCRLF
CALL    SEEK_TRACK_V           ;Seek with verify command to 1791/5
CALL    NZ, SHOW_ERRORS         ;If any show error flags

LD      HL, FBUFFER            ;Place sector data here
LD      (TADDR), HL           ;Setup DMA address for BIOS

CALL    WRITE_SECTOR           ;<<<<<<< WRITE 1 SECTOR >>>>>>
CALL    NZ, SHOW_ERRORS         ;If any, show error flags

CALL    SHOW_TSS_LOC           ;Announce current Track, Sec, (head if 2 sided)
CALL    SEC_DISPLAY            ;See if detail list the sector contents is req

INC     (IY+SCTR)              ;Get next sector
LD      A, (IY+SCTR)           ;Store new sec # in A
DEC     A                      ;Because sectors are numbered 1,2,3...
CP      A, (IX+NSCTRS)         ;Have we done a complete track yet

```



```

JR      NZ,SECTRK_DONE2

CALL    SWAP_SIDES      ;Sides swap check
LD      A,1             ;Back to sector 1 no matter what
LD      (IY+SCTR),A     ;Store for loop test
JR      NZ,SECTRK_DONE2 ;If B side (NZ), same track, back to sec 1

INC     (IY+TRK)        ;Need to bump track up one
LD      A,(IY+TRK)      ;Next track
CP      A,(IX+NTRKS)    ;Are we at the end yet
JP      Z,END_CMD       ;Yes we are done yet

;
SECTRK_DONE2:
CALL    CHECKABORT      ;SP will halt, ESC will abort
JR      Z,WRITE_LOOP1   ;Loop until abort or all tracks done
LD      A,(CRTDISP)     ;If detailed CRT display is on then one ESC
OR      A,A             ;will switch it off
JP      Z,END_CMD       ;If Z do not list the sector contents.
XOR     A,A             ;Just turn it off. Next time a real abort.
LD      (CRTDISP),A
JR      WRITE_LOOP1

;
; ----- SEQUENTIAL READ/WRITE SECTORS TEST -----
; This routine will read and write back to randomly selected tracks and
; sectors on a disk. Both sides, if a 2 sided disk.
;
SEC_RW_TEST:
LD      HL,RDWRSTMSG    ;Say R/W test
CALL    PMSG
CALL    ZCRLF           ;CR,LF at start.
XOR     A,A            ;Setup for the BIOS below
LD      (IY+TRK),A     ;Start with track 0
LD      (IY+SIDE),A    ;Start on Side A
INC     A
LD      (IY+SCTR),A    ;Start at sector 1
CALL    START_DRIVE_1  ;Select for "the current drive" in hardware, Side A
CALL    HOME
CALL    NZ,SHOW_ERRORS ;print out errors if any

CALL    CHECK_WP       ;See if disk is write protected
JP      NZ,END_CMD     ;Abort

;
RW_LOOP1:
CALL    ZCRLF
CALL    SEEK_TRACK_V   ;Seek with verify command to 1791/5
CALL    NZ,SHOW_ERRORS ;If any show error flags

LD      HL,FBUFFER     ;Place sector data here
LD      (TADDR),HL    ;Setup DMA address for BIOS

CALL    READ_SECTOR    ;<<<<<<< READ 1 SECTOR >>>>>>
CALL    NZ,SHOW_ERRORS ;If any, show error flags

CALL    SHOW_TSS_LOC   ;Announce current Track, Sec, (head if 2 sided)
CALL    SEC_DISPLAY    ;See if detail list the sector contents is req

CALL    WRITE_SECTOR   ;<<<<<<< WRITE 1 SECTOR >>>>>>
CALL    NZ,SHOW_ERRORS ;If any, show error flags

LD      HL,FBUFFER2    ;Place new sector data here
LD      (TADDR),HL    ;Setup DMA address for BIOS

CALL    READ_SECTOR    ;<<<<<<< READ 1 SECTOR >>>>>>
CALL    NZ,SHOW_ERRORS ;If any, show error flags

CALL    CMP_BUFFERS    ;Check for errors
JR      Z,COMPARE_OK
LD      HL,SEC_V_ERROR ;R/W Error found

```

```

CALL    PMSG
CALL    SHOW_TSS_LOC          ;trk,sec,head

COMPARE_OK:
INC     (IY+SCTR)            ;Get next sector
LD      A, (IY+SCTR)
DEC     A                    ;Because sectors are numbered 1,2,3...
CP      A, (IX+NSCTRS)       ;Have we done a complete track yet
JR      NZ, SECTRK_DONE3

CALL    SWAP_SIDES           ;Sides swap check
LD      A, 1                 ;Back to sector 1 no matter what
LD      (IY+SCTR), A         ;Store for loop test
JR      NZ, SECTRK_DONE3     ;If B side (NZ), same track, back to sec 1

INC     (IY+TRK)             ;Need to bump track up one
LD      A, (IY+TRK)         ;Next track
CP      A, (IX+NTRKS)       ;Are we at the end yet
JP      Z, END_CMD          ;Yes we are done yet

;
SECTRK_DONE3:
CALL    CHECKABORT          ;SP will halt, ESC will abort
JR      Z, RW_LOOP1         ;Loop until abort or all tracks done
LD      A, (CRTDISP)        ;If detailed CRT display is on then one ESC
OR      A, A                 ;Will switch it off
JP      Z, END_CMD          ;If Z do not list the sector contents.
XOR     A, A                 ;Just turn it off. Next time a real abort.
LD      (CRTDISP), A
JR      RW_LOOP1           ;try again

;
; ----- RANDOM TRK/SEC READ/WRITE SECTORS TEST -----
; This routine will read and write back to randomly selected tracks and
; sectors on a disk. Both sides, if a 2 sided disk.
;
RAND_SEC_TEST:
LD      HL, RAND_TST_MSG    ;Say R/W test
CALL    PMSG
CALL    ZCRLF               ;CR,LF at start.
XOR     A, A                 ;Setup for the BIOS below
LD      (IY+TRK), A         ;Start with track 0
LD      (IY+SIDE), A       ;Start on Side A
INC     A
LD      (IY+SCTR), A       ;Start at sector 1
CALL    START_DRIVE_1      ;Select for "the current drive" in hardware, Side A
CALL    HOME
CALL    NZ, SHOW_ERRORS     ;print out errors if any

CALL    CHECK_WP           ;See if disk is write protected
JP      NZ, END_CMD        ;Abort

;
RAND_LOOP1:
CALL    ZCRLF
CALL    SEEK_TRACK_V       ;Seek with verify command to 1791/5
CALL    NZ, SHOW_ERRORS     ;If any show error flags

LD      HL, FBUFFER        ;Place sector data here
LD      (TADDR), HL       ;Setup DMA address for BIOS

CALL    READ_SECTOR        ;<<<<<<<< READ 1 SECTOR >>>>>>>
CALL    NZ, SHOW_ERRORS     ;If any, show error flags

CALL    SHOW_TSS_LOC       ;Announce current Track, Sec, (head if 2 sided)
CALL    SEC_DISPLAY        ;See if detail list the sector contents is req

CALL    WRITE_SECTOR       ;<<<<<<<< WRITE 1 SECTOR >>>>>>>
CALL    NZ, SHOW_ERRORS     ;If any, show error flags

LD      HL, FBUFFER2       ;Place new sector data here

```

```

LD      (TADDR),HL      ;Setup DMA address for BIOS

CALL    READ_SECTOR    ;<<<<<<<< READ 1 SECTOR >>>>>>
CALL    NZ,SHOW_ERRORS ;If any, show error flags

CALL    CMP_BUFFERS    ;Check for errors
JR      Z,RAND_OK
LD      HL,SEC_V_ERROR ;R/W Error found
CALL    PMSG
CALL    SHOW_TSS_LOC    ;At trk,sec,head

RAND_OK:
CALL    RANDOM          ;Get a random sector#
LD      B,(IX+NSCTRS)
AND     A,B
OR      A,A             ;strip off extra bits
                        ;No sector 0
JR      NZ,RAND_SEC_OK
INC     A

RAND_SEC_OK:
LD      (IY+SCTR),A    ;Store for SEC display and sector read routine

CALL    RANDOM          ;Get a random track #
LD      B,(IX+NTRKS)
DEC     B               ;Tracks numbered 0,1,2...NTRKS-1
AND     A,B             ;strip off extra bits
LD      (IY+TRK),A    ;Store for SEC display and sector read routine

BIT     4,(IX+HW_BYTE) ;Is it a 1 or 2 sided disk
JR      Z,SECTRK_DONE4 ;If 1 sided then skip sides swap

CALL    RANDOM          ;Get a random SIDE
AND     02H            ;Isolate the SIDE bit
LD      (IY+SIDE),A    ;Will be either 0 or 02H
CALL    SET_SIDE

SECTRK_DONE4:
CALL    CHECKABORT     ;SP will halt, ESC will abort
JR      Z,RAND_LOOP1  ;Loop until abort
LD      A,(CRTDISP)    ;If detailed CRT display is on then one ESC
OR      A,A            ;will switch it off
JP      Z,END_CMD      ;If Z do not list the sector contents.
XOR     A,A            ;Just turn it off. Next time a real abort.
LD      (CRTDISP),A
JR      RAND_LOOP1    ;try again
;
; Simple random number generator
RANDOM: LD      A,R      ;Seed will be differentd each time
LD      B,A
ADD     A,A
ADD     A,A
ADD     A,B
RR      A
ADD     A,7
RET

;
;
; ----- CONTINUOUSLY READ SECTORS FROM ONE TRACK TEST -----
; This routine will read sectors continuously from one track. Both sides, if a
; a 2 sided disk. Can be used with a scope to do a 'CATS EYES' hardware test.
;
TRACK_TEST:
LD      HL,RDTST_MSG   ;Say read test
CALL    PMSG
XOR     A,A            ;Setup for the BIOS below
LD      (IY+TRK),A     ;Start with track 0
LD      (IY+SIDE),A    ;Start on Side A
INC     A
LD      (IY+SCTR),A    ;Start at sector 1
CALL    START_DRIVE_1  ;Select for "the current drive" in hardware, Side A

```

```

CALL HOME
CALL NZ,SHOW_ERRORS ;print out errors if any

CALL ZCRLF
LD HL,GET_TRACK_MSG ;get the required track
CALL PMSG
CALL GET_HEX ;get 2 digits
JP C,END_CMD ;Abort if C returned

LD B,(IX+NTRKS) ;Are we within disk range
CP A,B
JR C,GET_TSIDES ;Yes we are
LD HL,TRACK_ERROR
CALL PMSG
JR TRACK_TEST ;Try again

GET_TSIDES:
LD (IY+TRK),A ;Store requested track

BIT 4,(IX+HW_BYTE) ;Is it a 1 or 2 sided disk
JR Z,TRACK_TEST2 ;If 1 sided then skip side question

LD HL,GET_SIDE_MSG ;get required track
CALL PMSG
CALL GETCMD ;get the input option
CP A,ESC ;Abort if ESC character
JP Z,END_CMD
CP A,'B'
JR NZ,MUST_BE_TA
LD A,02H ;Set to side B Flag (Started off with 0=A)
LD (IY+SIDE),A
JR TRACK_TEST2

MUST_BE_TA:
CP A,'A'
JR Z,TRACK_TEST2 ;Already on A
LD HL,SIDE_ERROR ;Not 'A' or 'B' must be error
CALL PMSG
JR TRACK_TEST ;Try again

TRACK_TEST2:
CALL ZCRLF
CALL SET_SIDE ;Update the side hardware (if required)
CALL SEEK_TRACK_V ;seek to the new track position
CALL NZ,SHOW_ERRORS ;If any show error flags
;
READ_LOOP2:
CALL ZCRLF
LD HL,FBUFFER ;Place sector data here
LD (TADDR),HL ;Setup DMA address for BIOS

CALL READ_SECTOR ;<<<<<<< READ 1 SECTOR >>>>>>
CALL NZ,SHOW_ERRORS ;If any, show error flags

CALL SHOW_TSS_LOC ;Announce current Track, Sec, (head if 2 sided)
CALL SEC_DISPLAY ;See if detail list the sector contents is req

INC (IY+SCTR) ;Get next sector
LD A,(IY+SCTR)
DEC A ;Because sectors are numbered 1,2,3...
CP A,(IX+NSCTRS) ;Have we done a complete track yet
JR NZ,SEC_DONE4
LD A,1 ;Cycle back to sector 1
LD (IY+SCTR),A

SEC_DONE4:
CALL CHECKABORT ;SP will halt, ESC will abort
JR Z,READ_LOOP2 ;Loop until abort or all tracks done
LD A,(CRTDISP) ;If detailed CRT display is on then one ESC
OR A,A ;will switch it off

```

```

JP      Z,END_CMD           ;If Z do not list the sector contents.
XOR     A,A                 ;Just turn it off. Next time a real abort.
LD      (CRTDISP),A
JR      READ_LOOP2
;
;
;
;-----
;Read one whole track to RAM (5000H). This is used to see what the Format routine actually
;placed on the disk and/or to see what track formats other computers used.
;
;
TRACK_DUMP:
LD      HL,TRK_DUMP_MSG ;Say track dump msg
CALL    PMSG
XOR     A,A                 ;Setup for the BIOS below
LD      (IY+TRK),A         ;Start with track 0
LD      (IY+SIDE),A       ;Start on Side A
INC     A
LD      (IY+SCTR),A       ;Start at sector 1
CALL    START_DRIVE_1     ;Select for "the current drive" in hardware, Side A
CALL    HOME
CALL    NZ,SHOW_ERRORS ;print out errors if any

CALL    ZCRLF
LD      HL,GET_TRACK_MSG ;get required track
CALL    PMSG
CALL    GET_HEX           ;get 2 digits in [A]
JP      C,END_CMD
CALL    ZCRLF

LD      B,(IX+NTRKS)      ;Are we within disk range
CP      A,B
JR      C,GET_SIDES      ;Yes we are
LD      HL,TRACK_ERROR
CALL    PMSG
JR      TRACK_DUMP      ;Try again
GET_SIDES:
LD      (IY+TRK),A       ;Store requested track

BIT     4,(IX+HW_BYTE)    ;Is it a 1 or 2 sided disk
JR      Z,GET_TRK        ;If 1 sided then skip side question

LD      HL,GET_SIDE_MSG ;get required track
CALL    PMSG
CALL    GETCMD           ;get the input option
CP      A,ESC            ;Abort if ESC character
JP      Z,END_CMD
CP      A,'B'
JR      NZ,MUST_BE_A
LD      A,02H            ;Set to side B Flag (Started off with 0=A)
LD      (IY+SIDE),A
JR      GET_TRK
MUST_BE_A:
CP      A,'A'
JR      Z,GET_TRK        ;Already on A
LD      HL,SIDE_ERROR
CALL    PMSG
JR      TRACK_DUMP      ;Try again

GET_TRK:
CALL    ZCRLF
CALL    SET_SIDE         ;Update the side hardware (if required)

CALL    SEEK_TRACK_V     ;seek to the new track position          (verify)
CALL    NZ,SHOW_ERRORS ;If any show error flags
;
LD      HL,FBUFFER       ;Will build the complete sector image here (5000H)
LD      (TADDR),HL      ;Store the pointer here.

```

```

LD      D, (IX+TRK_SIZE+1) ;Need to find out how large the track is
LD      E, (IX+TRK_SIZE)   ;Number of bytes per track for this disk into [DE]
LD      A, D               ;check that it is not 0 (ie. data not filled in table)
OR      A, A
JR      NZ, SIZE_OK
LD      HL, TRK_SIZE_ERR
CALL    PMSG
JP      END_CMD

SIZE_OK:
LD      (TRACK_SIZE), DE ;Store for below also

LD      B, 0
Z_LOOP: LD      (HL), B      ;Fill RAM with 0's before starting
INC     HL
DEC     DE
LD      A, E
OR      A, A
JR      NZ, Z_LOOP
OR      A, D
JR      NZ, Z_LOOP        ;Have we done DE bytes yet

LD      HL, FBUFFER       ;Start again
LD      (TADDR), HL
LD      DE, (TRACK_SIZE)  ;Number of bytes per track

CALL    READ_TRACK       ;<<<<<< Read one whole track
CALL    NZ, SHOW_ERRORS  ;If any, show error flags

CALL    SHOW_TS_LOC

LD      HL, TRACK_CONTENTS
CALL    PMSG
LD      L, PAGE_SIZE     ;20 lines per page
LD      DE, (TRACK_SIZE) ;to count down
LD      BC, 0            ;to count up
EXX
LD      HL, FBUFFER       ;will use alt reg HL' for data pointer
EXX

RAMDUMP: EXX
LD      A, (HL)
CALL    PACC
INC     HL
EXX
INC     BC
LD      A, C
AND     A, 00011111B     ;32 characters /line
JR      NZ, NOCRX
CALL    ZCRLF
DEC     L
JR      NZ, NOCRX
PUSH   HL
LD      HL, MORE_MSG
CALL    PMSG
CALL    GETCMD           ;Ask if we wish to continue
POP    HL
CP     A, 'Y'
JR      NZ, DONECR
CALL    ZCRLF
LD      L, PAGE_SIZE
NOCRX: DEC     DE
LD      A, E
OR      A, A
JR      NZ, RAMDUMP
OR      A, D
JR      NZ, RAMDUMP     ;Have we sent DE bytes yet
DONECR: EXX             ;We are done
CALL    FRCINT           ;Need to clear the 1791/5 because we may have
;                          ;overrun reading the track
JP      END_CMD         ;All done back to main menu

```

```

;
;
;
;***** LOAD SECTORS FROM DISK TO RAM ROUTINE *****
;Note sectors from only one side.

LOAD_SECTORS:
    LD    HL,LOAD_MSG    ;Load msg
    CALL  PMSG
    XOR   A,A            ;Setup for the BIOS below
    LD    (IY+TRK),A     ;Start with track 0
    LD    (IY+SIDE),A    ;Start on Side A
    INC   A
    LD    (IY+SCTR),A    ;Start at sector 1
    CALL  START_DRIVE_1  ;Select for "the current drive" in hardware, Side A
    CALL  HOME
    CALL  NZ,SHOW_ERRORS ;print out errors if any

    CALL  ZCRLF
    LD    HL,GET_TRACK_MSG ;get required track
    CALL  PMSG
    CALL  GET_HEX        ;get 2 digits
    JP    C,END_CMD

    LD    B,(IX+NTRKS)   ;Are we within disk range
    CP    A,B
    JR    C,LGET_SIDES  ;Yes we are
    LD    HL,TRACK_ERROR
    CALL  PMSG
    JP    LOAD_SECTORS  ;Try again

LGET_SIDES:
    LD    (IY+TRK),A     ;Store requested track for below

    BIT   4,(IX+HW_BYTE) ;Is it a 1 or 2 sided disk
    JR    Z,GET_START_SEC ;If 1 sided then skip side question

    CALL  ZCRLF
    LD    HL,GET_SIDE_MSG ;get required track
    CALL  PMSG
    CALL  GETCMD        ;get the input option
    CP    A,ESC         ;Abort if ESC character
    JP    Z,END_CMD
    CP    A,'B'
    JR    NZ,LMUST_BE_A
    LD    A,02H         ;Set to side B Flag (Started off with 0=A)
    LD    (IY+SIDE),A
    JR    GET_START_SEC

LMUST_BE_A:
    CP    A,'A'
    JR    Z,GET_START_SEC ;Already on A
    LD    HL,SIDE_ERROR
    CALL  PMSG
    JP    LOAD_SECTORS  ;Try again

GET_START_SEC:
    LD    HL,GET_SEC_MSG ;Get starting sector
    CALL  PMSG
    CALL  GET_HEX        ;get 2 digits
    JP    C,END_CMD
    CALL  ZCRLF
    LD    B,(IX+NSCTRS) ;Are we within disk range
    CP    A,B
    JR    C,GSTART1     ;Yes we are
GSTART0:LD    HL,SEC_ERROR
    CALL  PMSG
    JP    LOAD_SECTORS  ;Try again
GSTART1:OR    A,A       ;Sectors numbered 1,2,3...
    JR    Z,GSTART0

```

```

LD      (IY+SCTR),A      ;Store start sector

LD      HL,SEC_COUNT_MSG ;Get no. of sectors to load
CALL    PMSG
CALL    GET_HEX          ;get 2 digits
JP      NC,GSTART2

GSTART3:
LD      HL,SEC_COUNT_ERR
CALL    PMSG
JP      END_CMD

GSTART2:OR  A,A          ;cannot have 0 sectors
JR      Z,GSTART3

LD      (SEC_COUNT),A   ;store count of sectors

LD      HL,GET_DMA_MSG  ;Get DMA Address
CALL    PMSG
CALL    GET_HEX4        ;get 4 digits
JP      NC,GSTART4

LD      HL,RAM_ERROR
CALL    PMSG
JP      END_CMD

GSTART4:
LD      (TADDR),HL     ;Store the DMA address
CALL    SET_SIDE       ;Update the side hardware (if required)
CALL    ZCRLF

MORE_LOAD:
CALL    SEEK_TRACK_V   ;seek to the new track position      (verify)
CALL    NZ,SHOW_ERRORS ;If any show error flags

;

LD      HL,LOADING_MSG
CALL    PMSG
CALL    SHOW_TSS_LOC
LD      HL,LOADING1_MSG
CALL    PMSG
LD      HL,(TADDR)
LD      A,H
CALL    PACC
LD      A,L
CALL    PACC
LD      HL,H_MSG       ;H. at end of message
CALL    PMSG

CALL    READ_SECTOR    ;<<<<<<<< READ 1 SECTOR >>>>>>
CALL    NZ,SHOW_ERRORS ;If any, show error flags

;

LD      HL,(DMA_NEXT)  ;Update the next RAM location
LD      (TADDR),HL

LD      A,(SEC_COUNT)
DEC     A
JR      Z,DONE_LOAD_OK
LD      (SEC_COUNT),A ;Store for next time

INC     (IY+SCTR)
LD      A,(IY+SCTR)
DEC     A              ;Because sectors are numbered 1,2,3...
CP      A,(IX+NSCTRS) ;Have we done a complete track yet
JR      NZ,LSECTRK_OK

LD      A,1            ;Back to sector 1 no matter what
LD      (IY+SCTR),A   ;Store for loop test

INC     (IY+TRK)       ;Need to bump track up one
LD      A,(IY+TRK)
CP      A,(IX+NTRKS)
JR      Z,ERROR_LOAD  ;Ran out of tracks

```



```

;
LSECTRK_OK:
    CALL    CHECKABORT        ;SP will halt, ESC will abort
    JR      NZ,DONE_LR
    JP      MORE_LOAD        ;Loop until abort or all sectors are done
DONE_LR:LD  A,(CRTDISP)        ;If detailed CRT display is on then one ESC
    OR      A,A                ;will switch it off
    JP      Z,END_CMD        ;If Z do not list the sector contents.
    XOR     A,A                ;Just turn it off. Next time a real abort.
    LD      (CRTDISP),A
    JP      MORE_LOAD
;
DONE_LOAD_OK:
    LD      HL,LOAD_DONE_MSG
    CALL    PMSG
    JP      END_CMD
;
ERROR_LOAD:
    LD      HL,ERR_TK_MSG
    CALL    PMSG
    JP      END_CMD
;
;
;
;***** DUMP SECTORS FROM RAM TO DISK ROUTINE *****
;
DUMP_SECTORS:
    LD      HL,DUMP_MSG        ;Load msg
    CALL    PMSG
    XOR     A,A                ;Setup for the BIOS below
    LD      (IY+TRK),A        ;Start with track 0
    LD      (IY+SIDE),A       ;Start on Side A
    INC     A
    LD      (IY+SCTR),A        ;Start at sector 1
    CALL    START_DRIVE_1     ;Select for "the current drive" in hardware, Side A
    CALL    HOME
    CALL    NZ,SHOW_ERRORS    ;print out errors if any

    CALL    CHECK_WP          ;See if disk is write protected
    JP      NZ,END_CMD

WR_T_OK:
    CALL    ZCRLF
    LD      HL,GET_TRACK_MSG  ;get required track
    CALL    PMSG
    CALL    GET_HEX            ;get 2 digits
    JP      C,END_CMD
    CALL    ZCRLF
    LD      B,(IX+NTRKS)      ;Are we within disk range
    CP      A,B
    JR      C,DGET_SIDES     ;Yes we are
    LD      HL,TRACK_ERROR
    CALL    PMSG
    JP      DUMP_SECTORS     ;Try again

DGET_SIDES:
    LD      (IY+TRK),A        ;Store requested track for below

    BIT    4,(IX+HW_BYTE)     ;Is it a 1 or 2 sided disk
    JR      Z,GET_START_DSEC   ;If 1 sided then skip side question
    CALL    ZCRLF

    LD      HL,GET_SIDE_MSG   ;get required track
    CALL    PMSG
    CALL    GETCMD            ;get the input option
    CP      A,ESC             ;Abort if ESC character
    JP      Z,END_CMD
    CP      A,'B'
    JR      NZ,DMUST_BE_A
    LD      A,02H             ;Set to side B Flag (Started off with 0=A)

```

```

        LD      (IY+SIDE),A
        JR      GET_START_DSEC
DMUST_BE_A:
        CP      A,'A'
        JR      Z,GET_START_DSEC      ;Already on A
        LD      HL,SIDE_ERROR
        CALL    PMSG
        JP      DUMP_SECTORS          ;Try again

GET_START_DSEC:
        LD      HL,GET_SEC_MSG ;Get starting sector
        CALL    PMSG
        CALL    GET_HEX      ;get 2 digits
        JP      C,END_CMD
        CALL    ZCRLF
        LD      B,(IX+NSCTRS) ;Are we within disk range
        CP      A,B
        JR      C,DSTART1      ;Yes we are
DSTART0:LD      HL,SEC_ERROR
        CALL    PMSG
        JP      DUMP_SECTORS    ;Try again
DSTART1:OR      A,A           ;Sectors numbered 1,2,3...
        JR      Z,DSTART0

        LD      (IY+SCTR),A    ;Store start sector

        LD      HL,SEC_COUNT_MSG ;Get no. of sectors to write
        CALL    PMSG
        CALL    GET_HEX      ;get 2 digits
        JP      NC,DSTART2

DSTART3:LD      HL,SEC_COUNT_ERR
        CALL    PMSG
        JP      END_CMD
DSTART2:OR      A,A           ;cannot have 0 sectors
        JR      Z,DSTART3

        LD      (SEC_COUNT),A ;store count of sectors

        LD      HL,GET_DMAD_MSG ;Get DMA Address
        CALL    PMSG
        CALL    GET_HEX4     ;get 4 digits
        JP      NC,DSTART4
        LD      HL,RAM_ERROR
        CALL    PMSG
        JP      END_CMD
DSTART4:LD      (TADDR),HL    ;Store the DMA address
        CALL    SET_SIDE     ;Update the side hardware (if required)
MORE_DUMP:
        CALL    SEEK_TRACK_V ;seek to the new track position      (verify)
        CALL    NZ,SHOW_ERRORS ;If any show error flags
;
        LD      HL,DUMPING_MSG
        CALL    PMSG
        CALL    SHOW_TSS_LOC
        LD      HL,DUMPING1_MSG
        CALL    PMSG
        LD      HL,(TADDR)
        LD      A,H
        CALL    PACC
        LD      A,L
        CALL    PACC
        LD      HL,H_MSG      ;H. at end of message
        CALL    PMSG

        CALL    WRITE_SECTOR  ;<<<<<<< WRITE 1 SECTOR >>>>>>
        CALL    NZ,SHOW_ERRORS ;If any, show error flags
;

```

```

LD     HL, (DMA_NEXT)           ;Update the next RAM location
LD     (TADDR), HL

LD     A, (SEC_COUNT)
DEC    A
JR     Z, DONE_DUMP_OK
LD     (SEC_COUNT), A           ;Store for next time

INC    (IY+SCTR)                ;Get next sector
LD     A, (IY+SCTR)
DEC    A                        ;Because sectors are numbered 1,2,3...
CP     A, (IX+NSCTRS)           ;Have we done a complete track yet
JR     NZ, DSECTRK_OK

LD     A, 1                      ;Back to sector 1 no matter what
LD     (IY+SCTR), A            ;Store for loop test

INC    (IY+TRK)                 ;Need to bump track up one
LD     A, (IY+TRK)
CP     A, (IX+NTRKS)           ;Are we at the end yet
JR     Z, ERROR_DUMP           ;Ran out of tracks
;
DSECTRK_OK:
CALL   CHECKABORT              ;SP will halt, ESC will abort
JR     NZ, DONE_LX
JP     MORE_DUMP               ;Loop until abort or all sectors are done
DONE_LX: LD A, (CRTDISP)         ;If detailed CRT display is on then one ESC
OR     A, A                     ;will switch it off
JP     Z, END_CMD              ;If Z do not list the sector contents.
XOR    A, A                     ;Just turn it off. Next time a real abort.
LD     (CRTDISP), A
JP     MORE_DUMP
;
DONE_DUMP_OK:
LD     HL, DUMP_DONE_MSG
CALL   PMSG
JP     END_CMD
;
ERROR_DUMP:
LD     HL, ERR_TK_MSG
CALL   PMSG
JP     END_CMD
;
;
;***** CURRENT DISK, FORMAT ROUTINE *****
;
FORMAT_DISK:
LD     (IX_OLD_STORE), IX       ;Save current IX in case of an abort

BIT    5, (IX+HW_BYTE)          ;See if current selected disk is 5" or 8"
JR     NZ, FORM_TABLE5
CALL   GET_TABLE_8              ;Setup IX to (new) table parameters for 8"
JR     Z, TABLE_OK
LD     HL, BADCMD
CALL   PMSG
JR     FABORT
FORM_TABLE5:
CALL   GET_TABLE_5              ;Setup IX to (new) table parameters for 5"
JR     Z, TABLE_OK
LD     HL, BADCMD
CALL   PMSG
JR     FABORT
TABLE_OK:
LD     HL, FORMATTING_MSG
CALL   PMSG
PUSH   IX                      ;IX->HL
POP    HL
LD     DE, TITLE                ;add in offset
ADD    HL, DE                   ;HL now points to the title entry of the selected disk

```

```

CALL    PMSG

CALL    START_DRIVE_1    ;Select for "the current drive" in hardware, Side A
CALL    HOME              ;Restore disk head
CALL    NZ,SHOW_ERRORS   ;print out errors if any

CALL    CHECK_WP         ;See if disk is write protected
JP      Z,FORMWP_OK      ;Go to the core test routine
LD      IX,(IX_OLD_STORE) ;Nothing altered go back to main menu
FABORT: CALL START_DRIVE_1 ;RE-select to old drive disk format, Side A
LD      A,(CMD_STORE)    ;So we dont pick up other menu menu items on
RET                                           ;return back to MENU

FORMWP_OK:
LD      HL,FBUFFER       ;Will build the complete sector image here
LD      (TADDR),HL      ;Store the pointer here.

LD      A,1
LD      (IY+SCTR),A      ;Start at sector 1
XOR     A,A              ;Setup for the BIOS below
LD      (IY+SIDE),A     ;Start on Side A
CALL    SET_SIDE        ;Make sure IX+HW_BYTE is set for correct side
XOR     A,A
                                           ;Fall through with 0 in [A] for trk 0
NEXT_TRK:
LD      (IY+TRK),A      ;store track info for each loop

CALL    BUILD_TRACK      ;Build a complete track image in RAM

CALL    FORMAT_INFO     ;Display the track if detailed info is on

LD      A,(IY+SIDE)
CP      A,02H           ;If B side no seek necessary
JR      Z,SKIP_SEEK

LD      A,(IY+TRK)      ;Get new track#
CALL    SEEK_TRACK_NV   ;Seek with verify command to 1791/5
CALL    NZ,SHOW_ERRORS  ;If any show error flags

SKIP_SEEK:
LD      DE,FBUFFER      ;Move Start of track image into DE
PUSH   DE               ;save it
LD      HL,(E_GAP4_MARK) ;End+1 of track header
DEC    HL
SBC    HL,DE
EX     DE,HL           ;Count now in DE

LD      HL,TRACK_SIZE   ;Side step, need to store track byte count
LD      (HL),D          ;for display at end of format etc.
INC    HL
LD      (HL),E
POP    HL              ;DE on stack to --> HL

CALL    WRITE_TRACK     ;<<<< Write track with hardware >>>>
CALL    NZ,SHOW_ERRORS

CALL    CHECKABORT     ;SP will halt, ESC will abort
JR      NZ,CHECK_ABORT

AGAIN1:
LD      A,(IX+HW_BYTE)  ;Check if side B is req
BIT    4,A
JR      Z,AGAIN        ;SS drive go to next track

LD      A,(IY+SIDE)
OR     A,A              ;If 0 we have done A side, now do B
JR      Z,FORM_B_SIDE  ;switch over to B side
XOR     A,A
LD      (IY+SIDE),A
CALL    SET_SIDE        ;update the hardware

```

```

        JR      AGAIN                ;Need track increase now we are back on side A
FORM_B_SIDE:
        LD      A,02H                ;Flag for B side
        LD      (IY+SIDE),A
        CALL    SET_SIDE              ;Update the hardware
        LD      A,(IY+TRK)
        JR      NEXT_TRK             ;Do everything again with this track on B side

AGAIN:   INC     (IY+TRK)             ;What was the last track number just done
        LD      A,(IY+TRK)
        CP      A,(IX+NTRKS)         ;Check if we are on the last tarck
        JR      NZ,NEXT_TRK          ;If not then back to updating tracks and doing it again.
;
        LD      HL,END_FORM_MSG
        CALL    PMSG
        LD      HL,TRACK_SIZE
        LD      A,(HL)                ;Fill in total byte count of a track
        CALL    PACC                  ;Is useful for building tables and track displ
        INC     HL                    ;was obtained from WRITE_1_TRACK below
        LD      A,(HL)
        CALL    PACC
        LD      HL,END_FORM1_MSG     ;end of formatting
        CALL    PMSG

        LD      A,(IX+SPECIAL_FLAG) ;Is any post formatting mods required?
        OR      A,A
        JR      Z,NO_MODS
        CP      A,CPM86_FLAG
        CALL    Z,INIT_CPM86         ;Must modify first 5" disk sector for CPM86

NO_MODS:
        CALL    START_DRIVE_1        ;Select disk with new IX table format, Side A
        LD      A,(CMD_STORE)        ;So we dont pick up other menu menu items on
        RET                          ;return back to MENU
;
CHECK_ABORT:
        LD      A,(CRTDISP)          ;If detailed CRT display is on then one ESC
        OR      A,A                  ;will switch it off
        JP      Z,FORM_ABORT         ;If Z do not list the sector contents.
        XOR     A,A                  ;Just turn it off. Next time a real abort.
        LD      (CRTDISP),A
        JR      AGAIN1
;
FORM_ABORT:
        LD      HL,FORM_ERRMSG       ;Say error formatting disk
        CALL    PMSG
        LD      SP,STACK
        JP      START                ;Must abort everything because disk status is unknown
;
;
;
; QUICK 8" IBM SSSD FORMAT of a blank disk in B: drive
; This is for quickly making CPM 8" disks. Not really a disgnostic
; This assumes an 8" disk is in B: drive.
;
IBMFORM_DISK:
        XOR     A,A
        LD      (ERRORS_FLAG),A      ;Will keep tab on errors during this routine

        LD      (IX_OLD_STORE),IX    ;Store current disk paramater table
        LD      A,(DRIVE_1)          ;Because we may not already be on B:
        LD      (DRIVE_STORE),A

        LD      HL,IBM_FORMAT
        CALL    PMSG

        LD      IX,STDSDT            ;Force current drive to 8" IBM SSSD
        LD      A,2
        LD      (DRIVE_1),A         ;Force B: drive hardware selection
        CALL    START_DRIVE_1        ;Select for "the current drive" in hardware, Side A

```

```

CALL HOME ;Restore disk head
CALL NZ,SHOW_ERRORS ;print out errors if any

CALL CHECK_WP ;See if disk is write protected
JP Z,IBMFORM_OK1 ;Go to the core routine

LD IX,(IX_OLD_STORE) ;Nothing altered go back to main menu
LD A,(DRIVE_STORE)
LD (DRIVE_1),A ;Get back original drive
CALL START_DRIVE_1 ;RE-select to old drive disk format, Side A
LD A,(CMD_STORE) ;So we dont pick up other menu menu items on
RET ;return back to MENU

IBMFORM_OK1:
LD HL,IBM_FORMAT1 ;Formatting started
CALL PMSG

IBMFORM_OK:
LD HL,FBUFFER ;Will build the complete sector image here
LD (TADDR),HL ;Store the pointer here.

LD A,1
LD (IY+SCTR),A ;Start at sector 1
XOR A,A ;Setup for the BIOS below
LD (IY+SIDE),A ;Side A
CALL SET_SIDE ;Make sure IX+HW_BYTE is set for correct side
XOR A,A ;Fall through with 0 in [A] for trk 0

IBM_NEXT_TRK:
LD (IY+TRK),A ;store track info for each loop
CALL BUILD_TRACK ;Build a complete track image in RAM

CALL SEEK_TRACK_NV ;Seek with verify command to 1791/5
CALL NZ,SHOW_ERRORS ;If any show error flags

LD HL,FORM_TRK_MSG ;At track xx
CALL PMSG
LD A,(IY+TRK)
CALL PACC

LD DE,FBUFFER ;Move Start of track image into DE
PUSH DE ;save it
LD HL,(E_GAP4_MARK) ;End+1 of track header
DEC HL
SBC HL,DE
EX DE,HL ;Count now in DE
POP HL ;DE on stack to --> HL

CALL WRITE_TRACK ;<<<< Write track with hardware >>>>
CALL NZ,SHOW_ERRORS

CALL CHECKABORT ;SP will halt, ESC will abort
JP NZ,IBMF_ABORT

INC (IY+TRK) ;what was the last track number just done
LD A,(IY+TRK)
CP A,(IX+NTRKS) ;Check if we are on the last tarck
JR NZ,IBM_NEXT_TRK ;If not then back to updating tracks and doing it again.

;
LD HL,COPY_SYSTRKS ;Do you wish to copy CPM from the system tracks of drive A:
CALL PMSG
CALL GETCMD ;Ask if we wish to continue
CP A,'Y'
JP NZ,IBMF_DONE1 ;If not return
LD HL,COPYING_CPM
CALL PMSG
LD A,(IX+SYS_TRKS)
LD (COPY_TRK_COUNT),A ;Count down 2 tracks

LD A,1 ;Second drive on A: set (DRIVE_2) to it

```

```

LD      (DRIVE_2),A
;
IBM_CPM_LOOP      ;Now copy CPM from A: Drive to B: drive. IX is already 8" IBM
CALL      START_DRIVE_2      ;Select for A: drive in hardware, Side A
CALL      HOME
CALL      NZ,SHOW_ERRORS      ;print out errors if any

LD      A,(COPY_TRK)      ;Because home sets TRK to 0
LD      (IY+TRK),A

CALL      SEEK_TRACK_V
CALL      NZ,SHOW_ERRORS      ;print out errors if any

LD      HL,FBUFFER      ;Will build the complete sector image here (5000H)
LD      (TADDR),HL      ;Store the pointer here.
LD      E,1      ;Start with 1st sector
LD      D,(IX+NSCTRS)      ;[D] contains the number of sectors to read
;
CALL      MULTI_SEC_RD      ;<<<<<< Read multiple sectors
CALL      NZ,SHOW_ERRORS      ;If any, show error flags
;
;---
;
CALL      START_DRIVE_1      ;<<<<<< WRITE TO Destination drive
CALL      HOME      ;Select for B: drive in hardware, Side A
CALL      NZ,SHOW_ERRORS      ;Not clear why this is needed. Get seek errors without it
;print out errors if any

LD      A,(COPY_TRK)      ;Because home sets TRK to 0
LD      (IY+TRK),A

CALL      SEEK_TRACK_V
CALL      NZ,SHOW_ERRORS      ;print out errors if any

LD      HL,FBUFFER      ;Will obtain the complete sector image from here
LD      (TADDR),HL      ;Store the pointer here.
LD      E,1      ;Start with 1st sector
LD      D,(IX+NSCTRS)      ;Count of sectors to read
;
CALL      MULTI_SEC_WR      ;<<<<<< Write multiple sectors
CALL      NZ,SHOW_ERRORS      ;If any, show error flags
;
LD      A,(COPY_TRK)      ;bump up a track
INC      A
LD      (COPY_TRK),A
LD      A,(COPY_TRK_COUNT)      ;Have we more tracks to do
DEC      A
LD      (COPY_TRK_COUNT),A
JP      Z,IBMF_DONE2
;
CALL      CHECKABORT      ;SP will halt, ESC will abort
JR      NZ,IBMF_DONE1      ;Loop until abort or all tracks done
JP      IBM_CPM_LOOP
;
IBMF_DONE2
LD      A,(ERRORS_FLAG)      ;Were there errors
OR      A,A
JR      NZ,IBMF_ABORT
LD      HL,END_FORM_MSG2
CALL      PMSG
CALL      START_DRIVE_1      ;Select for "the current drive" in hardware, Side A
LD      A,(CMD_STORE)      ;So we dont pick up other menu menu items on
RET      ;return back to MENU
;
IBMF_DONE1:
LD      A,(ERRORS_FLAG)      ;Were there errors
OR      A,A
JR      NZ,IBMF_ABORT
LD      HL,END_FORM_MSG1
CALL      START_DRIVE_1      ;Select for "the current drive" in hardware, Side A

```

```

LD      A, (CMD_STORE)      ;So we dont pick up other menu menu items on
RET                                           ;return back to MENU
;
IBMF_ABORT:
LD      HL, FORM_ERRMSG     ;Say error formatting disk
CALL    PMSG
LD      SP, STACK
JP      START               ;Must abort everything because disk status is unknown
;
;
;
;***** DISK TO DISK COPY *****
; Unlike much of this program, this module is somewhat hardware specific in that it assumes
; the source and destination disks are identical and are using the SAME disk formats.
; Normall they are 8" drives but it should be OK with two 5" drives as well. (5" was not tested)
; I decided not to use whole track R/W's because there is no CRC error checking of the data.
; Uses the 1791/5 multi sector read command. Will do both sides if DS disk
;
COPY_DISK:
XOR     A,A
LD      (ERRORS_FLAG),A      ;Will keep tab on errors during this routine

LD      HL,COPY_MSG         ;Announce the disk copy msg
CALL    PMSG
XOR     A,A                 ;Setup for the BIOS below
LD      (IY+TRK),A         ;Start with track 0
LD      (COPY_TRK),A
LD      (IY+SIDE),A        ;Start on Side A
CALL    START_DRIVE_1      ;Start "the current drive" hardware, Side A
CALL    HOME
CALL    NZ,SHOW_ERRORS     ;print out errors if any
JP      NZ,DONE_COPY

;
GET_CDEST:
; <<< DESTINATION DRIVE
LD      HL,COPY2_MSG       ;Destination disk
CALL    PMSG
CALL    SELECT_DR_2        ;Get second drive, put it in (DRIVE_2)
CP      A,0FFH
JR      Z,GET_CDEST        ;Invalid drive, start over
CP      A,ESC
JP      Z,DONE_COPY

;
FORMATS_OK:
CALL    START_DRIVE_2      ;Select for second drive in hardware, Side A
CALL    HOME
CALL    NZ,SHOW_ERRORS     ;print out errors if any
JP      NZ,DONE_COPY       ;If errors abort

CALL    CHECK_WP           ;See if disk is write protected
JP      NZ,DONE_COPY       ;Abort if second drive is write protected

;
COPYWP_OK:
LD      HL,HOW_MANY_TRKS
CALL    PMSG
CALL    GETCMD
CP      A,ESC              ;Abort if ESC character
JP      Z,DONE_COPY
CP      A,'S'
JR      NZ,NOT_S
LD      A,2
JR      GOT_TRK_CNT
NOT_S:  CP      A,'A'
JR      NZ,NOT_ALL
LD      A,(IX+NTRKS)
JR      GOT_TRK_CNT
NOT_ALL:LD HL,INVALID_TRK_CT ;Must be S or All
CALL    PMSG
JR      COPYWP_OK

```



```

;
GOT_TRK_CNT:
    LD        (COPY_TRK_COUNT),A        ;Store number or tracks to copy

COPY_R_LOOP:
    ; <<<< READ FROM Source drive
    CALL     START_DRIVE_1              ;Select for "the current drive" in hardware, Side A
    CALL     SET_SIDE                   ;Make sure drive is set for correct side
    CALL     HOME                       ;
    CALL     NZ,SHOW_ERRORS              ;print out errors if any

    LD      A,(COPY_TRK)                ;Because home sets TRK to 0
    LD      (IY+TRK),A

    CALL     SEEK_TRACK_V                ;Get appropriate track
    CALL     NZ,SHOW_ERRORS              ;not clear why this is so.

    LD      HL,COPY_AT_TRK
    CALL     PMSG
    CALL     SHOW_TS_LOC                 ;Announce current Track. Shows current (TRK)

    LD      HL,FBUFFER                   ;Will build the complete sector image here (5000H)
    LD      (TADDR),HL                   ;Store the pointer here.
    LD      E,1                           ;Start with 1st sector
    LD      D,(IX+NSCTRS)                 ;[D] contains the number of sectors to read

;
    CALL     MULTI_SEC_RD                 ;<<<<<< Read multiple sectors
    CALL     NZ,SHOW_ERRORS              ;If any, show error flags

;
;---
;
    ;<<<<< WRITE TO Destination drive
    CALL     START_DRIVE_2              ;Start "the second drive" in hardware, Side A
    CALL     SET_SIDE                   ;Update the side hardware
    CALL     HOME                       ;Not clear why this is needed. Get seek errors without it
    CALL     NZ,SHOW_ERRORS              ;print out errors if any

    LD      A,(COPY_TRK)                ;Because home sets TRK to 0
    LD      (IY+TRK),A

    CALL     SEEK_TRACK_V                ;Get appropriate track
    CALL     NZ,SHOW_ERRORS              ;not clear why this is so.

    LD      HL,WRITE_AT_TRK
    CALL     PMSG
    CALL     SHOW_TS_LOC                 ;Announce current Track. Shows current (TRK)

    LD      HL,FBUFFER                   ;Will obtain the complete sector image from here
    LD      (TADDR),HL                   ;Store the pointer here.
    LD      E,1                           ;Start with 1st sector
    LD      D,(IX+NSCTRS)                 ;Count of sectors to read

;
    CALL     MULTI_SEC_WR                 ;<<<<<< Write multiple sectors
    CALL     NZ,SHOW_ERRORS              ;If any, show error flags

;
    CALL     SWAP_SIDES                  ;Sides swap check
    JR      NZ,R_TRK_OK                  ;If B side (NZ), same track
;
; Else see if more tracks are required
    LD      A,(COPY_TRK)                  ;bump up a track
    INC     A
    LD      (COPY_TRK),A
    LD      A,(COPY_TRK_COUNT)           ;Have we more tracks to do
    DEC     A
    LD      (COPY_TRK_COUNT),A
    JP     Z,DONE_COPY2

;
R_TRK_OK:
    CALL     CHECKABORT                  ;SP will halt, ESC will abort
    JR      NZ,DONE_COPY                 ;Loop until abort or all tracks done
    JP     COPY_R_LOOP
;

```

```

DONE_COPY2:                ;We are done restore both drives
    LD    A,(ERRORS_FLAG)    ;Were there errors
    OR    A,A
    JR    NZ,BAD_COPY
    LD    HL,COPYING_DONE    ;Announce we are finished
    CALL  PMSG
DONE_COPY:
    CALL  START_DRIVE_1    ;Select for "the current drive" in hardware, Side A
    JP    END_CMD
;
BAD_COPY:LD    HL,BAD_COPY_MSG    ;errors seen so say bad copy
    CALL  PMSG
    JR    DONE_COPY
;
;***** VERIFY TWO DISKS ARE THE SAME SECTOR BY SECTOR*****
; This module assumes the disks sizes are identical and are using the same disk formats.
; Normall they are 8" drives but it should be OK with two 5" drives as well. (5" was not tested)
;
VERIFY_DISK:
    LD    HL,VERIFY_MSG    ;Announce the disk copy msg
    CALL  PMSG
    XOR   A,A                ;Setup for the BIOS below
    LD    (IY+TRK),A        ;Start with track 0
    LD    (COPY_TRK),A
    LD    (IY+SIDE),A      ;Start on Side A of A:
    INC  A
    LD    (IY+SCTR),A      ;Start at sector 1
    CALL  START_DRIVE_1    ;Start the drive hardware (A side)
    CALL  HOME
    CALL  NZ,SHOW_ERRORS   ;print out errors if any
    JP    NZ,END_CMD
;
GET_VDEST:                ;<<< DESTINATION DRIVE
    LD    HL,VERIFY2_MSG    ;Destination disk
    CALL  PMSG
    CALL  SELECT_DR_2      ;Ask user which drive for second drive
    CP    A,OFFH           ;Note IX will be the same for both disks
    JR    Z,GET_VDEST     ;Invalid drive, start over
    CP    A,ESC
    JP    Z,END_CMD
;
HOW_MANY:
    LD    HL,HOW_MANY_TRKS
    CALL  PMSG
    CALL  GETCMD
    CP    A,ESC            ;Abort if ESC character
    JP    Z,END_CMD
    CP    A,'S'
    JR    NZ,NOT_VS
    LD    A,2
    JR    VGOT_TRK_CNT
NOT_VS: CP    A,'A'
    JR    NZ,NOT_VALL
    LD    A,(IX+NTRKS)
    JR    VGOT_TRK_CNT
NOT_VALL:
    LD    HL,INVALID_TRK_CT ;Must be S or All
    CALL  PMSG
    JR    HOW_MANY
;
VGOT_TRK_CNT:
    LD    (COPY_TRK_COUNT),A ;Store number or tracks to copy
;
VERIFY_R_LOOP:
    CALL  START_DRIVE_1    ;Start the drive hardware (A side)
    CALL  SET_SIDE        ;Update the side hardware
    LD    A,(IY+SCTR)     ;Is it a new track
    CP    A,1
    JR    NZ,VR_1

```

```

CALL HOME ;If a new track need to seek
CALL NZ,SHOW_ERRORS ;If any show error flags
LD A,(COPY_TRK)
LD (IY+TRK),A

CALL SEEK_TRACK_V ;seek to the new track position
CALL NZ,SHOW_ERRORS ;If any show error flags
VR_1:
LD HL,READ_AT_TRK
CALL PMSG
CALL SHOW_TSS_LOC ;Announce current Track/Sec

LD HL,FBUFFER ;Will place the complete sector image here (5000H)
LD (TADDR),HL
;
CALL READ_SECTOR ;<<<<<< Read sector of A: Drive>>>>>>
CALL NZ,SHOW_ERRORS ;If any, show error flags
;
CALL START_DRIVE_2 ;Start the drive hardware (A side)
CALL SET_SIDE ;Update the side hardware -- assume side A only
LD A,(IY+SCTR) ;Is it a new track
CP A,1
JR NZ,VR_2
CALL HOME ;Not clear why but get a seek error without this!
CALL NZ,SHOW_ERRORS ;If any, show error flags
LD A,(COPY_TRK)
LD (IY+TRK),A

CALL SEEK_TRACK_V ;seek to the new track position
CALL NZ,SHOW_ERRORS ;If any show error flags
VR_2:
LD HL,VERIFY_AT_TRK
CALL PMSG
CALL SHOW_TSS_LOC ;Announce current Track

LD HL,FBUFFER2 ;Will place the complete sector image here (5000H)
LD (TADDR),HL

CALL READ_SECTOR ;<<<<<< Read sector on B: Drive
CALL NZ,SHOW_ERRORS ;If any, show error flags
;
CALL CMP_BUFFERS ;Check for errors
JR Z,VCOMPARE_OK
LD HL,SEC_V_ERROR ;R/W Error found
CALL PMSG
CALL SHOW_TSS_LOC ;trk,sec,head

VCOMPARE_OK:
CALL CHECKABORT ;SP will halt, ESC will abort
JP NZ,DONE_VERIFY1 ;Abort if requested

INC (IY+SCTR) ;Get next sector
LD A,(IY+SCTR)
DEC A ;Because sectors are numbered 1,2,3...
CP A,(IX+NSCTRS) ;Have we done a complete track yet
JP NZ,VERIFY_R_LOOP

CALL SWAP_SIDES ;Sides swap check
LD A,1 ;Back to sector 1 no matter what
LD (IY+SCTR),A ;Store for loop test
JP NZ,VERIFY_R_LOOP ;If B side (NZ), same track, back to sec 1

LD A,(COPY_TRK) ;bump up a track
INC A
LD (COPY_TRK),A
LD A,(COPY_TRK_COUNT) ;Have we more tracks to do
DEC A
LD (COPY_TRK_COUNT),A
JP NZ,VERIFY_R_LOOP

```

```

;
END_VERIFY:                                ;We are done restore both drives
      LD      HL,VERIFY_DONE
      CALL    PMSG
;
DONE_VERIFY1:
      CALL    START_DRIVE_1                ;Start the drive hardware (A side)
      CALL    HOME                          ;Will set TRK to 0
      CALL    NZ,SHOW_ERRORS                ;print out errors if any
      LD      A,(CMD_STORE)                 ;So we dont pick up other menu items
      RET                                       ;Back to main menu
;
;
;
; This routine will copy the CPM system tracks FROM another current disk and writes it TO
; the current disk. Its just a modification of COPY_DISK where the number of tracks set aside
; is picked off from the Disk paramater table
; It should be OK with any disk format but so far has just been checked out on
; standard 8" SSSD IBM 3740 disks. Both disks must use the same format. (untested for 5" disks)
;
COPY_CPM3_SYS:
      XOR     A,A
      LD      (ERRORS_FLAG),A                ;Will keep tab on errors during this routine

      LD      HL,SYS_COPY_MSG
      CALL    PMSG
      XOR     A,A                            ;Setup for the BIOS below
      LD      (IY+TRK),A                    ;Start with track 0
      LD      (COPY_TRK),A
      LD      (IY+SIDE),A                  ;Start on Side A

      CALL    START_DRIVE_1                ;Start the drive hardware (A side)
      CALL    HOME
      CALL    NZ,SHOW_ERRORS                ;print out errors if any
      JP     NZ,END_CMD

      CALL    CHECK_WP                      ;See if disk is write protected
      JP     NZ,END_CMD

GET_SDEST:                                ;<<<< SOURCE DRIVE
      LD      HL,SYS_COPY_MSG1              ;Source disk of CPM
      CALL    PMSG
      CALL    SELECT_DR_2                  ;Ask user which drive to get the CPM system tracks from
      CP     A,OFFH                        ;Note IX will be the same for both disks
      JR     Z,GET_SDEST                   ;Invalid drive, start over
      CP     A,ESC
      JP     Z,END_CMD

      CALL    START_DRIVE_2                ;Start the drive hardware (A side)
      CALL    HOME
      CALL    NZ,SHOW_ERRORS                ;print out errors if any
      JP     NZ,END_CMD

COPYWP_SOK:
      LD      A,(IX+SYS_TRKS)                ;How many tracks for CPM system on this disk
      LD      (COPY_TRK_COUNT),A            ;Store number or tracks to copy

SYS_R_LOOP:                                ; <<<<< READ FROM Source drive
      CALL    START_DRIVE_2                ;Start the source drive hardware (A side)
      CALL    SET_SIDE                      ;Make sure IX+HW_BYTE is set for correct side
      CALL    HOME
      CALL    NZ,SHOW_ERRORS                ;print out errors if any

      LD      A,(COPY_TRK)                 ;Because home sets TRK to 0
      LD      (IY+TRK),A
      CALL    SEEK_TRACK_V
      CALL    NZ,SHOW_ERRORS                ;print out errors if any

      LD      HL,COPY_SYS_TRK

```

```

CALL    PMSG
CALL    SHOW_TS_LOC          ;Announce current Track. Shows current (TRK)

LD      HL,FBUFFER          ;Will build the complete sector image here (5000H)
LD      (TADDR),HL         ;Store the pointer here.
LD      E,1                 ;Start with 1st sector
LD      D,(IX+NSCTRS)      ;[D] contains the number of sectors to read
;
CALL    MULTI_SEC_RD       ;<<<<<< Read multiple sectors
CALL    NZ,SHOW_ERRORS     ;If any, show error flags
;
;---
;
CALL    START_DRIVE_1      ;<<<<<< WRITE TO Destination drive
CALL    SET_SIDE           ;Start the destination drive hardware (A side)
CALL    HOME               ;Update the side hardware (A)
CALL    HOME               ;Not clear why this is needed. Get seek errors without it
CALL    NZ,SHOW_ERRORS     ;print out errors if any

LD      A,(COPY_TRK)      ;Because home sets TRK to 0
LD      (IY+TRK),A
CALL    SEEK_TRACK_V
CALL    NZ,SHOW_ERRORS     ;print out errors if any

LD      HL,WRITE_SYS_TRK
CALL    PMSG
CALL    SHOW_TS_LOC       ;Shows current (TRK,SIDE)

LD      HL,FBUFFER          ;Will obtain the complete sector image from here
LD      (TADDR),HL         ;Store the pointer here.
LD      E,1                 ;Start with 1st sector
LD      D,(IX+NSCTRS)      ;Count of sectors to read
;
CALL    MULTI_SEC_WR       ;<<<<<< Write multiple sectors
CALL    NZ,SHOW_ERRORS     ;If any, show error flags
;
CALL    SWAP_SIDES         ;Sides swap check
JR      NZ,R_TRK_SOK      ;If B side (NZ), same track
;
;Else see if more tracks are required
LD      A,(COPY_TRK)      ;bump up a track
INC     A
LD      (COPY_TRK),A
LD      A,(COPY_TRK_COUNT) ;Have we more tracks to do
DEC     A
LD      (COPY_TRK_COUNT),A
JP     Z,DONE_SYS2
;
R_TRK_SOK:
CALL    CHECKABORT        ;SP will halt, ESC will abort
JR      NZ,DONE_SYS1      ;Loop until abort or all tracks done
JP     SYS_R_LOOP
;
DONE_SYS2:
;We are done restore both drives
LD      A,(ERRORS_FLAG)   ;Were there errors
OR      A,A
JR      NZ,ERROR_SYS3
LD      HL,DONE_SYS_MSG   ;Announce we are finished
CALL    PMSG
DONE_SYS1:
CALL    START_DRIVE_1     ;Start the source drive hardware (A side)
CALL    HOME              ;Will set TRK to 0
CALL    NZ,SHOW_ERRORS    ;print out errors if any
LD      A,(CMD_STORE)     ;So we dont pick up other menu items
RET
;Back to main menu
;
ERROR_SYS3:
LD      HL,ERRORS_SEEN    ;errors encountered
CALL    PMSG
JR      DONE_SYS1

```

```

;
;
;
;=====
;===== 1791/5 COMMAND ROUTINES =====
;=====
;
; SEND A RESTORE COMMAND FOR THE CURRENT DISK.
; On return: Z if no errors. NZ if errors with errors in (ERSTAT)and [A]
; Note (TRK) IS set to 0, (IX+HW_BYTE) SIDE_BIT (7) is not changed
;
HOME:  LD      A,HOME_ERR_MASK          ;Error mask for Type I RESTORE CMD
      LD      (ERMASK),A              ;Save error mask for end/error routine
      XOR     A,A
      LD      (IY+TRK),A
HOME2: IN      A,(STATUS)
      AND     A,1
      JR      NZ,HOME2                ;Wait until 1791/5 is ready
      CALL    WAIT_OFF                ;Disable Wait State Hardware (just in case it is on)
      LD      A,RSCMD
      LD      (CHIP_CMDSV),A          ;Store it in case error flags are to be shown
      OUT     (CMD),A
      CALL    END_ROUTINE             ;Wait, check status and return
      RET
;
;
; SEND A HEAD STEP-IN COMMAND TO THE CURRENT DRIVE
; On return: Z if no errors. NZ if errors with errors in (ERSTAT)and [A]
; Note (TRK) is NOT updated to new track position, (IX+HW_BYTE) SIDE_BIT (7) is unchanged
;
STEP_IN_CMD:
      LD      A,SIN_ERR_MASK          ;Error mask for Type I CMD
      LD      (ERMASK),A              ;Save error mask for end/error routine
      LD      A,STEPIN                ;Send Step-in (with verify) CMD
      LD      (CHIP_CMDSV),A          ;Store it in case error flags are to be shown
STEP_IN2:
      IN      A,(STATUS)
      AND     A,1
      JR      NZ,STEP_IN2             ;Wait until 1791/5 is ready
      CALL    WAIT_OFF                ;Disable Wait State Hardware (just in case it is on)
      LD      A,STEPIN
      LD      (CHIP_CMDSV),A          ;Store it in case error flags are to be shown
      OUT     (CMD),A
      CALL    END_ROUTINE             ;Wait, check status and return
      RET
;
; Seek with track WITH verify. Assumes valid track in (TRK)
; On return: Z if no errors. NZ if errors, (ERSTAT)and [A] has errors
; Note (TRK) is NOT updated to new track position
;
SEEK_TRACK_V:
      LD      A,SK_ERR_MASK           ;Error mask for Type I Seek CMD
      LD      (ERMASK),A              ;Save error mask for end/error routine
      LD      A,SKCMD                 ;Send seek (with verify) CMD
      LD      (CHIP_CMDSV),A          ;Store it in case error flags are to be shown
      LD      A,SEEK_RETRY_MAX        ;In case of errors will retry seeking
      LD      (SEEK_RT_COUNT),A       ;a few times
      JR      SEEK_TRK
;
;
; Seek track with NO verify. Assumes valid track in (TRK)
; On return: Z if no errors. NZ if errors, (ERSTAT)and [A] has errors
; Note (TRK) is NOT updated to new track position
;
SEEK_TRACK_NV:
      LD      A,SK_ERR_MASK           ;Error mask for Type I Seek CMD
      LD      (ERMASK),A              ;Save error mask for end/error routine
      LD      A,SKNCMD                ;Send seek CMD ((with NO verify)

```

```

LD      (CHIP_CMDSV),A      ;Store it in case error flags are to be shown
LD      A,SEEK_RETRY_MAX   ;In case of errors will retry seeking
LD      (SEEK_RT_COUNT),A  ;a few times
SEEK_TRK:
IN      A,(STATUS)
AND     A,1
JR      NZ,SEEK_TRK        ;wait until 1791/5 is ready
CALL    WAIT_OFF           ;Disable Wait State Hardware (in case it was on)
LD      A,(IY+TRK)         ;Send required track to 1791/5
OUT     (DATA),A
LD      A,(CHIP_CMDSV)     ;Was it a seek or seek_nv
OUT     (CMD),A            ;Send seek cmd
CALL    END_ROUTINE       ;Wait, check status and return
RET     Z                  ;Return if no errors

LD      A,(SEEK_RT_COUNT)
DEC     A                  ;Retrys 2,1... Will reseek
LD      (SEEK_RT_COUNT),A
JR      Z,BAD_SEEK
JR      SEEK_TRK          ;Try re-reading the sector

BAD_SEEK:
XOR     A,A
DEC     A                  ;Set to NZ
LD      A,(ERSTAT)
RET

;
;
;
;
; Read an ID from the track. Assumes valid track in (TRK)
; Note any Sector ID from the track will be obtained depending where the head falls
; The currently selected side at (SIDE) is read
;
READ_ID:LD      A,ID_ERR_MASK
LD      (ERMASK),A        ;Store for error display
RDSC_1: IN     A,(STATUS)
AND     A,1
JR      NZ,RDSC_1        ;wait until 1791/5 is ready
LD      HL,IDSV          ;Will store the 6 bytes here
LD      BC,600H+DATA     ;6 bytes in B, Data port in C below
CALL    WAIT_ON          ;Enable hardware wait states
LD      A,RDACMD         ;Send the Read Track ID CMD
LD      (CHIP_CMDSV),A   ;Store it in case error flags are to be shown
DI      ;Just in case
OUT     (CMD),A
JR      MM2              ;Slight delay
MM2:    JR      MM3
MM3:    INIR          ;Block input 6 bytes
EI
CALL    END_ROUTINE     ;Wait, turn off wait hware, check status
RET     ;and return

;
;
;
; <<< CORE SECTOR READ ROUTINE>>>. Assumes valid track in (TRK)&(SEC)
; Address in (TADDR). The currently selected side at (SIDE) is setup
; Note: (TADDR) is unaltered, (DMA_NEXT) is updated for repetative calls to this routine
;
READ_SECTOR:
LD      A,RS_ERR_MASK    ;Error mask from 1791/5 after read is done
LD      (ERMASK),A
LD      A,SEC_RETRY_MAX  ;In case of errors will retry reading sector
LD      (SEC_RT_COUNT),A ;a few times
AGAIN_RD:
LD      HL,(TADDR)       ;Deposit sector info here
LD      A,(IY+SCTR)
OUT     (SECTOR),A
LD      C,DATA           ;C = DATA port for INIR below
CALL    WAIT_ON

```

```

if CHIP_1791
  LD  A,RDCMD91      ;Read sector command to 1791
  LD  (CHIP_CMDSV),A ;store cmd here for Error display (if any)
endif

if CHIP_1795
  LD  A,RDCMD95      ;Read sector command to 1795
  LD  (CHIP_CMDSV),A ;store cmd here for Error display (if any)
  OR  A,(IY+SIDE)    ;If B side, need to OR in bit 1 (02H).
endif

  DI
  OUT (CMD),A
  LD  A,(IX+SIZE)    ;128=0,256=1,512=2 or 1024=3 byte sector size
  LD  B,NBYTES
  OR  A,A            ;Block input [B] Bytes->[HL] from port [C]
  JR  Z,R128256     ;Do 128 byte read
  LD  B,0
  CP  A,1
  JR  Z,R128256     ;Do 256 byte read
  CP  A,2
  JR  Z,R512        ;Do 512 byte read
  INIR
  INIR
R512: INIR          ;256X2 bytes
R128256:INIR       ;128 or 256 bytes (B=128 or 0)
  EI
  LD  (DMA_NEXT),HL ;Store next byte of DMA
  CALL END_ROUTINE ;Wait, turn off wait hware, check status and return
  RET Z            ;RET Z if all is OK
                ;Read failed. Do we retry again

  LD  A,(CRTDISP)   ;Check if detailed display flag is on
  OR  A,A
  JR  Z,MM6
  LD  HL,SEC_READ_RETRY ;'Re-reading SECTOR XXH.'
  CALL PMSG
  LD  A,(IY+SCTR)
  CALL PACC
  LD  HL,H_MSG
  CALL PMSG
MM6:  LD  A,(SEC_RT_COUNT)
  DEC  A            ;Retrys 4,3 2,1... Will reseek on 3rd re-read try
  LD  (SEC_RT_COUNT),A
  JR  Z,BAD_RD
  CP  A,2
  JR  NZ,AGAIN_RD  ;Try re-reading the sector
  LD  A,(CRTDISP)   ;Check if detailed display flag is on
  OR  A,A
  JR  Z,MM7B
  LD  HL,SEC_RH_RETRY ;'Re-Seeking head for re-reading SECTOR XXH.'
  CALL PMSG
  LD  A,(IY+SCTR)
  CALL PACC
  LD  HL,H_MSG
  CALL PMSG
MM7B:
  LD  A,(IY+TRK)    ;If it failed a second time, home, reseek and try again
  PUSH AF
  CALL HOME         ;Restore to track 0 (Note assumes TRK is unchanged in HOME)
  POP AF
  LD  (IY+TRK),A
  CALL SEEK_TRACK_V ;Re-seek to track
  JP  AGAIN_RD
BAD_RD XOR  A,A
  DEC  A            ;Set to NZ
  LD  A,(ERSTAT)
  RET
;

```





```

DEC      D
JR       NZ,MR512A
JR       DONE_MR

MR1K:   LD      B,0
        POP     AF
        OUT     (CMD),A          ;send it to the 179x CMD port immediatly
MR1KA:  INIR
        INIR
        INIR
        LD      B,0
        DEC     D
        JR      NZ,MR1KA
        JR      DONE_MR

DONE_MR:
EI
LD      (DMA_NEXT),HL          ;Store "next location". Used by other routines
CALL    END_ROUTINE          ;Wait, turn off wait hware, check status and return
RET     Z                    ;RET Z if all is OK
XOR     A,A
DEC     A                    ;Set to NZ
LD      A,(ERSTAT)          ;Return with error bits
RET

;
;
; Read Track. Assumes head is over relevent track in (TRK)
; Will place data at (TADDR)
; The currently selected side at (SIDE) is read
; The total size of a track in bytes is in [DE]

READ_TRACK:
LD      A,RT_ERR_MASK        ;Setup for error checking
LD      (ERMASK),A
LD      HL,(TADDR)
LD      A,(IY+TRK)
OUT     (TRACK),A
CALL    WAIT_ON              ;Turn on VII board wait state hardware
LD      A,RDTCMD             ;setup for 179x read track command
LD      (CHIP_CMDSV),A      ;Store it in case error flags are to be shown
DI      ;Disable Interrupts
OUT     (CMD),A              ;send it to the 179x CMD port
JR      MM7                  ;Slight delay
MM7:    JR      R_LOOP

R_LOOP: IN      A,(DATA)      ;Get all track bytes
LD      (HL),A
INC     HL
DEC     DE                    ;Track size is in DE
LD      A,E
OR      A,A
JR      NZ,R_LOOP
OR      A,D
JR      NZ,R_LOOP          ;Have we sent DE bytes yet
EI
CALL    END_ROUTINE          ;Wait, turn off wait hware, check status and return
RET     Z                    ;RET Z if all is OK
XOR     A,A
DEC     A                    ;Set to NZ
LD      A,(ERSTAT)          ;Return with error bits
RET

;
;
; <<< CORE SECTOR WRITE ROUTINE >>> . Assumes valid track in (TRK) & (SEC)
; Address in (TADDR). The currently selected side at (SIDE) is setup
; Note: (TADDR) is unaltered, (DMA_NEXT) is updated for repetative calls to this routine
;
WRITE_SECTOR:

```

```

LD      A,WS_ERR_MASK ;Error mask from 1791/5 after read is done
LD      (ERMASK),A
LD      A,SEC_RETRY_MAX ;In case of errors will retry reading sector
LD      (SEC_RT_COUNT),A ;a few times
AGAIN_WR:
LD      HL,(TADDR) ;Deposit sector info here
LD      A,(IY+SCTR)
OUT     (SECTOR),A
LD      C,DATA ;C = DATA port for OTIR below
CALL   WAIT_ON

if CHIP_1791
LD      A,WRCMD91 ;Read sector command to 1791
LD      (CHIP_CMDSV),A ;store cmd here for Error display (if any)
endif

if CHIP_1795
LD      A,WRCMD95 ;Read sector command to 1795
LD      (CHIP_CMDSV),A ;store cmd here for Error display (if any)
OR      A,(IY+SIDE) ;If B side, need to OR in bit 1 (02H).
endif

DI
OUT     (CMD),A
LD      A,(IX+SIZE) ;128=0,256=1,512=2 or 1024=3 byte sector size
LD      B,NBYTES
OR      A,A ;Block input [B] Bytes->[HL] from port [C]
JR      Z,W128256 ;Do 128 byte read
LD      B,0
CP      A,1
JR      Z,W128256 ;Do 256 byte read
CP      A,2
JR      Z,W512 ;Do 512 byte read
OTIR
OTIR
W512: OTIR ;256X2 bytes
W128256:OTIR ;128 or 256 bytes (B=128 or 0)
EI
LD      (DMA_NEXT),HL ;Store next byte of DMA (for multiple sec R/W's)
CALL   END_ROUTINE ;Wait, turn off wait hware, check status and return
RET     Z ;RET Z if all is OK
;Read failed. Do we retry again
LD      A,(CRTDISP) ;Check if detailed display flag is on
OR      A,A
JR      Z,MM6W
LD      HL,SEC_WR_RETRY ;'Re-writing SECTOR XXH.'
CALL   PMSG
LD      A,(IY+SCTR)
CALL   PACC
LD      HL,H_MSG
CALL   PMSG
MM6W: LD      A,(SEC_RT_COUNT)
DEC     A ;Retrys 4,3 2,1... Will reseek on 3rd re-read try
LD      (SEC_RT_COUNT),A
JR      Z,BAD_WR
CP      A,2
JR      NZ,AGAIN_WR ;Try re-reading the sector
LD      A,(CRTDISP) ;Check if detailed display flag is on
OR      A,A
JR      Z,MM7W
LD      HL,SEC_WH_RETRY ;'Re-Seeking head for re-reading SECTOR XXH.'
CALL   PMSG
LD      A,(IY+SCTR)
CALL   PACC
LD      HL,H_MSG
CALL   PMSG
MM7W: LD      A,(IY+TRK) ;If it failed a second time, home, reseek and try again
PUSH   AF

```

```

CALL    HOME            ;Restore to track 0
POP     AF
LD      (IY+TRK),A
CALL    SEEK_TRACK_V   ;Re-seek to track
JP      AGAIN_RD
BAD_WR  XOR    A,A
DEC     A              ;Set to NZ
LD      A,(ERSTAT)    ;Return with error bits
RET

;
; Multi_sector Write of multiple sectors on a TARCK. Assumes head is over relevent track in (TRK)
; Will place data at (TADDR) The currently selected side at (SIDE) is read
; (DMA_NEXT) is updated for repetative calls to this routine.
; [D] contains the number of sectors (Max IX+NSCTRS), [E] the starting sector number.
; Note this can be easily modified for CPM3 to multi sector write by adjusting [D] & [E].
; Also note, actual timing for data I/O is quite tight for slow CPU's. Thus the strange layout.
;
MULTI_SEC_WR:
LD      A,WS_ERR_MASK ;Setup for error checking
LD      (ERMASK),A
LD      HL,(TADDR)
LD      A,(IY+TRK)
OUT     (TRACK),A
LD      A,E           ;first sector (usually 1)
LD      (IY+SCTR),A  ;For error dump (Actully error could be any sector on the track)
OUT     (SECTOR),A
LD      C,DATA       ;C = DATA port for INIR below

CALL    WAIT_ON      ;Turn on VII board wait state hardware

if CHIP_1791
LD      A,WRCMD91    ;Read sector command to 1791
LD      (CHIP_CMDSV),A ;store cmd here for Error display (if any)
endif

if CHIP_1795
LD      A,WRCMD95    ;Read sector command to 1795
LD      (CHIP_CMDSV),A ;store cmd here for Error display (if any)
OR      A,(IY+SIDE)  ;If B side need to OR in bit 1 (02H).
endif

SET     4,A          ;Multi sec bit
PUSH   AF           ;Store for now
DI      ;Disable Interrupts

LD      A,(IX+SIZE) ;128=0,256=1,512=2 or 1024=3 byte sector size
OR      A,A          ;Block input [B] Bytes->[HL] from port [C]
JR      Z,MW128      ;Do 128 byte read
CP      A,1
JR      Z,MW256      ;Do 256 byte read
CP      A,2
JR      Z,MW512      ;Do 512 byte read
JR      MW1K         ;Must be 1K sectors

MW128: LD      B,128  ;Note the code is spread out like this with
POP     AF           ;each sector having its own module because of timing
OUT     (CMD),A     ;send it to the 179x CMD port immediatly
MW128A: OTIR        ;128 bytes (B=128)
LD      B,128
DEC     D
JR      NZ,MW128A
JR      DONE_MW

MW256: LD      B,0
POP     AF
OUT     (CMD),A     ;send it to the 179x CMD port immediatly
MW256A: OTIR        ;256 bytes
LD      B,0
DEC     D

```

```

JR      NZ,MW256A
JR      DONE_MW

MW512: LD      B,0
      POP     AF
      OUT     (CMD),A          ;send it to the 179x CMD port immediatly
MW512A: OTIR                    ;512 = 256x2
      OTIR
      LD      B,0
      DEC     D
      JR      NZ,MW512A
      JR      DONE_MW

MW1K:   LD      B,0
      POP     AF
      OUT     (CMD),A          ;send it to the 179x CMD port immediatly
MW1KA:  OTIR                    ;1K = 256x4
      OTIR
      OTIR
      OTIR
      LD      B,0
      DEC     D
      JR      NZ,MW1KA
      JR      DONE_MW

DONE_MW:
      EI
      LD      (DMA_NEXT),HL      ;Store "next" location. Used by other routines
      CALL   END_ROUTINE        ;Wait, turn off wait hware, check status and return
      RET    Z                  ;RET Z if all is OK
      XOR    A,A
      DEC    A                  ;Set to NZ
      LD     A,(ERSTAT)        ;Return with error bits
      RET

;
;
;
; Write Track. Assumes head is over relevent track in (TRK)
; Track data at (TADDR)
; The currently selected side at (SIDE) is written to
; The total size of a track in bytes is in [DE]
;
WRITE_TRACK:
      LD     A,WT_ERR_MASK      ;Setup for error checking
      LD     (ERMASK),A
      LD     HL,(TADDR)
      LD     A,(IY+TRK)
      OUT    (TRACK),A
      CALL   WAIT_ON           ;Turn on VII board wait state hardware
      LD     A,WRTCMD           ;setup for 179x WRITE track command
      LD     (CHIP_CMDSV),A     ;Store it in case error flags are to be shown
      DI                      ;Disable Interrupts
      OUT    (CMD),A           ;send it to the 179x CMD port
W_LOOP: LD     A,(HL)
      OUT    (DATA),A          ;Send all track bytes
      INC    HL
      DEC    DE
      LD     A,E
      OR     A,A
      JR     NZ,W_LOOP
      OR     A,D
      JR     NZ,W_LOOP        ;Have we sent DE bytes yet
      EI
      CALL   END_ROUTINE        ;Wait, turn off wait hware, check status and return
      RET    Z                  ;RET Z if all is OK
      XOR    A,A
      DEC    A                  ;Set to NZ
      LD     A,(ERSTAT)
      RET

```

```

;
;
; END Routine comes after each Type I command is completed. It waits for the 1791/5
; chip to time out then switches off the wait state generator and checks for errors.
; Returns Z if no errors, otherwise NZ with error in (ERSTAT). If chip gets hung waiting
; for status it forces a 1791/5 Interrupt CMD after ~ 5 seconds. This is usually due to
; a hardware problem.
;
END_ROUTINE:
    CALL    DELAY            ;Delay for hardware
    PUSH    BC              ;Setup a loop count
    PUSH    DE
    PUSH    HL
    LD      BC,0
    LD      E,STATUS_DELAY ;time out about 5 seconds
ENDR2:    IN      A,(STATUS) ;Wait until chip is not busy
    AND     A,1
    JR      Z,ENDR2_OK
    DJNZ   ENDR2           ;Try for ~5 seconds
    DEC     B
    DEC     C
    JR      NZ,ENDR2
    DEC     B
    DEC     C
    DEC     E
    JR      NZ,ENDR2
    CALL    WAIT_OFF       ;Disable Hardware wait states
    LD      HL,TIMEOUT_ERR
    CALL    PMSG
    CALL    SHOW_STAT_BITS
    LD      (ERSTAT),A
    CALL    ZCRLF
    CALL    FRCINT
    POP     HL
    POP     DE
    POP     BC
    JR      ENDR3
ENDR2_OK:
    POP     HL
    POP     DE
    POP     BC
    CALL    WAIT_OFF       ;Disable Hardware wait states
    IN      A,(STATUS)    ;(May already be off anyway for Seeks etc)
    LD      D,A
    LD      A,(ERMASK)    ;Error mask was stored here
    AND     A,D
    LD      (ERSTAT),A    ;Save error status for error routine
    RET     Z             ;RET Z if all is OK
ENDR3:    XOR     A,A
    DEC     A             ;Set to NZ
    LD      A,(ERSTAT)
    RET
;
;
;
;
; SELECT DRIVE IN HARDWARE, WILL BE THE NEW CURRENT DRIVE
; (IX+HW_BYTE) points to relevent drive selection, density and
; side(s) info for the current disk. Note: Sets head to Side A - ALWAYS
;
START_DRIVE_1:
    LD      A,(DRIVE_1)   ;Get the drive selection bit
    JR      START_DR

START_DRIVE_2:
    LD      A,(DRIVE_2)   ;Get the drive selection bit

START_DR:
    OR      A,(IX+HW_BYTE) ;OR in the density,side and size bits
    LD      (IOBYTE),A    ;For diagnostic display

```

```

AND    A,7FH          ;Strip off the current side bit flag of (IOBYTE)
CPL                    ;Hardware is inverted
SET    4,A           ;Force to side A
OUT    (SELECT),A
;
CALL   ZBITS
CALL   DELAY
RDYCK: IN  A,(STATUS) ;Drive select delay
AND    80H
JP     NZ,RDYCK      ;Return when ready
RET

;
; Delay for drive select hardware. Different for 8" and 5" drives
;
DELAY: BIT    5,(IX+HW_BYTE) ;8" or 5"
LD     A,39
JR     Z,DELAY1
LD     A,60
DELAY1: LD    B,0
M0:    DJNZ  M0
DEC    A
JR     NZ,DELAY1
RET

;
; This routine switches the active side of the current disk.
; Returns Z if no sides OR new side is A, NZ if new side is B.
;
SWAP_SIDES:
BIT    4,(IX+HW_BYTE) ;Is it a 1 or 2 sided disk
RET    Z              ;If 1 sided then return Z

LD     A,(IY+SIDE)
OR     A,A           ;If 0 we have done A side now do B
JR     Z,GOTO_B_SIDE1 ;switch over to B side
XOR    A,A
LD     (IY+SIDE),A
CALL   SET_SIDE     ;update the hardware
XOR    A,A
RET                    ;Return Z

GOTO_B_SIDE1:
LD     A,02H
LD     (IY+SIDE),A
CALL   SET_SIDE     ;Update the hardware
XOR    A,A
DEC    A
RET                    ;Returns NZ

;
;
; SELECT DRIVE SIDE IN HARDWARE
; Set side for sector R/W if different than current side
; (SIDE) = 0 if A side, (SIDE)= 02H if B side.
; Note Hardware bit 7 of HW_BYTE is updated here!
;
SET_SIDE:
LD     A,(IX+HW_BYTE) ;Get current side info
BIT    4,A           ;Is it a 1 or 2 sided disk
RET    Z            ;Return if single sided

LD     A,(IY+SIDE)  ;Which side do we want. (0 for A, 02H for B)
OR     A,A
JR     NZ,SET_HW_B
IN     A,(SELECT)  ;get hardware selection
SET    4,A         ;Port hardware for side selection (1=A, 0=B)
OUT    (SELECT),A  ;Select side A in hardware (inverted)
RES    7,(IX+HW_BYTE) ;Set to side to A (Will show up in IOBYTE info)
JR     DONE_SET_SIDES

SET_HW_B:
IN     A,(SELECT)  ;get hardware selection
RES    4,A         ;Port hardware for side selection (1=A, 0=B)
OUT    (SELECT),A  ;Select side B in hardware

```

```

        SET      7,(IX+HW_BYTE) ;Set to side to B (Will show up in IOBYTE info)
;
DONE_SET_SIDES:
        PUSH    BC
        LD      B,0FFH          ;Slight delay for hardware
DELYS:  DJNZ    DELYS
        POP     BC
        RET
;
;
;
; Turn on or off the hardware wait state generator. This hardware stops and starts the CPU
; as bytes are read/written to the 179x/Disk
;
;         DISABLE WAIT STATES
WAIT_OFF:
        IN      A,(SELECT)
        OR      80H
        OUT     (SELECT),A
        RET
;
;         ENABLE WAIT STATES
WAIT_ON:
        IN      A,(SELECT)
        AND     7FH
        OUT     (SELECT),A
        RET
;
;
;         FORCE CHIP INTERRUPT
FRCINT: LD      A,0D0H
        OUT     (CMD),A
        LD      A,10
FRC1:   DEC     A
        JR      NZ,FRC1
        IN      A,(STATUS)
        RET
;
;
;
; General routine to show the current location of the active disk head.
; Will display track, sector and side if two sided drive/disk
; Format is 'At TRACK xxH, SCETOR yyH, SIDE A/B.'
; Note: NO CR/LF at start of string
;
SHOW_TSS_LOC:
        PUSH    HL
        LD      HL,ATTRK        ;Announce current track position
        CALL    PMSG
        IN      A,(TRACK)
        CALL    PACC
        LD      HL,ATSEC        ;and sector position
        CALL    PMSG
        IN      A,(SECTOR)
        CALL    PACC
SHOW_SIDES:
;Announce head if 2 sided disk
        BIT    4,(IX+HW_BYTE) ;Is it a 1 or 2 sided disk
        JR      Z,RW_SKIP_SIDES ;If 1 sided then skip side swap
        IN      A,(SELECT) ;Get hardware selection
        BIT    4,A ;Port Hardware for side selection (1=A, 0=B)
        JR      Z,RW_B_SIDE
        LD      HL,HEAD0_MSG ;Announce side 0 (or A)
        CALL    PMSG
        POP     HL
        RET
RW_B_SIDE:
        LD      HL,HEAD1_MSG ;Announce side 1 (or B)
        CALL    PMSG

```



```

        POP    HL
        RET
RW_SKIP_SIDES:
        LD     HL,H_MSG      ;If SS disk, no need for a HEAD# message
        CALL  PMSG
        POP    HL
        RET
;
;
;
; General routine to show the current TRACK location of the active disk head.
; Will display track only. The format is 'At TRACK xxH.'
; Note: NO CR,LF at start of string
;
SHOW_T_LOC:
        PUSH  HL
        LD    HL,ATTRK
        CALL  PMSG
        IN    A,(TRACK)      ;Say what track we are now on.
        CALL  PACC           ;Display current track number
        LD    HL,H_MSG
        CALL  PMSG
        POP   HL
        RET
;
;
; General routine to show the current TRACK & SIDE location of the active
; disk head. The format is 'At TRACK xxH Side A/B.'
; Note: NO CR,LF at start of string
;
SHOW_TS_LOC:
        PUSH  HL
        LD    HL,ATTRK
        CALL  PMSG
        IN    A,(TRACK)      ;Say what track we are now on.
        CALL  PACC           ;Display current track number
        JP    SHOW_SIDES
;
;
; General routine to print out the error status returned by the 1791/5 chip
; after seeks, sector reads etc.
; Note: will end with CRLF and error bits in [A]
;
SHOW_ERRORS:
        LD    A,(ERRORS_FLAG)
        INC  A
        LD    (ERRORS_FLAG),A      ;Assuming we have less than 256 errors!

        LD    A,(CHIP_CMDSV) ;What was the last command to the chip
        CP    A,RDCMD91      ;Was it a read sector 1791 CMD?
        JR    NZ,NOT_RDCMD91
        LD    HL,SEC_READ_ERROR
        JR    DONE_ERRS
NOT_RDCMD91:
        CP    A,RDCMD95      ;Was it a read sector 1795 CMD?
        JR    NZ,NOT_RDCMD95
        LD    HL,SEC_READ_ERROR
        JR    DONE_ERRS
NOT_RDCMD95:
        CP    A,SKNCMD       ;Was it a seek no verify cmd
        JR    NZ,NOT_SKNCMD
        LD    HL,SEEKNV_ERROR
        JR    DONE_ERRS
NOT_SKNCMD:
        CP    A,RSCMD        ;Was it a restore CMD
        JR    NZ,NOT_RSCMD
        LD    HL,RSCMD_ERROR
        JR    DONE_ERRS
NOT_RSCMD:

```

```

    CP    A,RDACMD      ;Was it a read track address command
    JR    NZ,NOT_RDACMD
    LD    HL,SEC_ID_ERROR
    JR    DONE_ERRS
NOT_RDACMD:
    CP    A,SKCMD      ;Was it seek cmd with verify
    JR    NZ,NOT_SKCMD
    LD    HL,SKCMD_ERROR
    JR    DONE_ERRS
NOT_SKCMD:
    CP    A,RDTCMD     ;Was it a read track command
    JR    NZ,NOT_RDTCMD
    LD    HL,RDTCMD_ERROR
    JR    DONE_ERRS
NOT_RDTCMD:
    CP    A,WRTCMD     ;Was it a write track cmd
    JR    NZ,NOT_WRTCMD
    LD    HL,WRTCMD_ERROR
    JR    DONE_ERRS
NOT_WRTCMD:
    CP    A,WRCMD91    ;Was it a sector write cmd 1791
    JR    NZ,NOT_WRCMD91
    LD    HL,WRCMD_ERROR
    JR    DONE_ERRS
NOT_WRCMD91:
    CP    A,WRCMD95    ;Was it a sector write cmd 1795
    JR    NZ,NOT_WRCMD95
    LD    HL,WRCMD_ERROR
    JR    DONE_ERRS
NOT_WRCMD95:
    CP    A,STEPIN     ;Was it a step-in CMD
    JR    NZ,NOT_STEPIN
    LD    HL,STEPIN_ERROR
    JR    DONE_ERRS
NOT_STEPIN:
    LD    HL,UNKNOWN_ERROR
DONE_ERRS:
    CALL PMSG
    LD    A,(ERSTAT)
    CALL ZBITS        ;Drop in bit pattern
    LD    HL,CLOSE_BRACKET
    CALL PMSG
    CALL ZCRLF        ;Always end with CRLF
    LD    A,(ERSTAT)  ;Return with error in A
    RET
;
;
; This routine simply checks the written and re-read sectors in memory are the same.
; Z if all OK. NZ, if error(s)
;
CMP_BUFFERS:
    LD    HL,FBUFFER
    LD    DE,FBUFFER2
    LD    C,0         ;Will flag errors
    LD    A,(IX+SIZE) ;128=0,256=1,512=2 or 1024=3 byte sector size
    LD    B,NBYTES
    OR    A,A         ;Block size count
    JR    Z,C128256   ;Do 128 byte compare
    LD    B,0
    CP    A,1
    JR    Z,C128256   ;Do 256 byte read
    CP    A,2
    JR    Z,C512      ;Do 512 byte read
    CALL CHECK_BLOCK  ;Must be 1024 byte sectors so 4X256
    CALL CHECK_BLOCK
C512:  CALL CHECK_BLOCK ;256X2 bytes
C128256:CALL CHECK_BLOCK
    XOR  A,A
    CP   A,C

```

```

        RET
CHECK_BLOCK:
    LD     A, (DE)
    CP     A, (HL)           ;Do they match
    CALL  NZ, MIS_MATCH
    INC   HL
    INC   DE
    DJNZ  CHECK_BLOCK      ;Decrease count
    RET
MIS_MATCH:
    LD     C, 1             ;Flag for error
    RET
;
; Fill memory buffer with a byte [C]. Is be used for sector write and CPM86
; disk initialization
; Return with [HL] pointing to end of sector buffer +1
FILL_BUFFER:
    LD     HL, FBUFFER
    LD     A, (IX+SIZE)     ;128=0,256=1,512=2 or 1024=3 byte sector size
    LD     B, NBYTES
    OR     A, A             ;Block size count
    JR     Z, B128256      ;Do 128 byte compare
    LD     B, 0
    CP     A, 1
    JR     Z, B128256      ;Do 256 byte read
    CP     A, 2
    JR     Z, B512         ;Do 512 byte read
    CALL  BUILD_BLOCK      ;Must be 1024 byte sectors so 4X256
    CALL  BUILD_BLOCK
B512:   CALL  BUILD_BLOCK  ;256X2 bytes
B128256:CALL  BUILD_BLOCK
    RET
BUILD_BLOCK:
    LD     (HL), C         ;Drop in the fill character
    INC   HL
    DJNZ  BUILD_BLOCK      ;Decrease count
    RET
;
;
;
;
;----- BUILD TRACK IMAGE IN RAM -----
; Assumes      IX points to the table or relevent Disk paramaters and
;              (TADDR) points to the buffer in RAM to build the track.
;              Requires valid (TRK), (SCTR) & (SIDE)
;              For maximum flexibility I have used seperate routines
;              for Single and Double density formats. This routine is
;              a very sensitive. Don't change unless you know what
;              you are doing.
;
;              Build in memory (@5000H)a complete SD Track.
;
BUILD_TRACK:
    PUSH  IY               ;Save the "main" IY pointer to TRACK, SIDE, SECTOR
    LD     A, (IY+TRK)
    LD     (F_TRK), A      ;store trk & side numbers locally
    LD     A, (IY+SIDE)
    LD     (F_SIDE), A

    LD     H, (IX+SKEW1)   ;IY Now points to the table containing the order of
    LD     L, (IX+SKEW)    ;the sector numbers on a track (usually 1,2,3,4...
    PUSH  HL               ;but not always
    POP   IY              ;HL->IY

    LD     HL, (TADDR)     ;This will be FBUFFER where track ins built in RAM
    BIT   6, (IX+HW_BYTE) ;Will build a different track image for SD or DD disks
    JP    NZ, DD_TRACKS   ;Seperate sector image for DD tracks

```

```

SD_TRACKS:
    CALL    SD_TRK_HEADER          ;Drop in header and Index mark BEFORE first sector

    LD      D, (IY+0)              ;sector# in D, usually 1 at the start
    LD      E, (IX+NSCTRS)        ;Total Sectors/side

NEXT_SEC:
    CALL    SD_BUILD_SEC          ;<<< Build a sector >>>>
    INC     IY                    ;point to next sector number in sec skew table
    LD      D, (IY+0)            ;store sector # in D
    DEC     E
    JR      NZ, NEXT_SEC          ;All sectors are in RAM

    CALL    SD_TRK_END            ;Now need to flush out track to end
    POP     IY                    ;Get back the main IY pointer
    RET

;
;    Build in memory (@5000H) a complete Single Density Track.
;
SD_TRK_HEADER:                    ;Lay down the track header before the 1st sector
    LD      A, (IX+GAP_FILL_CHAR) ;Do not alter [D]= Sec# or [E]= Sec/side
    LD      B, (IX+HEADR)        ;Header has 40 (FF's)
    CALL    DROP                  ;drop it at the end of the growing image (Count in B)
    XOR     A,A                  ;Now 6 0's
    LD      B, 6
    CALL    DROP
    LD      A, 0FCH              ;Index ID mark
    LD      (HL), A              ;drop into image
    INC     HL
    LD      A, (IX+GAP_FILL_CHAR) ;Now 26 (FF's)
    LD      B, 26                ;Header has the count of fill characters required
    CALL    DROP                  ;drop it at the end of the growing image (Count in B)
    LD      (INDEX_MARK), HL     ;pointer+1 to end of track header
    RET                          ;return with [HL] pointing to first sector byte

;
SD_BUILD_SEC:                    ;Lay down a sector at current [HL]. Do not alter [D] or [E]
    XOR     A,A
    LD      B, (IX+GAP1)         ;<---- (eg. 6,0's for IMB 3740, 8")
    CALL    DROP
    LD      A, 0FEH              ;Sector ID Address mark
    LD      (HL), A              ;drop it in the growing image
    INC     HL
    LD      A, (F_TRK)           ;Drop in the track #
    LD      (HL), A
    INC     HL
    LD      A, (F_SIDE)          ;Side#, 0 for A side, 02H for B side
    OR      A,A
    LD      A, 0
    JR      Z, BLD_ASIDE
    LD      A, 1

BLD_ASIDE:
    LD      (HL), A              ;0 here for A side, 1 for B side
    INC     HL
    LD      (HL), D              ;Drop in sector #
    INC     HL
    LD      A, (IX+SIZE)         ;128=0, 256=1, 512=2, 1024=3
    LD      (HL), A              ;drop in sector length byte
    INC     HL
    LD      A, 0F7H              ;Dropping in a 0F7 will cause the 179x
    LD      (HL), A              ;to write in the 2 CRC bytes
    INC     HL
    LD      A, (IX+GAP_FILL_CHAR)
    LD      B, (IX+GAP2)        ;<---- (eg. 11, FF's for IMB 3740, 8")
    CALL    DROP
    XOR     A,A
    LD      B, (IX+GAP1)         ;<---- (eg. 6,0's for IMB 3740, 8")
    CALL    DROP
    LD      A, 0FBH              ;Data address mark for 1791/5
    LD      (HL), A              ;to write in the 2 CRC bytes
    INC     HL

```

```

LD      A,D
CP      A,1                ;If first sector then store data marker
JR      NZ,SDATA_FIELD
LD      (S_DATA_MARK),HL  ;Pointer to start of sector data area
SDATA_FIELD:
LD      A,(IX+SIZE)       ;Now write in the sector data field itself
LD      B,NBYTES          ;128,256,512 or 1024 byte sector size
LD      C,1                ;1 loop of 128 bytes in WR_DATA_FIELD below
OR      A,A
JR      Z,SD_DATA_FIELD   ;Do 128 byte write (B=128)
LD      B,0                ;Need 256 bytes for the rest of possible sectors
LD      C,1                ;One loop
CP      A,1
JR      Z,SD_DATA_FIELD   ;Do 256 byte write
LD      C,2                ;2 loops of 256 bytes in WRITE_DATA_FIELD
CP      A,2
JR      Z,SD_DATA_FIELD   ;Do 512 byte read
LD      C,4                ;(must be 3) so 1024 byte sector
SD_DATA_FIELD:
LD      A,(IX+DATA_FILL_CHAR) ;get the sector fill character (usually E5)
SDF0:   CALL DROP
DEC     C
JR      NZ,SDF0           ;Decrease [C] to 0
LD      A,D
CP      A,1                ;If first sector then store image mark
JR      NZ,NOT_FIRST
LD      (E_DATA_MARK),HL  ;For first sector will diaplay data late
NOT_FIRST:
LD      A,0F7H            ;Dropping in a 0F7 will cause the 1791/5
LD      (HL),A            ;to write in the 2 CRC bytes
INC     HL
LD      A,(IX+GAP_FILL_CHAR)
LD      B,(IX+GAP3)       ;<---- (eg. 27,FF's for IMB 3740, 8")
CALL    DROP
LD      A,D
CP      A,1                ;If first sector then store image mark
RET     NZ
LD      (E_SEC_MARK),HL   ;For first sector will diaplay data later
RET

;
SD_TRK_END
LD      (S_GAP4_MARK),HL  ;Mark beginning of end of track field (GAP4)
LD      A,(IX+GAP_FILL_CHAR)
LD      B,(IX+GAP4)       ;<---- (eg 247,FF's for IMB 3740, 8")
LD      C,(IX+GAP4R)      ;Times to repeat DROP
SD_TRK1:CALL DROP
DEC     C
JR      NZ,SD_TRK1
LD      (E_GAP4_MARK),HL  ;Mark end of Track
RET

;
DROP:   LD      (HL),A     ;DATA block loader
INC     HL                ;B= byte count,HL pointer
DJNZ   DROP              ;A = value to drop into image. Count in B
RET

;
;
;
;   Build in memory (@5000H)a complete Double Density Track.
;
DD_TRACKS:
CALL    DD_TRK_HEADER    ;Same thing for DD Trdcks. Extra stuff req.
                        ;Drop in header and Index mark BEFORE first sector

LD      D,(IY+0)         ;sector# in D, usually 1 at the start
LD      E,(IX+NSCTRS)    ;Total Sectors/side
DD_NEXT_SEC:
CALL    DD_BUILD_SEC     ;<<< Build a DD sector >>>>
INC     IY                ;point to next sector number in sec skew table
LD      D,(IY+0)         ;store sector # in D

```

```

DEC      E
JR       NZ,DD_NEXT_SEC      ;All sectors are in RAM

CALL    DD_TRK_END          ;Now need to flush out track to end
POP     IY                  ;get back the main IY pointer
RET

;
;
DD_TRK_HEADER:              ;Lay down the track header before the 1st sector
LD      A,(IX+GAP_FILL_CHAR) ;Do not alter [D]= Sec# or [E]= Sec/side
LD      B,(IX+HEADR)        ;Header has 80 (4E's)
CALL    DROP                ;drop it at the end of the growing image (Count in B)
XOR     A,A                 ;Now 12 0's
LD      B,12
CALL    DROP
LD      A,0F6H              ;3 of F6's
LD      B,3
CALL    DROP
LD      A,0FCH              ;Index ID mark (FC)
LD      (HL),A              ;drop into image
INC     HL
LD      A,(IX+GAP_FILL_CHAR) ;Now 50 (4Es)
LD      B,50
CALL    DROP                ;Header has the count of fill characters required
LD      (INDEX_MARK),HL     ;drop it at the end of the growing image (Count in B)
RET     ;pointer+1 to end of track header
;return with [HL] pointing to first sector byte

;
DD_BUILD_SEC:                ;Lay down a sector at current [HL]. Do not alter [D] or [E]
XOR     A,A
LD      B,(IX+GAP1)         ;<---- (eg. 12,0's for IBM System 34 Format)
CALL    DROP
LD      A,0F5H              ;Special DD bytes
LD      B,3
CALL    DROP
LD      A,0FEH              ;Sector ID Address mark
LD      (HL),A              ;drop it in the growing image
INC     HL
LD      A,(F_TRK)           ;Drop in the track #
LD      (HL),A
INC     HL
LD      A,(F_SIDE)          ;Side#, 0 for A side, 02H for B side
OR      A,A
LD      A,0
JR      Z,DBLD_ASIDE
LD      A,1                  ;1 for side B

DBLD_ASIDE:
LD      (HL),A              ;0 here for A side, 1 for B side
INC     HL
LD      (HL),D              ;Drop in sector #
INC     HL
LD      A,(IX+SIZE)         ;128=0,256=1,512=2, 1024=3
LD      (HL),A              ;drop in sector length byte
INC     HL
LD      A,0F7H              ;Dropping in a 0F7 will cause the 179x
LD      (HL),A              ;to write in the 2 CRC bytes
INC     HL
LD      A,(IX+GAP_FILL_CHAR)
LD      B,(IX+GAP2)         ;<----(eg 22,4E's for IBM System 34 Format)
CALL    DROP
XOR     A,A
LD      B,(IX+GAP1)         ;<---- (eg 8,0's for IBM System 34 Format)
CALL    DROP
LD      A,0F5H              ;Special DD bytes
LD      B,3
CALL    DROP
LD      A,0FBH              ;Data address mark for 1791/5
LD      (HL),A              ;to write in the 2 CRC bytes
INC     HL
LD      A,D                  ;Get sec #

```

```

CP      A,1                ;If first sector then store data marker
JR      NZ,DDATA_FIELD
LD      (S_DATA_MARK),HL  ;Pointer to start of sector data area
DDATA_FIELD:
LD      A,(IX+SIZE)       ;Now write in the sector data field itself
LD      B,NBYTES         ;128,256,512 or 1024 byte sector size
LD      C,1               ;1 loop of 128 bytes in WR_DATA_FIELD below
OR      A,A
JR      Z,DD_DATA_FIELD   ;Do 128 byte write (B=128)
LD      B,0               ;Need 256 bytes for the rest of possible sectors
LD      C,1               ;One loop
CP      A,1
JR      Z,DD_DATA_FIELD   ;Do 256 byte write
LD      C,2               ;2 loops of 256 bytes in WRITE_DATA_FIELD
CP      A,2
JR      Z,DD_DATA_FIELD   ;Do 512 byte read
LD      C,4               ;(must be 3) so 1024 byte sector
DD_DATA_FIELD:
LD      A,(IX+DATA_FILL_CHAR) ;get the sector fill character (usually E5)
DDF0:   CALL DROP
DEC     C
JR      NZ,DDF0           ;Decrease [C] to 0
LD      A,D               ;Get back sec #
CP      A,1               ;If first sector then store image mark
JR      NZ,DNOT_FIRST
LD      (E_DATA_MARK),HL  ;For first sector will diaplay data late
DNOT_FIRST:
LD      A,0F7H           ;Dropping in a 0F7 will cause the 1791/5
LD      (HL),A           ;to write in the 2 CRC bytes
INC     HL
LD      A,(IX+GAP_FILL_CHAR)
LD      B,(IX+GAP3)      ;<---- (54,4E's for IBM System 34 Format)
CALL   DROP
LD      A,D
CP      A,1               ;If first sector then store image mark
RET     NZ
LD      (E_SEC_MARK),HL  ;For first sector will diaplay data later
RET

;
DD_TRK_END
LD      (S_GAP4_MARK),HL  ;Mark beginning of end of track field (GAP4)
LD      A,(IX+GAP_FILL_CHAR)
LD      B,(IX+GAP4)      ;<---- (eg 598 4E's for IBM System 34 Format)
LD      C,(IX+GAP4R)     ;Times to repeat DROP
DD_TRK1:CALL DROP
DEC     C
JR      NZ,DD_TRK1
LD      (E_GAP4_MARK),HL ;Mark end of Double density Track
RET

;
; This routine initalized the first sector of a CPM86 5" Disk.
; For DDDS CPM86 disks the last byte of the first sector has to be a 01.
; (For a DDSS CPM86 disk BTW it has to be 00)
;
INIT_CPM86:
LD      HL,INITCPM_MSG ;Say Initalizing disk for CPM86
CALL   PMSG
XOR     A,A               ;Setup for the BIOS below
LD      (IY+TRK),A       ;Track 0
LD      (IY+SIDE),A     ;Side A
INC     A
LD      (IY+SCTR),A     ;Sector 1
CALL   SET_SIDE         ;Make sure IX+HW_BYTE is set for correct side
CALL   START_DRIVE_1   ;Select the drive in hardware
CALL   HOME
CALL   NZ,SHOW_ERRORS ;print out errors if any

LD      HL,FBUFFER      ;Sector data to be written is here
LD      (TADDR),HL     ;Setup DMA address for BIOS

```

```

LD      C,0E5H      ;Data fill character (for CPM86 always 0E5H)
CALL    FILL_BUFFER ;Fill buffer with character E5's
DEC     HL          ;Backup to last data byte
LD      A,01
LD      (HL),A      ;drop in the 01 flag at end.

CALL    WRITE_SECTOR ;Write sector back to disk
CALL    NZ,SHOW_ERRORS ;If any, show error flags
XOR     A,A         ;Return Z so no further special commands will
RET     ;be picked up.

;
;
; Setup the "current drive" hardware selection bits for (A:,B:,C: or D:)
; Store bits in memory location (DRIVE_1)
; Return Z flag if OK, NZ & [A]= 0FFH if error, NZ and ESC if abort
;
SELECT_DR_1:
LD      HL,GETDRV_MSG ;Ask which drive is the current drive
CALL    PMSG
CALL    GETCMD       ;Get the input option
CP      A,ESC        ;Abort if ESC character
JP      Z,ABORT_CMD
CP      A,'A'
JR      NZ,NOT_A1
LD      A,1
JR      GOT_DRIVE1
NOT_A1: CP      A,'B'
JR      NZ,NOT_B1
LD      A,2
JR      GOT_DRIVE1
NOT_B1: CP      A,'C'
JR      NZ,NOT_C1
LD      A,4
JR      GOT_DRIVE1
NOT_C1: CP      A,'D'
JR      NZ,INVALID_DR
LD      A,8
GOT_DRIVE1:
LD      (DRIVE_1),A
XOR     A,A
RET

;
ABORT_CMD:
XOR     A,A          ;Abort with z flag but ESC in A
DEC     A            ;Make NZ
LD      A,ESC
RET

;
INVALID_DR:
XOR     A,A          ;Abort with z flag but FF in A
DEC     A            ;Make NZ
RET

;
;
; Setup the "second drive" hardware selection bits for (A:,B:,C: or D:)
; Store bits in memory location (DRIVE_2)
; Return Z flag if OK, NZ & [A]= 0FFH if error, NZ and ESC if abort
;
SELECT_DR_2:
LD      HL,GETDRV_MSG ;Ask which drive is the current drive
CALL    PMSG
CALL    GETCMD       ;Get the input option
CP      A,ESC        ;Abort if ESC character
JP      Z,ABORT_CMD
CP      A,'A'
JR      NZ,NOT_A2
LD      A,1
JR      GOT_DRIVE2

```



```

NOT_A2: CP      A, 'B'
        JR      NZ, NOT_B2
        LD      A, 2
        JR      GOT_DRIVE2
NOT_B2: CP      A, 'C'
        JR      NZ, NOT_C2
        LD      A, 4
        JR      GOT_DRIVE2
NOT_C2: CP      A, 'D'
        JR      NZ, INVALID_DR
        LD      A, 8
GOT_DRIVE2:
        LD      (DRIVE_2), A
        XOR     A, A
        RET

```

```
;-----Select the appropriate disk parameter table -----
```

```
; Return Z flag if OK, NZ & [A]= 0FFH if error, NZ and ESC if abort
; [IX] to Drive parameter table
```

```
;
SELECT_IX:
        CALL    ZCRLF
SELECT_IX1:
        LD      HL, GETSIZE_MSG ;Ask if 5" or 8"
        CALL    PMSG
        CALL    GETCMD          ;get the input option
        CP      A, ESC          ;Abort if ESC character
        JP      Z, ABORT_CMD
        CP      A, '5'
        JR      Z, GET_TABLE_5
        CP      A, '8'
        JR      Z, GET_TABLE_8
BAD_CMD: LD      HL, BADCMD
        CALL    PMSG
        XOR     A, A           ;NZ Flag to indicate no selection
        DEC     A
        RET

```

```
GET_TABLE_8:
        LD      HL, DISK_8_FORMATS ;Pointer for 8" disk table
        CALL    PMSG
        CALL    GETCMD          ;SELECT A DISK PARAMATER TABLE
        CP      A, ESC          ;Abort if ESC character
        JP      Z, ABORT_CMD
        CP      A, 'A'          ;128 bytes/sec
        LD      IX, STDSDT      ;Set to 128 byte sectors 8" SD IBM Disk
        RET     Z
        CP      A, 'B'
        LD      IX, STDDDT      ;Set to 128 byte sectors 8" DD format
        RET     Z
        CP      A, 'C'
        LD      IX, DDT256      ;Set to 256 byte sectors IBM DDDS disk
        RET     Z
        CP      A, 'D'
        LD      IX, DDT512      ;Set for 512 byte sectors 8" DDDS disk
        RET     Z
        CP      A, 'E'
        LD      IX, DDT1K       ;Set for 1024 byte sectors 8" DDSS disk
        RET     Z
        CP      A, 'F'
        LD      IX, DDT1K2      ;Set for 1024 byte sectors 8" DDDS disk
        RET     Z
        JP      BAD_CMD

```

```
GET_TABLE_5:
        LD      HL, DISK_5_FORMATS ;Pointer for 5" disk table
        CALL    PMSG
        CALL    GETCMD          ;SELECT A DISK PARAMATER TABLE

```

```

CP      A,ESC          ;Abort if ESC character
JP      Z,ABORT_CMD
CP      A,'A'
LD      IX,MINCPM      ;Set for 512 byte CPM-86 5" DD disk
RET     Z
CP      A,'B'
LD      IX,MINSDT      ;Set for 128 byte 5" SD disk
RET     Z
CP      A,'C'
LD      IX,MINDDT      ;Set for 128 byte 5" DD disk
RET     Z
CP      A,'D'
LD      IX,DEC         ;Set for 512 byte 5" DEC-VT180 DD disk
RET     Z
CP      A,'E'
LD      IX,TOSHIBA     ;Set for TOSHIBA DD disk
RET     Z
CP      A,'F'
LD      IX,CDOS        ;Set for CDOS SD disk
RET     Z
CP      A,'G'
LD      IX,CDOSDD      ;Set for CDOS DD disk
RET     Z
CP      A,'H'
LD      IX,EPSON       ;Set for EPSON QX-10 DD disk
RET     Z
CP      A,'I'
LD      IX,MORROW      ;Set for MORROW DD disk
RET     Z
CP      A,'J'
LD      IX,ZENITH      ;Set for ZENITH Z-100 DD disk
RET     Z
CP      A,'K'
LD      IX,SUPER       ;Set for SUPERBRAIN DD disk
RET     Z
CP      A,'L'
LD      IX,MSDOS       ;Set for MSDOS 1.1 DD disk
RET     Z
CP      A,'M'
LD      IX,MSDOS2      ;Set for MSDOS 2.2 DD disk
RET     Z
CP      A,'N'
LD      IX,TRS80       ;Set for TRS80 III DD disk
RET     Z
JP      BAD_CMD

;
;
;-----
; Describe on one line the type of disk format selected
; Assumes a valid IX pointer to relevent disk table
;
SHOW_HW_TITLE:
LD      HL,DISK_INFO    ;CR,LF,'Current drive:--'
CALL    PMSG
PUSH    IX              ;IX->HL
POP     HL
LD      DE,TITLE       ;add in offset
ADD     HL,DE           ;HL now points to the title entry of the selected disk
CALL    PMSG           ;Show the title line
RET

;
;-----
; Decode the IOBYTE byte and print out hardware disk parameters on one line
; Assumes a valid IX pointer to relevent disk table
;
SHOW_HW_BYTE:
LD      HL,DISK_INFO    ;CR,LF,'Drive '
CALL    PMSG
LD      A,(IOBYTE)     ;Obtain the current drive letter (A;,B:....)

```

```

AND      A,03H
CP       A,1
JR       NZ,NOT_A_DR
LD       A,'A'
JR       DRV_DONE
NOT_A_DR:
CP       A,2
JR       NZ,NOT_B_DR
LD       A,'B'
JR       DRV_DONE
NOT_B_DR
CP       A,4
JR       NZ,NOT_C_DR
LD       A,'C'
JR       DRV_DONE
NOT_C_DR
LD       A,'D'
DRV_DONE:
LD       C,A                ;Drop it on to the CRT/LCD
CALL    CO
LD       HL,DISK_INFO1
CALL    PMSG
LD       A,(IOBYTE)
CALL    ZBITS                ;Print out bit pattern in IOBYTE for hardware
LD       HL,DISK_INFO2
CALL    PMSG
BIT     5,(IX+HW_BYTE)      ;Say whether 8" or 5"
JR       Z,INCH8
LD       C,'5'
JR       NEXT2
INCH8:  LD       C,'8'
NEXT2:  CALL    CO
BIT     4,(IX+HW_BYTE)      ;Say whether Single or Double sided disk
JR       Z,S_SIDE
LD       HL,DISK_INFO3
JR       NEXT3
S_SIDE: LD       HL,DISK_INFO4
NEXT3:  CALL    PMSG
BIT     6,(IX+HW_BYTE)      ;Say whether SD or DD disk
JR       Z,DD_DSK
LD       HL,DISK_INFO5
JR       NEXT4
DD_DSK: LD       HL,DISK_INFO6
NEXT4:  CALL    PMSG
        RET
;
;
;-----
; Display the disk paramater table pointed to by IX
;
SHOW_IX_TABLE:
LD       HL,DISK_INFO7      ;Print sectors/track
CALL    PMSG
LD       A,(IX+NSCTRS)
CALL    PACC
LD       HL,DISK_INFO8      ;Print tracks/side
CALL    PMSG
LD       A,(IX+NTRKS)
CALL    PACC
LD       HL,DISK_INFO9      ;Sector Size
LD       A,(IX+SIZE)
OR       A,A
JR       NZ,NOT_128
LD       HL,DISK_INFO10     ;128 Bytes/sector
JR       DONE_IX_TABLE
NOT_128:CP      1
JR       NZ,NOT_256
LD       HL,DISK_INFO11     ;256 Bytes/Sector
JR       DONE_IX_TABLE

```

```

NOT_256:CP      2
              JR      NZ,NOT_512
              LD      HL,DISK_INFO12      ;512 Bytes/sector
              JR      DONE_IX_TABLE
NOT_512:LD      HL,DISK_INFO13      ;1024 Bytes/Sector
DONE_IX_TABLE:
              CALL    PMSG
              RET

;
;-----
; Display the current track ID Field.
; NOTE this is NOT the current sector read. It's whatever sector
; the head happens to be over at the time.
;
SHOW_ID:
              LD      HL,IDMSG0          ;'Track/Side/Sec/Size/(CRC) TRACK ID field='
              CALL    PMSG
              LD      HL,IDSV            ;Point to where data was saved
              LD      B,4                ;4+2 bytes in ID
IDLOOP1:LD     A,(HL)
              CALL    PACC                ;Print [A] in ASCII
              INC     HL
              DEC     B
              JR      NZ,IDLOOP1
              LD      C,' '
              CALL    CO
              LD      C,'('
              CALL    CO
              LD      B,2
IDLOOP5:LD     A,(HL)
              CALL    PACC                ;Print CRC value
              INC     HL
              DEC     B
              JR      NZ,IDLOOP5
              LD      C,')'
              CALL    CO
              RET

;
;
; Display Bits of Versafloppy II Status Port
; Return with bits in [A]
;
SHOW_STAT_BITS:
              LD      HL,STATUS_179x
              CALL    PMSG
              IN      A,(STATUS)
              CALL    ZBITS                ;display bit pattern of [A]
              RET                          ;Will return with error in [A]

;
; Check to see if disk is write protected
; Z if not protected, NZ if write protected
;
CHECK_WP:
              IN      A,(STATUS)          ;Remember data is inverted
              BIT     6,A
              RET     Z
              LD      HL,DISK_WP_MSG      ;This disk is currently write protected
              CALL    PMSG
              XOR     A,A
              DEC     A
              RET                          ;Will return with error in [A]

;
;
;----- DISPLAY SECTOR DATA CONTENTS -----
;Display the Sector read from disk to RAM location at (TADDR)
SEC_DISPLAY:
              LD      A,(CRTDISP)        ;Do we have the more detail display option on
              OR      A,A

```

```

RET      Z          ;If not just return
PUSH    AF
PUSH    BC
PUSH    DE
PUSH    HL
LD      HL,SEC_MSG
CALL    PMSG
LD      HL,(TADDR)  ;Move Transfer address into HL
LD      A,(IX+SIZE) ;Get sector size
OR      A,A
LD      DE,NBYTES  ;128 byte sectors
JR      Z,DISP1
CP      A,1
LD      DE,NBYTES*2 ;256 byte sectors
JR      Z,DISP1
CP      A,2
LD      DE,NBYTES*4 ;512 byte sectors
JR      Z,DISP1
LD      DE,NBYTES*8 ;Must be 1024 byte sectors

DISP1: LD      A,E
AND     A,00111111B ;64 characters /line
JR      NZ,NOCR
CALL    ZCRLF
NOCR:  LD      A,(HL)
AND     A,7FH
CP      ' '          ;FILTER OUT CONTROL CHARACTERS'
JR      NC,T33
T22:   LD      A, '.'
T33:   CP      A,07CH
JR      NC,T22
LD      C,A
CALL    CO
INC     HL
DEC     DE          ;Decrease DE count by 1
LD      A,E
OR      A,A          ;Not XX00 at least
JR      NZ,DISP1
OR      A,D
JR      NZ,DISP1    ;Not 0000
CALL    ZCRLF
NODISP: POP     HL
POP     DE
POP     BC
POP     AF
RET

;
;----- SHOW DETAILS OF RAW SECTOR IMAGE FOR TRACK WRITE -----
FORMAT_INFO:
PUSH    AF          ;Save everything since track formating uses all
PUSH    BC
PUSH    DE
PUSH    HL
LD      HL,FORM_TRK ;Formatting track...
CALL    PMSG
LD      A,(IY+TRK)
CALL    PACC

;
BIT     4,(IX+HW_BYTE) ;Is it a 1 or 2 sided disk
JR      Z,NO_SIDESF   ;If 1 sided then skip sides info
LD      A,(IY+SIDE)
OR      A,A          ;If 0 we have done A side now do B
JR      NZ,B_SIDEF   ;is B side
LD      HL,HEADO_MSG
JR      SEC_INFO1
B_SIDEF:LD HL,HEAD1_MSG
JR      SEC_INFO1
NO_SIDESF:
LD      HL,H_MSG     ;H.

```

```

SEC_INFO1:
    CALL    PMSG
    LD      A,(CRTDISP)    ;Do we have the more detail display option on
    OR      A,A
    JP      Z,NO_TDISPLAY ;If Z do not list the sector contents.

                                ;OK we are going to step along the image one
                                ;section at a time. Key locations were stored
                                ;previously in the SCTRIM: routine.

    LD      HL,TRACK_MSG
    CALL    PMSG
    LD      DE,FBUFFER    ;Move Start of track image into DE
    PUSH    DE            ;save it
    LD      HL,(INDEX_MARK) ;End+1 of track header
    DEC     HL
    SBC     HL,DE
    EX      DE,HL        ;Count now in DE
    POP     HL            ;DE on stack to HL
HDRX:     LD      A,(HL)
    CALL    PACC          ;Print out HEX values of HEADER Field
    INC     HL
    DEC     DE            ;Decrease DE count by 1
    LD      A,E
    OR      A,A            ;Not XX00 at least
    JR      NZ,HDRX
    OR      A,D
    JR      NZ,HDRX        ;Not 0000

                                ;Now display the FIRST sector ID field area
    LD      HL,SECTOR_MSG
    CALL    PMSG
    LD      DE,(INDEX_MARK) ;Move Start of sector image into DE
    PUSH    DE            ;save it
    LD      HL,(S_DATA_MARK) ;End+1 of sec ID field (allow the +1 to capture the FB byte)
    SBC     HL,DE
    EX      DE,HL        ;Count now in DE
    POP     HL            ;DE on stack to HL
HSECX:   LD      A,(HL)
    CALL    PACC          ;Print out HEX values of 1st Sector ID Field
    INC     HL
    DEC     DE            ;Decrease DE count by 1
    LD      A,E
    OR      A,A            ;Not XX00 at least
    JR      NZ,HSECX
    OR      A,D
    JR      NZ,HSECX        ;Not 0000

                                ;Now display the FIRST sector DATA field area
    LD      HL,SEC_DATA_MSG
    CALL    PMSG
    LD      DE,(S_DATA_MARK) ;Move Start of sector image into DE
    PUSH    DE            ;save it
    LD      HL,(E_DATA_MARK) ;End+1 of DATA field
    DEC     HL
    SBC     HL,DE
    EX      DE,HL        ;Count now in DE
    POP     HL            ;DE on stack to HL
HDATAX:  LD      A,(HL)
    CALL    PACC          ;Print out HEX values of 1st Sector DATA Field
    INC     HL
    DEC     DE            ;Decrease DE count by 1
    LD      A,E
    OR      A,A            ;Not XX00 at least
    JR      NZ,HDATAX
    OR      A,D
    JR      NZ,HDATAX        ;Not 0000

                                ;Now display the FIRST sector GAP3 field area
    LD      HL,SEC_GAP3_MSG

```

```

CALL    PMSG
LD      DE, (E_DATA_MARK)      ;Move Start of sector image into DE
PUSH    DE                      ;save it
LD      HL, (E_SEC_MARK)      ;End+1 of GAP3 field
DEC     HL
SBC     HL, DE
EX      DE, HL                ;Count now in DE
POP     HL                      ;DE on stack to HL
HGAP3X: LD    A, (HL)
CALL    PACC                    ;Print out HEX values of 1st Sector GAP3 Field
INC     HL
DEC     DE                      ;Decrease DE count by 1
LD      A, E
OR      A, A                    ;Not XX00 at least
JR      NZ, HGAP3X
OR      A, D
JR      NZ, HGAP3X            ;Not 0000

                                ;Now display the Track GAP4 field area
LD      HL, SEC_GAP4_MSG
CALL    PMSG
LD      DE, (S_GAP4_MARK)      ;Start of sector GAP4 into DE
PUSH    DE                      ;save it
LD      HL, (E_GAP4_MARK)      ;End+1 of GAP3 field
DEC     HL
SBC     HL, DE
EX      DE, HL                ;Count now in DE
POP     HL                      ;DE on stack to HL
HGAP4X: LD    A, (HL)
CALL    PACC                    ;Print out HEX values of 1st Sector GAP3 Field
INC     HL
DEC     DE                      ;Decrease DE count by 1
LD      A, E
OR      A, A                    ;Not XX00 at least
JR      NZ, HGAP4X
OR      A, D
JR      NZ, HGAP4X            ;Not 0000
CALL    ZCRLF

;
NO_TDISPLAY:
POP     HL
POP     DE
POP     BC
POP     AF
RET

;
;
;===== SUPPORT ROUTINES =====
;
PMSG:   LD      A, (HL)        ;PRINT MESSAGE STRING in [HL] up to 0
OR      A
RET     Z
LD      C, A
CALL    CO
INC     HL
JP      PMSG

;
GETCMD: CALL    CI              ;GET A CHARACTER, convert to UC, ECHO it
CALL    UCASE
CP      A, ESC
RET     Z                      ;Don't echo an ESC
PUSH    AF                    ;Save it
PUSH    BC
LD      C, A
CALL    CO                    ;Echo it
POP     BC
POP     AF                    ;get it back
RET

```

```

ZCRLF:  PUSH  AF          ;Send CR/LF to CRT
        PUSH  BC
        LD    C,CR
        CALL  CO
        LD    C,LF
        CALL  CO
        POP  BC
        POP  AF
        RET

;
; Convert LC to UC
UCASE:  CP    A,'a'      ;must be >= lowercase a
        RET    C        ; else go back...
        CP    A,'z'+1   ;must be <= lowercase z
        RET    NC       ; else go back...
        SUB   A,'a'-'A' ;subtract lowercase bias
        RET

;
; Return with 2 HEX digits in [A]. If abort, Carry flag set + ESC in [A]
GET_HEX:
        PUSH  BC
        CALL  GETCMD     ;Get a character from keyboard & ECHO
        CP    A,ESC
        JR    Z,HEX_ABORT
        CP    '/',      ;check 0-9, A-F
        JR    C,HEX_ABORT
        CP    'F'+1
        JR    NC,HEX_ABORT
        CALL  ASBIN      ;Convert to binary
        SLA  A
        SLA  A
        SLA  A
        SLA  A          ;Shift to high nibble
        LD    B,A        ;Store it
        CALL  GETCMD     ;Get 2nd character from keyboard & ECHO
        CP    A,ESC
        JR    Z,HEX_ABORT
        CP    '/',      ;check 0-9, A-F
        JR    C,HEX_ABORT
        CP    'F'+1
        JR    NC,HEX_ABORT
        CALL  ASBIN      ;Convert to binary
        OR   A,B        ;add in the first digit
        OR   A,A        ;To return NC
        POP  BC
        RET

HEX_ABORT:
        SCF            ;Set Carry flag
        LD    A,ESC
        POP  BC
        RET

;
; Put 4 HEX characters in [HL] (To set RAM location etc.)
;
GET_HEX4:
        LD    H,0000H
        CALL  GET_HEX    ;get 2 HEX digits
        JR    C,SCAN_ABORT
        LD    H,A
        CALL  GET_HEX    ;get 2 more HEX digits
        JR    C,SCAN_ABORT
        LD    L,A
        OR   A,A        ;To return NC
        RET

SCAN_ABORT:
        SCF            ;Set Carry flag
        RET

```



```

; ASCII TO BINARY CONVERSION ROUTINE
ASBIN: SUB     30H
      CP      0AH
      RET     M
      SUB     07H
      RET

;Print the accumulator value on CRT in HEX-ASCII
PACC:  PUSH   AF
      PUSH   BC
      PUSH   AF
      RRCA
      RRCA
      RRCA
      RRCA
      CALL   ZCONV
      POP    AF
      CALL   ZCONV
      POP    BC
      POP    AF
      RET

ZCONV: AND     A,0FH           ;HEX to ASCII
      ADD     90H
      DAA
      ADC     40H
      DAA
      LD      C,A
      CALL   CO
      RET

;
;DISPLAY BIT PATTERN IN [A]
;
ZBITS: PUSH   AF
      PUSH   BC
      PUSH   DE
      LD     E,A
      LD     B,8
BQ2:   SLA    E
      LD     A,18H
      ADC    A
      LD     C,A
      CALL   CO
      DJNZ  BQ2
      POP    DE
      POP    BC
      POP    AF
      RET

;
;
; CHECK IF AN ABORT (ESC) CHARACTER IS PRESENT AT THE CONSOL
; HOLD EVERYTHING IF SPACEBAR IS PRESSED
;
CHECKABORT:                ;see if an abort is required
      CALL   CONST
      RET    Z
CHECK1: CALL   CI
      CP     A,ESC           ;ESC to abort
      JR     Z,CABORT
      CP     A,' '           ;If spacebar then freeze CRT display
      JR     Z,CHECK1
      XOR    A,A
      RET
CABORT: XOR    A,A           ;Was ESC, so return with NZ flag
      DEC   A               ;return NZ, & 0FFH in A if ESC there
      RET

;
;
;

```

```

;----- Debug Routines -----
DEBUG_A:                ;Display contents of A of CRT
    PUSH    AF
    PUSH    BC
    PUSH    DE
    PUSH    HL
    PUSH    AF
    LD      C,'>'
    CALL    CO
    POP     AF
    CALL    PACC
    LD      C,'<'
    CALL    CO
    POP     HL
    POP     DE
    POP     BC
    POP     AF
    RET

;
DEBUG_B                ;Display A on CRT wait for keyboard
    CALL    DEBUG_A
    PUSH    AF
    CALL    CI
    POP     AF
    RET

;
DEBUG_HL              ;Display HL on CRT
    PUSH    AF
    LD      A,H
    CALL    DEBUG_A
    LD      A,L
    CALL    DEBUG_A
    POP     AF
    RET

;
;
;
;-----
;
SIGNON:               DB      CR,LF,LF
                    if CHIP_1795
                    DB      'Versafloppy II (for 1795 chip) Diagnostic Program. '
                    DB      '(V1.01 by John Monahan 2009)',CR,LF,0
                    endif

                    if CHIP_1791
                    DB      'Versafloppy II (for 1791 chip) Diagnostic Program. '
                    DB      '(V1.02 by John Monahan 2009)',CR,LF,0
                    endif

GETDRV_MSG:          DB      'Please select a drive. (A,B,C or D): ',0

XXXDRV:              DB      CR,LF, 'Sorry that is an invalid drive',0
GETSIZE_MSG:         DB      CR,LF, 'Is this a 5" or 8" Drive (5,8): ',0
DISK_8_FORMATS:      DB      CR,LF,LF,'Possible 8" disk formats:-'
                    DB      CR,LF, 'A = 128 Bytes/Sec, 26 Sec/Ttk. 8" SDSS. (IBM 3740 Format) '
                    DB      CR,LF, 'B = 128 Bytes/Sec, 50 Sec/Ttk. 8" DDSS. (SD Systems 8" DD Format) '
                    DB      CR,LF, 'C = 256 Bytes/Sec, 26 Sec/Ttk. 8" DDDS. (IBM System 34 Format) '
                    DB      CR,LF, 'D = 512 Bytes/Sec, 15 Sec/Trk. 8" DDDS. '
                    DB      CR,LF, 'E = 1024 Bytes/Sec, 9 Sec/Trk. 8" DDSS. (For CPM3 System) '
                    DB      CR,LF, 'F = 1024 Bytes/Sec, 9 Sec/Trk. 8" DDDS.'
                    DB      CR,LF, 'Please select a disk format: ',0
DISK_5_FORMATS:      DB      CR,LF,LF,'Possible 5" disk formats:-'
                    DB      CR,LF, 'A = 512 Bytes/Sec, 8 Sec/Ttk. 5" DDDS. (CPM3 & IBM PC CPM-86 Format) '
                    DB      CR,LF, 'B = 128 Bytes/Sec, 18 Sec/Ttk. 5" SDSS. (SD Systems 5" SD Format) '
                    DB      CR,LF, 'C = 128 Bytes/Sec, 29 Sec/Ttk. 5" DDSS. (SD Systems 5" DD Format) '
                    DB      CR,LF, 'D = 512 Bytes/Sec, 9 Sec/Ttk. 5" DDDS. (DEC VT180 Format) '
                    DB      CR,LF, 'E = 256 Bytes/Sec, 16 Sec/Ttk. 5" DDDS. (TOSHIBA T-100 Format) '
                    DB      CR,LF, 'F = 128 Bytes/Sec, 18 Sec/Ttk. 5" SDDS. (CROMEMCO-SD CDOS Format) '

```

```

DB      CR,LF, 'G = 512 Bytes/Sec, 10 Sec/Ttk. 5" DDDS. (CROMEMCO-DD CDOS Format)'
DB      CR,LF, 'H = 512 Bytes/Sec, 10 Sec/Ttk. 5" DDDS. (EPSON QX-10 Format)'
DB      CR,LF, 'I = 1024 Bytes/Sec, 5 Sec/Ttk. 5" DDDS. (MORROW MD3 Format)'
DB      CR,LF, 'J = 512 Bytes/Sec, 8 Sec/Ttk. 5" DDDS. (ZENITH Z-100 Format)'
DB      CR,LF, 'K = 256 Bytes/Sec, 16 Sec/Ttk. 5" DDDS. (SUPERBRAIN QD Format)'
DB      CR,LF, 'L = 512 Bytes/Sec, 8 Sec/Ttk. 5" DDDS. (IBM PC, MSDOS 1.1 Format)'
DB      CR,LF, 'M = 512 Bytes/Sec, 9 Sec/Ttk. 5" DDDS. (IBM PC, MSDOS 2.x Format)'
DB      CR,LF, 'N = 512 Bytes/Sec, 10 Sec/Ttk. 5" DDDS. (TRS-80 MOD III Format)'
DB      CR,LF, 'Please select a disk format: ',0
OPTION_ERROR: DB      CR,LF, 'Sorry invalid option',0
SEC_COUNT_ERR DB      CR,LF, 'Sector count error',0
RAM_ERROR     DB      CR,LF, 'Invalid RAM location error',0

MAIN_MENU0:   DB      CR,LF,LF,'VERSAFLOPPY II DIAGNOSTICS ---- MAIN MENU ---- (Detail Display OFF)',0
MAIN_MENU1:   DB      CR,LF,LF,'VERSAFLOPPY II DIAGNOSTICS ---- MAIN MENU ---- (Detail Display ON)',0
DISK_INFO     DB      CR,LF,'Current drive:- ',0
DISK_INFO0:   DB      CR,LF,'Drive ',0
DISK_INFO1:   DB          ': (IOBYTE)=' ,0
DISK_INFO2:   DB          ' Size=' ,0
DISK_INFO3:   DB          '" , Double Sided, ',0
DISK_INFO4:   DB          '" , Single Sided, ',0
DISK_INFO5:   DB          'Double Density disk.',0
DISK_INFO6:   DB          'Single Density disk.',0

DISK_INFO7:   DB      CR,LF, 'Current Disk [IX] Table: Sec/Track=' ,0
DISK_INFO8:   DB          'H, Tracks/Side=' ,0
DISK_INFO9:   DB          'H, Sec Size=' ,0
DISK_INFO10:  DB          'H, 128 Bytes/Sec.',0
DISK_INFO11:  DB          'H, 256 Bytes/Sec.',0
DISK_INFO12:  DB          'H, 512 Bytes/Sec.',0
DISK_INFO13:  DB          'H, 1024 Bytes/Sec.',0
IDMSG0:       DB          'Track/Side/Sec/Size/(CRC) TRACK ID field= ',0

MENU_OPTIONS  DB      CR,LF,LF, '0 = Select a DRIVE/DISK to analyze'
DB      CR,LF, '1 = Seek Test                2 = Sequential Read Sector Test'
DB      CR,LF, '3 = Sequential Write Sector Test 4 = Sequential R/W Sector Test'
DB      CR,LF, '5 = Random Track/Sector R/W Test 6 = Continously Read sectors from a Track'
DB      CR,LF, '7 = Examine one complet track    8 = Load disk sctors to RAM'
DB      CR,LF, '9 = Write RAM to disk sectors    C = Copy current disk to another disk'
DB      CR,LF, 'F = Format the current disk        S = Copy CPM system tracks from '
DB          'another disk.'
DB      CR,LF, 'V = Verify sectors disk to disk  I = Quick Format of 8" CPM SSSD Disk in B:',0

MENU1_MSG:    DB      CR,LF, 'D = Turn ON detailed display.    ESC To return to CPM'
DB      CR,LF, 'Please enter a command:- ',0

MENU2_MSG:    DB      CR,LF, 'D = Turn OFF detailed display.    ESC To return to CPM'
DB      CR,LF, 'Please enter a command:- ',0

SEKMSG        DB      CR,LF, 'Disk seek test.',CR,LF,0
ATTRK:        DB          ' TRACK ',0
ATSEC:        DB          'H, SECTOR ',0
HEAD0_MSG     DB          'H, SIDE A',0
HEAD1_MSG     DB          'H, SIDE B',0
H_MSG         DB          'H. ',0
ATHEAD0       DB          ' SIDE A ',0
ATHEAD1       DB          ' SIDE B ',0
SEC_MSG:      DB      CR,LF, 'Sector Contents=' ,0
FORMATTING_MSG: DB      CR,LF, 'Formatting Disk. Format=' ,0
BADCMD        DB      CR,LF,BELL, 'Invalid command selected.',0
FORM_ERRMSG:  DB      CR,LF,BELL, 'Disk formatting aborted or write error.',0
END_FORM_MSG: DB      CR,LF, 'Disk has been formatted correctly. Each track is ',0
END_FORM1_MSG DB          'H Bytes long.',0
CODE_NOT_DONE: DB      CR,LF, 'Sorry Code not done yet. HALT',0
FORM_TRK:     DB      CR,LF, 'Formatting TRACK ',0
TRACK_MSG:    DB          'Track Image:- '
DB      CR,LF, 'Index field up to First sector=' ,CR,LF,0
SECTOR_MSG:   DB      CR,LF, 'Sector ID Field=' ,CR,LF,0
SEC_DATA_MSG: DB      CR,LF, 'Sector DATA Field=' ,CR,LF,0

```

```

SEC_GAP3_MSG: DB CR,LF, 'Sector GAP3 Field=',CR,LF,0
SEC_GAP4_MSG: DB CR,LF, 'End of Track GAP4 Field=',CR,LF,0
GET_TRACK_MSG: DB CR,LF, 'Please enter track number. (Enter 2 HEX digits): ',0
GET_SIDE_MSG: DB CR,LF, 'Please select Disk SIDE A or B (A,B): ',0
SIDE_ERROR: DB CR,LF,BELL, 'Invalid SIDE selection',0
GET_SEC_MSG: DB CR,LF, 'Please enter sector number. (Enter 2 HEX digits): ',0
TRACK_CONTENTS: DB CR,LF, 'Track contents:-',CR,LF,0
MORE_MSG: DB CR,LF, 'Do you wish to see more (Y/N): ',0
SEC_ERROR: DB CR,LF, 'Invalid sector number selection',0
GAP3_MSG: DB CR,LF, 'GAP3=',0
GAP1_MSG: DB CR,LF, 'GAP1=',0
ID_MSG: DB CR,LF, 'Sector ID Mark=',0
GAP2_MSG: DB CR,LF, 'GAP2=',0
SYNC_MSG: DB CR,LF, 'Data Sync=',0
STATUS_179x: DB CR,LF, 'Status Bits of 1791/5 Chip= ',0
ENDMSG: DB CR,LF,BELL, 'Disk has been formatted ',CR,LF,0
ERRMSG: DB CR,LF,BELL, 'SORRY COULD NOT SEEK NEXT TRACK',CR,LF,0
TRACK_ERROR: DB CR,LF,BELL, 'That track is out of range for this disk',0
SEC_READ_RETRY: DB CR,LF, 'Re-reading SECTOR ',0
SEC_WR_RETRY: DB CR,LF, 'Re-writing SECTOR ',0
SEC_RH_RETRY DB CR,LF, 'Re-Seeking head for re-reading SECTOR ',0
SEC_WH_RETRY DB CR,LF, 'Re-Seeking head for re-writing SECTOR ',0
SEC_V_ERROR DB CR,LF,BELL, 'Sector Verify Error found -----',0
RDTST_MSG DB CR,LF, 'Sequential Read Sectors Test',0
WRTST_MSG DB CR,LF, 'Sequential Write Sectors Test',0
RDWRTST_MSG DB CR,LF, 'Sequential R/W Sectors Test',0
RAND_TST_MSG DB CR,LF, 'Random Track/Sector R/W Sectors Test',0
DISK_WP_MSG DB CR,LF,BELL, 'This disk is currently Write Protected.',0
INITCPM_MSG DB CR,LF, 'Special modification of first sector for CPM86 done.',0
TRK_SIZE_ERR DB CR,LF,BELL, 'Size of track (TRK_SIZE) in the disk parameter table is invalid',0
SEC_COUNT_MSG DB CR,LF, 'Number of sectors (Enter 2 HEX digits): ',0
GET_DMA_MSG DB CR,LF, 'Enter RAM loaction to recieve the data (Enter 4 HEX digits): ',0
GET_DMAD_MSG DB CR,LF, 'Enter RAM loaction of data to write (Enter 4 HEX digits): ',0
LOADING_MSG: DB CR,LF, 'Loading data ',0
LOADING1_MSG: DB ' To RAM at ',0
ERR_TK_MSG: DB CR,LF,BELL, 'ERROR. Ran out of tracks!',0
LOAD_DONE_MSG: DB CR,LF, 'Data read correctly.',0
TRK_DUMP_MSG DB CR,LF, 'Load a selected track from current disk into RAM',0
LOAD_MSG DB CR,LF, 'Load sector(s) from disk to RAM',0
DUMP_MSG DB CR,LF, 'Write sector(s) to disk with data from RAM location.',0
DUMPING_MSG: DB CR,LF, 'Writing data ',0
DUMPING1_MSG: DB ' From RAM at ',0
DUMP_DONE_MSG: DB CR,LF, 'Data written correctly.',0
TIMEOUT_ERR: DB CR,LF,BELL, 'Forced a 1791/5 chip RESET because the status port was hung up busy!'
DB CR,LF, 'Check hardware such as drive type, connections etc.',0
COPY_MSG: DB CR,LF, 'Copy current disk to another disk (Note: Disks MUST be same format).',0
COPY2_MSG DB CR,LF, 'For the destination disk: ',0
HOW_MANY_TRKS DB CR,LF, 'Copy just the System tracks or all tracks. (S or A): ',0
INVALID_TRK_CT DB CR,LF,BELL, 'Invalid track option.',0
COPYING_DONE DB CR,LF, 'Disk Copying done.',0
COPY_AT_TRK: DB CR,LF, 'Reading data from ',0
WRITE_AT_TRK: DB ' Writing data to ',0

READ_AT_TRK: DB CR,LF, 'Reading ',0
VERIFY_AT_TRK: DB ' Verify ',0
VERIFY_DONE DB CR,LF, 'Disk verifying done',0
VERIFY_MSG DB CR,LF, 'Sector by Sector verifying of two disks',0
VERIFY2_MSG DB CR,LF, 'For the second disk: ',0
CLOSE_BRACKET: DB ')',0
UNKNOWN_ERROR: DB CR,LF,BELL, 'Unknown Error. Status Bits= ',0
SYS_COPY_MSG: DB CR,LF, 'Will copy CPM system tracks FROM another disk and '
DB 'write it TO the current disk',0
SYS_COPY_MSG1: DB CR,LF, 'Source disk for CPM system tracks:',0
COPY_SYS_TRK DB CR,LF, 'Copying CPM from ',0
WRITE_SYS_TRK DB ' Writing CPM to ',0
DONE_SYS_MSG DB CR,LF, 'CPM System tracks copied OK.',0
IBM_FORMAT DB CR,LF, 'Quick formatting of a blank SSSD 8" IBM disk in B: drive for CPM.',0
IBM_FORMAT1 DB CR,LF, 'Formatting in progress',CR,LF,0
END_FORM_MSG1: DB CR,LF, 'Disk has been formatted as a SSSD 8" disk (IBM 3740 Format).',0

```

```

ERRORS_SEEN:  DB      CR,LF,LF,BELL, 'Errors encountered. Format not reliable',0
COPY_SYSTRKS  DB      CR,LF, 'Do you wish to copy CPM from the system tracks of drive A:'
              DB      CR,LF, 'Note: It must be on an IBM 3740 formatted disk. (Y/N)',0
COPYING_CPM   DB      CR,LF, 'Copying CPM from system tracks of Drive A:',0
END_FORM_MSG2:DB      CR,LF, 'Disk B: has been formatted and CPM system installed on it correctly.',0
FORM_TRK_MSG  DB      CR, 'At Track ',0
BAD_COPY_MSG  DB      CR,LF,LF,BELL, 'Errors encountered during disk copying. Copy may not be valid',0
;
;
;
SEC_READ_ERROR: DB      CR,LF,BELL,'SEC READ Error.'
                DB      ' Bits: DNR,0,Record Type,RNF,'
                DB      'CRC,DATA,DRQ,Busy (',0

SEEKNV_ERROR:  DB      CR,LF,BELL,'SEEK (NV) Error.'
                DB      ' Bits: DNR,WP,Head,Seek,'
                DB      'CRC,TRK0,INDEX,Busy (',0

RSCMD_ERROR:   DB      CR,LF,BELL,'RESTORE Error.'
                DB      ' Bits: DNR,WP,Head,Seek,'
                DB      'CRC,TRK0,INDEX,Busy (',0

SEC_ID_ERROR:  DB      CR,LF,BELL,'SEC ID Error.'
                DB      ' Bits: DNR,0,0,RNF,'
                DB      'CRC,DATA,DRQ,Busy (',0

SKCMD_ERROR:   DB      CR,LF,BELL,'SEEK Error.'
                DB      ' Bits: DNR,WP,Head,Seek,'
                DB      'CRC,TRK0,INDEX,Busy (',0

STEPIN_ERROR:  DB      CR,LF,BELL,'STEP-IN Error.'
                DB      ' Bits: DNR,WP,Head,Seek,'
                DB      'CRC,TRK0,INDEX,Busy (',0

RDTCMD_ERROR:  DB      CR,LF,BELL,'TRACK Read Error.'
                DB      ' Bits: DNR,0,0,0,'
                DB      '0,DATA,DRQ,Busy (',0

WRTCMD_ERROR:  DB      CR,LF,BELL,'TRACK Write Error.'
                DB      ' Bits: DNR,WP,WF,0,'
                DB      '0,DATA,DRQ,Busy (',0

WRCMD_ERROR:   DB      CR,LF,BELL,'SEC WRITE Error.'
                DB      ' Bits: DNR,WP,WF,0,'
                DB      'CRC,DATA,DRQ,Busy (',0
;
;
;-----
;
;      LOOKUP TABLES OF DISK PARAMETERS
;
;      8" SINGLE DENSITY DRIVE VARIABLES (IBM 3740 Format)
STDSDT: DB      26          ;SECTORS PER TRACK
        DB      77          ;TRACKS PER SIDE
        DB      00000000B   ;Disk HW_BYTE (SDSS)
        DB      40          ;HEADER GAP (SD-Systems has 100-27, IBM is 40!)
        DB      6           ;GAP 1 (0's)
        DB      11          ;GAP 2 (FF's)
        DB      27          ;GAP 3 (FF's)
        DB      247         ;GAP 4 (FF's)
        DB      1           ;GAPR (Flag for multiple repeats of GAP4)
        DB      0           ;128 Bytes/sec
        DB      0FFH       ;GAP Format fill character
        DB      0E5H       ;Data area fill character
        DW      1423H      ;Size in bytes of 1 formatted track
        DB      0H         ;No special post format
        DW      SKEW_IBM   ;Location of this disks sector skew table
        DB      1H         ;Each format will have a unique number. For disk to disk copy
        DB      2         ;Tracks set aside for operating system (eg CPM 2)

```

```

DB      '8" SINGLE DENSITY (IBM 3740 Format)',0
SKEW_IBM:
DB      1H,2H,3H,4H,5H,6H,7H,8H,9H,0AH,0BH,0CH,0DH,0EH,0FH
db      10H,11H,12H,13H,14H,15H,16H,17H,18H,19H,1AH

;
8" DOUBLE DENSITY (128 BYTE SECTORS)
STDDDT: DB      50          ;SECTORS PER TRACK
DB      77          ;TRACKS PER SIDE
DB      01000000B     ;Disk HW_BYTE (DDSS)
DB      80          ;HEADER GAP (SD-Systems has 100-16, IBM is 80!)
DB      8           ;GAP 1 (4E's)
DB      22          ;GAP 2 (4E's)
DB      16          ;GAP 3 (4E's)
DB      199         ;GAP 4 (4E's) (X3 = 597)
DB      3           ;GAPR (Flag for multiple repeats of GAP4)
DB      0           ;128 Bytes/sec
DB      4EH         ;GAP Format fill character
DB      0E5H        ;Data area fill character
DW      29A0H       ;Size in bytes of 1 formatted track
DB      0H          ;No special post formatting modifications of disk req
DW      SKEW_SDT    ;Location of this disks sector skew table
DB      2H          ;Each format will have a unique number. For disk to disk copy
DB      2           ;Tracks set aside for operating system (eg CPM 2)
DB      '8" DOUBLE DENSITY (SD_Systems Format)',0
SKEW_SDT:
DB      1H,2H,3H,4H,5H,6H,7H,8H,9H,0AH,0BH,0CH,0DH,0EH,0FH
db      10H,11H,12H,13H,14H,15H,16H,17H,18H,19H,1AH,1BH,1CH,1DH,1EH,1FH
db      20H,21H,22H,23H,24H,25H,26H,27H,28H,29H,2AH,2BH,2CH,2DH,2EH,2FH
db      30H,31H,32H

;
;
8" DOUBLE DENSITY (256 BYTE SECTORS) (IBM System 34 Format)
DDT256: DB      26          ;NBR SECTORS PER TRACK
DB      77          ;NBR TRACKS PER SIDE
DB      01010000B     ;Disk HW_BYTE (DDDS)
DB      80          ;HEADER GAP (SD-Systems has 100-54, IBM is 80!)
DB      12          ;GAP 1
DB      22          ;GAP 2
DB      54          ;GAP 3
DB      199         ;GAP 4 (4E's) (X3 = 597)
DB      3           ;GAPR (Flag for multiple repeats of GAP4)
DB      1           ;256 Bytes/sec
DB      4EH         ;GAP Format fill character
DB      0E5H        ;Data area fill character
DW      28ECH       ;Size in bytes of 1 formatted track
DB      0H          ;No special post formatting modifications of disk req
DW      SKEW_256    ;Location of this disks sector skew table
DB      3H          ;Each format will have a unique number. For disk to disk copy
DB      2           ;Tracks set aside for operating system (eg CPM 2)
DB      '8" DD (256 BYTE SECTORS) (IBM System 34 Format)',0
SKEW_256:
DB      1H,2H,3H,4H,5H,6H,7H,8H,9H,0AH,0BH,0CH,0DH,0EH,0FH
db      10H,11H,12H,13H,14H,15H,16H,17H,18H,19H,1AH

;
;
8" DOUBLE DENSITY (512 BYTE SECTORS)
DDT512: DB      15          ;NBR SECTORS PER TRACK
DB      77          ;NBR TRACKS PER SIDE
DB      01010000B     ;Disk HW_BYTE (DDDS)
DB      80          ;HEADER GAP (SD-Systems has 100-54, IBM is 80!)
DB      12          ;GAP 1
DB      22          ;GAP 2
DB      84          ;GAP 3
DB      200         ;GAP 4 (4E's) (X3 = 597)
DB      2           ;GAPR (Flag for multiple repeats of GAP4)
DB      2           ;512 Bytes/sec
DB      4EH         ;GAP Format fill character
DB      0E5H        ;Data area fill character

```

```

DW      0H          ;Size in bytes of 1 formatted track
DB      0H          ;No special post formatting modifications of disk req
DW      SKEW_512    ;Location of this disks sector skew table
DB      4H          ;Each format will have a unique number. For disk to disk copy
DB      2           ;Tracks set aside for operating system (eg CPM 2)
DB      '8" DDDS (512 BYTE SECTORS)',0
SKEW_512:
DB      1H,2H,3H,4H,5H,6H,7H,8H,9H,0AH,0BH,0CH,0DH,0EH,0FH

;
;      8" DOUBLE DENSITY (1024 BYTE SECTORS - Single Sided)
DDT1K:  DB      9           ;NBR SECTORS PER TRACK
DB      77          ;NBR TRACKS PER SIDE
DB      01000000B   ;Disk HW_BYTE (DDDS) ;
DB      80          ;INDEX HEADER GAP
DB      12          ;NBR GAP 1
DB      22          ;NBR GAP 2
DB      54          ;NBR GAP 3
DB      199        ;GAP 4
DB      3           ;GAPR (Flag for multiple repeats of GAP4)
DB      3           ;1024 Bytes/sec
DB      4EH        ;GAP Format fill character
DB      0E5H       ;Data area fill character
DW      2B5AH      ;Size in bytes of 1 formatted track
DB      0H          ;No special post formatting modifications of disk req
DW      SKEW_1K    ;Location of this disks sector skew table
DB      5H          ;Each format will have a unique number. For disk to disk copy
DB      1           ;Tracks set aside for operating system (eg CPM 2)
DB      '8" DOUBLE DENSITY, SINGLE SIDED (1024 BYTE SECTORS)',0
SKEW_1K:
DB      1H,2H,3H,4H,5H,6H,7H,8H,9H

;
;      8" DOUBLE DENSITY (1024 BYTE SECTORS - Double Sided)
DDT1K2: DB      9           ;NBR SECTORS PER TRACK
DB      77          ;NBR TRACKS PER SIDE
DB      01010000B   ;Disk HW_BYTE (DDDS) ;
DB      80          ;INDEX HEADER GAP
DB      12          ;NBR GAP 1
DB      22          ;NBR GAP 2
DB      54          ;NBR GAP 3
DB      199        ;GAP 4
DB      3           ;GAPR (Flag for multiple repeats of GAP4)
DB      3           ;1024 Bytes/sec
DB      4EH        ;GAP Format fill character
DB      0E5H       ;Data area fill character
DW      2B5AH      ;Size in bytes of 1 formatted track
DB      0H          ;No special post formatting modifications of disk req
DW      SKEW_1KDS  ;Location of this disks sector skew table
DB      6H          ;Each format will have a unique number. For disk to disk copy
DB      1           ;Tracks set aside for operating system (eg CPM 2)
DB      '8" DOUBLE DENSITY, DOUBLE SIDED (1024 BYTE SECTORS)',0
SKEW_1KDS:
DB      1H,2H,3H,4H,5H,6H,7H,8H,9H

;
;
; 5", 128 byte, SD SD-Systems Format
MINSDT: DB      18          ;sectors per track
DB      35          ;tracks per side
DB      00100000B   ;Disk HW_BYTE (SDSS)
DB      20-8        ;index header gap
DB      6           ;GAP 1
DB      11          ;GAP 2
DB      8           ;GAP 3
DB      221        ;GAP 4 (FF's)
DB      1           ;GAPR (Flag for multiple repeats of GAP4)
DB      0           ;128 Bytes/sec
DB      0FFH       ;GAP Format fill character

```

```

DB      0E5H          ;Data area fill character
DW      0CC7H        ;Size in bytes of 1 formatted track
DB      0H           ;No special post formatting modifications of disk req
DW      SKEW_MINSD   ;Location of this disks sector skew table
DB      7H           ;Each format will have a unique number. For disk to disk copy
DB      2            ;Tracks set aside for operating system (eg CPM 2)
DB      '5", SDSS, 128 byte, SD-Systems Format',0
SKEW_MINSD:
DB      1H,2H,3H,4H,5H,6H,7H,8H,9H,0AH,0BH,0CH,0DH,0EH,0FH,10H,11H

;
; 5", 128 byte, DD SD-Systems Format
MINDDT: DB      29          ;sectors per track
DB      35          ;tracks per side
DB      01100000B    ;Disk HW_BYTE (DDSS)
DB      100-16      ;index header gap
DB      8           ;GAP 1
DB      22          ;GAP 2
DB      16          ;GAP 3
DB      247         ;GAP 4
DB      1           ;GAPR (Flag for multiple repeats of GAP4)
DB      0           ;128 Bytes/sec
DB      4EH         ;GAP Format fill character
DB      0E5H        ;Data area fill character
DW      17C0H       ;Size in bytes of 1 formatted track
DB      0H          ;No special post formatting modifications of disk req
DW      SKEW_MINDD   ;Location of this disks sector skew table
DB      8H          ;Each format will have a unique number. For disk to disk copy
DB      2           ;Tracks set aside for operating system (eg CPM 2)
DB      '5", DDSS. 128 byte, SD-Systems Format',0
SKEW_MINDD:
DB      1H,2H,3H,4H,5H,6H,7H,8H,9H,0AH,0BH,0CH,0DH,0EH,0FH
db      10H,11H,12H,13H,14H,15H,16H,17H,18H,19H,1AH,1BH,1CH

;
; 5", 512 byte, DDDS, 8 sector IBM PC CPM-86 format
MINCPM: DB      8          ;sectors per track
DB      40          ;tracks per side
DB      01110000B    ;Disk HW_BYTE (DDDS)
DB      80          ;index header gap
DB      12          ;GAP 1
DB      22          ;GAP 2
DB      80          ;GAP 3
DB      207         ;GAP 4 (4E's) (1038)
DB      5           ;GAPR (Flag for multiple repeats of GAP4)
DB      2           ;512 Bytes/sec
DB      04EH        ;GAP Format fill character
DB      0E5H        ;Data area fill character (for CPM86)
DW      19C0H       ;Size in bytes of 1 formatted track
DB      CPM86_FLAG   ;Special post formatting modifications of disk req
DW      SKEW_CPM86   ;Location of this disks sector skew table
DB      9H          ;Each format will have a unique number. For disk to disk copy
DB      2           ;Tracks set aside for operating system (eg CPM 2)
DB      '5", DDDS, 512 byte, 8 sector IBM PC CPM-86 format',0
SKEW_CPM86:
DB      1H,2H,3H,4H,5H,6H,7H,8H

;
;
; 5", 512 byte, DDDS, 9 sector DEC VT180 format
DEC:    DB      9          ;sectors per track
DB      40          ;tracks per side
DB      01110000B    ;Disk HW_BYTE (DDDS)
DB      80          ;index header gap
DB      12          ;GAP 1
DB      22          ;GAP 2
DB      26          ;GAP 3
DB      218         ;GAP 4 (4E's) (872)
DB      4           ;GAPR (Flag for multiple repeats of GAP4)
DB      2           ;512 Bytes/sec

```



```

DB      04EH          ;GAP Format fill character
DB      0E5H          ;Data area fill character (for CPM)
DW      1971H         ;Size in bytes of 1 formatted track
DB      0             ;No special post formatting modifications of disk req
DW      SKEW_DEC      ;Location of this disks sector skew table
DB      0AH          ;Each format will have a unique number. For disk to disk copy
DB      2             ;Tracks set aside for operating system (eg CPM 2)
DB      '5", DDDS, 512 byte, 9 sector DEC VT180 format',0
SKEW_DEC:
DB      1H,2H,3H,4H,5H,6H,7H,8H,9H
;
;
; 5", 256 byte, DDDS, 16 sector TOSHIBA T-100 format
TOSHIBA: DB 16          ;sectors per track
DB      35            ;tracks per side
DB      01110000B     ;Disk HW_BYTE (DDDS)
DB      80            ;index header gap
DB      12            ;GAP 1
DB      22            ;GAP 2
DB      50            ;GAP 3
DB      183           ;GAP 4 (4E's) (366)
DB      2             ;GAPR (Flag for multiple repeats of GAP4)
DB      1             ;256 Bytes/sec
DB      04EH          ;GAP Format fill character
DB      0E5H          ;Data area fill character (for CPM)
DW      1928H         ;Size in bytes of 1 formatted track
DB      0             ;No special post formatting modifications of disk req
DW      SKEW_TOSH     ;Location of this disks sector skew table
DB      0BH          ;Each format will have a unique number. For disk to disk copy
DB      2             ;Tracks set aside for operating system (eg CPM 2)
DB      '5", DDDS, 256 byte, 16 sector TOSHIBA T-100 format',0
SKEW_TOSH:
DB      1H,2H,3H,4H,5H,6H,7H,8H,9H,0AH,0BH,0CH,0DH,0EH,0FH
;
;
; 5", 128 byte, CROMEMCO CDOS (SINGLE density) Format
CDOS:  DB 18          ;sectors per track
DB      40            ;tracks per side
DB      00110000B     ;Disk HW_BYTE (SDDS)
DB      20-8          ;index header gap
DB      6             ;GAP 1
DB      11            ;GAP 2
DB      8             ;GAP 3
DB      185           ;GAP 4 (FF's)
DB      1             ;GAPR (Flag for multiple repeats of GAP4)
DB      0             ;128 Bytes/sec
DB      0FFH          ;GAP Format fill character
DB      0E5H          ;Data area fill character
DW      0CA3H         ;Size in bytes of 1 formatted track
DB      0H            ;No special post formatting modifications of disk req
DW      SKEW_CDOS     ;Location of this disks sector skew table
DB      0CH          ;Each format will have a unique number. For disk to disk copy
DB      2             ;Tracks set aside for operating system (eg CPM 2)
DB      '5", SDDS, 128 byte, CROMEMCO CDOS Format',0
SKEW_CDOS:
DB      1H,2H,3H,4H,5H,6H,7H,8H,9H,0AH,0BH,0CH,0DH,0EH,0FH,10H,11H
;
;
; 5", 512 byte, CROMEMCO CDOS w/INTL TERM. CP/M Format
CDOSDD: DB 10          ;sectors per track
DB      40            ;tracks per side
DB      00110000B     ;Disk HW_BYTE (SDDS)
DB      80            ;index header gap
DB      12            ;GAP 1
DB      22            ;GAP 2
DB      30            ;GAP 3
DB      214           ;GAP 4 (FF's)
DB      1             ;GAPR (Flag for multiple repeats of GAP4)
DB      2             ;512 Bytes/sec
DB      0FFH          ;GAP Format fill character

```

```

DB      0E5H          ;Data area fill character
DW      188EH        ;Size in bytes of 1 formatted track
DB      0H           ;No special post formatting modifications of disk req
DW      SKEW_CDOS2   ;Location of this disks sector skew table
DB      0DH          ;Each format will have a unique number. For disk to disk copy
DB      2            ;Tracks set aside for operating system (eg CPM 2)
DB      '5", DDDS, 512 byte, CROMEMCO CDOS/CPM Format',0
SKEW_CDOS2:
DB      1H,2H,3H,4H,5H,6H,7H,8H,9H,0AH

; 5", 512 byte, EPSON QX-10 Format
EPSON:  DB      10          ;sectors per track
DB      40           ;tracks per side
DB      00110000B    ;Disk HW_BYTE (SDDS)
DB      80           ;index header gap
DB      12           ;GAP 1
DB      22           ;GAP 2
DB      30           ;GAP 3
DB      214          ;GAP 4 (FF's)
DB      1            ;GAPR (Flag for multiple repeats of GAP4)
DB      2            ;512 Bytes/sec
DB      0FFH        ;GAP Format fill character
DB      0E5H        ;Data area fill character
DW      188EH        ;Size in bytes of 1 formatted track
DB      0H           ;No special post formatting modifications of disk req
DW      SKEW_EPSON   ;Location of this disks sector skew table
DB      0EH          ;Each format will have a unique number. For disk to disk copy
DB      2            ;Tracks set aside for operating system (eg CPM 2)
DB      '5", DDDS, 512 byte, EPSON QX-10 Format',0
SKEW_EPSON:
DB      1H,3H,5H,7H,9H,2H,4H,6H,8H,0AH          ;<-- note skew table
;
;
; 5", 1K byte, DDDS, 5 sector MORROW MD3 format
MORROW: DB      5          ;sectors per track
DB      40           ;tracks per side
DB      01110000B    ;Disk HW_BYTE (DDDS)
DB      80           ;index header gap
DB      12           ;GAP 1
DB      22           ;GAP 2
DB      50           ;GAP 3
DB      192          ;GAP 4 (4E's) (574)
DB      3            ;GAPR (Flag for multiple repeats of GAP4)
DB      3            ;1024 Bytes/sec
DB      04EH        ;GAP Format fill character
DB      0E5H        ;Data area fill character (for CPM)
DW      1977H        ;Size in bytes of 1 formatted track
DB      0            ;No special post formatting modifications of disk req
DW      SKEW_MORROW  ;Location of this disks sector skew table
DB      0FH          ;Each format will have a unique number. For disk to disk copy
DB      2            ;Tracks set aside for operating system (eg CPM 2)
DB      '5", DDDS, 1024 byte, 9 sector MORROW MD3 format',0
SKEW_MORROW:
DB      1H,2H,3H,4H,5H
;
;
; 5", 512 byte, DDDS, 5 sector ZENITH Z-100 format
ZENITH: DB      8          ;sectors per track
DB      40           ;tracks per side
DB      01110000B    ;Disk HW_BYTE (DDDS)
DB      80           ;index header gap
DB      12           ;GAP 1
DB      22           ;GAP 2
DB      26           ;GAP 3
DB      242          ;GAP 4 (4E's) (1454)
DB      6            ;GAPR (Flag for multiple repeats of GAP4)
DB      2            ;512 Bytes/sec
DB      04EH        ;GAP Format fill character
DB      0E5H        ;Data area fill character (for CPM)

```

```

DW      1933H          ;Size in bytes of 1 formatted track
DB      0              ;No special post formatting modifications of disk req
DW      SKEW_ZENITH    ;Location of this disks sector skew table
DB      10H           ;Each format will have a unique number. For disk to disk copy
DB      2              ;Tracks set aside for operating system (eg CPM 2)
DB      '5", DDDS, 512 byte, 8 sector ZENITH Z-100 format',0
SKEW_ZENITH:
DB      1H,2H,3H,4H,5H,6H,7H,8H
;
;
; 5", 512 byte, DDDS, 10 sector SUPERBRAIN QD format
SUPER:  DB      10          ;sectors per track
DB      35              ;tracks per side
DB      01110000B      ;Disk HW_BYTE (DDDS)
DB      80              ;index header gap
DB      12              ;GAP 1
DB      22              ;GAP 2
DB      16              ;GAP 3
DB      177            ;GAP 4 (4E's) (354)
DB      2              ;GAPR (Flag for multiple repeats of GAP4)
DB      2              ;512 Bytes/sec
DB      04EH           ;GAP Format fill character
DB      0E5H           ;Data area fill character (for CPM)
DW      193AH          ;Size in bytes of 1 formatted track
DB      0              ;No special post formatting modifications of disk req
DW      SKEW_SUPER     ;Location of this disks sector skew table
DB      11H           ;Each format will have a unique number. For disk to disk copy
DB      2              ;Tracks set aside for operating system (eg CPM 2)
DB      '5", DDDS, 512 byte, 10 sector SUPERBRAIN QD format',0
SKEW_SUPER:
DB      1H,2H,3H,4H,5H,6H,7H,8H,9H,0AH
;
;
; 5", IBM PC, MSDOS 1.1, 512 byte, DDDS, 8 sector format
MSDOS:  DB      8          ;sectors per track
DB      40              ;tracks per side
DB      01110000B      ;Disk HW_BYTE (DDDS)
DB      80              ;index header gap
DB      12              ;GAP 1
DB      22              ;GAP 2
DB      80              ;GAP 3
DB      193            ;GAP 4 (4E's)
DB      2              ;GAPR (Flag for multiple repeats of GAP4)
DB      2              ;512 Bytes/sec
DB      04EH           ;GAP Format fill character
DB      0E5H           ;Data area fill character
DW      16B2H          ;Size in bytes of 1 formatted track
DB      0H             ;Special formatting modifications of disk req (+++ NOT DONE YET)
DW      SKEW_DOS1     ;Location of this disks sector skew table
DB      12H           ;Each format will have a unique number. For disk to disk copy
DB      2              ;Tracks set aside for operating system (eg CPM 2)
DB      '5", DDDS, 512 byte, 8 sector IBMPC MSDOS 1.1 format',0
SKEW_DOS1:
DB      1H,2H,3H,4H,5H,6H,7H,8H
;
;
; 5", IBM PC, MSDOS 2.x, 512 byte, DDDS, 9 sector format
MSDOS2: DB      9          ;sectors per track
DB      40              ;tracks per side
DB      01110000B      ;Disk HW_BYTE (DDDS)
DB      80              ;index header gap
DB      12              ;GAP 1
DB      22              ;GAP 2
DB      80              ;GAP 3
DB      193            ;GAP 4 (4E's)
DB      2              ;GAPR (Flag for multiple repeats of GAP4)
DB      2              ;512 Bytes/sec
DB      04EH           ;GAP Format fill character
DB      0E5H           ;Data area fill character

```

```

DW      193EH          ;Size in bytes of 1 formatted track
DB      0H             ;Special formatting modifications of disk req (+++ NOT DONE YET)
DW      SKEW_DOS2     ;Location of this disks sector skew table
DB      13H           ;Each format will have a unique number. For disk to disk copy
DB      2              ;Tracks set aside for operating system (eg CPM 2)
DB      '5", DDSS, 512 byte, 9 sector IBMPC MSDOS 2.x format',0
SKEW_DOS2:
DB      1H,2H,3H,4H,5H,6H,7H,8H,9H
;
;
; 5", TRS-80 MOD-III, 512 byte, DDSS, 10 sector format
TRS80:  DB      10          ;sectors per track
DB      40              ;tracks per side
DB      01100000B      ;Disk HW_BYTE (DDSS)
DB      80              ;index header gap
DB      12              ;GAP 1
DB      22              ;GAP 2
DB      26              ;GAP 3
DB      137            ;GAP 4 (4E's)
DB      2              ;GAPR (Flag for multiple repeats of GAP4)
DB      2              ;512 Bytes/sec
DB      04EH           ;GAP Format fill character
DB      0E5H           ;Data area fill character
DW      1976H          ;Size in bytes of 1 formatted track
DB      0H             ;Special formatting modifications of disk req (+++ NOT DONE YET)
DW      SKEW_TRS      ;Location of this disks sector skew table
DB      14H           ;Each format will have a unique number. For disk to disk copy
DB      2              ;Tracks set aside for operating system (eg CPM 2)
DB      '5", DDSS, 512 byte, 10 sector TRS-80 MOD-III format',0
SKEW_TRS:
DB      1H,2H,3H,4H,5H,6H,7H,8H,9H,0AH
;
;
;-----
;
;
; THE FOLLOWING RAM LOCATIONS ARE REQ
;
IX_OLD_STORE  DW      0000H
ERRORS_FLAG  DB      0H          ;Will keep track of errors during a routine
CRTDISP      DB      0H
IDSV         DS      6H
CMD_STORE    DB      0H          ;Menu CMD Store (Note: NOT CHIP CMD,CHIP_CMDSV)
ERMASK       DB      0H          ;ERROR MASK
ERSTAT       DB      0H          ;PRESENT ERROR STORE
CHIP_CMDSV   DB      0H          ;COMMAND TO 1791/5 SAVE
SP_SAVE      DW      0000H      ;SP SAVE
SEC_RT_COUNT DB      0H          ;Number of sector reads before setting error flag
SEEK_RT_COUNT DB      0H          ;Number of seek retrys
TRK_RT_COUNT DB      0H          ;Number of track reads before setting error flag
TRACK_SIZE   DW      0000H      ;Size in bytes of current formatted track
SEC_COUNT    DB      0H          ;number of sectors to load
DMA_NEXT     DW      0           ;Store for next (TADDR) for multiple sec R/W's
COPY_TRK_COUNT: DB      0H          ;Used for disk to disk track copying
COPY_TRK     DB      0H
DRIVE_STORE  DB      0H          ;For 8' quick formatting
;
F_TRK        DB      0H          ;for building format track image
F_SIDE       DB      0H          ;for building format track image
INDEX_MARK   DW      0000H      ;End of Index field +1
S_DATA_MARK  DW      0000H      ;Pointer to start of Data area
E_DATA_MARK  DW      0000H      ;Pointer to end of Data area+1
E_SEC_MARK   DW      0000H      ;End Sector image +1
S_GAP4_MARK  DW      0000H      ;start GAP4 area
E_GAP4_MARK  DW      0000H      ;End track +1
;
DS      100H
STACK EQU    $

```

;END