```
title CP/M 2.2 BIOS -- A.D.C. SUPER QUAD - SUPER SIX SBC -- VERSION 2.22

;--------------------------------
; AUTHOR  Marcus G. Calescibetta
; DATE    October 15, 1983
; VERSION 2.22
;--------------------------------

      .z80                  ; M80 z80 code pseudo op

      include    SUPRBIOS.LIB          ; contains all macros used

;--------------------------------
; INTERUPT OR POLLED CONSOLE I/O
;--------------------------------

intrin      equ   true             ; intr con in  (Type ahead)
introut     equ   false            ; intr con out (Buff con output)

inbfsz      equ   32               ; buf siz for intr in,  power of 2
outbfsz     equ   64               ; buf siz for intr out, power of 2

ints  equ   intrin or introut

;----------------------------
; PARALLEL OR SERIAL PRINTER
;----------------------------

parprnt     equ   true             ; true if par prn, false if ser prn

;------------------------
; CP/M PAGE 0 ADDRESSES
;------------------------

iobyte      equ   00003h                ; iobyte addr
cdisk equ   00004h                ; default disk and user number addr

;----------------------------
; CP/M DEBLOCKING PARAMETERS
;----------------------------

wrtuse      equ   0                ; write to previously allocated block
wrtdir      equ   1                ; write to directory (must flush)
wrtfre      equ   2                ; write to newly allocated block
rtrys equ   5                ; retry count in case of disk error

;---------------------------------
; FLOPPY CONTROLLER PORT ADDRESSES
;---------------------------------

fbase equ   0ch              ; base port addr of wd-1793 or wd-2793
fcmd  equ   fbase+0               ; command register
fstat equ   fbase+0               ; status register
ftrk  equ   fbase+1               ; track register
```

```
fsec  equ   fbase+2               ; sector register
fdata equ   fbase+3               ; data register
fwait equ   014h             ; wait register
fdsd  equ   014h             ; density - size - drive register

;----------------------------
; FLOPPY CONTROLLER COMMANDS
;----------------------------

fstpr equ   002h             ; floppy step rate (0,1,2 or 3)

frst  equ   000h + fstpr          ; restore hds
fsek  equ   01ch + fstpr          ; seek track
frd   equ   084h             ; read sector
fwrt  equ   0a4h             ; write sector
frda  equ   0c4h             ; read track id

;-------------------------------
; FLOPPY CONTROLLER BIT TESTERS
;-------------------------------

dden  equ   08h              ; double density (bit 3)
sden  equ   00h              ; single density (bit 3)
s5    equ   10h              ; 5" floppy      (bit 4)
s8    equ   00h              ; 8" floppy  (bit 4)
s0    equ   00h              ; side select 0  (bit 2)
s1    equ   04h              ; side select 1  (bit 2)

;-----------------------------------
; HDC-1001 CONTROLLER PORT ADDRESSES
;-----------------------------------

hbase equ   0e0h             ; base port address of hdc-1001
hdata equ   hbase+0               ; data register
herror      equ   hbase+1               ; error register
hwrtpre     equ   hbase+1               ; write pre-compensation register
hseccnt     equ   hbase+2               ; sector count register
hsec  equ   hbase+3               ; sector number register
hcyllow     equ   hbase+4               ; cylinder low register
hcylhi      equ   hbase+5               ; cylinder high register
hsdh  equ   hbase+6               ; size - drive - head register
hstatus     equ   hbase+7               ; status register port
hcmd  equ   hbase+7               ; command register port

;-------------------
; HDC-1001 COMMANDS
;-------------------

cmdrst      equ   010h               ; restore command
cmdrd equ   020h             ; read sector command
cmdwrt      equ   030h               ; write sector command

;-----------------------------
; STATUS REGISTER BIT TESTERS
```

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

bsybit      equ   080h             ; busy bit
rdybit      equ   040h             ; data request bit
errbit      equ   001h             ; error bit

;------------------
; SIO PORT ADDRESS
;------------------

siobase     equ   0
sioad equ   siobase+0        ; SIO channel A data
sioac equ   siobase+1        ; SIO channel A control/status
siobd equ   siobase+2        ; SIO channel B data
siobc equ   siobase+3        ; SIO channel B control/status

;------------------
; PIO PORT ADDRESS
;------------------

piobase     equ   4
pioad equ   piobase+0        ; PIO channel A data
pioac equ   piobase+2        ; PIO channel A control
piobd equ   piobase+1        ; PIO channel B data
piobc equ   piobase+3        ; PIO channel B control

;----------------------
; MEMORY CONTROL PORT
;----------------------

memry equ   16h              ; memory control port

;----------------------------
; LOCATION OF BIOS IN MEMORY
;----------------------------

      .phase      0EA00h                ; starting address of this bios

;------------------
; BIOS JUMP VECTOR
;------------------

start:      jp    coldboot          ; cold start
wboota:     jp    wboot             ; warm start
      jp    const             ; console status
      jp    conin             ; console character input
      jp    conout                ; console character output
      jp    list              ; list    character output
      jp    punch             ; punch   character output
      jp    reader                ; reader  character output
      jp    home              ; move head to home position
      jp    seldsk                ; select disk
      jp    settrk                ; set track number
      jp    setsec                ; set sector number
```

```
      jp    setdma                  ; set dma address
      jp    read              ; read disk
      jp    write             ; write disk
      jp    liststst                ; return list status
      jp    sectran                 ; sector translate

ccpstart:
      dw    start-1600h       ; ccp  addr
bdosjmp:
      dw    start-0dfah       ; bdos addr

;-----------------
; WARM BOOT TABLE
;-----------------

;
; filled in by CPMLDR
;
; contains entries of form dw track, dw first sector, dw last sector
; it is a map of where the image of the CCP and BDOS reside on disk
;

bootbl::
      dw    0,0,0             ; space for four entries
      dw    0,0,0             ;
      dw    0,0,0             ;
      dw    0,0,0             ;
      dw    0                 ; extra track to exit properly

;-----------------------
; SERIAL INTERUPT TABLE
;-----------------------

;
;     serial channel interrupt vector table
;     contains one entry for each interrupt that
;     can be generated by sio/dart
;

      if    ints              ; interrupt driven i/o
      ds    16-(($-start) mod 16)

serinttbl:
      dw    intret                  ; channel B transmitter buffer empty
      dw    intret                  ; channel B external interrupts
      dw    intret                  ; channel B receiver ready
      dw    intret                  ; channel B special receive interrupt

      if    introut
      dw    tran0int          ; channel A transmitter buffer empty
      else
      dw    intret                  ; channel A transmitter buffer empty
      endif
```

```
      dw    intret                  ; channel A external interrupts

      if    intrin
      dw    recv0int          ; channel A receiver ready
      else
      dw    intret                  ; channel A receiver ready
      endif

      dw    recv0int          ; channel A special recieve interrupt
;     dw    intret                  ; channel A special receive interrupt
      endif

;
; Console / List Vectors
;

const:
      ld    D,0
      jr    conjmp

conin:
      ld    D,3
      jr    conjmp

conout:
      ld    D,6
      jr    conjmp

list:
      ld    D,9
      jr    lstjmp

listst:
      ld    D,0ch

lstjmp:
      ld    A,(iobyte)
      rlca
      rlca              ;make 0-3
      ld    HL,lsttbl
      jr    jmp2

conjmp:
      ld    A,(iobyte)
      ld    HL,contbl
jmp2:
      and   00000011b   ;get console field
      jr    dskcon

reader:
      ld    d,3
      ld    hl,rdrtbl
      ld    a,(iobyte)
      jr    gordr
```

```
punch:
      ld    d,6
      ld    hl,puntbl
      ld    a,(iobyte)
      rra
      rra
gordr:
      rra
      rra
      jr    jmp2

dskcon:
      add   a,a
      ld    e,a
      ld    a,d
      ld    d,0
      add   hl,de
      ld    d,(hl)
      inc   hl
      ld    e,a
      ld    a,(hl)
      ex    de,hl
jphl:
      jp    (hl)

;-----------------------
; CONSOLE DRIVER TABLE
;-----------------------

contbl      equ   $+1

      dw    cpu
      dw    cpu             ; default=serial 0
      db    1               ; alternate=serial 1
      db    0,0,0           ; space for two more console drivers

;--------------------
; LIST DRIVER TABLE
;--------------------

lsttbl      equ   $+1

      dw    cpu

      if parprnt
      dw    cpu+2           ; default=parallel
      db    1               ; alternate=serial 1
      else
      dw    cpu+1           ; default=serial 1
      db    2               ; alternate=parallel
      endif

      db    0,0,0
```

```
;---------------------
; READER DRIVER TABLE
;---------------------

rdrtbl      equ   $+1

      dw    cpu
      dw    cpu+1           ; default=serial 1
      db    0               ; alternate=serial 0
      db    0,0,0           ; space for two more readers drivers

;--------------------
; PUNCH DRIVER TABLE
;--------------------

puntbl      equ   $+1

      dw    cpu
      dw    cpu+1           ; default=serial 1
      db    0               ; alternate=serial 0
      db    0,0,0,0              ; space for two more punch drivers

;-----------
; WARM BOOT
;-----------

;
; Wboot reads in the CCP and BDOS from the CPM.SYS file
; CPMLDR initializes the bootbl to be a list of items of
;
; track (dw), first sector(db), last sector(db)
;

wboot:
      ei

      ld    sp,0100h        ; set stack ptr at default dma  buf end

      call  flush           ; check if any good data to flush
      call  rstlvec              ; reset floppy login vector

      ld    c,0             ; load c-reg w. drive to boot off
      call  seldsk               ; sel drv A (use same drv bootbl frm)
      ld    a,h             ; check if sel drv returned illegal dph
      or    l               ; or low byte w high byte
      jp    z,booterr       ; bad select if both zero

      ld    e,(hl)               ; put dph tran vec addr in de-reg
      inc   hl              ; inc ptr to tran vec addr
      ld    d,(hl)               ; now get high byte of tran vec addr
      ld    (bootxlt),de         ; save tran vec addr

      ld    hl,(ccpstart)        ; ccp start is where to put first sec
```

```
      ld    iy,bootbl-6       ; bt tbl has loc of ccp & bdos on dsk
nxttrk:
      push  hl                ; save reg we use, cause calls clobber

      ld    de,6              ; point to nxt entry in bt tbl
      add   iy,de             ; six bytes per entry
      ld    c,(iy)                ; put track no. in bc-reg
      ld    b,(iy+1)          ; each entry is a word

      ld    a,b               ; check if trk is zero
      or    c                 ; or high and low byte together
      jr    z,bootdone        ; if trk = zero then done

      call  settrk

      ld    c,(iy+2)          ; get nxt sec to rd frm bt tbl
      ld    b,(iy+3)          ; even sectors are store as words
      dec   bc                ; pre decrement it, increment later
      ld    (nextsc),bc       ; save it

nxtsec:
      pop   bc                ; ccp addr is tos, put in bc for setdma
      push  bc                ; save ccp current addr on stack
      call  setdma                ; set dma addr as ccp cur addr

      scf                     ; clear carry flag, first set it to 1
      ccf                     ; then complement it, now zero for sbc

      ld    bc,(nextsc)       ; get next sector to read
      inc   bc                ; update it, this is where we inc later

      ld    l,(iy+4)          ; get last sector frm bt tbl
      ld    h,(iy+5)          ; get high byte of last sec
      sbc   hl,bc             ; cmp w nxt sec to rd
      pop   hl                ; fix stack in case done w. trk
      jr    c,nxttrk          ; if lst sec to rd = nxt sec, dne w trk

      ld    (nextsc),bc       ; save updated next sector
      ld    de,128                ; update ccp curr addr
      add   hl,de             ; we rd 128 byt sec at a time
      push  hl                ; save updated ccp on stack

      ld    de,(bootxlt)          ; now do actual read of sector
      call  sectran               ; ld de w tran vec addr, bc w. sec no
      ld    c,l               ; translated sector returned in hl
      ld    b,h               ; put sector no in bc for setsec
      call  setsec                ; setsec to read
      call  read              ; read sec, and put at ccp curr addr
      or    a                 ; check if read returned an error
      jr    nz,booterr        ; if error then exit warm boot
      jr    nxtsec                ; else loop to get next sector

bootdone:
      ld    a,0c3h                ; set page zero jump vectors
```

```
      ld    (00),a                  ; jp instruction = 0c3h
      ld    hl,wboota         ; load hl w. warm boot address
      ld    (01h),hl          ; loc. 0 = jp wboota

      ld    (05),a                  ; set bdos jump vector
      ld    hl,(bdosjmp)            ; ld hl w. bdos start addr
      ld    (06h),hl          ; loc. 5 = jp bdos

      ld    a,(cdisk)         ; pass current drive and user to ccp
      ld    c,a               ; ccp expects drv in c-reg

      ld    hl,(ccpstart)           ; set up for jp (hl)
      jp    (hl)              ; go to CP/M

nextsc:     dw   0                  ; nxt sec to rd during wboot
bootxlt: dw 0                 ; translate vector address

booterr:
      ld    hl,badbootmsg           ; let user know whats going on
      call  pmsg              ; send msg to console
      halt                    ; halt cpu so must reset system

badbootmsg:
      db    0dh,0ah,'Warm boot error-reset system'
crlf:
      db    0dh,0ah,'   $'

;
; Console/list drivers
; All have entry A=driver number, other parameters per CP/M
; A=0 serial port 0; A=1 serial port 1; A=2 parallel (list only)
;
; standard console/list routines for CPU serial ports
;

cpuio       equ   (($ AND 0FF00H) + 0100H) - $)
      ds    cpuio,0

cpu:
      jp    pserin            ;
      jp    serin       ; these three go direct to serial
      jp    serout            ;
      jp    lstout
      jp    pollst

pserin:

;
;     poll serial in-return A=0ff if char ready, else 0
;

      add   A,A

      if    intrin
```

```
      jr    z,polbuf          ; for first serial port
      endif

      inc   A                 ; A = command port
      ld    C,A
      in    A,(C)
      and   1
      ret   z                 ; no character waiting
      ld    A,0ffh
      ret

      if    intrin
polbuf:
      ld    hl,(outptri)            ; get pointers input and output locs in
buffer
      ld    a,h
      sub   l                 ; any thing in buffer
      ret   z                 ; nothing there
      ld    a,0ffh                ; there is something
      ret
      endif

serin:
      ld    B,A
      call  pserin
      ld    A,B
      jr    z,serin                 ; loop until character received
      add   A,A

      if    intrin
      jr    z,bufin                 ; for first serial port
      endif

      ld    C,A
      in    A,(C)
      and   7fh               ; mask high order bit
      ret

      if    intrin
bufin:
      di                      ; disable interrupts for saftey
      ld    h,0               ; make into offset to retrieve from
      ld    de,inpbuf         ; get address of buffer
      add   hl,de             ; get address
      ld    a,l               ; update pointer
      inc   a
      and   inbfsz-1
      ld    (outptri),a
      ei                      ; ok to turn interrupts on now
      ld    a,(hl)                ; get char
      ret                     ; all done

recv0int:
      ld    (svstk),SP        ; save user stack pointer
```

```
      ld    SP,locstk
      push  HL                ; save registers
      push  DE
      push  AF
      ld    HL,(outptri)           ; check if overflow
      inc   H
      ld    A,L
      and   inbfsz-1
      cp    H
      jò    nz,rcv0±          ; skið iæ rooí foò more
      IN    A,(0)             ; clear input port
      xor   A                 ; clear channel no.
      ld    C,07              ; send bell
      call  serout
      jr    rcv02             ; exit

rcv01:
      ld    de,(inptri)       ; get pointer to store address
      ld    d,0
      ld    hl,inpbuf         ; get address of base of buffer
      add   hl,de             ; find next address in buffer
      in    a,(0)             ; get input char
      and   7fh               ; mask out parity
      ld    (hl),a                ; save char
      ld    a,l               ; update pointer
      inc   a
      and   inbfsz-1
      ld    (inptri),a
rcv02:
      pop   AF
      pop   DE
      pop   HL                ; restore registers
      ld    SP,(svstk)
      endif

      if    ints              ; enable for bad interrupts
intret:
      ei                      ; enable interrupts
      reti                    ; return from interrupt
      endif

serout:
      ld    B,C               ; character to output
      add   A,A

      if    introut
      jr    z,bufout          ; for first serial port if interrupts
      endif

      inc   A
      ld    C,A
serst:
      in    A,(C)
      and   4
```

```
      jr    z,serst
      dec   C
      out   (C),B
      ret

      if    introut
bufout:
      di                      ; kill interrupts for a bit
      ld    hl,(inptro)       ; get buffer pointers
      ld    a,l               ; anything in buffer
      sub   h
      jr    z,chkout          ; if not go check if anything in transmitter
      ei                      ; interrupts on again for a bit
bfot1:
      ld    hl,(inptro)       ; get buffer pointers
      inc   l                 ; any room in buffer
      ld    a,l
      and   outbfsz-1
      sub   h
      jr    z,bfot1               ; wait if not
      di                      ; turn interrupts off some more
      dec   l                 ; set up pointer into buffer
      ld    h,0
      ld    de,outbuf
      add   hl,de
      ld    (hl),c                ; put char into buffer
      ld    a,l               ; update input pointer
      inc   a
      and   outbfsz-1
      ld    (inptro),a
      ei                      ; interrupts on
      ret                     ; and return

chkout:
      in    a,(1)             ; get transmitter status
      and   4                 ; is transmitter empty
      jr    z,bfot1               ; if not go put char into buffer
      ld    a,c               ; otherwise send char
      out   (0),a
      ei                      ; and now return with interrupts on
      ret

tran0int:
      ld    (svstk),SP        ; save user stack pointer
      ld    SP,locstk
      push  HL                ; save registers
      push  DE
      push  AF
      ld    hl,(inptro)       ; anything more to send
      ld    a,l
      sub   h
      jr    z,trret               ; if not leave
      ld    l,h               ; get address of next char to send
      ld    h,0
```

```
      ld    de,outbuf
      add   hl,de
      ld    a,(hl)                ; get char to send
      out   (0),a             ; send it
      inc   l                 ; update pointer
      ld    a,l
      and   outbfsz-1
      ld    (outptro),a
      jr    trint1
trret:
      ld    a,28h             ; reset tx interrupt
      out   (1),a
trint1:
      if    intrin
      jp    rcv02             ; if interrupt input use that return
      else
      pop   AF                ; restore registors
      pop   DE
      pop   HL
      ld    SP,(svstk)
      ei                      ; interrupts back on
      reti                    ; and return
      endif                   ; intrin
      endif                   ; introut

;
;     these are the drivers modified for use with the printer
;

lstout:
      cp    2                 ; parallel?
      jr    z,plist
      ld    B,A               ; must be serial
      ld    E,C
lstot1:
      call  plserout
      or    A
      ld    A,B
      jr    z,lstot1
      dec   C
      out   (C),E
      ret

pollst:
      cp    2
      jr    z,polplist

plserout:
      add   A,A               ; serial out poll for prn-chk bsy line
      inc   A
      ld    C,A
      ld    A,10h             ; rset DART latch to get bsy cur val
      out   (C),A
      in    A,(C)
```

```
      and   0ch
      cp    0ch
      ld    A,0ffh
      ret   z
      cpl
      ret

polplist:

;
;     parallel printer driver poll
;

      ld    A,(frstpar)       ; first time parallel called?
      or    A
      jr    z,noinit          ; no, interface already initialized
      out   (pioac),A         ; set channel A to output
      ld    A,0cfh
      out   (piobc),A         ; set channel B to bit mode
      ld    A,00011111b       ; bit 7-5 output, 4-0 input
      out   (piobc),A
      xor   A
      out   (piobd),A
      ld    (frstpar),A       ; reset first
noinit:
      in    A,(piobd)
      and   1                 ; test busy bit
      ret   z                 ; printer busy
      ld    A,0ffh
      ret                     ; printer not busy

plist:                              ; parallel printer driver
      call  polplist
      or    A
      jr    z,plist
      ld    A,C
      out   (pioad),A         ; character on PIO A channel
      ld    A,80h
      out   (piobd),A         ; strobe printer
      xor   A
      out   (piobd),A         ; reset strobe
      ret

frstpar: db 0fh               ; frst init code for parallel
                              ; prn flg that PIO has not been init
;-------------
; SELECT DISK
;-------------

seldsk:
      ld    a,c               ; c contains logical drive no.
      ld    (seklrv),a        ; save it to see if drives changed

      ld    hl,drvmap-6       ; convert logical drv to dphadr
```

```
      ld    de,006                ; size of drive map entry
dphlp:
      add   hl,de             ; point to start of drive map
      dec   a                 ; drv map addr point to dph s
      jp    p,dphlp               ; loop until at correct drv map entry
      ld    (smapadr),hl          ; save current drv map addr for chkown

      ld    a,(hl)                ; check if logical drive configured
      cp    0ffh              ; if =0ff then illegal drive
      jr    z,badsel          ; return w. zero in hl

      inc   hl                ; point to phy drv entry in map
      inc   hl                ; point to dph addr entry in drv map
      ld    e,(hl)                ; put dph addr into d and e
      inc   hl                ; now high byte
      ld    d,(hl)                ; save it in d
      push  de                ; save dph addr on stack for exit

      call  chkhrd                ; check if hard disk
      jr    z,movddb          ; if hard then dont try to login fpy

      call  logfpy                ; login fpy (get dsk den, set dpb ect.)
      jr    z,movddb          ; jmp ovr bad sel if fpy now logged in
      pop   hl                ; could not login floppy, clean stack

badsel:
      ld    hl,0              ; return w. bad select value
      ret

movddb:
      call  getddb                ; get addr of dsk deblk blk in hl

      ld    de,dbconst        ; move dblk pars to usage area
      ld    bc,13d                ; 14 bytes of dblk pars
      ldir                    ; move as block

      pop   hl                ; restore dph addr to hl

      ret


;---------
; SET DMA
;---------

setdma:
      ld    (dmaadr),bc       ; save dma addr here
      ret

;------
; HOME
;------

home:
```

```
      ld    bc,0              ; fall through to settrk

;-----------
; SET TRACK
;-----------

settrk:

      ld    (sektrk),bc       ; bc contains selected track no.
      ret

;--------------------
; SECTOR TRANSLATION
;--------------------

sectran:
      ld    a,d               ; de contains sec trn vec addr
      or    e                 ; chk if zero
      jr    z,sectrnx         ; if de =0 then no tran

      ex    de,hl             ; put tran vector address in hl
      add   hl,bc             ; add in sector no. offset
      ld    c,(hl)                ; get translated sector number
      ld    b,0               ; trn sec no range, 256 > sec no. >= 0
sectrnx:
      ld    l,c               ; return tran or non-tran sec no. in hl
      ld    h,b               ; now high byte of sectro no.
      ret

;------------
; SET SECTOR
;------------

setsec:
      ld    (seksec),bc       ; bc contains selected sector no.
      ret

;-------------
; READ SECTOR
;-------------

read:
      xor   a                 ; clear accumulator
      ld    (rtcnt),a         ; reset retry count
      ld    a,1               ; set operation to read
      ld    (oper),a          ; save it for when we do xfer

      call  deblk             ; deblock phy sec, buf adr, and blk sec
      call  inbuf             ; check if new sector is in buf
      call  xfer              ; transfer data out of buf into dma

      ld    a,(rtcnt)         ; if rtcnt not zero then error

      ret
```

```
;--------------
; WRITE SECTOR
;--------------

write:
      xor   a                 ; set operation to write
      ld    (oper),a          ; save it for when we do xfer
      ld    (rtcnt),a         ; reset operation retry count

      ld    a,c               ; enter w. containing write type
      ld    (wrttyp),a        ; save for write to directory check
      cp    wrtfre                  ; check if write to free block
      jp    nz,useblk         ; jump if write to used block
freblk:
      call  deblk             ; deblock phy sec, buf adr, and blk sec
      call  flush             ; flush buffer of previouse contents
      call  rsetusg                 ; set all block records as free
      call  sethst                  ; set hst vars to deblk vars
      call  xfer              ; transfer dat out of dma into buf
      call  update                  ; update usage vars to show blk used
      jp    dirchk                  ; check if directory write
useblk:
      call  deblk             ; deblock phy sec, buf adr, and blk sec
      call  inbuf             ; check if sec in buff, if not get it
      call  xfer              ; transfer data out of buf into dma
      call  update                  ; update usage vars to show blk used
dirchk:
      ld    a,(rtcnt)         ; check if error on write
      and   a                 ; if err, don't set write pend flg
      jr    nz,wrtend         ; jmp over flush if err

      ld    a,0ffh                  ; always a write pending after a write
      ld    (wrtpnd),a        ; set write pending flag

      ld    a,(wrttyp)        ; check if write to directory
      cp    wrtdir                  ; compare with write directory var
      call  z,flush                 ; if wrttyp = wrtdir then flush buf
wrtend:
      ld    a,(rtcnt)         ; pass bdos 0 if no err, not 0 if err
      ret

;-----------------------------------
; DEBLOCK PHY SEC, BUF ADDR, BLK SEC
;-----------------------------------

deblk:
      ld    a,(pdrv)          ; get phy drv from dsk deblk parameters
      ld    (sekprv),a        ; save it as sek phy drv

      ld    a,(secshf)        ; convert sek sec to phy sec
      ld    de,(seksec)       ; secshf is log2 cpm sps
      call  shfr16                  ; shift de-reg, a-reg times
      ld    a,(hstspt_1)            ; de now contains sec rel to cyl
```

```
      and   e                 ; mask for sec rel to platter
      ld    (dsec),a          ; save it as deblocked sector

      ld    a,(hdshf)         ; convert seksec to phy head
      ld    de,(seksec)       ; hdshf is log2 cpm spt
      call  shfr16                ; hd no is high bits, so no msk needed
      ld    a,(hdoff)         ; add in head off set for partioning
      add   a,e               ; cartriage drv must be part. by hds
      ld    (dhd),a               ; save it as deblocked head no.

      lä    a,(cpmsps_1©          » converô sekseã tï hsô buæ no.
      ld    de,(seksec)       ; mask out low order bits of sec no
      and   e                 ; since buf no <= 8, need only lsb
      ld    e,a               ; mul buf no x128 to get rel adr in buf
      ld    d,0               ; zero out high order of multiplicitan

      ld    a,7               ; convert hst buf no. to hst buf adr
      call  shfl16                ; mult by 128 (shift lf 7)
      ld    hl,hstbuf         ; base addr of hst buf
      add   hl,de             ; add in offset
      ld    (dadr),hl         ; save it as deblocked buf addr

      ld    a,(blkshf)        ; convert seksec to blk no.
      ld    de,(seksec)       ; blkshf = log2 cpm spb, what we are
      call  shfr16                ; doing is, bl=int(sesc/cpmspb)
      ld    (dblk),de         ; save as blk no. on this trk

      ld    a,(hstspb_1)          ; convert seksec to hst sec no. in blk
      ld    de,(dsec)         ; must use sec just deblocked
      and   e                 ; and deblk sec w. hst spb
      ld    (dblsec),a        ; save it as deblocked block sec

      ret

;----------------------------
; CHECK IF NEW SEC IS IN BUF
;----------------------------

inbuf:
      ld    a,(seklrv)        ; check if new drv = old drv
      ld    hl,hstlrv         ; first check if log drives are same
      cp    (hl)              ; compare w. last accessed drv
      jp    nz,difblk         ; if drvs are dif, then so is blk

      ld    de,(sektrk)       ; check if new trk = old trk
      ld    hl,(hsttrk)       ; this is the old trk
      call  cmp16             ; so far drv same
      jp    nz,difblk         ; if trks are dif, then so is blk

      ld    a,(dhd)               ; check if new head = old head
      ld    hl,hsthd          ; so far drv, trk same
      cp    (hl)              ; if dif hd, could be
      jp    nz,ckblk          ; same blk if blk size > phy trk siz
```

```
      ld    a,(dsec)          ; check if new sec = old sec
      ld    hl,hstsec         ; this is the old sec
      cp    (hl)              ; so far drv, trk, hd same
      jp    nz,ckblk          ; could be same blk even if dif sec

      call  sethst                ; everything same, set buf addr though

      ret

ckblk:
      ld    hl,(dblk)         ; check if new sec is in same blk
      ld    de,(hstblk)       ; if the new blk equal old blk
      call  cmp16             ; then do not reset usage vars
      jp    nz,difblk         ; regardless fall through to prerd

samblk:
      call  flush             ; sam blk, but dif blk sec so flush old
      ld    hl,usgblk         ; check if this sec free in block
      ld    de,(dblsec)       ; look in block usage vector
      ld    d,0               ; index down to correct entry in vector
      add   hl,de             ; hl now contains addr of blk usg entry

      ld    b,(hl)                ; get blk usage flag
      ld    a,(oper)          ; get oper
      or    b                 ; if (sec not fre) or (oper is read)
      jp    nz,prerd          ; then need to pre-read sector

nprerd:
      call  sethst                ; sector not allocated
      ret
difblk:
      call  flush             ; flush any old stuff out
      call  setusg                ; sector not in block
prerd:
      call  sethst                ; sector in blk but alloc, or dif blk
      call  rdhst             ; read in new setor
      ret

;--------------------
; FLUSH HOST BUFFER
;--------------------

flush:
      ld    a,(wrtpnd)        ; check if host buffer active
      and   a                 ; wrtpnd =0 if inactive, =0ff if active
      call  nz,wrthst         ; physicaly flush buffer if active

      xor   a                 ; clear write pending
      ld    (wrtpnd),a        ; host buffer now in active

      ret

;----------------------------------
; SET DEBLOCKED VARS TO HOST VARS
```

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

sethst:
      ld    hl,sekvars+2          ; blk move sek & dblk vars to hst vars
      ld    de,hstvars       ; dont need seksec so sekvars+2
      ld    bc,16d                ; this is how many to move
      ldir                   ; set host variables
      ret

;--------------------------------------
; TRANSFER DATA TO/FROM BUFFER AND DMA
;--------------------------------------

xfer:
      ld    bc,080h               ; transfer 128 bytes
      ld    de,(dmaadr)      ; load cpm addr (dma addr storage)
      ld    hl,(hstadr)      ; load buf addr

      ld    a,(oper)         ; check if read or write operation
      or    a                ; oper =0 if read, =1 if write
      jp    nz,transf        ; jump if read

      ex    de,hl            ; read operation so switch directions
transf:
      ldir                   ; send 128 byte block

      ret

;--------------------------------
; SET OR RESET BLOCK USAGE FLAGS
;--------------------------------

rsetusg:
      ld    b,0              ; set all block sectors in blk to free
      jp    setblk                ; sec in blk not allocated if =0
setusg:
      ld    b,0ffh                ; set all block sectors in blk to used
setblk:
      ld    a,(hstspb_1)          ; get no of host sector per block
      ld    hl,usgblk        ; get addr of blk sector usage vector
setflg:
      ld    (hl),b                ; set or reset block usage flag
      inc   hl               ; point to next flag loc
      dec   a                ; dec count of block sectors to go
      jp    p,setflg         ; loop if more to set
      ret

;------------------------------
; UPDATE BLOCK USAGE VARIABLES
;------------------------------

update:
      ld    de,(blksec)      ; we have just written to a sec in blk
      ld    d,0              ; so set block sec usage flag to used
```

```
      ld    hl,usgblk         ; first point to flag
      add   hl,de             ; de contains sector no. in block
      ld    a,0ffh                 ; 0ff means sector in block is not free
      ld    (hl),a                 ; set flag
      ret

;------------
; WRITE HOST
;------------

wrthst:
      call  chkhhrd                ; check if host logical dsk is hard dsk
      jr    z,hwrthst         ; jmp if hard to hard wrt host else fpy

fwrthst:
      call  fsettsk                ; set up cntr reg & gen housekeeping
      call  fwrtsec                ; write sector to floppy
      call  retrys                 ; check floppy cntr status for errors
      jr    nz,fwrthst        ; loop if error
      ret

hwrthst:
      call  hsettsk                ; set up hdc1001 ctrler registers
      call  hwrtsec                ; send wrt cmd and data to ctrler
      call  retrys                 ; chk hdc1001 status for errors
      jp    nz,hwrthst        ; loop if error
      ret

;-----------
; READ HOST
;-----------

rdhst:
      call  chkhhrd                ; chk host dsk selected is hard disk
      jr    z,hrdhst          ; set zero flag if hard disk

frdhst:
      call  fsettsk                ; set controller reg & do housekeeping
      call  frdsec                 ; read a sector from floppy
      call  retrys                 ; chk status for errors
      jr    nz,frdhst         ; loop if error and under retry count
      ret

hrdhst:
      call  hsettsk                ; first set up controller registers
      call  hrdsec                 ; transfer data
      call  retrys                 ; check for errors
      jp    nz,hrdhst         ; retrys sets zero flg if no errors
      ret

;--------------------
; HARD DISK SET TASK
;--------------------
```

```
hsettsk:
      ld    a,(hstprv)       ; get physical drive no.
      ld    b,a              ; save in b so can use a for mem fetch
      sla   b                ; rotate phy drive to correct position
      sla   b                ; drive is expected in bits 3 and 4
      sla   b                ; last shift for drive
      ld    a,(hstsdh)       ; get sdh sect size setting
      or    b                ; or it in w. rotated phy drive no
      ld    b,a              ; save result in b
      ld    a,(hsthd)        ; get host head no.
      or    b                ; or it in w. drv and sec siz
      out   (hsdh),a         ; send it siz drv hd register

      ld    bc,(hsttrk)      ; host track is really host cylinder
      ld    a,b              ; move msb to a-reg for out inst
      out   (hcylhi),a       ; send high byte to cylhi reg
      ld    a,c              ; move lsb  to a-reg
      out   (hcyllow),a      ; send to cyl low

      ld    a,(hstsec)       ; get host sector no
      out   (hsec),a         ; send to sector register

      ret

;----------------------
; FLOPPY DISK SET TASK
;----------------------

fsettsk:
      ld    a,(hstsdh)       ; get dsd setting
      res   2,a              ; reset no. of sides on drive (0-1)
      ld    bc,(hsthd)       ; get deblocked head
      ld    b,c              ; save it in b-reg
      sla   c                ; rotate it to correcd dsd-reg setting
      sla   c                ; bit 2 is where head select goes
      or    c                ; or hd sel w. phy drv, den, & drv siz
      out   (fdsd),a         ; send it to floppy controller reg

      ld    hl,trkvec        ; trk vec contains hd loc of ldrvs
      ld    de,(hstlrv)      ; get logical drive no.
      ld    d,0              ; prepare for double add
      add   hl,de            ; add in logical drive as offset
      ld    a,(hl)               ; get track no. where head positioned
      out   (ftrk),a         ; set trk reg to current hd position

      ld    d,a              ; check if seek needed, d =last trk
      ld    a,(hstlrv)       ; get new drive
      ld    e,a              ; e =drive to seek

      ld    a,(lastdrv)      ; last drive
      cp    e                ; drive to seek
      jr    nz,diftrk        ; do sek to reload heads

      ld    a,(lasthd)       ; get last head used
```

```
      cp    b               ; cmp w. new hd sel saved at begining
      jr    nz,diftrk       ; if hds are dif, must do sek to ld hds

      ld    a,(hsttrk)      ; hds, drv same, chk if trks same
      cp    d               ; cmp new trk w. old trk
      jr    z,samtrk        ; if trks dif, must do sek to new trk

diftrk:
      ld    a,e             ; save new drive as old drive
      ld    (lastdrv),a     ; this will mak sure we ld hds if need
      ld    a,b             ; save new head as old head
      ld    (lasthd),a      ; see you next time around

      ld    a,(hsttrk)      ; trk to seek passed in a-reg
      ld    (hl),a              ; save it in trk vec as new hd position
      call  fseek           ; go find track

samtrk:
      ld    a,(hstsec)      ; now set sector no. reg.
      ld    b,a             ; save sec. to rd/wrt in b-reg
      ld    a,(hstsdh)      ; check density of dsk
      bit   3,a             ; density is bit 3
      jr    z,fsndsec       ; jmp ovr if single density
      inc   b               ; dbl den, so mak sec no frm 0-7 to 1-8

fsndsec:
      ld    a,b             ; put sec no back in a-reg
      out   (fsec),a        ; send it to fpy controller sec reg
      ret

;------------------------
; HARD DISK WRITE SECTOR
;------------------------

hwrtsec:
      ld    a,cmdwrt        ; load a-reg w. cmd to wrt sec
      out   (hcmd),a        ; send wrt cmd to hdc-1001
      call  snddta              ; send data to cmd reg
      call  polbsy              ; wait until contrller done w. wrt
      ret

;------------------------
; HARD DISK READ SECTOR
;------------------------

hrdsec:
      ld    a,cmdrd             ; load a with command to read sector
      out   (hcmd),a        ; send read command to command reg
      call  polbsy              ; wait until not busy
      call  rxdta           ; transfer data from cnt buf to hst buf
      ret

;-----------
; SEND DATA
```

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

snddta:
      ld    a,(hstsiz)       ; this is how many bytes
      ld    b,a              ; set up for otir
      ld    c,hdata              ; this is where the data comes fr
      ld    hl,hstbuf        ; this is where the data goes
      otir                   ; block move data in c to (hl)

      ld    a,(hstsiz+1)         ; check if moving 512 bytes
      and   2                ; msb =2 if 512,set flag if not zero
      jp    z,sndout         ; if zero then move another 256 bytes
      otir                   ; b is already 0, hl and c are set
sndout:
      ret

;--------------
; RECIEVE DATA
;--------------

rxdta:
      ld    a,(hstsiz)       ; this is how many bytes to move
      ld    b,a              ; set up for inir
      ld    c,hdata              ; this is where the data comes from
      ld    hl,hstbuf        ; this is where the data goes
      inir                   ; block move data into hst buf

      ld    a,(hstsiz+1)         ; check if moving 512 bytes
      and   2                ; msb would be 2 if 512 bytes
      jp    z,rxout              ; jump over if moveing 128 or 256 only
      inir                   ; b should be zero, hl and c set also
rxout:
      ret

;-----------
; POLL BUSY
;-----------

polbsy:
      in    a,(hstatus)      ; read status port
      and   a                ; set flags
      jp    m,polbsy         ; loop if busy bit set
      and   errbit               ; mask for error bit
      ret

;---------------------
; SET RETRY CONDITION
;---------------------

retrys:
      call  chkhhrd
      jr    z,hretry

fretry:
```

```
      in    a,(fstat)
      and   a
      jr    z,rtout
      jr    rterr

hretry:
      in    a,(hstatus)       ; read status register
      and   errbit                  ; mask for error bit
      jr    z,rtout                 ; jump to exit rtry

rterr:
      ld    a,(rtcnt)         ; get no. of retrys so far
      inc   a                 ; increment retry count
      ld    (rtcnt),a         ; save it for next time
      cp    rtrys             ; set not z flg, unless rtcnt = rtrys
      ret   nz                ; return w. flag set or reset

      xor   a                 ; clear write pending flag
      ld    (wrtpnd),a        ; don't try and flush buffer if wrt err

      ld    bc,0ffffh
      ld    (hsttrk),bc
      ret

rtout:
      xor   a                 ; clear zero flag
      ld    (rtcnt),a         ; clear retry cnt in case had to retry

      ret                     ; return w. no errors

;-------------------------------
; CHECK IF HARD DRIVE SELECTED
;-------------------------------

chkhrd:
      ld    hl,(smapadr)            ; load hl w. seek drive's map addr
      ld    a,(hl)                  ; first entry is drive type
      and   a                 ; hard disk =0, ret w. z-flg set if hrd
      ret

;---------------------------------
; CHECK IF HOST DRIVE IS HARD DISK
;---------------------------------

chkhhrd:
      ld    hl,(hmapadr)            ; load hl w. hst drive's map addr
      ld    a,(hl)                  ; fetch drive type byte
      and   a                 ; set z-flag if hard disk
      ret

;--------------------
; LOGIN FLOPPY DRIVE
;--------------------
```

```
logfpy:
      ld    hl,logvec         ; load hl w. login vec base addr
      ld    de,(seklrv)       ; fetch logical drv no.
      ld    d,0               ; zero out high byt to prep for dbl add
      add   hl,de             ; add in offset to login vector

      ld    a,(hl)                ; get log vec entry for logical drv
      ld    (sdhdsd),a        ; put in wrking var in case alrdy loged
      bit   7,a               ; logvec (ldrv) =0ff if not loged in

      push  hl                ; save login vector addr
      jr    z,logedin         ; jmp ovr if already logged in
flogin:
      call  getds             ; get fpy drv siz (8-5) & pdrv # (0-3)
      call  frestore          ; restore fpy drv hds
      jr    nz,logerr         ; ret if drv not rdy for select error

      calì  getden                ; get dsk density (sgl-dbl)
      call  getsd             ; get number of sides on dsk (0-1)
      call  setdpb                ; set dpb addr in dph to ind. den & sd
      call  settran               ; set sec trn vec adr in dph if sgl den

logedin:
      call  setddb                ; set ddb addr in drv map

      pop   hl                ; restore logvec addr
      ld    a,(sdhdsd)        ; put new dsd reg setting in a-reg
      ld    (hl),a                ; save it as a logvec entry
      xor   a
      ret
logerr:
      pop   hl                ; clean stack and return w. nz set
      ret

;-------------------------------------------
; GET DRV SIZE AND PHY DRV NO. FROM DRV MAP
;-------------------------------------------

getds:
      ld    hl,(smapadr)            ; point to drvtyp in dmap
      ld    a,(hl)                  ; put drive type in a-reg
      cp    2                 ; if =2 then 5"- 96 tpi fpy drv
      ld    b,0               ; b-reg contains dsd setting
      jr    z,dsiz596         ; jmp to 96 tpi set-up
      jr    c,dsiz548         ; if =1 then 5" - 48 tpi fpy drv
dsiz8:
      jr    getpdrv                 ; if =3 then 8" fpy drv
dsiz596:
      set   5,b               ; set bit 5 to indicate 96 tpi
dsiz548:
      set   4,b               ; set bit 4 to indicate 5" fpy
getpdrv:
      inc   hl                ; phy drv no. is second entry in dmap
      ld    a,(hl)                ; put phy drv no. in a-reg
```

```
      or    b               ; or in drive size bit setting
      ld    (sdhdsd),a      ; save it in den - dsiz - dno
      out   (fdsd),a        ; set floppy den,dsiz,dno in controller

      ret

;------------------
; GET DISK DENSITY
;------------------

getden:
      ld    a,1             ; do sek to trk 1, sgl den
      call  settkv
      call  fseek           ; do seek to trk 1
      reô   z               ; return if succesful (dsk is sgl den)
dbldens:
      ld    hl,sdhdsd       ; else assume dbl den (saves time)
      set   3,(hl)              ; set double density bit setting
      ret

;---------------------
; GET NUMBER OF SIDES
;---------------------

getsd:
      ld    a,(sdhdsd)      ; get dsd reg setting as calc so far
      bit   3,a             ; test if single density
      ret   z               ; if bit 3 =0, then sgl den

      set   2,a             ; set side select bit
      out   (fdsd),a        ; send it to the floppy controller

      ld    a,1             ; do seek, trk 1, side 1
      call  settkv
      call  fseekntst       ; sek w.out rdy tst,(never rdy if 1 sd)
      ret   nz              ; jump ovr if error on sek

      call  frdadr              ; must rd adr to get side #
      ret   nz              ; sgl hd drvs rd hd0 when hd1 selected

      ld    a,(sideno)      ; get side # from track id buffer
      and   a               ; if side # =0 when hd1 selected then
      ret   z               ; ret if single headed drive

      ld    hl,sdhdsd       ; sek to trk 1, side 1 OK
      set   2,(hl)
      ret

;----------------
; SET DPB IN DPH
;----------------

setdpb:
      call  getdph              ; get dph adr of sekldrv in hl
```

```
      ld    de,0ah                ; pnt to dpb entry of dph
      add   hl,de             ; dbp addr in dph is 10th entry
      push  hl                ; save addr of dpb address on stk
tstsiz:
      ld    a,(sdhdsd)        ; fetch dsd register setting

      ld    hl,fdpb8          ; ld hl w. dpb base for 8" floppy
      bit   4,a               ; check drv siz bit
      jr    z,tstden          ; jump if 8" drv

      ld    hl,fdpb548        ; ld hl w. dpb base for 5"-48 tpi fpy
      bit   5,a               ; check 96 tpi bit (** not standard **)
      jr    z,tstden          ; jump if 48 tpi drive

      lä    hl,fdpb596        ; ld hl w. dpb base for 5"-96 tpi fpy
tstden:
      bit   3,a               ; test density bit of dsd
      jr    z,movdpb          ; sgl den dpb = dpb base
setdd:
      ld    de,15d                ; double den, sgl side is next dpb
      add   hl,de             ; index down to next ddb
tstsid:
      bit   2,a               ; test if double sided
      jr    z,movdpb          ; hl is pointing to dbl den, sgl sd
setds:
      add   hl,de             ; dbl den, dbl side is next dpb
movdpb:
      pop   de                ; this is where dpb addr goes in dph
      ex    de,hl             ; de =dpb addr, hl =loc in dph of dpb
      ld    (hl),e                ; save low byte of addr in dph
      inc   hl                ; inc pointer to high byte of addr
      ld    (hl),d                ; save high byte of addr in dph
      ret

;---------------------------------------
; SET TRANSLATION VECTOR ADDRESS IN DPH
;---------------------------------------

settran:
      call  getdph                ; return w dph addr in hl-reg

      ld    a,(sdhdsd)        ; get dsd setting
      bit   3,a               ; chk denisty bit
      ld    de,0              ; initalize de-reg w. tran vec addr
      jr    nz,sett               ; jump if double density, (no trn vec)

      ld    de,tran8          ; initalize tran vec addr for 8" tran
      bit   4,a               ; chk drive size bit, =0 for 8", =1 5"
      jr    z,sett                ; jump if 8" drive

      ld    de,tran548        ; init tran vec addr for 5-48tpi sglden
      bit   5,a               ; check bit 5 for 96 tpi drive
      jr    z,sett                ; jump if 48 tpi drive
```

```
      ld    de,tran596       ; de gets 5"-96 tpi sgl den tran vec
sett:
      ld    (hl),e                ; save tran vec addr as #1 entry in dph
      inc   hl               ; vec addr is a word
      ld    (hl),d                ; now store high byte
      ret

;----------------------
; SET DDB IN DRIVE MAP
;----------------------

setddb:
      ld    a,(sdhdsd)       ; need this to fig what kind of dsk

      ld    hl,fddb8         ; init hl for 8" sgl sid, sgl den
      bit   4,a              ; check bit 4 for siz, 8"=0, 5"=1
      jr    z,setddd         ; jump to density chk if 8" drv

      ld    hl,fddb548       ; init hl w 5"-48 tpi sgl sid, den
      bit   5,a              ; check bit 5 for tpi, 0=48tpi, 1=96tpi
      jr    z,setddd         ; jump to chk den if 48 tpi

      ld    hl,fddb596       ; set hl w. 5"96tpi ddb addr
setddd:
      bit   3,a              ; chk bit 3 for density, 0=sgl 1=dbl
      jr    z,mvddb               ; jmp if sgl, hl alrdy has addr
      ld    de,13            ; point to next ddb, (sgl sid, dbl den)
      add   hl,de            ; add in offset to next ddb

      bit   2,a              ; chk # sides, 0 =1 side, 1 =2 side
      jr    z,mvddb               ; jump ovr if sgl sided
      add   hl,de            ; add in offset again for next ddb
mvddb:
      push  hl               ; save ddp addr
      ld    hl,(smapadr)          ; destination of ddb addr is in dmap
      ld    de,4             ; fourth entry of logical drv in dmap
      add   hl,de            ; hl now has dmap addr destination
      pop   de               ; restore ddb addr into de-reg

      ld    (hl),e                ; hl-reg now has dmap ddb-entry addr
      inc   hl               ; pnt to where high byte goes
      ld    (hl),d                ; finish storing ddp addr in dmap

      ld    hl,2             ; offset into ddb
      add   hl,de            ; add in base address of ddb
      ld    a,(sdhdsd)       ; store sdhdsd in pdrv of ddb
      ld    (hl),a                ; for when do blk move fm. ddb -> ddcon
      ret

;----------------------
; RESTORE FLOPPY HEADS
;----------------------

frestore:
```

```
      ld    a,(sdhdsd)       ; get physical drive to do restore on
      out   (fdsd),a         ; send it to floppy controller

      call  tstrdy                 ; test it drive is ready
      ret   nz               ; return if user abort

      di                     ; disable console interupts
      ld    a,frst                 ; load floppy restore at proper stp rte
      out   (fcmd),a         ; send restore command
      in    a,(fwait)        ; wait until command finished executing
      ei

      xor   a                ; set track vec to indicate at trk0
      call  settkv
      ret

;---------------------
; SEEK TRACK IN A-REG
;---------------------

fseek:
      ld    b,a              ; track to seek passed in a-reg
      call  tstrdy                 ; wait for drive to become ready
      ret   nz               ; return w nz-flay set if user abort
      ld    a,b              ; move trk no bk to a, (tstrdy uses a)

fseekntst:
      di                     ; disable console interupts for disk io
      out   (fdata),a        ; load trk to sek into data register
      ld    a,fsek                 ; load a-reg w seek cmd
      out   (fcmd),a         ; tell controller to seek the track

      call  delay            ; must delay 28 us before reading stat
      in    a,(fwait)        ; wait until done with seek
      in    a,(fstat)        ; read floppy staus register
      and   018h             ; check for crc and/or seek error
      ei                     ; return w. nz set if error

      ret

;-------------------------
; WRITE A SECTOR TO FLOPPY
;-------------------------

fwrtsec:
      in    a,(fstat)        ; first check if write protected
      bit   6,a              ; if bit 6 of stat =1, then protected
      jr    nz,wrtpro        ; tell user if protected

      ld    c,fdata                ; destination port
      ld    hl,hstbuf        ; source starting address

      di                     ; dont want to be intr or will lose dta
      ld    a,fwrt                 ; start write by sending command
```

```
      out   (fcmd),a          ; tell ctrler to wrt sec
      call  delay             ; mak sure wait 28 us before stat rd
fwrtlp:
      in    a,(fwait)         ; check if in floppy interupt
      or    a                 ; interupt is bit 7 of fwait, 0=active
      jp    p,fwrtout         ; if zero then no more data   1=nactive
      outi                    ; else send dta, inc buf addr
      jr    fwrtlp                ; loop until done

fwrtout:
      in    a,(fstat)         ; read status for errors
      and   a                 ; set nz flg if no errors
      ei                      ; enable console interupts
      ret

wrtpro:
      ld    de,wprtmsg        ; pass write prot msg in de to dskerr
      call  dskerr                ; call rout to prn msg and wait for cio
      jr    z,fwrtsec         ; try again if user didnt type ctrl-c
      ret                     ; else return w nz flg set

;---------------------------
; READ A SECTOR FROM FLOPPY
;---------------------------

frdsec:
      ld    c,fdata               ; destination port
      ld    hl,hstbuf         ; source starting address

      di                      ; disable console interupts
      ld    a,frd             ; start read by sending command
      out   (fcmd),a          ; tell ctrler to read sector
      call  delay             ; mak sure wait 28 us before rding stat
frdlp:
      in    a,(fwait)         ; wait until ctrler rdy to snd dta
      or    a                 ; check if interupt active
      jp    p,frdout          ; if bit 7 =0, then inter active
      ini                     ; else rd dta, increment buff addr
      jr    frdlp             ; loop until interupt =0

frdout:
      in    a,(fstat)         ; check for any errors
      and   a                 ; set nz-flag if errors
      ei                      ; enable cnsole interupts
      ret

;----------------------
; READ FLOPPY TRACK ID
;----------------------

frdadr:
      ld    c,fdata               ; data port to get data from
      ld    hl,trkid          ; data buffer to put data
```

```
      di                    ; dont want an interupt and lose data
      ld    a,frda                  ; floppy read address command
      out   (fcmd),a        ; send command to controller
      call  delay           ; must wait 28ms before reading stat
frdalp:
      in    a,(fwait)       ; check if done with read
      or    a               ; bit 7 is set if done, fpy ctr intr
      jp    p,frdax                 ; exit loop if done
      ini                   ; read data into buffer
      jr    frdalp                  ; loop until done
frdax:
      in    a,(fstat)       ; get status and check for errors
      and   018h            ; check if seek or crc error
      ei                    ; we can enable interupts now

      ret

;-------------------------------
; TEST IF FLOPPY DRIVE IS READY
;-------------------------------

tstrdy:
      ld    a,0d0h                  ; reset floppy controller
      out   (fcmd),a        ; set status reg to current state

      call  delay           ; must delay before reading status

      ld    a,(sdhdsd)      ; test if 8 inch drive
      bit   4,a             ; bit 4 =0 if 8", bit 4 =1 if 5"
      ld    de,0ffffh       ; set up time out counter
      jr    z,tready        ; skip index pulse tests if 8 in. drv

      ld    hl,tstnidx      ; test for no index pulse if 5 in. drv
tstnidx:
      in    a,(fstat)       ; check no idx to make sure dsk in drv
      bit   1,a             ; bit 1 =1 if index detected
      jr    nz,decde        ; decrement counter if index found

      ld    hl,tstidx       ; test for index pulse if 5 in. drv
tstidx:
      in    a,(fstat)       ; check for an index pulse
      bit   1,a             ; bit 1 on status is =0 if no index
      jr    z,decde                 ; dec time out counter if no index yet

tready:
      ld    hl,tstrdx       ; test for drive ready 8-5 in. drv
tstrdx:
      in    a,(fstat)       ; get floppy status
      bit   7,a             ; test not ready bit in status
      ret   z               ; return if not ready bit =0

decde:
      call  delay           ; delay even move before decrement
```

```
      dec   de              ; decrement counter in de-reg
      ld    a,d             ; check if counter is zero
      or    e               ; or high - low byte together
      jr    z,notready      ; jump if time out error
      jp    (hl)            ; jump to tstnidx, tstidx or tstrdx

notready:
      ld    de,nrdymsg      ; pass disk err actual error msg
      call  dskerr                ; display msg and drv on console
      jr    z,tstrdy        ; retry if user didn't type ctrl-c
      ret

;-------------------
; FLOPPY DISK ERROR
;-------------------

dskerr:
      ld    hl,bioerr       ; point to general bios error message
      call  pmsg            ; display message on console

      ld    a,(seklrv)      ; display most recently sel drive
      add   a,041h                ; adjust to ascii
      call  pchar           ; disp drv (won't be right if flushing
                            ; buf fm dif drv)

      ex    de,hl           ; exact error msg addr passed in de
      call  pmsg            ; display this msg on console

      push  bc              ;
      call  conin           ; wait for user to type a character
      pop   bc              ;

      cp    003h            ; check if ctrl-c typed
      jr    z,diskerr       ; jmp ovr retry if ctrl-c

      xor   a               ; set z-flag
      ld    a,0d0h                ; reset floppy controler
      out   (fcmd),a        ; return to retry command
      ret

diskerr:
      and   0ffh            ; if here, ctrl-c entered
      ld    a,rtrys-1       ; set nz-flag
      ld    (rtcnt),a       ; don't do any retrys

      ret

;--------------------------
; PRINT MESSAGE ON CONSOLE
;--------------------------

pmsg:
      ld    a,(hl)                ; enter with hl pointing to msg
      cð    '$'             ; dollar sign is msg ending
```

```
      call  pchar             ; pass char to print a
      inc   hl                ; inc msg pointer
      jr    nz,pmsg                ; loop until $ found
      ret

;----------------------------
; PRINT CHARACTER ON CONSOLE
;----------------------------

pchar:
      push  hl                ;
      push  de                ;
      push  bc                ;
      push  af                ;
      ld    c,a               ;
      call  nz,conout         ;
      pop   af                ;
      pop   bc                ;
      pop   de                ;
      pop   hl                ;
      ret

;--------------------
; RESET LOGIN VECTOR
;--------------------

rstlvec:
      ld    hl,logvec         ; set hl to logvec base addr
      ld    b,16              ; set b-reg to # entrys in logvec
lveclp:
      ld    (hl),0ffh         ; reset logvec entry
      inc   hl                ; increment entry no.
      djnz  lveclp
      ret

;---------------------
; UPDATE TRACK VECTOR
;---------------------

settkv:
      ld    hl,trkvec
      ld    de,(seklrv)
      ld    d,0
      add   hl,de
      ld    (hl),a
      ret

;--------------------------------
; GET DPH ADDRESS FROM DRIVE MAP
;--------------------------------

getdph:
      ld    hl,(smapadr)           ; get logical drive base addr in dmap
      inc   hl                ; inc past drive type entry
```

```
      inc   hl                ; inc past phyical drive no. entry
      ld    e,(hl)                ; put low byte of dph addr in e
      inc   hl                ; inc to high byte
      ld    d,(hl)                ; put high byte of dph addr in d
      ex    de,hl             ; return w dph addr in hl

      ret

;--------------------------------
; GET DDB ADDRESS FROM DRIVE MAP
;--------------------------------

getddb:
      ld    hl,(smapadr)          ; logical drive base addr in in dmap
      ld    de,4              ; ddb addr for ldrv is 4 bytes down
      add   hl,de             ; add in offset
      ld    e,(hl)                ; fetch low byte of ddb addr
      inc   hl                ; inc pointer
      ld    d,(hl)                ; fectch high byte of ddb addr
      ex    de,hl             ; return w ddb addr in hl
      ret

;--------------------------------
; DELAY 28u SECONDS AT 6 mhz
;--------------------------------

delay:
      ld    a,3               ; initialize loop counter
delaylp:
      ex    (sp),hl               ; 19 cycles
      ex    (sp),hl               ; 19 cycles
      dec   a                 ;  4 cycles
      jr    nz,delaylp        ; 10 cycles = 52 cyc = 8.7 @ 6 mhz
      ret

;----------------
; SHIFT RIGHT 16
;----------------

shfr16:
      and   a                 ; shift de reg right a-reg times
shfrlp:
      ret   z
      srl   d                 ; shift msb first, bit 0 into carry
      rr    e                 ; rotate carry into bit 7 lsb
      dec   a                 ; decrement shift counter
      jr    shfrlp                ; loop util finished

;---------------
; SHIFT LEFT 16
;---------------

shfl16:
      and   a                 ; shift de-reg left a-reg times
```

```
shfllp:
      ret   z
      sla   e                 ; shf lsb first, c = bit 7, bit 0 = 0
      rl    d                 ; shf msb, bit 0 = carry
      dec   a                 ; decrement shift counter
      jr    shfllp                  ; loop until a-reg eq zero

;------------------------------
; COMPARE 16-BIT DE AND HL REG
;------------------------------

cmp16:
      ld    a,e               ; compare e-reg w. l-reg
      xor   l                 ; xor will zero accum. if e = l
      ret   nz                ; return if not equal
      ld    a,d               ; compare d-reg w. h-reg
      xor   h                 ; set zero flag if same
      ret

;------------------------------------------------
; DATA STORAGE
;------------------------------------------------

bioerr:  db 0dh,0ah,'Bios Err on $'

nrdymsg: db ': Drive Not Ready$'
wprtmsg: db ': Write Protected Disk$'

lastdrv: db 0ffh
lasthd:      db   0ffh
dmaadr:      dw   0080h             ; dma address storage
rtcnt:       db   0                 ; retry counter

logvec:      ds   16,0ffh                ; drive login vector
trkvec:      ds   16,0ffh                ; current head position on drive

sekvars:                        ; seek variables

seksec:     dw    0                 ; seek sector
smapadr:dw  0                   ; seek drive drive-map address
seklrv:     db    0                 ; seek logical drive
sekprv:     db    0                 ; seek physical drive
sektrk:     dw    0                 ; seek track

dhd:  db    0                   ; deblocked head
dsec: db    0                   ; deblocked sector
dadr: dw    0                   ; deblocked buffer address
dblk: dw    0                   ; deblocked block no. in cylinder
dblsec:     db    0                 ; deblocked host sector no. in block

dbconst:                        ; deblocking constants (calc in ddb)
secsiz:       dw  0                 ; host sector size
sdhdsd:       db  0                 ; hd - shd ; fpy - dsd
pdrv:   db  0                   ; physical drive
```

```
hdoff:        db  0                ; head offset
stprte:       db  0                ; cmd reg step rate
hstspt_1: db      0                ; host sector per track
hstspb_1: db      0                ; host sectors per block
hdshf:        db  0                ; log2 cpm spt
blkshf:       db  0                ; log2 cpm spb
hdmsk:        db  0                ; heads - 1
secshf:       db  0                ; log2 cpm sps
cpmsps_1: db      0                ; cpm sps - 1

hstvars:                      ; host drive variables

hmapadr:dw  0                 ; host drive drive-map address
hstlrv: db  0                 ; last logical drive operated on
hstprv:     db    0                 ;       physical drive
hsttrk:     dw    0                 ;       track (equiv to cylinder)
hsthd:      db    0                 ;       head
hstsec:     db    0                 ;       sector
hstadr:     dw    0                 ;       buffer address
hstblk:     dw    0                 ;       block no. in current cylinder
blksec:     db    0                 ;       host sector no. in block
hstsiz:     dw    0                 ;       physical sector size
hstsdh:     db    0                 ;       sdh register sector size setting

wrtpnd:     db    0                 ; write pending (host buffer active)
wrttyp:     db    0                 ; write type (0=use, 1=dir, 2=free)

oper: db    0                 ; operation (0=write, 1=read)

trkid:                            ; track id buffer for read address cmd
trkno:      db    0                 ; track number
sideno:     db    0                 ; side number
secno:      db    0                 ; sector number
seclen:     db    0                 ; sector size 0,1,2,3
crc1: db    0                 ; crc code 1
crc2: db    0                 ; crc code 2

;-------------------------------------------------------------------------
------
;                     DRIVE PARAMETER DEFINITIONS
;-------------------------------------------------------------------------
------
;
; dskdef < drvtyp0, drvtyp1, ... , drvtypm>
;
;-------------------------------------------------------------------------
------

      dskdef      <f8,f8,f548,f596>

;-----------------------------------
; BUFFERS FOR INTERRUPT CONSOLE I/O
;-----------------------------------
```

```
      if    intrin
outptri: db 0
inptri:     db   0
      endif

      if    introut
inptro:     db   0
outptro: db 0
      endif

      if    intrin
tmplen      defl  (($-start) mod inbfsz)

      if    tmplen
      ds    inbfsz-tmplen
      endif

inpbuf:
      ds    inbfsz
      endif

      if    introut
tmplen      defl  (($-start) mod outbfsz)

      if    tmplen
      ds    outbfsz-tmplen
      endif

outbuf:     ds    outbfsz
      endif

      if    ints
      ds    20              ; local stack for interrupt routines
locstk:
svstk:      ds    2               ; stack pointer storage area
      endif

;-------------------------
; COLD BOOT INITALIZATION
;-------------------------

coldboot:

;---------------------------------
; INITALIZE BAUD RATE ON SUPER SIX
;---------------------------------

      in    a,(015h)         ; this should not affect SUPER QUAD
      and   07fh             ; read baud switchs, mask off #dsk sds
      out   (018h),a         ; send it to baud set port

;--------------------------------
; INITALIZE HARD DISK STEP RATES
;--------------------------------
```

```
      ld    hl,drvmap-6       ; drive map base addr - 6
      ld    de,6              ; entry size in drive map
      ld    a,16d             ; drive counter

chkhd:

      add   hl,de             ; point to next entry in dmap

      push  hl                ; save dmap pointer
      push  de                ; save entry size
      push  af                ; save ldrv counter

      ld    a,(hl)                ; load a-reg w. drv type, zero = hard
      and   a                 ; set zero flag if hard

      jr    nz,nextdrv        ; jump if not hard disk

      inc   hl                ; select physical drive from dmap
      ld    b,(hl)                ; next entry in dmap is pdrv
      sla   b                 ; shift it over to bits 3 and 4
      sla   b                 ; for sdh reg setting
      sla   b                 ;

      inc   hl                ; put ddb address in d & e-reg
      inc   hl                ; last entry / entry in dmap is ddb adr
      inc   hl                ; hl points to ddb address of drive
      ld    e,(hl)                ; load d & e-reg w. ddb address
      inc   hl                ; point to high byte
      ld    d,(hl)                ; put it in d

      ld    hl,2              ; sdh reg setting is 2nd entry in ddb
      add   hl,de             ; add in sdh offset to ddb addr
      ld    a,(hl)                ; get sdh value
      or    b                 ; or in physical drive no
      out   (hsdh),a          ; send result to hdc-1001

      xor   a                 ; clear cylinder no. registers
      out   (hcylhi),a        ; first high
      out   (hcyllow),a       ; then low

      ld    hl,5              ; offset into ddb for step rate
      add   hl,de             ; add in base address of ddb
      ld    a,(hl)                ; move step rate into a-reg
      add   a,070h                ; or in command to restore
      out   (hcmd),a          ; send restore w. proper stp rte
      call  polbsy

nextdrv:
      pop   af                ; restore registers
      pop   de                ; these reg contain values need
      pop   hl                ; in hdc initalization loop
      dec   a                 ; decrement drive no count
      jr    nz,chkhd          ; loop until done w. all 16 drives
```

```
      iæ    ints

;---------------------
; SET UP SIO CHANNELS
;---------------------

      ld    hl,isioa          ; initalize SIO channel A
      ld    bc,isioal*256+1
      otir

      ld    hl,isiob          ; initalize SIO channel B
      ld    bc,isiobl*256+3
      otir

      ld    hl,serinttbl            ; initialize interrupt vector
      ld    a,h
      ld    i,a
      im    2

      endif

      xor   a                 ; initalize iobyte, disk, and user
      ld    (iobyte),a        ; iobyte =0
      ld    (cdisk),a         ; disk = user =0

      ld    hl,signon
      call  pmsg

      ld    a,(wboota+2)            ; warm boot vector
      add   a,6
      srl   a
      srl   a                 ; A=memory size in K
      call  twodec                  ; get ASCII digits for memory size
      push  bc
      ld    c,b
      call  conout
      pop   bc
      call  conout

      ld    hl,sign2
      call  pmsg

      lä    a,(contbl+1©            ; defaulô console
      add   a,031h                  ; make 1-2
      ld    c,a
      call  conout

      ld    hl,prnmsg
      call  pmsg

      ld    A,(lsttbl+1)
      cp    2
```

```
      jr    c,serprn

      ld    hl,parprn
      call  pmsg
      jr    lastmsg
serprn:
      push  af
      ld    hl,serpmsg
      call  pmsg
      pop   af
      add   a,031h
      ld    c,a
      call  conout
lastmsg:
      ld    hl,sign3
      call  pmsg
      jp    wboot

;-----------------------------------------------
; CONVERT A-REG < 100 TO TWO ASCII DIGITS IN BC
;-----------------------------------------------

twodec:
      ld    bc,0ff0ah
nxtten:
      inc   b
      sub   c
      jr    nc,nxtten
      add   a,c
      add   a,030h
      ld    c,a
      ld    a,030h
      add   a,b
      ld    b,a
      ret

;---------------------------
; COLD BOOT SIGNON MESSAGES
;---------------------------

signon:
      dâ    0dh,0ah,'Supeò Bioó v® 2.22',0dh,0ah

      if    ints
      if    intrin
      db    'Typeahead '
      endif

      if    intrin and introut
      db    'and '
      endif

      if    introut
      db    'Buffered output '
```

```
      endif

      db    'installed',0dh,0ah
      endif                   ;ints

      db    0dh,0ah,'$'

sign2:      db    'K CP/M 2.2 installed',0dh,0ah,0ah
      db    'Default console is serial port $'

prnmsg:     db    0dh,0ah,'Default printer is $'

parprn:     db    'parallel printer driver$'

serpmsg:
      db    'serial port $'

sign3:      db    0dh,0ah,0ah,'$'

      if    ints
;---------------------------------------
; INITIALIZATION TABLES FOR SIO DEVICES
;---------------------------------------

;-----------
; CHANNEL A
;-----------

ISIOA:
      DB    018H              ; channel reset
      DB    001H              ; write reg 1

      if    intrin and introut
      DB    01AH              ; Int on all recv and tx char
      else
      if    introut
      DB    002H              ; Int on tx ready
      else
      DB    018H              ; Int on all recv char (no parity)
      endif
      endif

      DB    003H              ; write reg 3
      DB    0C1H              ; Rx 8-bit char, Rx enable
      DB    004H              ; write reg 4
      DB    044H              ; X16 clock, 1 stop bit
      DB    005H              ; write reg 5
      DB    0EAH              ; DTR, Tx 8-bit char, Tx enable, RTS
      DB    000H              ; read reg 0
ISIOAL      EQU   $-ISIOA                ; length of sequence

;-----------
; CHANNEL B
;-----------
```

```
ISIOB:
      DB    018H            ; channel reset
      DB    001H            ; write reg 1
      DB    004H            ; Status affects interrupt vector
      DB    002H            ; write reg 2
      DB    LOW(serinttbl-start)   ; Interrupt vector table start
      DB    003H            ; write reg 3
      DB    0C1H            ; Rx 8-bit char, Rx enable
      DB    004H            ; write reg 4
      DB    044H            ; X16 clock, 1 stop bit
      DB    005H            ; write reg 5
      DB    0EAH            ; DTR, Tx 8-bit char, Tx enable, RTS
      DB    000H            ; read reg 0
ISIOBL     EQU   $-ISIOB               ; length of sequence

      endif

      end
```