

D>PIP LST:=CBIOS&.ASM[T8]

\*\*\*\*\*

\* Morrow Designs CBIOS for CP/M Version 2.2.

\* This CBIOS can be configured to run with the following devices.  
\* The disks may be configured to run with any or all of the disk  
\* systems. The logical order of the disks can be set to any order.

\* Disk systems:

- \* HDCA 10, 20 and 26 megabyte hard disks.
- \* HDDMA 5, 10, 16, megabyte hard disk systems.
- \* DJDMA floppy disk controller with 8 and 5 1/4 inch disks.
- \* DJ 2D/B floppy disk controller with 8 inch disks.

\* Console I/O:

- \* Disk Jockey 2D/B serial.
- \* Disk Jockey DMA serial.
- \* Multi I/O serial.
- \* Decision I serial.

\* Printer I/O:

- \* Multi I/O serial with handshaking.
- \* Multi I/O Diablo 1620 simulator for the Hytype II.

\* Note: Floppy systems diskette (drive A:) has to have 1024 byte sectors in order for the cold and warm boot loaders to work. Be sure to format all new system diskettes with 1024 byte sectors. The system diskette can be either single or double sided. The sector size on normal (non A: drive) diskettes is not restricted. Thus if you have a diskette with software that is supposed to run on the A: drive then you should mount the diskette in the B: drive and then PIP it over to a 1024 byte sector system diskette.

\* Written by Les Kent and Marc Kupper 3/4/82

Date	Programmer	Description
**11 20 82	Marc	Public release of revision E.31
* 11 19 82	Marc	Changed HDC3 equate to HDCA
* 11 19 82	Marc	Changed blank IO routines from RET to JMP \$
* 11 19 82	Marc	Converted BIOSLN to a byte value
* 11 9 82	Marc	Reduced bad map size to 1 for non MW systems
* 11 8 82	Marc	Deleted baud rate test from Multio drivers
* 11 4 82	Marc	Added initial IOBYTE to IOCONF
* 11 3 82	Marc	Added the North Star character I/O system
* 11 2 82	Marc	Added character redirection code for the IOBYTE
* 11 1 82	Marc	Changed serial i/o entry names to IOBYTE names
* 10 18 82	Marc	Fixed SETHIGH for 2 sided DJDMA 8 inch disks
* 10 18 82	Marc	Deleted the HyType drivers
**10 1 82	Marc	Public release of revision E.3
* 9 29 82	Marc	40H now points to the HDDMA command channel
* 9 28 82	Marc	MW's now have 1024 directory entries
* 9 28 82	Marc	Deleted the Centronics drivers
* 9 27 82	Marc	Changed login message to look like a label
* 9 27 82	Marc	Changed the login messages to say M5, M10, ...
* 9 27 82	Marc	Redefined the dparam table structure
* 9 22 82	Marc	Added a serial console for the Switchboard
* 9 22 82	Marc	Added initialization code for serial group 2
* 9 22 82	Marc	Added sector size byte to the HDCA DPB's
* 9 22 82	Marc	Added sector size parameter to DPBGEN
* 9 9 82	Marc	Fixed system length checks for 64K systems

1/16/83

ORIG M20/DJDMA

CBIOS

NEW DECISION

64K  
BIOS offset = 3700  
BOOT offset = 0B00

NOTE: changed ABOOT&.ASM AS FOLLOWS:  
MSIZE EQU 64. THAT'S OVER/CHG!

```

PROC: DDT CPM64.COM
      IABOOT&.HEX
      ROBOOT
      ICBIOS&.HEX
      R3700
      ↑C
      SYSGEN (DEST = HD)

```

CBIOS&5.ASM: 6/10/84 ADDED SPOOL-2-Q  
LST:=PRN: SPOOL 2 Q

```

* 9 9 82 Marc      SETHIGH was botching 2 sided DPB pointers
* 8 31 82 Marc    Changed TRACKS in HD driver to HDTRAK
* 8 27 82 Marc    Added code/system length checker
* 8 27 82 Marc    mwreset save/restores the track number
* 8 26 82 Marc    mwreset now sets *step and *dir for CMI
* 8 20 82 Marc    Added 'equ'ed handshaking to the serial LST:
* 8 19 82 Marc    Removed clock switching code from HDCA driver
* 8 18 82 Marc    Added handshake configuration code
* 8 18 82 Marc    Added handshake configuration bytes
* 8 18 82 Marc    Removed 'equ'ed handshaking from LST:
* 8 12 82 Marc    Added configuration entries for a0 & d0
* 8 11 82 Marc    Added the autostart command structure
* 8 11 82 Marc    Redefined the configuration table
* 8 11 82 Marc    Added DJDMA drive parameter table
* 8 9 82 Marc     Added clock switching to HDCA code
* 8 9 82 Marc     Added seek complete clearing in HDCA
* 8 6 82 Marc     Added buffer disable on home
* 8 6 82 Marc     Fixed 8250 UART initialization sequence
* 8 6 82 Marc     Strip parity on conout to clear up glitches
* 8 6 82 Marc     Fixed the 8 inch dpb256ss DPB's EXM
* 8 6 82 Marc     Increased the HD capacities slightly
* 8 6 82 Marc     Deleted all non-supported MW drives
* 8 6 82 Marc     Deleted call to flush in conout
* 8 6 82 Marc     Moved printer back to port 3
* 7 28 82 Marc    Moved conin flush call to conout
* 7 27 82 Marc    Fixed double sided head settle time
* 7 14 82 Marc    Optimized MWissue
* 7 14 82 Marc    Clean up login message for HD a bit
* 6 30 82 Marc    Fixed MF multi density problems
* 6 29 82 Marc    Added Olivetti HD561/1 HD561/2 drives
* 6 28 82 Marc    Added a MW error reporter
* 6 18 82 Marc    Added nonstandard system mode flag
* 6 17 82 Marc    Added a buffer error flag
* 6 17 82 Marc    Added save/restore of 50-52 to MW driver
* 6 17 82 Marc    Fixed Centronics drivers
* 6 7 82 Marc     Fixed allocation map sizes
* 6 7 82 Marc     Fixed MW partitioning
* 6 7 82 Marc     Fixed HD partitioning (again)
* 5 13 82 Marc    Fixed illegal MAC labels
* 5 11 82 Marc    Fixed North Star drive configurations
* 4 30 82 Marc    Fixed Quantum Q2040 tracks to 512
* 4 29 82 Marc    Fixed ST412 step constant to 0
* 4 26 82 Marc    Added unallocated writing
* 4 22 82 Marc    Fixed HD partition overlap
* 4 20 82 Marc    Started testing and debugging of E.3
* 4 19 82 Marc    Added 1 sector to HD warm boot loader
* 4 19 82 Marc    Added mod. number to CBIOS rev. number
* 4 19 82 Marc    Clean up login message 'if's
* 4 15 82 Marc    Fixed MCR Initialization for LST:
* 4 15 82 Marc    Added Seagate ST412 drive
* 4 6 82 Marc     Moved serial LST: device to port 2
* 4 1 82 Marc     Added common group select routines
* 4 1 82 Marc     Fixed Diablo HyType II initialization
* 4 1 82 Marc     Fixed LISTST for PROM driver
* 3 16 82 Marc    Added Tandon TM602 and TM603 drives
* 3 16 82 Marc    Use 'part number' equates for MW drives
* 3 15 82 Marc    Dropped hdrev and mwrev equates
* 3 15 82 Marc    Seagate ST506 head settle is 0 ms.
* 3 15 82 Marc    Added MiniScribe 1006 and 1012 drives
* *3 1 82 Marc    Public release of revision E.2
* 2 -- 82 Marc    Pre-release testing and debugging
* 2 1 82 Les + Marc Initial coding of revision E
*

```

\*\*\*\*\*

```
revnum equ 53 ;CBIOS revision number 5.x = E
cpmrev equ 22 ;CP/M revision number 2.2
```

```
*****
*
* The following flags set a 'non-standard' system mode and an
* assembly time debugger.
*
* If this CBIOS is used with the CP/M 2.2 system that is shipped on
* a Morrow Designs diskette then NOSTAND can be set to 1. This
* will allow the CBIOS to use various data areas found inside of
* the CP/M 2.2 BDOS. If the CBIOS is used with a different
* operating system then NOSTAND should be set to 0.
*
* The DEBUG flag merely causes various internal values and
* addresses to be printed during the assembly process. This
* printing is forced via assembly errors and thus should not
* affect the resulting code in any way.
*
```

```
nostand equ 1 ;Set to 1 for non-standard mode
debug equ 0 ;Set to 1 for debugging mode
```

```
*****
*
* The following is set to the memory size of the CP/M the CBIOS is
* being created for.
*
```

```
msize equ 4864 ;Memory size of target CP/M
biosln equ 16h ;BIOS length. Also in ABOOT.ASM
```

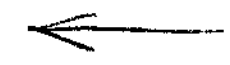
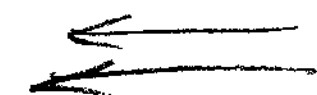
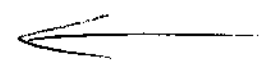
```
*****
*
* The following equates set up the disk systems to be included
* along with the types of drives and the logical order of the
* drives.
*
```

```
maxhd equ 1 ;Set to number of HDCA hard disk drives
maxmw equ 0 ;Set to number of HDDMA hard disks
maxfd equ 0 ;Set to number of 2D/B floppies
maxdm equ 2 1 ;Set to number of DJ DMA floppies 8 inch
maxmf equ 2 0 ;Set to number of DJ DMA floppies 5 1/4 inch
```

```
hdorder equ 1 ;Set the order of logical drives ELSE 0 if
mworder equ 0 ; not included.
fdorder equ 0
dmorder equ 2
mforder equ 3 0
```

```
ml0f equ 0 ;HDCA controller disk drives. Set only one
m20 equ 1 ;Fujitsu M2301B
m26 equ 0 ;Fujitsu M2302B
ml0m equ 0 ;Shugart SA4000
;Memorex
```

```
mwquiet equ 0 ;HDDMA controller disk drives. Set only one
st506 equ 0 ;Set for no names printed on login
st412 equ 0 ;Seagate ST-506
cm5619 equ 0 ;Seagate ST-412
;CMI CM-5619
```



```
wmdrive equ 0 ;Device to warm boot from. This is the
; CP/M logical drive number.

if maxmw ne 0 ;Only HDDMA drives use the bad map
badsiz equ 32 ;Number of badmap entries
else
badsiz equ 1 ;Leave one entry as filler
endif
```

```
*****
*
* Since most hard disk drives hold more than 8 megabytes we
* partition the drive. We partition our drives using two different
* formulas.
*
* One is the so called 'standard partitioning' where we try to
* create as many 8 megabyte partitions as possible plus a small
* partition to take up the slack on the end of the drive.
*
* Another way the drives are partitioned is the so called 'even
* partition' formula. This means that the drive is split into
* equale sized partitions with the only restriction being that no
* partition be over 8 megabytes in length.
*
* All hard disk drives shipped from Morrow Designs are partitioned
* using the standard partition formula. If the user wishes to
* implement even partitioning then he/she must set HDPART or MWPART
* to the number of partitions desired.
*****
```

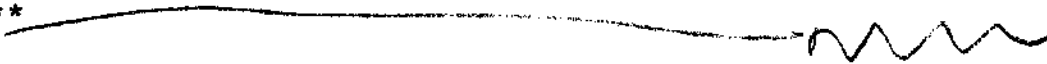
```
hdpart equ 0 ;Set to number of non standard partitions
mwpart equ 0 ;Set to number of non standard partitions
```

```
*****
*
* The following equates define the console and printer environments.
*****
```

```
*****
*
* Define the console driver to be used.
*
* CONTYP is: 0 Nothing, used for patching to PROM's.
* 1 Provide for 128 bytes of patch space.
* 2 Multi I/O or Decision I driver.
* 3 2D/B driver.
* 4 DJDMA serial port
* 5 Switchboard serial port
* 6 North Star motherboard (2 serial + 1 parallel)
*
* Set CBAUD to the divisor latch value for the console. For an
* explanation of the values look at the DEFCON table.
*****
```

```
contyp equ 2
cbaud equ 12 6 NOT IN FIRST
```

```
*****
*
* Define the printer driver to be used.
*
* LSTTYP is: 0 Nothing, used for patching to PROM's.
*****
```



*Auto boot*  
*need ~~INIT~~ INIT*

```

*       1       Provide for 128 bytes of patch space.
*       2       Multio serial, no protocol.
*       3       Multio serial, Clear To Send protocol.
*       4       Multio serial, Data Set Ready protocol.
*       5       Multio serial, Xon/Xoff protocol.

```

```

* Note: The Decision board is functionally identical to the Multi
*       I/O board for serial printer I/O. Selections 2 to 5 will
*       work on the Wunderbuss i/o board. To use drivers 6 or 7
*       the MULTR3 equate will have to be set.

```

```

* Set pbaud to the divisor latch value for the printer. For an
* explanation of the values see the deflst table.

```

```

*****

```

```

lsttyp equ 3
lbaud equ 96

```

```

*****

```

```

* The next equate determines if you have a Multi I/O Rev 3 or a
* Decision I mother board for parallel i/o. If are not using
* either of these boards then you need not worry about this equate.
* If you are using a Multi I/O rev. other than 3.x or 4.x then you
* should set MULTR3 to 0.

```

```

*****

```

```

multr3 equ 0 ;0 = Decision, 1 = Multi I/O rev. 3 or 4
congrp equ 1 ;Cosole port (1 = p1, 2 = p2, 3 = p3)
lstgrp equ 3 ;Printer port (1 = p1, 2 = p2, 3 = p3)

```

```

*****

```

```

* The following equates are internal to the CBIOS.

```

```

*****

```

```

ml0 equ ml0f or ml0m

hdlog if hdpart ne 0 ;Use non standard partitions
equ hdpart
else
hdlog equ ml0*2+m20*3+m26*3 ;Logical disks per drive for HDCA
endif

mwlog if mwpart ne 0 ;Use non standard partitions
equ mwpart
else
mwlog set st506+st412*2++cm5619*2 ;Logical disks per drive for HDDMA
endif

hdca equ m26 or m20 or ml0 ;HDCA controller
fujitsu equ m20 or ml0f
hdspt equ 32*m26+21*m20+21*ml0 ;Sectors per track

hdma set st506 or st412 or cm5619 ;HD DMA controller
mwspt equ 9 ;Sectors per track

maxlog equ (maxhd*hdlog)+(maxmw*mwlog)+maxfd+maxdm+maxmf

```

```

*****

```

```

*

```

```

* CP/M system equates.
*
*****

ccpln equ 800h
bdosln equ 0e00h

size equ (msize*1024)
ccp equ size-(biosln*100h+ccpln+bdosln)
bdos equ ccp+ccpln
bios equ ccp+ccpln+bdosln

offsetc equ 2100h-bios ;Offset for sysgen

if debug
dbgtmp set offsetc ;DDT offset ! <debug>
dbgtmp set ccp ;CCP address ! <debug>
dbgtmp set bdos ;BDOS address ! <debug>
dbgtmp set bios ;CBIOS address ! <debug>
endif

cdisk equ 4 ;Address of last logged disk
buff equ 80h ;Default buffer address
tpa equ 100h ;Transient memory
iobyte equ 3 ;IOBYTE location
wbot equ 0 ;Warm boot jump address
entry equ 5 ;BDOS entry jump address

if nostand ne 0
cblock equ bios-19h ;Current actual block# * blkmsk
;Used for unallocated writting
endif

```

```

*****
*
* The following are internal Cbios equates. Most are misc. constants.
*
*****

```

```

retries equ 10 ;Max retries on disk i/o before error
clear equ 'Z'-64 ;Clear screen on an ADM 3

anul equ 0 ;Null
aetx equ 'C'-64 ;ETX character
aack equ 'F'-64 ;ACK character
abel equ 'G'-64 ;Bell
abs equ 'H'-64 ;Back Space
aht equ 'I'-64 ;Horizontal tab
alf equ 'J'-64 ;Line feed
avt equ 'K'-64 ;Vertical tab
aff equ 'L'-64 ;Form Feed
acr equ 'M'-64 ;Carriage return
xon equ 'Q'-64 ;Xon character
xoff equ 'S'-64 ;Xoff character
aesc equ 1bh ;Escape character
ars equ leh ;RS character
aus equ lfh ;US character
asp equ ' ' ;Space
adel equ 7fh ;Delete

```

```

*****
*
* The following are the macros used in generating the DPH, DPB and
* allocation tables.
*
*****

```

```

dpgen macro nam,log,dspt,dbsh,dblm,dexm,ddsm,ddrm,dal0,dall,dcks,doff,ssiz
dpg&nam&log equ $
dw dspt
dw db dbsh
dw db dblm
dw db dexm
dw dw dds
dw dw ddr
dw db dal0
dw db dall
dw dw dck
dw dw doff
dw db ssiz
endm

```

```

dphgen macro nam,log,dpb1,dpb2
dph&nam&log equ $
dw 0
dw 0,0,0
dw dirbuf
dw &dpb1&dpb2
dw csv&nam&log
dw alv&nam&log
endm

```

```

alloc macro nam,log,al,cs
csv&nam&log: ds cs
alv&nam&log: ds al
endm

```

```

*****
*
* The following marco is used in generating the logical order of the
* CP/M drives.
*
*****

```

```

order macro num
if num eq horder
dw hdst
endif

if num eq mworder
dw mwdst
endif

if num eq fdorder
dw fdst
endif

if num eq dmorder
dw dmdst
endif

if num eq mforder
dw mfdst
endif
endm

```

```

*****
*
* The following are offset numbers of Device Specification Tables.
*
*****

```

```

d$wboot equ 0 ;Warm boot
d$stran equ 1 ;Sector translation
d$sel1 equ 2 ;Drive select, Return DPH
d$sel2 equ 3 ;Drive select
d$home equ 4 ;Home drive
d$strk equ 5 ;Set track
d$ssec equ 6 ;Set sector
d$sdma equ 7 ;Set DMA address
d$read equ 8 ;Read a physical sector
d$write equ 9 ;Write a physical sector
d$bad equ 10 ;Return pointer to bad sector info

```

```

*****
*
* The jump table below must remain in the same order, the routines
* may be changed, but the function executed must be the same.
*
*****

```

```

org bios ;Cbios starting address

wboote: jmp cboot ;Cold boot entry point
        jmp wboot ;Warm boot entry point

        if contyp ne 0
const: jmp conist ;Console status routine
cin:   jmp conin ;Console input
cout:  jmp costrp ;Console output
        else
const: jmp $ ;Console status routine PROM pointer
cin:   jmp $ ;Console input PROM pointer
cout:  jmp $ ;Console output PROM pointer
        endif

pout:   if (lsttyp ne 0) or (contyp eq 6)
        jmp lstout ;List device output
        else
pout:   jmp cout ;List device output
        endif

        if contyp eq 6
        jmp punout ;Punch device output
        else
        jmp cout ;Use console I/O
        endif

        if contyp eq 6
        jmp rdrin ;Reader device input
        else
        jmp cin ;Use console I/O
        endif

        jmp home ;Home drive
        jmp setdrv ;Select disk
        jmp settrk ;Set track
        jmp setsec ;Set sector
        jmp setdma ;Set DMA address
        jmp read ;Read the disk
        jmp write ;Write the disk

        if lsttyp ne 0
        jmp lstost ;List device status
        else
        jmp donop ;List device status
        endif

```



```

        jmp      sectran      ;Sector translation
;
;   The following jumps are extended BIOS calls defined by Morrow Designs
;
        if      maxfd ne 0
        jmp     fdssel      ;Hookup for SINGLE.COM program
        else
        jmp     donop
        endif

        jmp     0           ;End of the jump table

*****
*
* Drive configuration table.
*
*****

drconf: db      0           ;Revision 0 structure
        db      32         ;32 bytes long now

*****
*
* The following is the table of pointers to the Device
* Specification Tables. The order of this table defines the
* logical order of the CP/M drives.
*
*****

dsttab: equ     $

dn      set     1
        rept   16
        order   %dn
dn      set     dn+1
        endm

*****
*
* I/O configuration table.
*
* At this CBIOS revision 11 bytes are defined for this table.
* Several extensive changes are planned for the table. Future
* revision of the IOCONF table will have independant entries for
* three serial ports and will be used by several character drivers.
* Also the IOBYTE will be implemented for all the character
* drivers. I might even write an external program to edit this
* table.
*
* The first two bytes show the I/O configuration that the CBIOS was
* assembled with. These bytes are used by external software to
* determine the configuration options that are available.
*
* The next byte is the initial IOBYTE value. This value is written
* to location 3 on cold boots. See the CP/M 2 alternation guide
* for a description of the IOBYTE.
*
* The next byte is to make sure that the group select byte on the
* Mult I/O or Decsion I stays consistant throughout the Cbios.
* Only the group bits themselves (bits 0 and 1) should be changed
* as you output to the group port. If you modify one of the other
* bits (such as driver-enable) then you should modify the same bit
* in this byte. For example:
*

```

```

*          lda      group      ;Select console group
*          ori      congrp     ;Get group byte
*          out      grpsel     ;Select the console port
*
*          ;Modify a bit in the group byte
*          lda      group      ;Get group byte
*          ori      bank       ;Set the bank bit
*          sta      group      ;Save new group setting
*          ori      group2     ;Select second serial port
*          out      grpsel     ;Select the desired group

```

```

* Note: You should not set the group bits themselves in the
*       group byte.

```

```

* The following two words define the default baud rates for the
* console and the list devices. These words are provided so that
* the user can easily modify them and that they will also be used
* in the future by Morrow Designs software.

```

```

* The following is a list of possible baud rates and the decimal
* value needed for the defcon or deflst words.

```

Baud rate	defcon/deflst	Baud rate	defcon/deflst
50	2304	2000	58
75	1536	2400	48
110	1047	3600	32
134.5	857	4800	24
150	768	7200	16
300	384	9600	12
600	192	19200	6
1200	96	38400	3
1800	64	56000	2

```

* The next two bytes are used to configure the hardware handshaking
* protocol used by the serial list drivers with the Multio or
* Wunderbuss I/O boards. The first of these two bytes is a mask.
* This mask is ANDed with the 8250's MODEM Status Register to strip
* out the desired handshake lines. Next the result of the ANDing
* is XORed with the second of the two bytes. This XORing allows
* the handshake lines to be inverted. Common byte values are
* shown below.

```

```

* cts equ 10h          ;Clear To Send status mask
*
* db cts              ;Morrow Designs 'Clear To Send'
* db 0
*
* db cts              ;Inverted Clear To Send
* db cts
*
* db 0                ;No handshaking
* db 0ffh

```

```

* The last byte in the revision one structure is the last character
* that was received from the printer. This byte is used to
* implement Xon/Xoff software handshaking. This handshaking
* protocol should not bother printers that have not implemented
* Xon/Xoff protocol so this driver is enabled all the time.

```

```

*****

```

```

db      11          ;11 bytes long now
db      contyp     ;Console device driver number
db      lsttyp     ;List device drive number

iobyte equ  $      ;The initial IOBYTE is kept here
db      00$00$00$00b ;All devices go to CON:

group:  db      0          ;Group byte
defcon: dw      cbaud     ;Console baud rate divisor value
deflst: dw      lbaud     ;Printer baud rate divisor value

        if      (lsttyp ne 3) and (lsttyp ne 4) ;Xon/Xoff protocol
lstand: db      0          ;Serial list handshake mask
lstxor: db      0ffh      ;Serial list inversion flag
endif

        if      lsttyp eq 3 ;Clear To Send protocol
lstand: db      cts       ;Serial list handshake mask
lstxor: db      0         ;Serial list inversion flag
endif

        if      lsttyp eq 4 ;Data Set Ready protocol
lstand: db      dsr       ;Serial list handshake mask
lstxor: db      0         ;Serial list inversion flag
endif

lastch: db      xon       ;Last character recieved from the printer

```

← DB ~~10~~ 11\$00\$00\$00b

```

*****
*
* The following table are drive parameters for drives connected to
* the DJDMA floppy disk controller. There is one entry for each of
* the the eight drive that the controller can address. The first
* four entries are for the 8 inch drives and the last four are for
* the 5 1/4 inch drives. Users with fast stepping 8 inch drives
* (SA850/1) or slow 5 1/4 inch drives (SA400) should adjust this
* table for optimal device performace.
*
* Each table entry contains four fixed length fields. The fields
* are defined as follows:
*
*   tracks This byte contains the number of tracks on the
*           drive. Most 8 inch drives have 77 tracks and
*           most 5 1/4 inch drives have 35 or 40 tracks.
*
*   config This a a flag byte that indicates as to whether
*           or not this drive has been configured. Set to
*           0 to force reconfiguration.
*
*   step   This word contains the stepping rate constant.
*           The DJDMA's delay routines tick 34.1 times per
*           millisecond. Thus the step constant would be the
*           drive manufactors recomended stepping delay times
*           34.1. Example. Shugart SA 850's step at 3
*           milliseond intervals. The step constant would be
*           3 * 43.1 or 102.
*
*   rfu    The next two words are reserved for future use.
*           They must be zero.
*
*   settle This word is similar to the previously defined
*           step word. This specifies the head settle timing
*           after the heads have been stepped. Example,
*           Shugart's SA 850 head settle time is 15
*           milliseconds. The settle constant would be 15 *
*           34.1 or 512.
*
*****

```

```

* An assembler macro (dconf) has been provided to assist in
* generating the dparam table. This macros parameters are the
* number of tracks, the step rate in milliseconds, and the head
* settle time in milliseconds. For example:
*
*           ;Shugart SA 850
* dconf 77, 3, 15 ;77 tracks, 3 ms step, 15 ms settle
*
*           ;Shugart SA 400
* dconf 35, 40, 10 ;35 tracks, 40 ms step, 10 ms settle
*
* Note: Caution should be used when defining the drive parameters.
* Incorrect definations may damage the floppy disk drive. Morrow
* Designs takes no responsibility for damage that occures through
* the misuse of this macro.

```

```

*****

```

```

        if      (maxdm ne 0) or (maxmf ne 0)      ;DJDMA present?

dconf macro tracks, step, settle
db      tracks          ;Number of tracks
db      0                ;Reset the calibrated flag
dw      step*341/10     ;Step time
dw      0                ;Reserved for future use, must be zero
dw      0                ;Reserved for future use, must be zero
dw      settle*341/10  ;Head settle time
endm

```

```

dmarap: db      0, 10*8      ;Revision 0, length 80 bytes

```

```

dparam: equ     $           ;Drive parameter table

```

```

*****
*
* Define 8 inch drive parameters
* Use SA800 parameters: 77 tracks, 8 ms step, 8 ms settle
*
*****

```

```

dconf 77, 8, 8      ;Drive 0
dconf 77, 8, 8      ;Drive 1
dconf 77, 8, 8      ;Drive 2
dconf 77, 8, 8      ;Drive 3

```

```

*****
*
* Define 5 1/4 inch drive parameters
* Use Tandon parameters: 40 tracks, 5 ms step, 15 ms settle
*
*****

```

```

dconf 40, 5, 15     ;Drive 0
dconf 40, 5, 15     ;Drive 1
dconf 40, 5, 15     ;Drive 2
dconf 40, 5, 15     ;Drive 3

```

```

endif

```

```

*****
*
* Console driver routines.
*
* Routine used depends on the value of CONTYP. Possible CONTYP
* values are listed as follows:

```

```

*
* CONTYP is:  0      Nothing, used for patching to PROM's
*             1      Provide for 128 bytes of patch space
*             2      Multi I/O or Decision I driver
*             3      2D/B driver
*             4      DJDMA serial port
*             5      Switchboard serial port
*             6      North Star motherboard (2 serial + 1 parallel)
*

```

```

*****

```

```

*****

```

```

* This routine is an experiment to reduce missed and garbled
* characters on console output.
*

```

```

*****

```

```

        if      contyp ne 0

costrp: mov     a,c           ;Strip parity on conout
        ani    7fh
        mov    c,a
        jmp   conout

        endif

```

```

*****

```

```

* The folowing equates will define the Decision I mother
* board I/O or the Multi I/O environments if needed.
*

```

```

*****

```

```

multio equ      (contyp eq 2) or (lsttyp ge 2) ;Multi I/O board used?

        if      multio           ;Define Multi I/O environment
mbase  equ      48h              ;Base address of Multi I/O or Decision I
grp sel equ      mbase+7        ;Group select port
dll    equ      mbase           ;Divisor (lsb)
dlm    equ      mbase+1        ;Divisor (msb)
ier    equ      mbase+1        ;Interupt enable register
clk    equ      mbase+2        ;WB14 printer select port
lcr    equ      mbase+3        ;Line control register
mcr    equ      mbase+4
lsr    equ      mbase+5        ;Line status register
msr    equ      mbase+6
rbr    equ      mbase           ;Read data buffer
thr    equ      mbase           ;Tranmitter data buffer
dlab   equ      80h            ;Divisor latch access bit
thre   equ      20h            ;Status line THRE bit
cts    equ      10h            ;Clear to send
dsr    equ      20h            ;Data set ready
dr     equ      1              ;Line status DR bit
wls0   equ      1              ;Word length select bit 0
wls1   equ      2              ;Word length select bit 1 for 8 bit word
stb    equ      4              ;Stop bit count - 2 stop bits

```

```

; Define multi I/O ports addresses for group zero

```

```

gzero  equ      0
daisy0 equ      mbase           ;Daisy input ports
daisy1 equ      mbase+1
sensesw equ     mbase+1        ;Sense switches

```

```

        if      multir3 eq 0 ;Daisy output ports are different

```

```

daisi0 equ mbase ; for Decision I and Multi I/O.
daisi1 equ mbase+1 ;These two are the Decision I ports
else
daisi0 equ mbase+1 ; and these are the Multi I/O's.
daisi1 equ mbase
endif

```

```

; Define daisy 0 status input bits

```

```

ribbon equ 01h ;End of ribbon
paper equ 02h ;Paper out
cover equ 04h ;Cover open
pfrdy equ 08h ;Paper feed ready
crrdy equ 10h ;Carriage ready
pwrdy equ 20h ;Print wheel ready
check equ 40h ;Printer check (error)
ready equ 80h ;Printer ready

```

```

; Define daisy 0 status input bits for Diablo HyType II driver

```

```

crstrd equ 1020h ;Carriage ready
pfstrd equ 810h ;Paper feed ready
pwstrd equ 2040h ;Print wheel ready

```

```

; Define daisy 0 output bits

```

```

d9 equ 01h ;Data bit 9
d10 equ 02h ;Data bit 10
d11 equ 04h ;Data bit 11
d12 equ 08h ;Data bit 12

pfstb equ 10h ;Paper feed strobe
crstb equ 20h ;Carriage strobe
pwstb equ 40h ;Print wheel strobe
rest equ 80h ;Printer restore (Ribbon lift on Multi I/O)

```

```

; Define clock select bits

```

```

rlift equ 40h ;Ribbon lift
pselect equ 80h ;Select (Not used by Diablo)

```

```

; Define Modem Control Register bits

```

```

dtrenb equ 1 ;DTR enable
rtsenb equ 2 ;RTS enable

```

```

; Define group select bits

```

```

s0 equ 01h ;Group number (0-3)
s1 equ 02h
smask equ 03h
bank equ 04h
enint equ 08h
restor equ 10h ;Printer restore on Multi I/O
denable equ 20h ;Driver enable on Multi I/O

```

```

; Define special constants for the HyTyp II driver

```

```

cperi equ 10 ;Default to 10 characters per inch
lperi equ 6 ;Default lines per inch
hinc equ 120 ;Horizontal increments per inch
vinc equ 48 ;Vertical increments per inch
numtabs equ 160 ;Number of horizontal tabs
maxchrs equ 1024 ;Maximum number of printer characters to queue
maxrgt equ 1584 ;Maximum carriage position
dfrmln equ 110 ;Forms length times 10

```

```
autolf equ 0 ;Default to noIAuto line feed
```

```
endif
```

```
*****
*
* CONTYP: 2 Multi I/O or Decision I console driver
*
*****
```

```
if contyp eq 2
```

```
*****
*
* This driver on cold boot will inspect bits 1-3 of the sense
* switches. If the value found is in the range 0-6 then the
* console baud rate will be taken from the rate table. Otherwise
* the baud rate will be set from the DEFCON word which is found
* just below the regular Cbios jump table. The standard divisor
* table is given below.
*
```

```
* Sense switch: 123 (0 = off, 1 = on)
* 000 = 110
* 001 = 300
* 010 = 1200
* 011 = 2400
* 100 = 4800
* 101 = 9600
* 110 = 19200
* defcon = 9600
*
```

```
* Note: If you are using a Multio then the switches will not be
* available so the baud rate will be taken from DEFCON.
*
```

```
*****
*
* Due to its length, the TTYSET routine driver is below the CBOOT
* CBOOT routine.
*
```

```
*****
*
* Read a character from the serial port.
*
```

```
conin: call selcon ;Select console

coninl: in lsr ;Read status register
        ani dr ;Wait till character ready
        jz coninl
        in rbr ;Read character
        ani 7fh ;Strip parity
        ret
```

```
*****
*
* Output a character to serial port.
*
```

```
conout: call selcon ;Select console
```

```

conoutl: in    lsr          ;Read status
          ani    thre       ;Wait till transmitter buffer empty
          jz     conoutl
          mov    a,c        ;Character is in (c)
          out   thr        ;Output to transmitter buffer
          ret

```

```

*****
*
* Return serial port status. Returns zero if character is not
* ready to be read. Else returns 255 if ready.
*
*****

```

```

conist: call   selcon      ;Select console

          in    lsr        ;Read status register
          ani    dr
          rz     ;No character ready
          mvi   a,0ffh    ;Character ready
          ret

          endif          ;Multi I/O or Decision I

```

```

*****
*
* CONTYP: 3      2DB console driver
*
*****

```

```

          if    contyp eq 3

conout: jmp    fdcout      ;Console output

conin:  jmp    fdcin      ;Console input

conist: call   fdtstat    ;Console status
          mvi   a,0ffh
          rz
          inr   a
          ret

          endif          ;2DB

```

```

*****
*
* CONTYP: 4      DJDMA console driver
*
*****

```

```

          if    contyp eq 4

conout: lxi    h,dmchan
          mvi   m,serout   ;Command for serial output
          inx   h
          mov   m,c
          jmp   docmd

conin:  lxi    h,serin+1   ;Serial input status
          xra   a

ci2:   cmp    m           ;Wait till 40h deposited at 3fH
          jz   ci2
          mov   m,a       ;Clear status
          dcx   h         ;Point to input data
          mvi   a,7fh     ;For masking out parity
          ana   m
          ret

```



```

conist: lda      serin+1      ;Pick up serial input status
        ora      a
        rz              ;If zero then no character ready
        mvi      a,0ffh      ;Set character ready
        ret
        endif

```

```

*****
*
* CONTYP: 5      Switchboard as serial console
*
*****

```

```

        if      contyp eq 5

swbase equ      0              ;Base of the SWITCHBOARD

conist: in      swbase+2      ;Get the first ports status
        ani      4              ;Mask the data ready bits
        rz              ;Return console not ready
        mvi      a,0ffh

ttyset: ret              ;NULL terminal initialization

conin:  in      swbase+2      ;Get switchboard status
        ani      4              ;Test for data ready
        jz              conin
        in      swbase      ;Get a character
        ani      7fh          ;Strip off parity
        ret

conout: in      swbase+2      ;Check status
        ani      8              ;Wait till output buffer empty
        jz      conout
        mov      a,c          ;Write a character
        out      swbase
        ret

        endif

```

```

*****
*
* Multio/Wunderbuss group select routines
*
*****

```

```

        if (contyp eq 2) or (lsttyp ge 2)      ;Need group select routines?

selg0:  lda      group          ;Select group zero
        out      grpsel
        ret

selcon: lda      group          ;Select console group
        ori      congrp
        out      grpsel
        ret

selrdr: lda      group          ;Select reader/punch group
        ori      5-lstgrp      ;Use 'other' serial port
        out      grpsel
        ret

sellst: lda      group          ;Select printer group
        ori      lstgrp
        out      grpsel
        ret

```

endif

```

*****
*
* The following byte determines if an initial command is to be
* given to CP/M on warm or cold boots. The value of the byte is
* used to give the command to CP/M:
*
* 0 = never give command.
* 1 = give command on cold boots only.
* 2 = give the command on warm boots only.
* 3 = give the command on warm and cold boots.
*
*****

```

```

autost: db      0          ;Revision 0 structure
        db      100h - (low $) ;The rest of the page is used for this stuff

```

```

autoflg:db      0          ;Auto command feature enable flag

```

```

coldmes:dw      coldcm      ;Pointer to the cold start command
warmes: dw      warmcm      ;Pointer to the warm start command

```

```

*****
*
* If there is a command inserted here, it will be passed to the
* CCP if the auto feature is enabled. For Example:
*
*      coldcm: db      coldend-coldcm
*              db      'MBASIC MYPROG'
*      coldend equ    $
*
* will execute Microsoft BASIC, and MBASIC will execute the
* "MYPROG" BASIC program. Note: The command line must be in
* upper case for most commands.
*
*****

```

```

coldcm: db      coldend-coldcm      ;Length of cold boot command
        db      ''                  ;Cold boot command goes here
coldend equ    $

```

```

warmcm: db      warmend-warmcm      ;Length of warm boot command
        db      ''                  ;Warm boot command goes here
warmend equ    $

```

```

*****
*
* At the first page boundry following the CBIOS we have a series of
* pointers that point to various internal tables. At the start of
* each of these tables we have a revision byte and a length byte.
* The revision byte is the current revision number for that
* particular structure and the length byte is the length of that
* structure. This length does not include the revision byte nor
* the length byte itself.
*
*      Revision      Description
*      E.0           1 and 2 defined
*      E.3           This table is moved to a page boundry
*      E.3           0, 3 and 4 defined
*

```

\* The pointers defined so far are as follows:

```

* 0) High byte is the page number of the CBIOS. Low byte is
* the CBIOS revision number. Used to determine pointer

```

```

* structure.
*
* 1) This points to the drive configuration table.
*
* 2) This points to the I/O configuration bytes for the serial
* drivers. Eg, the console, printer, reader, and punch
* devices.
*
* 3) This points to the drive parameter table for DJDMA floppy
* disk drives. If no DJDMA is present then this pointer is
* null (0).
*
* 4) This points to the autostart command structures. Used to
* automatically invoke a command on cold or warm boot
*
* 5) This will be a null (0) pointer. It marks the end of the
* table.
*
*****

```

```

if      $ gt bios+256      ;Test for code overlap
'Fatal error, pointer table placement.'
else
ds      bios+256-$        ;Start at a page boundry
endif

db      high ($-1)        ;CBIOS page number
db      revnum            ;Cbios revision number
dw      drconf            ;Drive configuration table pointer
dw      ioconf            ;I/O configuration table pointer
if      (maxdm ne 0) or (maxmf ne 0) ;DJDMA present?
dw      dmarap            ;Drive parameter table pointer
else
dw      0
endif
dw      autost            ;Auto command structure pointer
dw      0                 ;End of table marker

```

```
*****
```

```

* The following code performs the mapping of logical to physical
* serial I/O devices. The physical entry points are CONIN, CONOUT,
* CONIST, RDRIN, PUNOUT, LSTOUT, and LSTOST. These entry points
* are mapped via the Intel standard I/O byte (IOBYTE) at location 3
* in the base page to the low level device drivers.
*

```

```

* Note: A naming convention has been chosen to reduce label
* colisions. The first three characters of a name indicate the
* device drivers name, the following three characters indicated the
* function performed by that particular device routine. The device
* names are defined and described in the "An Introduction to CP/M
* Features and Facilities" manual in the section on the STAT
* command and in the "CP/M Interface Guide" in the IOBYTE section.
* The device function postfixes are as follows.
*

```

```

* devSET      Initial device setup and initialization
* devIN       Read one character from the device
* devOUT      Write one character to the device
* devIST      Return the device character input ready status
* devOST      Return the device character output ready status
*

```

```

* The setup routine initializes the device and returns. The input
* routine returns one character in the A register (parity reset).
* The output routine write one character from the C register. The
* input status routine returns in the A register a 0 if the device
* does not have a character ready for input for 0ffh if a character

```

```

* is ready for input. The output status routine returns in the A
* register a 0 if the device is not ready accept a character and a
* 0ffh if the device is ready. The input and output routines
* should wait untill the device is ready for the desired operation
* before the doing the operation and returning.
*

```

```

* Not all of these functions need to be implemented for all the
* devices. The following is a table of the entry points needed for
* each device handler.
*

```

device name	setup	input	output	input status	output status
CON:		CONIN	CONOUT	CONIST	
RDR:		RDRIN		RDRIST	
PUN:			PUNOUT		
LST:			LSTOUT		LSTOST
TTY:	TTYSET	TTYIN	TTYOUT	TTYIST	TTYOST
CRT:	CRTSET	CRTIN	CRTOUT	CRTIST	CRTOST
UC1:	UC1SET	UC1IN	UC1OUT	UC1IST	
PTR:	PTRSET	PTRIN		PTRIST	
UR1:	UR1SET	UR1IN		UR1IST	
UR2:	UR2SET	UR2IN		UR2IST	
PTP:	PTPSET		PTPOUT		
UP1:	UP1SET		UP1OUT		
UP2:	UP2SET		UP2OUT		
LPT:	LPTSET		LPTOUT		LPTOST
UL1:	UL1SET		UL1OUT		UL1OST

```

* The CONIN, CONOUT, CONIST, RDRIN, RDRIST, PUNOUT, LSTOUT, and
* LSTOST routines are the logical device driver entry points
* provided by this device mapper. The other entry names must be
* provided by the physical device drivers.
*

```

```

*****

```

```

        if      contyp eq 6          ;I/O byte implemented for North Star
        ;      drivers. Other drivers to follow

conin:  mvi     e,1                  ;Console input
        call   redir                ;      IOBYTE: 76543210
        dw    ttyin                 ;CON: = TTY:   xxxxxx00
        dw    crtin                 ;CON: = CRT:   xxxxxx01
        dw    rdrin                 ;CON: = BAT:   xxxxxx10
        dw    uclin                 ;CON: = UC1:   xxxxxx11

conout: mvi     e,1                  ;Console output
        call   redir                ;      IOBYTE: 76543210
        dw    ttyout                ;CON: = TTY:   xxxxxx00
        dw    crtout                ;CON: = CRT:   xxxxxx01
        dw    lstout                ;CON: = BAT:   xxxxxx10
        dw    uclout                ;CON: = UC1:   xxxxxx11

conist: mvi     e,1                  ;Console input status
        call   redir                ;      IOBYTE: 76543210
        dw    ttyist                ;CON: = TTY:   xxxxxx00
        dw    crtist                ;CON: = CRT:   xxxxxx01
        dw    rdrist                ;CON: = BAT:   xxxxxx10
        dw    uclist                ;CON: = UC1:   xxxxxx11

rdrin:  mvi     e,7                  ;Reader input
        call   redir                ;      IOBYTE: 76543210

```

```

dw      ttyin      ;RDR: = TTY:      xxxx00xx
dw      ptrin      ;RDR: = PTR:      xxxx01xx
dw      urlin      ;RDR: = UR1:      xxxxl0xx
dw      ur2in      ;RDR: = UR2:      xxxxl1xx

rdrist: mvi        e,7      ;Reader input status
        call       redir    ; IOBYTE: 76543210
dw      ttyist     ;RDR: = TTY:      xxxx00xx
dw      ptrist     ;RDR: = PTR:      xxxx01xx
dw      urlist     ;RDR: = UR1:      xxxxl0xx
dw      ur2ist     ;RDR: = UR2:      xxxxl1xx

punout: mvi        e,5      ;Punch output
        call       redir    ; IOBYTE: 76543210
dw      ttyout     ;PUN: = TTY:      xx00xxxx
dw      ptpout     ;PUN: = PTP:      xx01xxxx
dw      uplout     ;PUN: = UP1:      xx10xxxx
dw      up2out     ;PUN: = UP2:      xx11xxxx

        endif

lstout: mvi        e,3      ;List output
        call       redir    ; IOBYTE: 76543210
dw      ttyout     ;LST: = TTY:      00xxxxxx
dw      crtout     ;LST: = CRT:      01xxxxxx
dw      lptout     ;LST: = LPT:      10xxxxxx
dw      ullout     ;LST: = UL1:      11xxxxxx

lstost: mvi        e,3      ;List output status
        call       redir    ; IOBYTE: 76543210
dw      ttyost     ;LST: = TTY:      00xxxxxx
dw      crtost     ;LST: = CRT:      01xxxxxx
dw      lptost     ;LST: = LPT:      10xxxxxx
dw      ullost     ;LST: = UL1:      11xxxxxx

redir:  lda        iobyte   ;Get the INTEL standard iobyte
redir0: rlc        ;Shift the next field in
        dcr        e        ;Bump the shift count
        jnz        redir0

redir1: ani        110b     ;Mask the redirection field
        mov        e,a      ;Make the word table offset
        mvi        d,0
        pop        h        ;Get the table base
        dad        d        ;Offset into our table
        mov        a,m      ;Load the low level i/o routine pointer
        inx        h
        mov        h,m
        mov        l,a
        pchl          ;Execute the low level i/o driver

        endif ;IOBYTE redirector

```

ADDED  
6/10/84

```

SZQ100 EQU 10 ; SPool-2-Q at ANT 10
SZQOUT CALL SZQOST ; GET STATUS
ORA A ; TEST FOR 0 IN ACC
JZ SZQOUT ; WAIT UNTIL READY
MOV A,C ; PUT CHAR IN A
OUT SZQ100 ; SEND IT
RET

SZQOST IN SZQ100 ; GET SPool 2 Q STATUS
ANL 1 ; CHECK BIT 0
MVI A,0 ; ZERO IN ACC MEANS NOT READY
RNZ ; RETURN WITH 0 IF SZQ IS BUSY
CMA ; MAKE ACC FFH
RET ; READY RETURN, A=FFH

OKOUT IN 2 ; INPUT FROM PORT 2
ANI 8 ; WAIT UNTIL OK TO SEND
JZ OKOUT

OKOUT1 IN 5 ; BUFFER FULL?
ANI 1 ;
JZ OKOUT1 ; WAIT UNTIL PUNCH READY
MOV 0,C ; OUTPUT THE CHARACTERS
OUT 0
RET

diouta IN 2 ; OUTPUT FROM DIR. GET STATUS
ANI 80h ; WAIT UNTIL OK TO SEND
JZ diouta
MOV A,C ; OUTPUT THE CHARACTERS
OUT 1
RET

dioutb call diouta ; OUTPUT THE CHARACTERS
        lda        COUNT
        dcr        a
        sta        COUNT
        RNZ
        MVI        A,78
        STA        COUNT
        MVI        C,ACTX
        call       diouta
dioutb IN 2 ; INPUT FM Diablo
        ani        40h
        JZ        dioutb
        IN        1

```

```

*****
*
* CONTYP: 1      Blank space for console driver
*
* The driver entries CONOUT, CONIN, CONIST are defined in the CP/M
* alternation guide. Eg. Input parameters are in register C and
* results are returned in register A. The TTYSET routine is used
* for initialization code. It should execute a RET when complete.
*
* The TTYSET routine could be placed just below the CBOOT routine.
* This space (below CBOOT) is recycled for use as a disk buffer
* after CBOOT is done.
*
*****

```

```

if      contyp eq 1      ;User defined IO area
ttyset  equ  $          ;Console initialization
conout  equ  $          ;Console output
conin   equ  $          ;Console input
conist  equ  $          ;Console input status
        jmp  $
        ds  125
endif
;User IO

```

*ans 7FH ; Strip parity*  
*CP1 aack*  
*J#Z d100tb*  
*Ret*

```

*****
*
* CONYTP: 6      North Star
*
* The following code implements the North Star console I/O system.
* This system is for users who purchase a Morrow Designs disk
* system to replace their North Star disk system.  The Mapping of
* the logical to physical entry points is performed as follows:
*
* Device name      Left      Right      Parallel
*                  serial    serial    port
*
* Console          CON: =  TTY:      CRT:      UC1:
* Reader           RDR: =  TTY:      PTR:      UR1:
* Punch            PUN: =  TTY:      PTP:      UP1:
* List             LST: =  TTY:      CRT:      UL1:
*
* For example, to use a printer connected to the right serial port,
* use the CP/M command:
*
*      STAT LST:=CRT:
*
* Likewise, the CP/M command "STAT LST:=UL1:" is used if you have a
* printer connected to the parallel port.
*
*****

```

```

if      contyp eq 6      ;Use North Star I/O?

nsl-dat equ  2          ;Left serial port data port
nslsta equ  3          ;Left serial port status port

nsr-dat equ  4          ;Right serial port data port
nsrsta equ  5          ;Right serial port status port

nsstbe equ  1          ;Transmitter buffer empty status bit
nssrbr equ  2          ;Receiver buffer ready status bit

;See the 8251 data sheets for more
; configuration information.

nslinl equ  0ceh       ;Left serial port initialization # 1
nsrinl equ  0ceh       ;Right serial port initialization # 1
;76543210 Bit definitions
;11001110 Default configuration
;xxxxxx00 Synchronous mode
;xxxxxx01 1X clock rate
;xxxxxx10 16X clock rate
;xxxxxx11 64X clock rate
;xxxx00xx 5 bit characters
;xxxx01xx 6 bit characters
;xxxx10xx 7 bit characters
;xxxx11xx 8 bit characters
;xxx0xxxx Parity disable
;xxx1xxxx Parity enable
;xx0xxxxx Odd parity generation/check
;xx1xxxxx Even parity generation/check

```

```

;00xxxxxx Invalid
;01xxxxxx 1 stop bit
;10xxxxxx 1.5 stop bits
;11xxxxxx 2 stop bits

nslin2 equ 37h ;Left serial port initialization # 2
nsrin2 equ 37h ;Right serial port initialization # 2
;76543210 Bit definations
;00110111 Default configuration
;xxxxxx1 Enable transmitter
;xxxxxx1x Assert DTR*
;xxxxxx1xx Enable reciever
;xxxxlxxx Send break character, TxD low
;xxxlxxx Reset PE, OE, FE error flags
;xxlxxxxx Assert RTS*
;xlxxxxxx Internal reset
;lxxxxxxx Enter hunt mode (for sync)

nspdats equ 0 ;Parallel data port
nspstas equ 6 ;Parallel status port

nsprbr equ 1 ;Reciever buffer ready status bit
nsptbe equ 2 ;Transmitter buffer empty status bit

nsram equ 0c0h ;North Star memory parity port,
; set to 0 for no North Star RAM

```

```

*****
*
* Left serial port routines. Use TTY: device.
*
*****

```

```

ttyin: ;Read a character
in nslsta
ani nssrbr
jz ttyin ;Wait till a character is ready
in nsldat ;Get the character
ani 7fh ;Strip parity
ret

```

```

ttyout: ;Write a character
in nslsta
ani nsstbe
jz ttyout ;Wait till the buffer is empty
mov a,c ;Write the character
out nsldat
ret

```

```

ttyist: ;Return input buffer status
in nslsta
ani nssrbr
rz ;Return not ready
mvi a,0ffh
ret ;There is a character ready

```

```

ttyost: ;Return output buffer status
in nslsta
ani nsstbe
rz ;Return not ready
mvi a,0ffh
ret ;Return ready

```

```

*****
*
* Right serial port routines. Use CRT:, PTR:, and PTP: devices.
*

```

```

*
*****
crtin:          ;Read a character
ptrin:
    in          nsrsta
    ani         nssrbr
    jz          crtin          ;Wait till a character is ready
    in          nsrdat        ;Get the character
    ani         7fh           ;Strip parity
    ret

```

```

crtout:        ;Write a character
ptpout:
    in          nsrsta
    ani         nsstbe
    jz          crtout        ;Wait till the buffer is empty
    mov         a,c           ;Write the character
    out         nsrdat
    ret

```

```

crtist:        ;Return input buffer status
ptrist:
    in          nsrsta
    ani         nssrbr
    rz          ;Return not ready
    mvi         a,0ffh
    ret          ;There is a character ready

```

```

crtost: OKI05          ;Return output buffer status
    in          nsrsta
    ani         nsstbe
    rz          ;Return not ready
    mvi         a,0ffh
    ret          ;Return ready

```

} MVI A,OFFH  
 RET

diaOST IN 2  
 ANI 80H  
 RZ  
 OKI05T MVI A,OFFH  
 RET

```

*****
*
* Parallel port routines. Use UCl:, URl:, UR2:, UP1:, UP2:, LPT:,
* and ULl: devices.
*
*****

```

```

uclin:         ;Read a character
urlin:
ur2in:
    in          nspsta
    ani         nsprbr
    jz          uclin          ;Wait till a character is ready
    in          nspdats        ;Get the character
    push        psw
    mvi         a,30h          ;Reset the parallel input flag
    out         nspsta
    pop         psw
    ani         7fh           ;Strip parity
    ret

```

```

uclout:        ;Write a character
uplout:
up2out:
lptout:
ullout:
    in          nspsta
    ani         nsptbe
    jz          uclout        ;Wait till the buffer is empty
    mvi         a,20h        ;Reset the parallel output flag

```



```

out      nspsta
mov      a,c          ;Write the character, strobe bit 7
nspout:  ori      80h
out      nspdat
ani      7fh
out      nspdat
ori      80H
out      nspdat
ret

```

```

uclist:          ;Return input buffer status
urlist:
ur2ist:

```

```

in      nspsta
ani      nsprbr
rz          ;Return not ready
mvi      a,0ffh
ret          ;Return ready

```

```

lptost:          ;Return output buffer status

```

```

ullost:
in      nspsta
ani      nsptbe
rz          ;Return not ready
mvi      a,0ffh
ret          ;Return ready

```

```

endif          ;North Star I/O configuration

```

```

*****
*
* LST: device driver routines.
*
* Routine used depends on the value of lsttyp. Possible LSTTYP
* values are listed as follows:
*
* LSTTYP is:  0      Nothing, used for patching to PROM's
*             1      Provide for 128 bytes of patch space
*             2      Multio serial, no protocol
*             3      Multio serial, Clear To Send protocol
*             4      Multio serial, Data Set Ready protocol
*             5      Multio serial, Xon/Xoff protocol
*
*****

```

```

*****
*
* lsttyp: 1      Blank space for printer driver
*
* The driver entries LSTOUT and LSTOST are defined in the CP/M
* alternation guide. Eg. Input parameters are in register C and
* results are returned in register A. The LSTSET routine is used
* for initialization code. It should execute a RET when complete.
*
* The LSTSET routine could be placed just below the CBOOT routine.
* This space (below CBOOT) is recycled for use as a disk buffer
* after CBOOT is done.
*
*****

```

```

if      lsttyp eq 1
lstset  equ      $          ;Printer initialization
lstout  equ      $          ;Printer output
lstost  equ      $          ;Printer output status
jmp     $
ds      125

```

OOT

endif

;User IO

```
*****
*
* lsttyp: 2, 3, 4, or 5 Serial printer, multi protocol
*
*****
```

```
        if      (lsttyp ge 2) and (lsttyp le 5)

lstout: call   lstost      ;Check printer status
        ora     a
        jz     lstout     ;Loop if not ready

        mov    a,c        ;Print the character
        out   thr
        ret

lstost: call   sellst     ;Printer status routine

        in     lsr        ;Check if transmitter buffer empty
        ani   thre
        rz     ;Return busy if buffer is not empty

        lhld  lstand     ;Fetch handshake mask bits

        in     msr       ;Get MODEM Status Register
        ana   l          ;Strip out hand-shake lines
        xra   h          ;Invert status
        rz     ;Return busy if printer is busy

        lda   lastch     ;Get last character recieved from the printer
        mov   b,a
        in   lsr        ;Check for a character from the printer
        ani  dr
        jz   xskip      ;Skip if no character present
        in   rbr        ;Get the character
        ani  7fh        ;Strip parity
        sta  lastch     ;Save last character recieved
        mov  b,a
xskip:  mov   a,b
        sui  xoff       ;Check for Xoff char (control S)
        jnz  xsdone     ;Printer ready
        ret   ;Printer not ready (return zero)

xsdone: mvi   a,0ffh    ;Printer ready for data
        ret

        endif          ;Multi I/O serial driver
```

```
*****
*
* Gocpm is the entry point from cold boots, and warm boots. It
* initializes some of the locations in page 0, and sets up the
* initial DMA address (80h).
*
*****
```

```
gocpm: lxi    h, buff      ;Set up initial DMA address
        call  setdma
        mvi  a, (jmp)     ;Initialize jump to warm boot
        sta  wbot
        sta  entry       ;Initialize jump to BDOS
        lxi  h, wboote    ;Set up low memory entry to CBIOS warm boot
        shld wbot+1
        lxi  h, bdos+6   ;Set up low memory entry to BDOS
```

```

shld    entry+1
xra     a                ;A ← 0
sta     bufsec          ;Set buffer to unknown state
sta     bufwrtn        ;Set buffer not dirty flag
sta     error           ;Clear buffer error flag

lda     cwflg           ;Get cold/warm boot flag
ora     a
lxi     h,coldmes      ;Pointer to initial cold command
jz      cldcmdnd
lxi     h,warmes       ;Pointer to initial warm command
cldcmdnd:mov e,m        ;Do one level of indirection
inx     h
mov     d,m
ldax   d                ;Get command length
inr    a                ;Bump length to include length byte itself
lxi    h,ccp+7         ;Command buffer (includes length byte)
mov    c,a              ;Set up for block move
mvi    b,0
call   movbyt          ;Move command to internal CCP buffer
lda    cwflg           ;Figure out whether or not to send message
ora    a
lda    autoflg
jz     cldbot
rar
cldbot: rar
lda    cdisk           ;Jump to CP/M with currently selected disk in C
mov    c,a
jc     ccp              ;Enter CP/M, send message
jmp    ccp+3           ;Enter CP/M, no message

```

```

cwflg: db 0             ;Cold/warm boot flag

```

```

*****
*
* WBOOT loads in all of CP/M except the CBIOS, then initializes *
* system parameters as in cold boot. See the Cold Boot Loader *
* listing for exactly what happens during warm and cold boots. *
*
*****

```

```

wboot: lxi    sp,tpa    ;Set up stack pointer
mvi    a,1
sta    cwflg          ;Set cold/warm boot flag

mvi    h,wmdrive     ;Move drive to warm boot off of into (h)
mvi    l,d$wboot     ;Perform warm boot operation
call   jumper
jnc    gocpm          ;No error

hlt
db     0

jmp    wboot          ;In case user restarts the computer

```

```

*****
*
* Setsec just saves the desired sector to seek to until an *
* actual read or write is attempted. *
*
*****

```

```

setsec: mov    h,b      ;Enter with sector number in (bc)
mov    l,c
shld   cpmsec
donop: ret

```

```
*****
*
* Setdma saves the DMA address for the data transfer.
*
*****
```

```
setdma: mov     h,b           ;Enter with DMA address in (bc)
        mov     l,c
        shld   cpmdma        ;CP/M dma address
        ret
```

```
*****
*
* Home is translated into a seek to track zero.
*
*****
```

```
home:   lda     bufwrtn       ;Test buffer dirty flag
        ora     a
        jnz    dohome        ;Skip buffer disable if buffer dirty
        xra     a            ;Invalidate buffer on home call
        sta     bufsec
dohome: lxi     b,0           ;Track to seek to
```

```
*****
*
* Settrk saves the track # to seek to. Nothing is done at this
* point, everything is deffered until a read or write.
*
*****
```

```
settrk: mov     h,b           ;Enter with track number in (bc)
        mov     l,c
        shld   cpmtrk
        ret
```

```
*****
*
* Sectran translates a logical sector number into a physical
* sector number.
*
*****
```

```
sectran:lda     cpmdrv        ;Get the Drive Number
        mov     h,a          ;Drive in (h)
        mvi    l,d$stran
        jmp     jumper       ;See device level sector translation routines
```

```
*****
*
* Setdrv selects the next drive to be used in read/write
* operations. If the drive has never been selected it calls
* a low level drive select routine that should perform some
* sort of check if the device is working. If not working then
* it should report an error. If the logical drive has been
* selected before then setdrv just returns the DPH without
* checking the drive.
*
*****
```

```
setdrv: mov     a,c           ;Save the logical drive number
        sta     cpmdrv
        cpi    maxlog        ;Check for a valid drive number
        jnc    zret          ;Illegal drive
```

```

mov     a,e           ;Check if bit 0 of (e) = 1
ani     1
jnz     setd3         ;Drive has allready been accessed

mov     h,c           ;Move logical drive into (h)
mvi     1,d$sel1
call    jumper       ;Call low level drive select
mov     a,h           ;Check if the low level drive select returned
ora     1             ; zero to indicate an error
jz      zret         ;Yes, an error so report to CP/M

push    h             ;Save DPH address
call    gdph         ;Get entry if DPH save table
pop     d             ;DPH -> (de)
mov     m,e           ;Put address of DPH in table
inx     h
mov     m,d
inx     h
mov     m,c           ;Put sector size in table
inx     h
mov     a,m           ;Check if bad map has ever been read for this
ora     a             ; drive
cz      getbad       ;Never been read so read in bad map
xchg    ;DPH -> (hl)

setd0:  mov     a,c           ;Move sector size code into (a)
        sta     secsiz      ;Save sector size
        xra     a
setd1:  dcr     c             ;Create number of (128 bytes/physical sector)-1
        jz      setd2
        rlc
        ori     1
        jmp     setd1
setd2:  sta     secpssec     ;Save for deblocking
        lda     cpmdrv      ;Save current drive as old drive
        sta     lastdrv    ; in case of select errors
        ret

setd3:  push    d             ;Save DPH address
        mov     h,c         ;Drive in (h)
        mvi     1,d$sel2    ;Select drive
        call    jumper     ;Quick select
        call    gdph
        pop     d
        mov     e,m         ;DPH -> (de)
        inx     h
        mov     d,m
        inx     h
        mov     c,m         ;Sector size -> (c)
        xchg    ;DPH -> (hl)
        jmp     setd0

gdph:   lda     cpmdrv      ;Return pointer to DPH save area
        rlc
        rlc
        mov     e,a
        mvi     d,0
        lxi     h,dphtab   ;DPH save area table
        dad     d           ;Add offset
        ret               ;(hl) = DPH save area for current drive

zret:   lxi     h,0         ;Seldrv error exit
        lda     lastdrv    ;Get last selected drive
        mov     c,a
        lda     cdisk      ;Pick up user/drive
        ani     0f0h       ;Save user number

```

```
ora c ;Put together with old drive
sta cdisk
ret
```

```
*****
*
* DPH save area. Each entry is 4 bytes long:
* 0 - LSB of DPH address
* 1 - MSB of DPH address
* 2 - Sector size code (1 = 128, 2 = 256, 3 = 512...)
* 3 - Bad map has been initialized (0 = Uninitialized)
*
*****
```

```
dphtab: rept maxlog*4
db 0
endm
```

```
*****
*
* Getbad - Check if a device has a bad map. If the device has
* a bad sector map then append bad entries to end of badmap
* table.
*
*****
```

```
getbad: mvi m,1 ;Set drive initialized
push b
push d
lda cpmdrv ;Pick up current drive
mov h,a ;Call drive routine to return a pointer to
mvi l,d$bad ;the track and sector of the bad map
call jumper

mov a,h ;If routine returns 0 then the device has
ora l ; no bad sector map
jz badret

mov e,m ;Pick up track number of bad sector map -> (de)
inx h
mov d,m
inx h
xchg
shld cpmtrk
xchg
mov a,m ;Pick up sector number of of bad sector map
inx h
mov h,m
mov l,a
shld truesec
call fill ;Read in bad sector map into the buffer
rc

lhld badptr ;Pick up bad map pointer
lxi d,buffer ;Start at beginning of buffer
badl: ldax d ;Pick up an entry from the buffer
ora a
jz bade ;All done
mov a,m ;Pick up entry from bad map table
inr a
jz overflo ;Bad map overflow
lda cpmdrv ;Put drive in table
mov m,a
inx h
lxi b,8
call movbyt ;Move the rest of information into the table
```

```

        jmp      badl

bade:   shld    badptr      ;Restore new bad map pointer
badret: pop     d
        pop     b
        ret

overflo: lxi    h,omes
        call   message
        jmp    badret

omes:   db     0dh, 0ah, 'BAD MAP OVERFLOW!', 0dh, 0ah, 0

nobad:  lxi    h,0          ;Used by device drives to indicate no bad
        ret          ; sector map

badptr: dw     badmap      ;Pointer to next available bad map entry

*****
*
* Write routine moves data from memory into the buffer. If the
* desired CP/M sector is not contained in the disk buffer, the
* buffer is first flushed to the disk if it has ever been
* written into, then a read is performed into the buffer to get
* the desired sector. Once the correct sector is in memory, the
* buffer written indicator is set, so the buffer will be
* flushed, then the data is transferred into the buffer.
*
*****

write:  mov     a,c          ;Save write command type
        sta    writtyp
        mvi    a,1          ;Set write command
        jmp    rwent

*****
*
* Read routine to buffer data from the disk. If the sector
* requested from CP/M is in the buffer, then the data is simply
* transferred from the buffer to the desired dma address. If
* the buffer does not contain the desired sector, the buffer is
* flushed to the disk if it has ever been written into, then
* filled with the sector from the disk that contains the
* desired CP/M sector.
*
*****

read:   xra     a          ;Set the command type to read
        if     nostand ne 0
        sta    unalloc    ;Clear unallocated write flag
        endif
rwent:  sta    rdwr        ;Save command type

*****
*
* Redwrt calculates the physical sector on the disk that
* contains the desired CP/M sector, then checks if it is the
* sector currently in the buffer. If no match is made, the
* buffer is flushed if necessary and the correct sector read
* from the disk.
*
*****

redwrt: mvi     b,0          ;The 0 is modified to contain the log2
secsiz equ    $-1          ; of the physical sector size/128
        ; on the currently selected disk

```

```

    lhd    cpmsec      ;Get the desired CP/M sector #
    mov    a,h
    ani    80h         ;Save only the side bit
    mov    c,a        ;Remember the side
    mov    a,h
    ani    7fh        ;Forget the side bit
    mov    h,a
    dcx    h          ;Temporary adjustment
divloop:dcr    b      ;Update repeat count
    jz    divdone
    ora    a
    mov    a,h
    rar
    mov    h,a
    mov    a,l
    rar          ;Divide the CP/M sector # by the size
                ; of the physical sectors

    mov    l,a
    jmp    divloop ;
divdone:inx    h
    mov    a,h
    ora    c          ;Restore the side bit
    mov    h,a
    shld   truesec   ;Save the physical sector number
    lxi    h,cpmdrv  ;Pointer to desired drive,track, and sector
    lxi    d,bufdrv  ;Pointer to buffer drive,track, and sector
    mvi    b,6       ;Count loop
dtslop:dcr    b      ;Test if done with compare
    jz    move       ;Yes, match. Go move the data
    ldax   d         ;Get a byte to compare
    cmp    m         ;Test for match
    inx    h         ;Bump pointers to next data item
    inx    d
    jz    dtslop     ;Match, continue testing

*****
*
* Drive, track, and sector don't match, flush the buffer if
* necessary and then refill.
*
*****

    call   fill      ;Fill the buffer with correct physical sector
    rc          ;No good, return with error indication

*****
*
* Move has been modified to cause either a transfer into or out
* the buffer.
*
*****

move:   lda    cpmsec ;Get the CP/M sector to transfer
    dcr    a      ;Adjust to proper sector in buffer
    ani    0      ;Strip off high ordered bits
secpsec equ    $-1 ;The 0 is modified to represent the # of
                ; CP/M sectors per physical sectors

    mov    l,a    ;Put into HL
    mvi    h,0
    dad    h      ;Form offset into buffer
    dad    h
    dad    h
    dad    h
    dad    h
    dad    h
    dad    h
    dad    h

```



```

    lxi    d,buffer      ;Beginning address of buffer
    dad    d              ;Form beginning address of sectgr to transfer
    xchg   ;DE = address in buffer
    lxi    h,0           ;Get DMA address, the 0 is modified t/
    ;          contain the DMA address

cpmdma  equ    $-2
    mvi    a,0           ;The zero gets modified to contain
    ;          a zero if a read, or a 1 if write

rdwr    equ    $-1
    ana    a              ;Test which kind of operation
    jnz    into          ;Transfer data into the buffer

outof:  call   movl28
    lda    error         ;Get the buffer error flag
    ret

into:   xchg           ;
    call   movl28       ;Move the data, HL = destination
    ;          DE = source

    mvi    a,1          ;Set buffer written into flag
    sta    bufwrtn      ;Check for directory write
    mvi    a,0
writtyp equ    $-1
    dcr    a             ;Test for a directory write
    mvi    a,0
    rnz    ;No error exit

```

```

*****
*
* Flush writes the contents of the buffer out to the disk if
* it has ever been written into.
*
*****

```

```

flush:  mvi    a,0           ;The 0 is modified to reflect if
    ;          the buffer has been written into

bufwrtn equ    $-1
    ora    a              ;Test if written into
    rz     ;Not written, all done
    mvi    a,d$write
    sta    rwop+1
    call   prep           ;Do the physical write
    sta    error         ;Set up the error flag
    ret

```

```

*****
*
* Prep prepares to read/write the disk. Retries are attempted.
* Upon entry, H&L must contain the read or write operation
* address.
*
*****

```

```

prep:   call   alt         ;Check for alternate sectors
    di          ;Reset interrupts
    xra    a           ;Reset buffer written flag
    sta    bufwrtn

    mvi    b,retries    ;Maximum number of retries to attempt
retrylp:push  b         ;Save the retry count

    mvi    l,d$sel2     ;Select drive
    call   jumpbuf

    lhld   alttrk       ;Track number -> (hl)

    mov    a,h          ;Test for track zero

```

```

ora      l
push     h          ;Save track number
mvi     l,d$home
cz      jumpbuf
pop      b          ;Restore track #

mvi     l,d$strk
call    jumpbuf

lhld    altsec     ;Sector -> (hl)
mov     b,h
mov     c,l

mvi     l,d$ssec
call    jumpbuf

lxi     b,buffer   ;Set the DMA address
mvi     l,d$sdma
call    jumpbuf

rwop:   mvi     l,0   ;Get operation address
call    jumpbuf

pop     b          ;Restore the retry counter
mvi     a,0        ;No error exit status
rnc     ;Return no error
dcr     b          ;Update the retry counter
stc     ;Assume retry count expired
mvi     a,0ffh    ;Error return
rz      ;Return sad news
mov     a,b
cpi     retries/2
jnz     retrylp   ;Try again
push    b          ;Save retry count
mvi     l,d$home  ;Home drive after (retries/2) errors
call    jumpbuf
pop     b
jmp     retrylp   ;Try again

*****
*
* Fill fills the buffer with a new sector from the disk.
*
*****

fill:   call    flush      ;Flush buffer first
rc      ;Check for error
lxi     d,cpmdrv   ;Update the drive, track, and sector
lxi     h,bufdrv
lxi     b,5        ;Number of bytes to move
call    movbyt     ;Copy the data

lda     rdwr       ;Test read write flag
ora     a
jz      fread     ;Skip write type check if reading
lda     writtyp    ;0 = alloc, 1 = dir, 2 = unalloc

if     nostand ne 0 ;Do non standard (but quick and dirty) check
ora     a
jnz     fnalloc    ;Skip if not an allocated write

lda     unalloc    ;Check unallocated write in progress flag
ora     a
jz      fwritin    ;We are doing an allocated write
lhld    cblock     ;Get current block address

```

```

xchg      lhld      oblock      ; and old block address
mov       a,d        ;Compare old versus new
cmp       h
jnz      awritin    ;Different, clear unallocated writing mode
mov       a,e
cmp       l
jnz      awritin
lxi      h,cpmdrv   ;Test for different drive
lda      unadr     ;
cmp       m
jnz      awritin    ;Drive is different, clear unallocated mode
ret       ;Unallocated write, do nothing...

fnaloc:  dcr       a
jz       awritin    ;Do a directory write
                    ;We are now doing an unallocated write
                    ;Save current block number
lhld     cblock
shld     oblock
lda      cpmdrv    ;Save drive that this block belongs to
sta      unadr
mvi     a,l        ;Set unallocated write flag
sta      unaloc
ret       ; and we do nothing about the write

awritin: xra       a        ;Clear unallocated writing mode
sta      unaloc

        else      ;Do standard unallocated test

sui      2          ;Test for an unallocated write
rz

endif

fwritin: lda      secsiz   ;Check for 128 byte sectors
dcr      a
rz       ;No deblocking needed

fread:  mvi      a,d$read
sta      rwop+1
call     prep       ;Read the physical sector the buffer
sta      error     ;Set the error status
ret

```

```

*****
*
* Jumpbuf, jumper are used to dispatch to a low level device
* subroutine. Jumper is called with the drive in (h) and the
* routine number (see description above) in (l). It passes
* along the (bc) and (de) registers unaltered. Jumpbuf is
* a call to jumper with the drive number from bufdrv.
*
*****

```

```

jumpbuf: lda      bufdrv   ;Dispatch with bufdrv for drive
mov      h,a

jumper:  push     d
push     b
push     h
mov      a,h          ;Logical drive into (a)
lxi     d,dsttab     ;Drive specification pointer table
jump1:  mov      c,a      ;Save logical in (c)
ldax    d
mov     l,a

```

```

inx      d
ldax    d
mov     h,a      ;Get a DST pointer in (hl)
inx     d
mov     a,c      ;Logical in (a)
sub     m        ;Subtract from first entry in DST
jnc     jumpl    ;Keep scanning table till correct driver found

inx     h        ;Bump (hl) to point to start of dispatch table
pop     d        ;Real (hl) -> (de)
mov     a,e      ;Move offset number into (a)
rlc     ;Each entry is 2 bytes
mov     e,a      ;Make an offset
mvi     d,0
dad     d        ;(hl) = **Routine
mov     a,m      ;Pick up address of handler for selected
inx     h        ; function
mov     h,m
mov     l,a      ;(hl) = *routine
mov     a,c      ;Logical in (a)
pop     b        ;Restore saved registers
pop     d
pchl

```

```

*****
*
* Check for alternate sectors in bad sector table.  If an
* alternate sector is found replace attrk and altsec with
* new sector number else pass along unaltered.
*
*****

```

```

alt:    lxi     h,badmap      ;Address of bad map -> (hl)
        lda     bufdrv      ;Pick up drive number currently working on
        mov     c,a        ;Move drive into (c) for speed in search
all:    xchg    badptr      ;Get bad map pointer
        lhld   badptr      ; -> (de)
        mov     a,d        ;Check if at end of bad map table
        cmp    h
        jnz    alt2       ;Still more
        mov     a,e
        cmp    l
        jnz    alt2       ;Still more
        lhld   buftrk      ;No alternate sector so use selected sector
        shld   attrk
        lhld   bufsec
        shld   altsec
        ret

```

```

alt2:   push   h            ;Save current bad map entry address
        mov   a,c          ;Move drive into (a)
        cmp   m            ;Check if drive in table matches
        jnz  altmis       ;Does not match skip this entry
        inx  h             ;Point to LSB of alternate track
        lda  buftrk        ;Pick up LSB of buffer track
        cmp  m
        jnz  altmis
        inx  h             ;Point to MSB alternate track
        lda  buftrk+1     ;Pick up MSB of buffer track
        cmp  m
        jnz  altmis
        inx  h             ;Point to LSB of alternate sector
        lda  bufsec       ;Pick up LSB of buffer sector
        cmp  m
        jnz  altmis

```

```

inx      h          ;Point to MSB of alternate sector
lda      bufsec+1   ;Pick up MSB of buffer sector
cmp      m
jnz      altmis     ;Found an alternate sector
inx      h          ;Point to real info on the alternate sector
lxi      d,alttrk
xchg
push     b
lxi      b,4
call    movbyt     ;Move alternate sector info in correct place
pop      b
pop      h
ret

```

```

altmis:  pop      h          ;Current alternate did not match
         lxi      d,9       ;Bump pointer by the length of an entry
         dad      d
         jmp      all       ;Loop for more

```

```

*****
*
* Mover moves 128 bytes of data. Source pointer in DE, Dest
* pointer in HL.
*
*****

```

```

movl28: lxi      b,128     ;Length of transfer
movbyt: xra      a         ;Check if host processor is a Z80
         adi      3
         jpo      z80mov   ;Yes, Its a Z80 so use block move

```

```

m8080:  ldax     d         ;Get a byte of source
         mov      m,a     ;Move it
         inx     d         ;Bump pointers
         inx     h
         dcx     b         ;Update counter
         mov      a,b     ;Test for end
         ora     c
         jnz     m8080
         ret

```

```

z80mov: xchg
         dw      0b0edh   ;ldir
         xchg
         ret

```

```

*****
*
* Return DPH pointer. Enter with (de) with DPH base address
* and (a) with logical drive number. Returns with DPH address
* in (hl).
*
*****

```

```

retdph  mov      l,a       ;Move logical drive into (l)
         mvi     h,0
         dad     h         ;Multiply by 16 (size of DPH)
         dad     h
         dad     h
         dad     h
         dad     d         ;(hl) = pointer to DPH
         ret

```

```

*****
*
* Utility routine to output the message pointed at by (hl)
*

```

```

* terminated with a null.
*
*****
message:mov    a,m          ;Get a character of the message
            inx    h          ;Bump text pointer
            ora    a          ;Test for end
            rz     ;Return if done
            push   h          ;Save pointer to text
            mov    c,a        ;Output character in C
            call   cout       ;Output the character
            pop    h          ;Restore the pointer
            jmp    message    ;Continue until null reached

```

```

*****
*
* The following code is for the Diskus Hard disk
*
*****

```

```

            if      hdca ne 0          ;Want HDC3 or 4 controller included ?

hdorg      equ     50h                ;Hard Disk Controller origin

hdstat     equ     hdorg              ;Disk Status
hdcntl     equ     hdorg              ;Disk Control
hdreslt    equ     hdorg+1            ;Disk Results
hdcmdnd    equ     hdorg+1            ;Disk Commands
hdskomp    equ     hdorg+2            ;Seek complete clear port (on HDC4)
hdfunc     equ     hdorg+2            ;Function port
hddata     equ     hdorg+3            ;Data port

;      Status port (50)

tkzero     equ     01h                ;Track zero
opdone     equ     02h                ;Operation done
complt     equ     04h                ;Seek complete
tmout      equ     08h                ;Time out
wfault     equ     10h                ;Write fault
drvrdy     equ     20h                ;Drive ready
index      equ     40h                ;Delta index

;      Control port (50)

hdfren     equ     01h                ;Enable external drivers
hdrun      equ     02h                ;Enable controllers state machine
hdclok     equ     04h                ;Clock source control bit, high = disk
hdwpert    equ     08h                ;Write protect a drive

;      Result port (51)

retry      equ     02h                ;Retry flag

;      Command port (51)

idbuff     equ     0                  ;Initialize data buffer pointer
rsect      equ     1                  ;Read sector
wssect     equ     5                  ;Write sector
isbuff     equ     8                  ;Initialize header buffer pointer

;      Function port (52)

pstep      equ     04h                ;Step bit
nstep      equ     0ffh-pstep         ;Step bit mask
null       equ     0fch                ;Null command

```

```

; Misc constants

hdrlen equ 4 ;Sector header length
seclen equ 512 ;Sector data length

*****
*
* Device Specification Table for HDCA controller driver
*
*****

hddst: db maxhd*hdlog ;Number of logical drives
        dw hdwarm ;Warm boot
        dw hdtran ;Sector translation
        dw hldrv ;First time select
        dw hddrv ;General select
        dw hdhome ;Home current selected drive
        dw hdseek ;Seek to selected track
        dw hdsec ;Select sector
        dw hddma ;Set DMA address
        dw hdread ;Read a sector
        dw hdwrite ;Write a sector
        dw nobad ;No bad sector map

hdwarm: call divlog ;Get physical drive number in (c)
        xra a
        lxi h,ccp-200h ;Initial DMA address
        push h
        sta head ;Select head zero
        inr a ; 1 -> (a)
        push psw ;Save first sector - 1
        call hdd2 ;Select drive
        mvi c,0
        call hdhome ;Home the drive
hdwrl: pop psw ;Restore sector
        pop h ;Restore DMA address
        inr a
        sta hdsect
        cpi 13 ;Past BDOS ?
        rz ;Yes, all done
        inr h ;Update DMA address
        inr h
        shld hdadd
        push h
        push psw
hdwrrd: lxi b,retries*100h+0 ;Retry counter
hdwr: push b ;Save the retry count
        call hdread ;Read the sector
        pop b
        jnc hdwrl ;Test for error
        dcr b ;Update the error count
        jnz hdwr ;Keep trying if not too many errors
        stc ;Error flag
        ret

hdtran: mov h,b ;Sector translation is handled via
        mov l,c ; physical sector header skewing
        inx h
        ret

hldrv: sta hcur ;Save logical disk
        call divlog ;Divide by logical disks per drive
        mov a,c
        sta hddisk ;Save new physical drive
        call hdptr ;Get track pointers
        mov a,m ;Get current track

```

```

inr      a                ;Check if -1
jnz     hd12             ;Nope, allready accessed
ori     null             ;Select drive
out     hdfunc
mvi     a,hdfren+hdclck ;Enable drivers
out     hdcntl
mvi     c,239           ;Wait 2 minutes for disk ready
lxi     h,0
hdtdel: dcx     h
mov     a,h
ora     l
cz     dcrc
jz     zret             ;Drive not ready error
in     hdstat           ;Test if ready yet
ani     drvrdy
jnz     hdtdel

if     not fujitsu
lxi     h,0             ;Time one revolution of the drive
mvi     c,index
in     hdstat
ana     c
mov     b,a             ;Save current index level in B
hdinxdl:in     hdstat
ana     c
cmp     b               ;Loop untill index level changes
jz     hdinxdl
hdindx2:inx     h
in     hdstat           ;Start counting untill index returns to
ana     c               ; previous state
cmp     b
jnz     hdindx2

if     m10              ;Memorex M10's have 40 ms head settle
dad     h               ;HL*2
endif

if     m26              ;Shugart M26's have 30 ms head settle
xra     a               ;HL/2 + HL (same as HL*1.5)
mov     a,h
rar
mov     d,a
mov     a,l
rar
mov     e,a
dad     d
endif

shld    settle         ;Save the count for timeout delay
endif

call    hdhome

hd12:   lda     hcur     ;Load logical drive
lxi     d,dphhd0       ;Start of hard disk DPH's
mvi     c,3            ;Hard disk sector size equals 512 bytes
jmp     retsph

dcrc:   dcr     c       ;Conditional decrement C routine
ret

divlog: mvi     c,0
divlx:  sui     hdlog
rc
inr     c
jmp     divlx

```



```

hddrv: sta hcur
      call divlog ;Get the physical drive #
hdd2:  mov a,c
      sta hddisk ;Select the drive
      ori null
      out hdfunc
      mvi a,hdfren+hdrun+hdclck+hdwprt ;Write protect
      out hdcntl
      ret

hdhome: call hdptr ;Get track pointer
      mvi m,0 ;Set track to zero
      in hdstat ;Test status
      ani tkzero ;At track zero ?
      rz ;Yes

      if not fujitsu
hdstepo:in hdstat ;Test status
      ani tkzero ;At track zero ?
      jz hddelay
      mvi a,1
      stc
      call accok ;Take one step out
      jmp hdstepo
      else
      xra a
      jmp accok
      endif

      if not fujitsu
hddelay:lhld settle ;Get hddelay
deloop: dcx h ;Wait 20ms
      mov a,h
      ora l
      inx h
      dcx h
      jnz deloop
      ret
      endif

hdseek: call hdptr ;Get pointer to current track
      mov e,m ;Get current track
      mov m,c ;Update the track
      mov a,e ;Need to seek at all ?
      sub c
      rz
      cmc ;Get carry into direction
      jc hdk2
      cma
      inr a
      if fujitsu
hdk2: jmp accok
      else
hdk2: call accok
      jmp hddelay
      endif

accok: mov b,a ;Prep for build
      call build

sloop: ani nstep ;Get step pulse low
      out hdfunc ;Output low step line
      ori pstep ;Set step line high
      out hdfunc ;Output high step line

```

```

dcr      b                ;Update repeat count
jnz      sloop            ;Keep going the required # of tracks
jmp      wsdone

hddma:   mov      h,b      ;Save the DMA address
         mov      l,c
         shld    hdadd
         ret

wsdone:  in        hdstat   ;Wait for seek complete to finish
         ani      complt
         jz       wsdone
         in        hdskomp  ;Clear sdone bit on an HDCA4
         ret

hdsec:   if      m26
         mvi      a,01fh    ;For compatibility with Cbios revs.
                               ; 2.3 and 2.4
         ana      c        ;Mask in sector number (0-31)
         cz       getspt   ;Translate sector 0 to sector 32
         sta      hdsect   ;Save translated sector number (1-32)
         mvi      a,0e0h   ;Get the head number
         ana      c
         rlc
         rlc
         rlc
         sta      head     ;Save the head number
getspt:  mvi      a,hdspt
         ret

         else

hdsec:   mov      a,c
         call    divspt
         adi     hdspt
         ana     a
         cz     getspt
         sta     hdsect
         mov     a,c
         sta     head
getspt:  mvi      a,hdspt
         dcr     c
         ret

divspt:  mvi      c,0
divsx:   sui      hdspt
         rc
         inr     c
         jmp     divsx
endif

hddread: call     hdprep
         rc
         xra     a
         out    hdcmd
         cma
         out    hddata
         out    hddata
         mvi    a,rsect ;Read sector command
         out    hdcmd
         call   process
         rc
         xra     a
         out    hdcmd
         mvi    b,seclen/4
         lhld   hdadd

```

```

in      hddata
in      hddata
rtloop: in      hddata      ;Move four bytes
mov     m,a
inx     h
in      hddata
mov     m,a
inx     h
in      hddata
mov     m,a
inx     h
in      hddata
mov     m,a
inx     h
dcr     b
jnz     rtloop
ret

hdwrite:call  hdprep      ;Prepare header
rc
xra     a
out     hdcmdnd
lhld   hdadd
mvi     b,seclen/4
wtloop: mov     a,m      ;Move 4 bytes
out     hddata
inx     h
mov     a,m
out     hddata
inx     h
mov     a,m
out     hddata
inx     h
mov     a,m
out     hddata
inx     h
dcr     b
jnz     wtloop
mvi     a,wsect      ;Issue write sector command
out     hdcmdnd
call   process
rc
mvi     a,wfault
ana     b
stc
rz
xra     a
ret

process:in   hdstat      ;Wait for command to finish
mov     b,a
ani     opdone
jz      process
mvi     a,hdfren+hdrun+hdclk ;Write protect
out     hdcntl
in      hdstat
ani     tmout      ;Timed out ?
stc
rnz
in      hreslt
ani     retry      ;Any retries ?
stc
rnz
xra     a
ret

```

```

hdprep:  in      hdstat
        ani      drvrdy
        stc
        rnz
        mvi      a,isbuff      ;Initialize pointer
        out      hdcmdnd
        call     build
        ori      0ch
        out      hdfunc
        lda      head
        out      hddata      ;Form head byte
        call     hdptr      ;Get pointer to current drives track
        mov      a,m      ;Form track byte
        out      hddata
        ana      a
        mvi      b,80h
        jz      zkey
        mvi      b,0
zkey:   lda      hdsect      ;Form sector byte
        out      hddata
        mov      a,b
        out      hddata
        mvi      a,hdfren+hdrun+hdclok ;Write protect
        out      hdcntl
        mvi      a,hdfren+hdrun+hdclok+hdwprt ;Write protect
        out      hdcntl
        xra      a
        ret

hdptr:  lhld     hddisk      ;Get a pointer to the current drives
        mvi      h,0      ; track position
        xchg
        lxi      h,hdtrak
        dad      d
        ret

build:  lda      head      ;Build a controller command byte
        ral
        ral
        ral
        ral
        lxi      h,hddisk
        ora      m
        xri      0f0h
        ret

hdcur:  db      0      ;Current logical disk
hdadd:  dw      0      ;DMA address
hddisk: db      0      ;Current physical disk number
head:   db      0      ;Current physical head number
hdsect: db      0      ;Current physical sector number

hdtrak: db      0ffh     ;Track pointer for each drive
        db      0ffh     ;All drive default to an uncalibrated
        db      0ffh     ; state (ff)
        db      0ffh

settle: dw      0      ;Time delay constant for head settle

        endif

```

\*\*\*\*\*

\*

\* The following equates relate the Morrow Designs 2D/B

\* controller. If the controller is non standard (0F800H)

\* only the FDORIG equate need be changed.

\*

\*

\*

\*

```

*
*****
if      maxfd ne 0      ;Include Discus 2D ?
fdorig  equ  0f800h     ;Origin of Disk Jockey PROM
fdboot  equ  fdorig+00h ;Disk Jockey 2D initialization
fdcin   equ  fdorig+03h ;Disk Jockey 2D character input routine
fdcout  equ  fdorig+06h ;Disk Jockey 2D character output routine
fdhome  equ  fdorig+09h ;Disk Jockey 2D track zero seek
fdseek  equ  fdorig+0ch ;Disk Jockey 2D track seek routine
fdsec   equ  fdorig+0fh ;Disk Jockey 2D set sector routine
fddma   equ  fdorig+12h ;Disk Jockey 2D set DMA address
fdread  equ  fdorig+15h ;Disk Jockey 2D read routine
fdwrite equ  fdorig+18h ;Disk Jockey 2D write routine
fdsel   equ  fdorig+1bh ;Disk Jockey 2D select drive routine
fdtstat equ  fdorig+21h ;Disk Jockey 2D terminal status routine
fdstat  equ  fdorig+27h ;Disk Jockey 2D status routine
fderr   equ  fdorig+2ah ;Disk Jockey 2D error, flash led
fd den  equ  fdorig+2dh ;Disk Jockey 2D set density routine
fdside  equ  fdorig+30h ;Disk Jockey 2D set side routine
fdram   equ  fdorig+400h ;Disk Jockey 2D RAM address
dblaid  equ  20h        ;Side bit from controller
io      equ  fdorig+3f8h ;Start of I/O registers
dreg    equ  io+1
cmdreg  equ  io+4
clrcmd  equ  0d0h

```

```

*****
*
* Device Specification Table for the Disk Jockey 2D/B
*
*****

```

```

fddst:  db      maxfd      ;Number of logical drives
        dw      fdwarm     ;Warm boot
        dw      fdtran     ;Sector translation
        dw      fdldrv     ;Select drive 1
        dw      fd sel2    ;Select drive 2
        dw      fd lhome   ;Home drive
        dw      fd seek    ;Seek to specified track
        dw      fd ssec    ;Set sector
        dw      fddma     ;Set DMA address
        dw      fd read    ;Read a sector
        dw      fd write   ;Write a sector
        dw      nobad     ;No bad sector map

```

```

*****
*
* Floppy disk warm boot loader
*
*****

```

```

fdwarm: mov     c,a
        call    fd sel     ;Select drive A
        mvi    c,0        ;Select side 0
        call    fd side
wrmfail: call    fd home    ;Track 0, single density
        jc     wrmfail     ;Loop if error

;The next block of code re-initializes
; the warm boot loader for track 0
        mvi    a,5-2      ;Initialize the sector to read - 2
        sta    newsec
        lxi    h,cop-100h ;First revolution DMA - 100h
        shld   newdma
;Load all of track 0

```

```

t0boot: mvi    a,5-2          ;First sector - 2
newsec  equ    $-1
        inr    a              ;Update sector #
        inr    a
        cpi    27             ;Size of track in sectors + 1
        jc     nowrap         ;Skip if not at end of track
        jnz    tlboot        ;Done with this track
        sui    27-6          ;Back up to sector 6
        lxi    h,ccp-80h     ;Memory address of sector - 100h
        shld   newdma
nowrap:  sta    newsec        ;Save the updated sector #
        mov    c,a
        call   fdsec         ;Set up the sector
        lxi    h,ccp-100h    ;Memory address of sector - 100h
newdma  equ    $-2
        lxi    d,100h        ;Update DMA address
        dad    d
nowrp:  shld   newdma        ;Save the updated DMA address
        mov    b,h
        mov    c,l
        call   fddma        ;Set up the new DMA address
        lxi    b,retries*100h+0;Maximum # of errors, track #
wrmfred:push b
        call   fdseek        ;Set up the proper track
        call   fdread        ;Read the sector
        pop    b
        jnc    t0boot        ;Continue if no error
        dcr    b
        jnz    wrmfred       ;Keep trying if error
        jmp    fderr         ;Too many errors, flash the light

;Load track 1, sector 1, sector 3 (partial), sector 2 (1024 byte sectors)

tlboot: mvi    c,1           ;Track 1
        call   fdseek
        lxi    b,ccp+0b00h   ;Address for sector 1
        lxi    d,10*100h+1   ;Retry count + sector 1
        call   wrmread
        lxi    b,ccp+0f00h   ;Address for sector 2
        lxi    d,10*100h+3   ;Retry count + sector 3
        call   wrmread

        lxi    b,0300h       ;Size of partial sector
        lxi    d,ccp+1300h   ;Address for sector 3
        lxi    h,ccp+0f00h   ;Address of sector 3

wrmcpy: mov    a,m           ;Get a byte and
        stax   d              ; save it
        inx   d              ;Bump pointers
        inx   h
        dcx   b              ;Bump counter
        mov   a,b            ;Check if done
        ora   c
        jnz   wrmcpy        ; if not, loop

        lxi    b,ccp+0f00h   ;Address for sector 2
        lxi    d,10*100h+2   ;Retry count + sector 2
        call   wrmread

        xra   a              ;Clear error indicator
        ret

wrmread:push d
        call   fddma        ;Set DMA address
        pop    b

```

```

call      fdsec      ;Set sector
wrmfrd:  push      b      ;Save error count
call      fdread     ;Read a sector
jc        wrmerr     ;Do retry stuff on error
call      fdstat     ;Sector size must be 1024 bytes
ani      0ch        ;Mask length bits
sui      0ch        ;Carry (error) will be set if < 0c0h
wrmerr:  pop        b      ;Fetch retry count
rnc      ;Return if no error
dcr      b          ;Bump error count
jnz      wrmfrd
jmp      fderr      ;Error, flash the light

fdtran:  inx        b
push     d          ;Save table address
push     b          ;Save sector #
call    fdget      ;Get DPH for current drive
lxi     d,10       ;Load DPH pointer
dad     d
mov     a,m
inx     h
mov     h,m
mov     l,a
mov     a,m        ;Get # of CP/M sectors/track
ora     a          ;Clear carry
rar     ;Divide by two
sub     c          ;Subtract sector number
push    psw        ;Save adjusted sector
jm     sidetwo

sidea:   pop        psw  ;Discard adjusted sector
pop     b          ;Restore sector requested
pop     d          ;Restore address of xlt table
sideone:xchg      ;hl <- &(translation table)
dad     b          ;bc = offset into table
mov     l,m        ;hl <- physical sector
mvi    h,0
ret

sidetwo:call     fdgsid  ;Check out number of sides
jz     sidea     ;Single sided
pop    psw      ;Retrieve adjusted sector
pop    b
cma     ;Make sector request positive
inr    a
mov    c,a     ;Make new sector the requested sector
pop    d
call   sideone
mvi   a,80h   ;Side two bit
ora   h       ; and sector
mov   h,a
ret

fdldrv: sta     fdlog   ;Save logical drive
mov    c,a    ;Save drive #
mvi   a,0    ;Have the floppies been accessed yet ?
flopflg equ    $-1
ana   a
jnz   flopok

mvi   b,17   ;Floppies havn't been accessed
lxi   h,fdboot ;Check if 2D controller is installed
mvi   a,(jmp)
clopp: cmp    m       ;Must have 17 jumps
jnz   zret
inx   h
inx   h

```

```

inx      h
dcr      b
jnz      clopp
lxi      d,fdinit      ;Initialization sequence
lxi      h,fdorig+7e2h ;Load address
lxi      b,30          ;Byte count
call     movbyt        ;Load controller RAM
mvi      a,0ffh        ;Start 1791
sta      dreg
mvi      a,clrcmd      ;1791 reset
sta      cmdreg
mvi      a,1          ;Set 2D initialized flag
sta      flopflg

flopok:  call     flush      ;Flush buffer since we are using it
lda      fdlog         ;Select new drive
mov      c,a
call     fdsl         ;
call     fdlhome      ;Recalibrate the drive
lxi      h,1          ;Select sector 1 of track 2
shld    truesec
inx      h
shld    cpmtrk
xra      a            ;Make sure we are doing a read
sta      rdwr
call     fill         ;Fill in buffer with sector
jc       zret         ;Test for error return
call     fdstat       ;Get status on current drive
sta      fdldst       ;Save drive status
ani      0ch          ;Mask in sector size bits
push    psw          ;Used to select a DPB
rar
lxi      h,xlts       ;Table of XLT addresses
mov      e,a
mvi      d,0
dad      d
push    h            ;Save pointer to proper XLT
call     fdget        ;Get pointer to proper DPH
pop      d
lxi      b,2          ;Copy XLT pointer into DPH
call     movbyt
lxi      d,8          ;Offset to DPB pointer in DPH
dad      d            ;HL <- &DPH.DPB
push    h
call     fdgsid       ;Get pointer to side flag table entry
lda      fdldst       ;Get drive status
ani      dblsid       ;Check double sided bit
mov      m,a          ;Save sides flag
lxi      d,dpbl28s    ;Base for single sided DPB's
jz      sideok
lxi      d,dpbl28d    ;Base of double sided DPB's

sideok:  xchg         ;(HL) -> DPB base, (DE) -> &DPH.DPB
pop      d
pop      psw         ;Offset to correct DPB
ral
ral
;Make 0, 10, 20, 30
mov      c,a
mvi      b,0         ;Make offset
dad      b           ;(hl) is now a DPB pointer
xchg         ;Put proper DPB address in DPH.DPB
mov      m,e
inx      h
mov      m,d
lxi      h,15        ;Offset to DPB.SIZ
dad      d
mov      c,m        ;Fetch sector size code

```



```

fdget:  lda    fdlog      ;Return proper DPH
        lxi    d,dphfd0
        jmp    retdph

fdsel2: sta    fdlog
        mov    c,a
        jmp    fdssel

fdlhome:mvi    c,0        ;Select side 0
        call   fdside
        jmp    fdhome    ;Do actual home

fdssec: push   b          ;Save sector number
        mov   a,b        ;Check side select bit
        rlc                   ;Move high bit to bit zero
        ani   1
        mov   c,a
        call  fdside      ;Call select side 0 = side A, 1 = Side B
        pop   b
        jmp  fdsec

fdgsid: lxi    h,fdlsid   ;Side flag table
        lda    fdlog      ;Drive number
        push   d
        mov   e,a        ;Make offset
        mvi   d,0
        dad   d           ;Offset to proper entry
        pop   d
        mov   a,m        ;Set up flags
        ora   a
        ret

fdinit: dw     0          ;Initialization bytes loaded onto 2D/B
        dw    1800h      ;Head loaded timeout
        dw     0          ;DMA address
        db     0          ;Double sided flag
        db     0          ;Read header flag
        db    07eh      ;Drive select constant
        db     0          ;Drive number
        db     8          ;Current disk
        db     0          ;Head loaded flag
        db     9          ;Drive 0 parameters
        db    0ffh      ;Drive 0 track address
        db     9          ;Drive 1 parameters
        db    0ffh      ;Drive 1 track address
        db     9          ;Drive 2 parameters
        db    0ffh      ;Drive 2 track address
        db     9          ;Drive 3 parameters
        db    0ffh      ;Drive 3 track address
        db     9          ;Current parameters
        db     0          ;Side desired
        db     1          ;Sector desired
        db     0          ;Track desired

        db     0          ;Header image, track
        db     0          ;Sector
        db     0          ;Side
        db     0          ;Sector
        dw     0          ;CRC

fdlog:  db     0
fdldst: db     0        ;Floppy drive status byte

fdlsid: rept   maxfd
        db    0ffh      ;Double sided flag 0 = single, 1 = double
        endm

```

endif

if (maxfd ne 0) or (maxdm ne 0)

```
*****
*
* Xlts is a table of address that point to each of the xlt
* tables for each sector size.
*
*****
```

```
xlts:  dw      xlt128      ;Xlt for 128 byte sectors
      dw      xlt256      ;Xlt for 256 byte sectors
      dw      xlt512      ;Xlt for 512 byte sectors
      dw      xlt124      ;Xlt for 1024 byte sectors
```

```
*****
*
* Xlt tables (sector skew tables) for CP/M 2.0. These tables
* define the sector translation that occurs when mapping CP/M
* sectors to physical sectors on the disk. There is one skew
* table for each of the possible sector sizes. Currently the
* tables are located on track 0 sectors 6 and 8. They are
* loaded into memory in the Cbios ram by the cold boot routine.
*
*****
```

```
xlt128: db      0
      db      1,7,13,19,25
      db      5,11,17,23
      db      3,9,15,21
      db      2,8,14,20,26
      db      6,12,18,24
      db      4,10,16,22
```

```
xlt256: db      0
      db      1,2,19,20,37,38
      db      3,4,21,22,39,40
      db      5,6,23,24,41,42
      db      7,8,25,26,43,44
      db      9,10,27,28,45,46
      db      11,12,29,30,47,48
      db      13,14,31,32,49,50
      db      15,16,33,34,51,52
      db      17,18,35,36
```

```
xlt512: db      0
      db      1,2,3,4,17,18,19,20
      db      33,34,35,36,49,50,51,52
      db      5,6,7,8,21,22,23,24
      db      37,38,39,40,53,54,55,56
      db      9,10,11,12,25,26,27,28
      db      41,42,43,44,57,58,59,60
      db      13,14,15,16,29,30,31,32
      db      45,46,47,48
```

```
xlt124: db      0
      db      1,2,3,4,5,6,7,8
      db      25,26,27,28,29,30,31,32
      db      49,50,51,52,53,54,55,56
      db      9,10,11,12,13,14,15,16
      db      33,34,35,36,37,38,39,40
      db      57,58,59,60,61,62,63,64
      db      17,18,19,20,21,22,23,24
      db      41,42,43,44,45,46,47,48
```

```
*
* Each of the following tables describes a diskette with the
* specified characteristics.
*
```

```
*****
*
* The following DPB defines a diskette for 128 byte sectors,
* single density, and single sided.
*
```

```
dpb128s:dw      26          ;CP/M sectors/track
           db        3          ;BSH
           db        7          ;BLM
           db        0          ;EXM
           dw      242          ;DSM
           dw        63          ;DRM
           db      0c0h         ;AL0
           db        0          ;AL1
           dw       16          ;CKS
           dw        2          ;OFF
           db        1          ;128 byte sectors
```

```
*****
*
* The following DPB defines a diskette for 256 byte sectors,
* double density, and single sided.
*
```

```
dpb256s:dw      52          ;CP/M sectors/track
           db        4          ;BSH
           db       15          ;BLM
           db        1          ;EXM
           dw      242          ;DSM
           dw       127          ;DRM
           db      0c0h         ;AL0
           db        0          ;AL1
           dw       32          ;CKS
           dw        2          ;OFF
           db        2          ;256 byte sectors
```

```
*****
*
* The following DPB defines a diskette as 512 byte sectors,
* double density, and single sided.
*
```

```
dpb512s:dw      60          ;CP/M sectors/track
           db        4          ;BSH
           db       15          ;BLM
           db        0          ;EXM
           dw      280          ;DSM
           dw       127          ;DRM
           db      0c0h         ;AL0
           db        0          ;AL1
           dw       32          ;CKS
           dw        2          ;OFF
           db        3          ;512 byte sectors
```

```
*****
*
```

\* The following DPB defines a diskette as 1024 byte sectors,  
\* double density, and single sided.  
\*  
\*\*\*\*\*

dpl024s:dw 64 ;CP/M sectors/track  
db 4 ;BSH  
db 15 ;BLM  
db 0 ;EXM  
dw 299 ;DSM  
dw 127 ;DRM  
db 0c0h ;AL0  
db 0 ;AL1  
dw 32 ;CKS  
dw 2 ;OFF  
db 4 ;1024 byte sectors

\*\*\*\*\*  
\*  
\* The following DPB defines a diskette for 128 byte sectors,  
\* single density, and double sided.  
\*  
\*\*\*\*\*

dpb128d:dw 52 ;CP/M sectors/track  
db 4 ;BSH  
db 15 ;BLM  
db 1 ;EXM  
dw 242 ;DSM  
dw 127 ;DRM  
db 0c0h ;AL0  
db 0 ;AL1  
dw 32 ;CKS  
dw 2 ;OFF  
db 1 ;128 byte sectors

\*\*\*\*\*  
\*  
\* The following DPB defines a diskette as 256 byte sectors,  
\* double density, and double sided.  
\*  
\*\*\*\*\*

dpb256d:dw 104 ;CP/M sectors/track  
db 4 ;BSH  
db 15 ;BLM  
db 0 ;EXM  
dw 486 ;DSM  
dw 255 ;DRM  
db 0f0h ;AL0  
db 0 ;AL1  
dw 64 ;CKS  
dw 2 ;OFF  
db 2 ;256 byte sectors

\*\*\*\*\*  
\*  
\* The following DPB defines a diskette as 512 byte sectors,  
\* double density, and double sided.  
\*  
\*\*\*\*\*

dpb512d:dw 120 ;CP/M sectors/track  
db 4 ;BSH  
db 15 ;BLM  
db 0 ;EXM

```

dw      561      ;DSM
dw      255      ;DRM
db      0f0h     ;AL0
db      0        ;AL1
dw      64       ;CKS
dw      2        ;OFF
db      3        ;512 byte sectors

```

```

*****
*
* The following DPB defines a diskette as 1024 byte sectors,
* double density, and double sided.
*
*****

```

```

dpl024d:dw      128      ;CP/M sectors/track
db      4          ;BSH
db      15         ;BLM
db      0          ;EXM
dw      599        ;DSM
dw      255        ;DRM
db      0f0h       ;AL0
db      0          ;AL1
dw      64         ;CKS
dw      2          ;OFF
db      4          ;1024 byte sectors

```

```
endif
```

```

*****
*
* The following equates relate the Morrow Designs DJDMA
* controller.
*
*****

```

```

if      (maxdm ne 0) or (maxmf ne 0)
dmchan  equ      50h      ;Default channel address
dmkick  equ      0efh     ;Kick I/O port address

rdsect  equ      20h      ;Read sector command
wrsect  equ      21h      ;Write a sector command
gstat   equ      22h      ;Get drive status
dmsdma  equ      23h      ;Set DMA address
intrqc  equ      24h      ;Set Interrupt request
dmhaltc equ      25h      ;Halt command
bracha  equ      26h      ;Channel branch
setcha  equ      27h      ;Set channel address
setcrc  equ      28h      ;Set CRC retry count
rdtrck  equ      29h      ;Read track command
wrtrck  equ      2ah      ;Write track command
serout  equ      2bh      ;Serial output through bit banger serial port
senabl  equ      2ch      ;Enable serial input
trksiz  equ      2dh      ;Set number of tracks
setlog  equ      2eh      ;Set logical drives
readm   equ      0a0h     ;Read from controller memory
writem  equ      0a1h     ;Write to controller memory

dmfstp  equ      3*341/10 ;Fast stepping rate constant is 3 ms * 34.1
dmfset  equ      15*341/10 ;Fast settling rate constant is 15 ms * 34.1

n$dubl  equ      80h      ;Double density
n$2side equ      40h      ;2 sided drive

serin   equ      03eh     ;Address of serial input data, (status - 1)

```

```

*****
*
* Device Specification Table for the Disk Jockey DMA floppy
*
*****

```

```

dmdst:  if      maxdm ne 0
        db      maxdm      ;Number of logical drives
        dw      dmwarm     ;Warm boot
        dw      dmtran     ;Sector translation
        dw      dmldrv     ;Select drive 1
        dw      dmselr     ;Select drive 2
        dw      dmhome     ;Home drive
        dw      dmseek     ;Seek to specified track
        dw      dmssec     ;Set sector
        dw      dmdma      ;Set DMA address
        dw      dmread     ;Read a sector
        dw      dmwrite    ;Write a sector
        dw      nobad      ;No bad sector map

dmtrck  equ      22*128      ;Amount of code on track 0

dmwarm: call    dmselr      ;Select drive 0
        lxi     h,dmchan    ;Set up branch
        mvi     m,bracha
        inx     h
        mvi     m,(low dmwchn) ;Low address byte
        inx     h
        mvi     m,(high dmwchn) ;High address byte
        inx     h
        mvi     m,0

dmwbad: lxi     h,dmwend-1   ;Pointer to end of command structure
        call    docmd      ;Read in tracks
        lda     dmwst      ;Get track read status
        ani     40h
        jz     dmwbad      ;Loop on 'terrible' errors like no disk
        lxi     b,300h     ;3/4 K bytes of sector 3 needs to be moved
        lxi     d,buffer   ;Sector 3 is in our buffer
        lxi     h,ccp+1300h ; and this is where we want it to go...
        call    movbyt
        xra     a
        ret

dmwchn: db      dmsdma      ;Set track 0 DMA address
        dw      ccp-512     ;First track DMA address - boot loader
        db      0
        db      rdtrck     ;Read track command
        db      0          ;Track 0
        db      0          ;Side 0
        db      0          ;Drive 0
        dw      dmwsec     ;Sector load/status map

dmwst:  db      0          ;Track read status
        db      dmsdma
        dw      ccp+dmtrck ;DMA address for track 1
        db      0
        db      rdtrck
        db      1          ;Track 1
        db      0          ;Side 0
        db      0          ;Drive 0
        dw      dmwsec+26  ;Map is loaded right after track 0 status map
        db      0
        db      0          ;Track read status
        db      dmsdma
        dw      buffer     ;Sector 3 gets loaded in system buffer
        db      0

```

```

db      rdsect
db      1          ;Track 1
db      3          ;Side 0, sector 3
db      0          ;Drive 0
dmwend: db      0          ;Read status
dw      0          ;Room for the halt

dmwsec: dw      0ffffh, 0ffffh      ;Do not load boot loader
dw      0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ;22 sectors to be loaded
dw      0, 0ffffh, 0ffffh, 0ffffh  ;First 2 sectors on track 2

dmselr: sta      dmlog
mvi     b,0          ;8 inch logical drives start at zero
jmp     dmsel2

dmtran: inx      b
push    d          ;Save table address
push    b          ;Save sector #
call    dmget
lxi     d,10
dad     d
mov     a,m
inx     h
mov     h,m
mov     l,a
mov     a,m      ;Get # of CP/M sectors/track
ora     a        ;Clear carry
rar     ;Divide by two
sub     c
push    psw      ;Save adjusted sector
jm      dmside2

dmsidea: pop     psw      ;Discard adjusted sector
pop     b        ;Restore sector requested
pop     d        ;Restor address of xlt table
dmsidel: xchg    ;hl <- &(translation table)
dad     b        ;bc = offset into table
mov     l,m      ;hl <- physical sector
mvi     h,0
ret

dmside2: call    dmstat
ani     20h
jz      dmsidea
pop     psw      ;Retrieve adjusted sector
pop     b
cma     ;Make sector request positive
inr     a
mov     c,a     ;Make new sector the requested sector
pop     d
call    dmsidel
mvi     a,80h   ;Side two bit
ora     h       ;      and sector
mov     h,a
ret

dmldrv: sta      dmlog
call    dmunit  ;Test for a drive
jc      zret
lxi     h,1     ;Select sector 1 of track 2
shld   truesec
inx     h
shld   cpmtrk
xra     a       ;Make sure we are doing a read
sta     rdwr
call    fill    ;Flush buffer and refill
jc      zret    ;Test for error return

```

```

call    dmstat      ;Get status on current drive
ani     0ch         ;Mask in sector size bits
push    psw         ;Used to select a DPB
rar
lxi     h,xlts      ;Table of XLT addresses
mov     e,a
mvi     d,0
dad     d
push    h           ;Save pointer to proper XLT
call    dmget
pop     d
lxi     b,2         ;Number of bytes to move
call    movbyt      ;Move the address of XLT
lxi     d,8         ;Offset to DPB pointer
dad     d           ;HL <- &DPH.DPB
push    h
call    dmstat
ani     20h         ;Check double sided bit
lxi     d,dpbl28s   ;Base for single sided DPB's
jz      dmsok
call    sethigh     ;Set controller to know about fast stepping
lxi     d,dpbl28d   ;Base of double sided DPB's
dmsok: xchg         ;HL <- DBP base, DE <- &DPH.DPB
pop     d           ;Restore DE (pointer into DPH)
pop     psw        ;Offset to correct DPB
ral
ral
mov     c,a
mvi     b,0
dad     b
xchg         ;Put DPB address in DPH
mov     m,e
inx     h
mov     m,d
lxi     h,15
dad     d
mov     c,m
dmget:  lda         dmlog
lxi     d,dphdm0
jmp     retsph

```

```

;
; The current drive is double sided. Thus is it safe to set the
; stepping rate to 3 ms with 15 ms settling.
;

```

```

sethigh:lhld    dmlog      ;Get the current drive number
mvi     h,0         ;Drive number is a byte
dad     h           ;Ten bytes per parameter table entry
mov     d,h
mov     e,1
dad     h
dad     h
dad     d
lxi     d,dparam+1   ;Parameter table address
dad     d           ;Skip the track size byte
mvi     m,0         ;Force reparamitization of this drive
inx     h           ;Offset to the Stepping rate constant
mvi     m,(low dmfstp) ;Fast stepping rate constant
inx     h
mvi     m,(high dmfstp)
lxi     d,5         ;Skip over the reserved fields
dad     d
mvi     m,(low dmfset) ;Fast settling rate constant
inx     h

```



```
mvi m, (high dmfset)
call dmparm ;Set drive parameters for the SA850
ret
```

```
endif
```

```
*****
*
* Drive specification table for DJDMA 5 1/4 inch drives
*
*****
```

```
mfdst:  if      maxmf ne 0
        db      maxmf      ;Number of logical drives
        dw      mfwarm     ;Warm boot
        dw      mftran     ;Sector translation
        dw      mfldrv     ;Select drive 1
        dw      mfsel2     ;Select drive 2
        dw      dmhome     ;Home drive
        dw      mfseek     ;Seek to specified track
        dw      mfssec     ;Set sector
        dw      dmdma      ;Set DMA address
        dw      dmread     ;Read a sector
        dw      dmwrite    ;Write a sector
        dw      nobad     ;No bad sector map

mftrck  equ      9*512     ;Amount of code on track 0

mfwarm: call    mfsel2     ;Select drive 0
        lxi     h, dmchan  ;Set up branch
        mvi     m, bracha
        inx     h
        mvi     m, (low mfwchn) ;Low address byte
        inx     h
        mvi     m, (high mfwchn) ;High address byte
        inx     h
        mvi     m, 0

mfwfal: lxi     h, mfwend-1 ;Pointer to end of command structure
        call    docmd     ;Read in tracks
        lda     mfwst     ;Check out drive status
        ani     40h       ;Test for ok
        jz      mfwfal    ;Failed, loop
        xra     a         ;Return no error
        ret

mfwchn: db      dmsdma     ;Set track 0 DMA address
        dw      ccp-512    ;First track DMA address - boot loader
        db      0
        db      rdtrck    ;Read track command
        db      0         ;Track 0
        db      0         ;Side 0
        db      0         ;Drive 0
        dw      mfwsec     ;Sector load/status map
        db      0

mfwst   db      0         ;Track read status
        db      dmsdma
        dw      ccp+mftck  ;DMA address for track 1
        db      0
        db      rdtrck
        db      1         ;Track 1
        db      0         ;Side 0
        db      0         ;Drive 0
        dw      mfwsec+10  ;Map is loaded right after track 0 status map
        db      0

mfwend: db      0         ;Track read status
        dw      0         ;Room for the halt
```

```

mfwsec: dw      0ffh, 0, 0, 0, 0      ;Do not load boot loader
         dw      0, 0ffffh, 0ffffh, 0ffffh, 0ffffh ;first two sectors loaded

mfssec: dcr     c                      ;Minnie floppy sectors start at zero
         lda     dblflg                 ;Get double sided flags
         ora     a
         jz      dmssec                 ;Nope, single sided
         mvi     b,80h                 ;Set high bit for double sided select
         jmp     dmssec

dblflg: db      0

mfseek: xra     a                      ;Clear double sided select
         sta     dblflg
         lda     mfpcon
         ani     n$2side
         jz      dmseek                 ;Only single sided

         mov     a,c                    ;Move selected track in (a)
         sbi     35                     ;Subtract by track by number of tracks
         jc      dmseek                 ;Less than track 35

         mov     d,a                    ;Save adjusted track number
         mvi     a,34
         sub     d                      ;Adjust to count tracks back out
         mov     c,a                    ;Resave new track number

         mvi     a,0ffh                 ;Set double sided flag
         sta     dblflg
         jmp     dmseek

mfsel2: sta     mflog

         mov     c,a                    ;Get proper physical configuration byte
         mvi     b,0
         lxi     h,mfscon
         dad     b
         mov     a,m
         sta     mfpcon
         mov     a,c                    ;Shhh, pretend that nothing happened

         mvi     b,4                    ;5 1/4 inch drives start at drive 4
         jmp     dmsel2

mftran: lda     mfpcon
         ani     n$dubl
         lxi     h,mfxltd                 ;Point to double sided sector translation table
         jnz     mftdubl                 ;Single density sector translation
         lxi     h,mfxlts

mftdubl: dad     b                      ;Add offset sector number to table
         mov     l,m                    ;Pick up sector number from table
         mvi     h,0                    ;MSB of sector number equal 0
         ret

mfldrv: sta     mflog
         call    dminit                 ;Test for a controller
         jc      zret

         lda     mflog                   ;Get proper physical configuration byte
         mov     c,a
         mvi     b,0
         lxi     h,mfscon
         dad     b
         mvi     a,n$dubl
         mov     m,a

```

```

sta      mfpcon

lxi      h,1          ;Select sector 1 of track 0
shld    truesec
dcx     h
shld    cpmtrk
xra     a            ;Make sure we are doing a read
sta     rdwr
call    fill         ;Flush buffer and refill
jc      zret         ;Test for error return
lda     buffer+5ch   ;Get diskette configuration byte

push    psw          ;Save configuration byte
lxi     h,1
shld    cpmtrk       ;Load track 1 sector 1
call    fill         ;This is to fix bug with DJDMA firmware on
jc      zret         ; returning single density status on track 0
pop     psw

ora     a
jnz     mfl9         ;Non zero

mvi     a,90h        ;Double density default configuration
call    dmstat       ;If zero then determine sector size
ani     80h          ;Check density bit
jnz     mfl9         ;Its double density
mvi     a,10h        ;Single density default configuration byte

mfl9:   mov     c,a   ;Move configuration byte into (c)

mfl2:   lxi     h,mfs   ;Address of configuration table -> (hl)
mov     a,m          ;Get an entry
ora     a            ;Check for end of the table
jz      zret         ;Yes, select error
cmp     c            ;Check if entry matches selected drive
jz      mfl3
inx     h            ;Skip onfiguration byte
inx     h            ;Skip drive type
inx     h            ;Skip DPB address
inx     h
jmp     mfl2

mfl3:   inx     h
mov     a,m          ;Pick up drive type
sta     mfpcon
mov     e,a

push    h
lda     mflog        ;Get proper physical configuration byte
mov     c,a
mvi     b,0
lxi     h,mfscon
dad     b
mov     m,e
pop     h

inx     h
mov     a,m
inx     h
mov     h,m
mov     l,a         ;DPB address -> (hl)
push   h           ;Save DPB address
call   mfgdph      ;Get DPH
lxi     d,10       ;Offset to DPB address in DPH
dad     d
pop     d

```

```

mov     m,e           ;Store DPB address in DPH
inx     h
mov     m,d
call    mfgdph
push   h
call    dmstat       ;Get status
pop     h
ani     80h          ;Check density bit
mvi     c,3          ;512 byte sectors
rnz
mvi     c,2          ;256 byte sectors
ret

mfgdph  lda     mflog
        lxi     d,dphmf0
        jmp    retdph

mfpccon: db     0           ;Physical configuration byte
mflog:  db     0

mfsccon: db     0, 0, 0, 0 ;Saved physical configuration bytes

mfs:    db     10h        ;North Star CP/M 1.4
        db     0          ;Single density, 35 tracks, single sided
        dw     dpbmf0     ;1K groups

        db     90h        ;North Star CP/M 1.4
        db     n$dubl     ;Double density, 35 tracks, single sided
        dw     dpbmf1     ;1K groups

        db     0b0h       ;North Star CP/M 2.x
        db     n$dubl     ;Double density, 35 tracks, single sided
        dw     dpbmf2     ;2K groups

        db     0f0h       ;North Star CP/M 2.x
        db     n$dubl+n$2side ;Double density, 35 tracks, double sided
        dw     dpbmf3     ;2K groups

        db     0e5h       ;North Star CP/M 1.4
        db     n$dubl     ;Double density, 35 tracks, single sided
        dw     dpbmf1     ;1K groups

        db     0a0h       ;North Star CP/M 2.x (fake 40 track)
        db     n$dubl     ;Double density, 35 tracks, single sided
        dw     dpbmf2     ;2K groups

        db     0d0h       ;North Star CP/M 2.x (fake 40 track)
        db     n$dubl+n$2side ;Double density, 35 tracks, double sided
        dw     dpbmf3     ;2K groups

        db     0           ;End of configuration table

mfxltd  db     1, 2, 3, 4
        db     21,22,23,24
        db     5, 6, 7, 8
        db     25,26,27,28
        db     9,10,11,12
        db     29,30,31,32
        db     13,14,15,16
        db     33,34,35,36
        db     17,18,19,20
        db     37,38,39,40

mfxlts  db     1, 2
        db     3, 4
        db     5, 6

```

```

db      7, 8
db      9,10
db      11,12
db      13,14
db      15,16
db      17,18
db      19,20
endif

```

```

*****
*
* Common routines for the DJDMA with 8 and 5 1/4 inch drives
*
*****

```

```

dmsel2: mov     c,a           ;Move drive into (c)
        lxi     h,dmchan
        mvi     m,setlog     ;Set logical drives
        inx     h
        mov     m,b         ;Drive in (b)
        push    b
        call   docmd
        pop     b
        jmp     dmsel

```

```

dmssec: push    b           ;Save sector number
        mov     a,b
        rlc
        ani     1
        mov     c,a
        call   dmsside
        pop     b
        jmp     dmsec

```

```

dmdma  lxi     h,dmchan     ;Default channel address
        mvi     m,dmsdma   ;Set DMA address
        inx     h
        mov     m,c       ;Low byte first
        inx     h
        mov     m,b       ;High byte next

```

```

docmd  xra     a
        inx     h
        mov     m,a

```

```

docmd2 inx     h
        mvi     m,dmhaltc
        inx     h
        mov     m,a
        out     dmkick

```

```

tests  ora     m
        jz     tests
        ret

```

```

dminit: lxi     h,dmchan     ;See if controller will halt
        mvi     m,dmhaltc
        inx     h
        mvi     m,0
        out     dmkick     ;Start controller
        lxi     d,0        ;Set up timeout counter

```

```

dminwt mov     a,m
        ora     a
        jnz    dmiok      ;Controller has responded
        dcx    d          ;Bump timeout counter
        mov     a,d
        ora     e
        jnz    dminwt

```

```

        stc
        ret
        ;Set error flag

dmiok  push    h
        call   dmparm
        pop    h
        dcx   h
        ;Back to start of command
        mvi   m, setcrc
        ;Set CRC error retry count to one
        inx   h
        mvi   m, 1
        xra   a
        jmp   docmd2
        ;Do command

;
;   Set floppy drive parameters
;
;   This routine reads the dparam table and if the a drive has not
;   previously been calibrated then that drives track count,
;   stepping rate, and head settling time are loaded.
;

dmparm: mvi   a, 8
        lxi   d, 1340h
        lxi   h, dparam+1
        ;Eight drives
        ;Start with drive 0's table
        ;Drive parameter table

dmstr0: push   psw
        mov   a, m
        ora   a
        jnz   dmstr1
        ;Save the drive count
        ;Load flags
        ;Does the drive need to be calibrated?
        ;No, do not fiddle around
        push  h
        push  d
        dcr   m
        dcx   h
        shld  dmnrk
        inx   h
        inx   h
        shld  dmnspar
        xchg
        shld  dmloc0
        inx   h
        inx   h
        inx   h
        inx   h
        shld  dmlocl
        lxi   h, dmwcon
        lxi   d, 17
        call  dmdoit
        pop   d
        pop   h
        ;Set the stepping constants pointer
        ;Set the local parameter table pointer
        ;Offset to the stepping parameters

dmstr1: lxi   b, 10
        dad   b
        xchg
        lxi   b, 16
        dad   b
        xchg

        pop   psw
        dcr   a
        jnz   dmstr0
        ;Retrieve drive count
        ;Bump count
        ;Set up next drive

        ret

dmhome xra   a
        mov   c, a
        ;Put a zero into (c) for track zero

```

```

dmseek  mov    a,c          ;Enter with track in (c)
        sta    lltrk       ;Save for use later
        ret

dmsec   lda    llss        ;Load sector
        ani    80h        ;Save side select bit
stores  ora    c
        sta    llss
        ret

dmside: mov    a,c          ;Move side bit into (a)
        ani    1
        rrc          ;Move around to bit 7
        mov    c,a        ;Resave in (c)
        lda    llss
        ani    7fh        ;Mask out old side select bit
        jmp    stores

dmssel: mov    a,c          ;Move drive into (a)
        sta    lldrv
dmdden: ret                ;Double density only

```

```

;
; Return status in the (a) register in the form:
;
;
```

```

;
;          7  6  5  5  3  2  1  0
;          ^  ^  ^  ^  ^  ^  ^  ^
; Density  -----+ | | | | | |
; Side select -----+ | | | | |
; Double sided -----+ | | | | |
; 5 1/4 -----+ | | | | |
; Sector size MSB -----+ | | | |
; Sector size LSB -----+ | | | |
; Drive select MSB -----+ | | |
; Drive select LSB -----+ | |
;
;
```

```

dmstat  lxi    h,dmchan
        mvi    m,gstat    ;Set up read status
        inx    h
        lda    lldrv      ;Get last selected drive
        mov    m,a        ;Store drive in command
        inx    h          ;Skip over returned status
        inx    h
        inx    h
        call   docmd      ;Issue command
        lda    llss      ;Get side bit of last operation
        ani    80h
        rrc          ;Move to bit 7
        mov    c,a
        lxi    h,dmchan+1 ;Point to drive
        mov    a,m        ;Load drive
        ora    c
        ani    4          ;Mask upper drive select bit for 5 1/4
        rlc
        rlc          ;Move to bit 4
        ora    m          ;Put together with lower drive bits
        ora    c
        mov    c,a
        inx    h
        mvi    a,10h     ;Double density bit
        ana    m
        rlc          ;20h
        rlc          ;40h
        rlc          ;80h for density bit
        ora    c

```

```

mov     c,a
inx     h
mvi     a,3           ;Sector length mask
ana     m             ;And in
rlc     ;             ;Move to bits 2 & 3
rlc
ora     c
mov     c,a
inx     h
mvi     a,4           ;Mask for double sided bit
ana     m
rlc     ;8
rlc     ;10
rlc     ;20
ora     c
ret

dmwrite mvi     a,wrsect
db      01           ;Ugh...
dmread  mvi     a,rdsect
lxi     h,dmchan
lxi     d,lltrk-1
mvi     b,4
cload  mov     m,a
inx     h
inx     d
ldax   d
dcr     b
jnz     cload
dcx     h
call    docmd
lda     dmchan+4
cpi     80h
cmc
ret

;
; Execute a DJDMA command, no command status is returned
;
; Entry:
; DE = offset to the halt status
; HL = pointer to the start of the command
;
; Returns:
; nothing
;

dmdoit: mvi     a,bracha           ;Branch channel command
sta     dmchan
shld   dmchan+1           ;Load command vector
xra    a                 ;Clear extended address
sta     dmchan+3

dad     d                 ;Offset to the halt status
mov     m,a              ;Clear the halt status indicator

out     dmkick           ;Start the controller

dmwait: ora     m           ;Wait for the operation complete status
jz     dmwait

ret

dmwcon: db      writem           ;Write track size
dmntrk: dw      0              ;Number of tracks + desync
db      0                    ;X-address

```



```

dmloc0: dw 2 ;Two bytes
          dw 0 ;Local controller address

          db writem ;Write stepping rate data
dmspar: dw 0 ;Pointer to the stepping parameters
          db 0
          dw 8
dmloc1: dw 0

          db dmhaltc ;Controller halt
          db 0 ;Status

;
; Driver variables
;

lltrk db 0
llss db 1
lldrv db 0
dmlog db 0

endif

*****
*
* The following equates are for the HDDMA hard disk controller *
*
*****

if maxmw ne 0 ;HDDMA controller present ?
if st506 ;Specifications for a Seagate Technology 506
cyl equ 153 ;Number of cylinders
heads equ 4 ;Number of heads per cylinder
precomp equ 64 ;Cylinder to start write precompensation
lowcurr equ 128 ;Cylinder to start low current
stepdly equ 30 ;Step delay (0-12.7 milliseconds)
steprcl equ 30 ;Recalibrate step delay
headdly equ 0 ;Settle delay (0-25.5 milliseconds)
endif

if st412 ;Specifications for a Seagate ST412
cyl equ 306
heads equ 4
precomp equ 128
lowcurr equ 128
stepdly equ 0
steprcl equ 30
headdly equ 0
endif

if cm5619 ;Specifications for an CMI 5619
cyl equ 306
heads equ 6
precomp equ 128
lowcurr equ 128
stepdly equ 2
steprcl equ 30
headdly equ 0
endif

sectsiz equ 7 ;Sector size code (must be 7 for this Cbios)
; 0 = 128 byte sectors
; 1 = 256 byte sectors
; 3 = 512 byte sectors
; 7 = 1024 byte sectors (default)
; f = 2048 byte sectors

```

```

;Define controller commands
dmaread equ 0 ;Read sector
dmawrit equ 1 ;Write sector
dmarhed equ 2 ;Find a sector
dmawhed equ 3 ;Write headers (format a track)
dmalcon equ 4 ;Load disk parameters
dmassta equ 5 ;Sense disk drive status
dmanoop equ 6 ;Null controller operation

```

```

reset equ 54h ;Reset controller
attn equ 55h ;Send a controller attention

```

```

chan equ 50h ;Default channel address
stepout equ 10h ;Step direction out
stepin equ 0 ;Step direction in
band1 equ 40h ;No precomp, high current
band2 equ 0c0h ;Precomp, high current
band3 equ 80h ;precomp, low current
track0 equ 1 ;Track zero status
wflt equ 2 ;Write fault from drive
dready equ 4 ;Drive ready
sekcmp equ 8 ;Seek complete

```

```

*****
*
* Drive Specification Table for the HD DMA hard disk controller *
*
*****

```

```

mwdst: db maxmw*mwlog ;Number of logical drives
       dw mwwarm ;Warm boot
       dw mwtran ;Sector translation
       dw mwldrv ;Select logical drive 1 (First time select)
       dw mwdrv ;Select logical drive 2 (General select)
       dw mwhome ;Home current selected drive
       dw mwseek ;Seek to selected track
       dw mwsec ;Select sector
       dw mwdma ;Set DMA address
       dw mwread ;Read a sector
       dw mwwrite ;Write a sector
       if heads > 2 ;Test if drive is big enough for a bad spot map
       dw mwbad ;Return bad sector map info
       else
       dw nobad
       endif

```

```

*****
*
* The following are the lowest level drivers for the Morrow *
* Designs Hard Disk DMA controller. *
*
*****

```

```

mwwarm xra a
       call mwdrv ;Select drive A
       call mwhome ;Home and reset the drive
       lxi b,0 ;Make sure we are on track 0
       call mwseek
       xra a
       sta mwhead ;Select head zero
       sta mwsectr ;Select sector 1
       lxi h,buffer ;Load sector 1 into buffer
       shld dmadma
       call mwwread ;Read CCP into buffer
       rc ;Return if error

```

```

    lxi    d,buffer+200h
    lxi    h,ccp
    lxi    b,200h        ;Move 200h bytes
    call   movbyt
    lxi    h,ccp-200h    ;Initial DMA address
    push  h
    xra   a
    push  a              ;Save first sector -1
mwwlod  pop  psw         ;Restore sector
        pop  h          ;Restore DMA address
    inr   a
    sta   mwsectr
    cpi   6              ;Past BDOS ?
    rz    ;Yes, all done
    inr   h              ;Update DMA address by 1024 bytes
    inr   h
    inr   h
    inr   h
    shld  dmadma
    push  h
    push  psw
    call  mwwread       ;Read in a sector
    jnc   mwwlod
    ret    ;Return with error

mwwread mvi   c,retries ;Retry counter
mwwerr  push  b         ;Save the retry count
        call  mwwread   ;Read the sector
        pop  b
        rnc
        dcr   c         ;Update the error count
        jnz   mwwerr    ;Keep trying if not too many errors
        stc
        ret

mwldrv  sta   mwcurl     ;Save current logical drive
        call  mwreset   ;Reset controller card
        jc    zret      ;Controller failure

        lda   mwcurl
        call  mwdrv     ;Select drive
        jc    zret      ;Select error

        call  mwstat    ;Get drive status
        ani   dready    ;Check if drive ready
        jnz   zret

        call  mwhome    ;Home drive

        lxi   d,dphmw0  ;Start of hard disk DPH's
        lda   mwcurl
        mov   l,a
        mvi   h,0
        dad   h
        dad   h
        dad   h
        dad   h
        dad   d         ;(hl) = pointer to DPH
        mvi   c,4       ;Return sector size of 1024
        ret

mwdrv   sta   mwcurl
        call  mwdlog
        mov   a,c
        sta   mwdrive   ;Save new selected drive
mwsel   mvi   a,dmanoop

```

```

jmp      mwprep      ;Execute disk command

mwdlog: mvi      c,0
mwllx:  sui      mwlog
        rc
        inr      c
        jmp      mwllx

mwstat  mvi      a,dmassta      ;Sense status operation code
        jmp      mwprep      ;Execute disk command

mwhome  call     mwreset      ;Reset controller, do a load constants
        lxi     h,dmargl      ;Load arguments
        mvi     m,steprcl      ;Load step delay (slow rate)
        inx     h
        mvi     m,headdly      ;Head settle delay
        call    mwissue      ;Do load constants again
        call    mwptr      ;Get pointer to current cylinder number
        mvi     m,0ffh      ;Fake at cylinder 65535 for max head travel
        inx     h
        mvi     m,0ffh
        lxi     b,0      ;Seek to cylinder 0
        call    mwseek      ;Recal slowly
        jmp     mwreset      ;Back to fast stepping mode

mwbad:  lxi     h,mwbtabs      ;Return pointer to bad sector location
        ret

mwbtabs: dw      0      ;Track 0
        dw      19      ;Head 2, sector 0 = (2 * SPT + 0) + 1

mwseek  call     mwptr      ;Get track pointer
        mov     e,m      ;Get old track number
        inx     h
        mov     d,m
        dcx     h
        mov     m,c      ;Store new track number
        inx     h
        mov     m,b
        mov     l,c      ;Build cylinder word
        mov     h,b
        shld   dmarg0      ;Set command channel cylinder number
        mov     a,d
        inr     a
        lxi     h,0ffffh
        jnz    mwskip0
        mvi     c,stepout
        jmp     mwskip

mwskip0:mov     h,b      ;(hl) = new track, (de) = old track
        mov     l,c
        call    mwhlmdc
        mvi     c,stepout
        mov     a,h
        ani     80h      ;Check hit bit for negative direction
        jnz    mwsout      ;Step in
        mvi     c,0
        jmp     mwskip

mwsout: call    mwneghi
mwskip: shld   dmastep
        lda     mwdrive
        ora     c
        sta     dmasel0

        mvi     a,dmanoop      ;No-operation command for the channel
        call    mwprep      ;Step to proper track

```

```

    lxi    h,0          ;Clear step counter
    shld  dmastep
    ret

mwdma   mov    h,b      ;Set DMA address
        mov    l,c
        shld  dmadma
        ret

mwsec   mov    a,c      ;Load sector number
        dcr    a        ;Range is actually 0-16
        call  mwdspt    ;Figure out head number -> (c)
        adi   mwspt     ;Make sector number
        sta   mwsectr
        mov   a,c
        sta   mwhead    ;Save head number
        ret

mwdspt  mvi    c,0      ;Clear head counter
mwdsptx sui   mwspt     ;Subtract a tracks worth of sectors
        rc      ;Return if all done
        incr  c        ;Bump to next head
        jmp   mwdsptx

mwreset lhld   chan     ;Save the command channel for a while
        shld  tempb
        lda   chan+2
        sta   tempb+2
        out  reset     ;Send reset pulse to controller
        lxi  h,dmachan ;Address of command channel
        shld chan      ;Default channel address
        xra  a
        sta  chan+2    ;Clear extended address byte
        shld 40h       ;Set up a pointer to the command channel
        sta  42h
        lhld dmarg0    ;Save the track number
        push h
        lxi  h,dmasell ;Load arguments
        lda  mwdrive    ;Get the currently selected drive
        ori  03ch      ;Raise *step and *dir
        mov  m,a        ;Save in drive select register
        lxi  d,5       ;Offset to dmarg1
        dad  d
        mvi  m,stepdly ;Load step delay
        inx  h
        mvi  m,headdly ;Head settle delay
        inx  h
        mvi  m,sectsiz ;Sector size code
        inx  h
        mvi  m,dmalcon ;Load constants command
        call mwissue    ;Do load constants
        pop  h         ;Restore the track number
        shld dmarg0
        push psw       ;Save status
        lhld tempb     ;Restore memory used for the channel pointer
        shld chan
        lda  tempb+2
        sta  chan+2
        pop  psw
        ret

mwread  mvi    a,dmread ;Load disk read command
        jmp   mwprep

mwwrite mvi    a,dmawrit ;Load disk write command

```

```

mwprep: sta      dmaop      ;Save command channel op code

        mvi      c,band1
        lhd      dmarg0
        lxi      d,precomp
        call     mwhlcde
        jc       mwpreps

        mvi      c,band2
        lxi      d,lowcurr
        call     mwhlcde
        jc       mwpreps

mwpreps mvi      c,band3      ;cylinder > low_current
        lda      mwhead      ;Load head address
        sta      dmarg2
        cma
        ani      7           ;Negative logic for the controller
                                ;3 bits of head select
        rlc
        rlc           ;Shove over to bits 2 - 4
        ora      c           ;Add on low current and precomp bits
        mov      c,a
        lda      mwdrive     ;Load drive address
        ora      c           ;Slap in drive bits
        sta      dmasell     ;Save in command channel head select
        lda      mwsectr     ;Load sector address
        sta      dmarg3

mwissue if      0           ;Set to 1 for MW error reporter
        call     mwdoit     ;Do desired operation
        rnc
        push    psw         ;Do nothing if no error
                                ;Save error info
        call     hexout     ;Print status
        call     dspout     ; and a space
        lxi      h,dmachan
        mvi      c,16       ;16 bytes of status
mwerr:  push    b
        push    h
        mov     a,m
        call   hexout     ;Print a byte of the status line
        call   spout
        pop    h
        pop    b
        inx   h           ;Bump command channel pointer
        dcr   c
        jnz   mwerr
        mvi   c,0ah
        call  pout
        mvi   c,0dh
        call  pout
        pop  psw         ;Restore error status
        ret

dspout: call   spout     ;Print two spaces
spout:  mvi   c,' '
        jmp   pout

hexout: push   psw       ;Poor persons number printer
        rrc
        rrc
        rrc
        rrc
        call  nibout
        pop  psw
nibout: ani   0fh
        adi   '0'

```

```

        cpi      '9'+1
        jc      nibok
        adi      27h
nibok:  mov      c,a
        jmp      pout

mwdoit  equ      $

        else

mwissue equ      $          ;Do a disk command, handle timeouts + errors

        endif

        lxi     h,dmastat   ;Clear status byte
        mvi     m,0
        out     attn       ;Start the controller
        lxi     d,0         ;Time out counter (65536 retries)
mwiloop mov      a,m        ;Get status
        ora     a          ;Set up CPU flags
        rm      ;Return no error (carry reset)
        stc     ;Return error status
        rnz     ;Waste some time
        xthl
        xthl
        xthl
        xthl
        dcx     d          ;Bump timeout counter
        mov     a,d
        ora     e
        jnz     mwiloop    ;Loop if still busy
        stc     ;Set error flag
        ret

mwptr   lda     mwdrive    ;Get currently select drives track address
        rlc
        mov     e,a
        mvi     d,0
        lxi     h,mwtab
        dad     d          ;Offset into track table
        ret

mwtran: mov     h,b
        mov     l,c
        inx     h
        ret

mwneghl:mov     a,h
        cma
        mov     h,a
        mov     a,l
        cma
        mov     l,a
        inx     h
        ret

mwhlmdc:chg     mwneghl
        call    mwneghl
        chg
        dad     d
        ret

mwhlcde:mov     a,h
        cmp     d
        rnz
        mov     a,l

```

```

cmp      e
ret

mwtab    equ      $           ;Collection of track addresses
rept    maxmw
db      0ffh           ;Initialize to (way out on the end of the disk)
db      0ffh
endm
db      0ffh

mwcurl   db      0           ;Current logical drive
mwdrive  db      0ffh        ;Currently selected drive
mwhead   db      0           ;Currently selected head
mwsectr  db      0           ;Currently selected sector

dmachan  equ      $           ;Command channel area
dmasel0  db      0           ;Drive select
dmastep  dw      0           ;Relative step counter
dmasell  db      0           ;Head select
dmadma   dw      0           ;DMA address
         db      0           ;Extended address
dmarg0   db      0           ;First argument
dmarg1   db      0           ;Second argument
dmarg2   db      0           ;Third argument
dmarg3   db      0           ;Fourth argument
dmaop    db      0           ;Operation code
dmastat  db      0           ;Controller status byte
dmalnk   dw      dmachan     ;Link address to next command channel
         db      0           ;extended address

```

endif

```

*****
*
* Cbios ram locations that don't need initialization.
*
*****

```

```

         if      nostand ne 0   ;Unallocated writting variables
unalloc: db      0           ;Unallocated write in progress flag
oblock:  dw      0           ;Last unallocated block number written
unadrv:  db      0           ;Drive that the block belongs to
        endif

cpmsec:  dw      0           ;CP/M sector #

cpmdrv:  db      0           ;CP/M drive #
cpmtrk:  dw      0           ;CP/M track #
truesec: dw      0           ;Physical sector that contains CP/M sector

error:   db      0           ;Buffer's error status flag
bufdrv:  db      0           ;Drive that buffer belongs to
buftrk:  dw      0           ;Track that buffer belongs to
bufsec:  dw      0           ;Sector that buffer belongs to

alttrk:  dw      0           ;Alternate track
altsec:  dw      0           ;Alterante sector
lastdrv: db      0           ;Last selected drive

```

```

*****
*
* DPB and DPH area.
*
*****

```

if maxhd ne 0



```

dphdsk set 0 ;Generate DPH's for the HDCA hard disks
rept maxhd
ldsk set 0
rept hdlog
dphgen hd,%dphdsk,dpbhd,%ldsk
ldsk set ldsk+1
dphdsk set dphdsk+1
endm
endm

```

```

if hdpart ne 0 ;Use non-standard partitioning

```

```

*****
*
* hdsectp is the number of 128 byte sectors per cylinder.
*
* hptrks is the total number of data cylinders. Eg. it is
* the number of cylinders on the drive minus the number of
* cylinders that are used for the system. If the number of
* 'system tracks' is not one then the initial value of
* 'off' should be adjusted accordingly.
*
* hptrks = tracks - 1
*
*****

```

```

if m10 ne 0
hdsectp equ 336 ;Sectors per track
hptrks equ 243 ;Total data tracks
endif

```

```

if m20 ne 0
hdsectp equ 672
hptrks equ 243
endif

```

```

if m26 ne 0
hdsectp equ 1024
hptrks equ 201
endif

```

```

ldsk set 0 ;Use non-standard partitioning
tracks set hptrks/hdlog ;Number of tracks per partition
dsm set hdsectp/8*tracks/4-1 ;Number of groups per partition
off set 1

```

```

rept hdlog
dpbgen hd,%ldsk,%hdsectp,5,31,1,%dsm,511,0ffh,0ffh,0,%off,3
off set off+tracks
ldsk set ldsk+1
endm

```

```

else ;Else use standard DPB's

```

```

dphdsk0 if m26 ne 0
dw 1024 ;CP/M sectors/track
db 5 ;BSH
db 31 ;BLM
db 1 ;EXM
dw 2015 ;DSM
dw 511 ;DRM
db 0ffh ;AL0
db 0ffh ;AL1
dw 0 ;CKS
dw 1 ;OFF

```

```

db 3 ;SECSIZ

dpbhd1 dw 1024 ;CP/M sectors/track
db 5 ;BSH
db 31 ;BLM
db 1 ;EXM
dw 2015 ;DSM
dw 511 ;DRM
db 0ffh ;AL0
db 0ffh ;AL1
dw 0 ;CKS
dw 64 ;OFF
db 3 ;SECSIZ

dpbhd2 dw 1024 ;CP/M sectors/track
db 5 ;BSH
db 31 ;BLM
db 1 ;EXM
dw 2047 ;DSM
dw 511 ;DRM
db 0ffh ;AL0
db 0ffh ;AL1
dw 0 ;CKS
dw 127 ;OFF
db 3 ;SECSIZ
endif

if m10 ne 0
dpbhd0 dw 336 ;CP/M sectors/track
db 5 ;BSH
db 31 ;BLM
db 1 ;EXM
dw 1269 ;DSM
dw 511 ;DRM
db 0ffh ;AL0
db 0ffh ;AL1
dw 0 ;CKS
dw 1 ;OFF
db 3 ;SECSIZ

dpbhd1 dw 336 ;CP/M sectors/track
db 5 ;BSH
db 31 ;BLM
db 1 ;EXM
dw 1280 ;DSM
dw 511 ;DRM
db 0ffh ;AL0
db 0ffh ;AL1
dw 0 ;CKS
dw 122 ;OFF
db 3 ;SECSIZ
endif

if m20 ne 0
dpbhd0 dw 672 ;CP/M sectors/track
db 5 ;BSH
db 31 ;BLM
db 1 ;EXM
dw 2036 ;DSM
dw 511 ;DRM
db 0ffh ;AL0
db 0ffh ;AL1
dw 0 ;CKS
dw 1 ;OFF
db 3 ;SECSIZ

```

```

dpbhd1 dw 672 ;CP/M sectors/track
      db 5 ;BSH
      db 31 ;BLM
      db 1 ;EXM
      dw 2036 ;DSM
      dw 511 ;DRM
      db 0ffh ;AL0
      db 0ffh ;AL1
      dw 0 ;CKS
      dw 98 ;OFF
      db 3 ;SECSIZ

dpbhd2 dw 672 ;CP/M sectors/track
      db 5 ;BSH
      db 31 ;BLM
      db 1 ;EXM
      dw 1028 ;DSM
      dw 511 ;DRM
      db 0ffh ;AL0
      db 0ffh ;AL1
      dw 0 ;CKS
      dw 195 ;OFF
      db 3 ;SECSIZ
endif
endif
endif ;End of HD DPH's and DPB's

if maxmf ne 0
dpbgen mf, 0, 20, 3, 7, 0, 04fh, 63, 0c0h, 0, 16, 3, 2
dpbgen mf, 1, 40, 3, 7, 0, 0a4h, 63, 0c0h, 0, 16, 2, 3
dpbgen mf, 2, 40, 4, 15, 1, 051h, 63, 80h, 0, 16, 2, 3
dpbgen mf, 3, 40, 4, 15, 1, 0a9h, 63, 80h, 0, 16, 2, 3

dn set 0
rept maxmf
dphgen mf,%dn,dpbmf,%dn
dn set dn+1
endm
endif

if maxfd ne 0
dn set 0
rept maxfd
dphgen fd,%dn,0,0
dn set dn+1
endm
endif

if maxdm ne 0
dn set 0
rept maxdm
dphgen dm,%dn,0,0
dn set dn+1
endm
endif

if maxmw ne 0

```

```

*****
*
* mwsectp is the number of 128 byte sectors per cylinder.
* mwsectp = 72 * heads
*
* mwtrks is the total number of data cylinders.
* mwtrks = tracks - 1
*

```

\*  
\*\*\*\*\*

```
if st506 ne 0
mwsect equ 288 ;Sectors per track
mwtrks equ 152 ;Total data tracks
endif
```

```
if st412 ne 0
mwsect set 288
mwtrks set 305
endif
```

```
if cm5619 ne 0
mwsect set 432
mwtrks set 305
endif
```

```
dphdsk set 0 ;Generate DPH's for the HDDMA hard disks
rept maxmw
ldsk set 0
rept mwlog
dphgen mw,%dphdsk,dpbmw,%ldsk
dphdsk set dphdsk+1
ldsk set ldsk+1
endm
endm
```

```
if mwpart ne 0 ;Generate DPB's for a HDDMA hard disk
```

```
ldsk set 0 ;Use non-standard partitioning
tracks set mwtrks/mwlog ;Number of tracks per partition
dsm set mwsectp/8*tracks/4-1 ;Number of groups per partition
off set 1
```

```
rept mwlog
dphgen mw,%ldsk,%mwsect,5,31,1,%dsm,1023,0ffh,0ffh,0,%off,4
off set off+tracks
ldsk set ldsk+1
endm
```

```
else ;Use standard partitioning
```

```
off set 1 ;Initial system track offset
trkoff set 8192/(mwsect/8)+1 ;The number of tracks in a partition
blocks set mwsect/8*mwtrks ;The number of blocks on the drive
psize set trkoff*(mwsect/8) ;The number of blocks in a partition
ldsk set 0
```

```
rept blocks/8192 ;Generate some 8 megabyte DPB's
dphgen mw,%ldsk,%mwsect,5,31,1,2047,1023,0ffh,0ffh,0,%off,4
off set off+trkoff
blocks set blocks-psize
ldsk set ldsk+1
endm
```

```
blocks set blocks/4
if blocks gt 256 ;If there is any stuff left, then use it
blocks set blocks-1
dphgen mw,%ldsk,%mwsect,5,31,1,%blocks,1023,0ffh,0ffh,0,%off,4
endif
endif
endif
```

```
buffer equ $
```

\*\*\*\*\*

\* Signon message output during cold boot. \*

\*\*\*\*\*

```
prompt: db      104i, 7EM  
          db      80h, clear          ;Clean buffer and screen  
          db      acr, alf, alf  
          db      'Morrow Designs '  
          db      '0'+msize/10      ;CP/M memory size  
          db      '0'+(msize mod 10)  
          db      'K CP/M '         ;CP/M version number  
          db      cpmrev/10+'0'  
          db      '.'  
          db      (cpmrev mod 10)+'0'  
          db      '.'  
          db      (revnum/10)+'A'-1  
          db      (revnum mod 10)+'0'  
          db      acr, alf
```

```
; Print a message like:  
; AB: DJDMA 8", CD: DJDMA 5 1/4", E: HDDMA M5  
;
```

```
msdrv set 0 ;Start with drive A:
```

```
msbump macro ndrives ;Print a drive name  
if dn gt 1  
db ', '  
endif  
rept ndrives  
db msdrv+'A'  
msdrv set msdrv+1  
endm  
db ': '  
endm
```

```
prhex macro digit ;Write a byte in hex  
prnib digit/10h  
prnib digit  
endm
```

```
prnib macro digit ;Write a digit in hex  
temp set digit and 0fh  
if temp < 10  
db temp + '0'  
else  
db temp - 10 + 'A'  
endif  
endm
```

```
dn set 1 ;Generate the drive messages  
rept 16 ;Run off at least 16 drives  
if dn eq hdorder ;Generate the HDCA's message  
msbump maxhd*hdlog  
db 'HDCA '  
if maxhd gt 1  
db '(', maxhd+'0', ')'  
endif  
if m10 ne 0  
if m10m ne 0  
db 'Memorex'  
else
```

```

db      'Fujitsu'
endif
db      ' M10'
endif
if      m20 ne 0
db      'Fujitsu M20'
endif
if      m26 ne 0
db      'Shugart M26'
endif
endif

if      dn eq mworder          ;Generate the HDDMA's message
msbump maxmw*mwlog
db      'HDDMA'
if      mwquiet eq 0
db      ' '
if      maxmw gt 1
db      '(' , maxmw+'0' , ')'
endif
if      st506 ne 0
db      'M5'
endif
if      st412 ne 0
db      'M10'
endif
if      cm5619 ne 0
db      'M16'
endif
endif
endif

if      dn eq fdorder          ;Generate the 2D/B message
msbump maxfd
db      'DJ2D/B @'
prhex  fdorig/100h
prhex  fdorig
endif

if      dn eq dmorder          ;Generate the DJDMA 8 message
msbump maxdm
db      'DJDMA 8"'
endif

if      dn eq mforder          ;Generate the DJDMA 5 1/4 message
msbump maxmf
db      'DJDMA 5 1/4"'
endif

dn      set      dn+1
endm

db      acr,alf
db      0          ;End of message

```

```

*****
*
* Cboot is the cold boot loader. All of CP/M has been loaded in *
* when control is passed here.
*
*****

```

```

cboot: lxi      sp,tpa          ;Set up stack

xra      a          ;Clear cold boot flag
sta      cwflg

```

```

sta      group      ;Clear group select byte
sta      cpmdrv     ;Select disk A:
sta      cdisk

lxi      h,bios+3   ;Patch cold boot to warm code
shld    bios+1

lda      iobyte     ;Initialize the IOBYTE
sta      iobyte

lxi      d,badmap   ;Clear out bad map
stax    d
lxi      h,badmap+1
lxi      b,9*badsiz ;32 map entries
call    movbyt
mvi      m,0ffh    ;End marker

if      contyp ne 6 ;Non IOBYTE inits
if      contyp ne 0 ;Do not call TTYSET for PROM's
call    ttyset     ;Initialize the terminal
endif

if      lsttyp ne 0 ;Do not call LSTSET for PROM's
call    lstset     ;Initialize the list device
endif
else    ;Do IOBYTE inits
lxi      h,devset  ;Device setup routine pointer table
cboot0: mov     e,m  ;Load a routine address
inx      h
mov      d,m
inx      h
mov      a,d      ;Test for the end of the table
ora      e
jz      cboot2
push    h         ;Save the table pointer
lxi      h,cboot1 ;Return address
push    h
xchg
pchl
cboot1: pop     h  ;'CALL' a device setup routine
jmp     cboot0    ;Restore the table pointer

devset: dw     ttyset, crtset, uclset ;Device setup routine pointers
dw     ptrset, urlset, ur2set
dw     ptpset, uplset, up2set
dw     lptset, ullset, 0

cboot2  equ     $
endif

lxi      h,prompt  ;Prep for sending signon message
call    message   ;Send the prompt
jmp     gocpm

```

```

*****
*
* Console and list device initialization routines follow.
*
*****

```

```

if      contyp eq 2 ;Multi I/O, Decision I

```

```

*****
*
* Terminal initialization routine. This routine reads the sense
* switch on the WB-14 and sets the speed accordingly.
*

```

```

*
*****
ttyset: call    selg0      ;Select group 0
        in      sensesw   ;Get sense switch (ff on a Multio)
        push   psw
        call   selcon    ;Select console
        pop    psw
        push   psw
        call   tini0     ;Initialize the console
        pop    psw
        push   psw
        call   selrdr    ;Select the reader/punch
        pop    psw
        call   tini0     ;Initialize the reader/punch
        ret

tini0:  ani     0e0h      ;Mask in upper three bits
        rlc
        rlc           ;Move into lower 3 bits
        rlc
        cpi     7        ;check for sense = 7 (Default setting)
        jz      dfbaud   ;Use default baud rate

        lxi    h, btab   ;Pointer to baud rate table
        add    a
        mov    e, a      ;Table of words so double
        mvi    d, 0      ;Make a 16 bit number into (de)
        dad    d
        mov    e, m      ;Get a pointer into baud rate table
        inx   h          ;Get lower byte of word
        mov    d, m      ;Bump to high byte of word
        jmp   setit      ;Get upper byte. (de) now has divisor
                          ;Set baud rate

dfbaud: lhld   defcon    ;Use default baud rate
        xchg

setit:  mvi    a, dlab+wls1+wls0+stb ;Enable divisor access latch
        out   lcr        ;Set the baud rate in (de)
        mov   a, d
        out   dlm        ;Set upper divisor
        mov   a, e
        out   dil        ;Set lower divisor

        mvi   a, wls1+wls0+stb ;Clear Divisor latch
        out   lcr
        xra   a
        out   ier        ;Set no interrupts
        out   lsr        ;Clear status
        mvi   a, dtrenb+rtsenb ;Enable DTR and RTS outputs to terminal
        out   mcr
        in   msr         ;Clear MODEM Status Register
        in   lsr         ;Clear Line Status Register
        in   rbr         ;Clear reciever buffers
        in   rbr
        ret

btab:   dw     1047      ;110 Baud      000
        dw     384      ;300           001
        dw     96       ;1200          010
        dw     48       ;2400          011
        dw     24       ;4800          100
        dw     12       ;9600          101
        dw     6        ;19200         110
        dw           ;DEFCON          111

```



```

endif                                     ;Multi I/O, Decision I

if      contyp eq 3                       ;2D/B console initialization
ttyset: call   fdtstat                    ;Clean input buffer
        rnz    ;All empty
        call   fdcin
        jmp    ttyset

endif                                     ;2D/B console

if      contyp eq 4
ttyset: call   dminit                     ;See if controller present
        rc     ;No controller, return
        lxi   d,dmaci                    ;Console initialization sequence
        lxi   h,dmchan
        lxi   b,10                        ;Command length
        call  movbyt
        dcx   h
        xra   a                           ;Clear serial input status
        sta   serin+1
        jmp   docmd2                      ;Do stuff and return

dmaci:  db    writem                       ;Zot monitor disable flag
        dw    ttyset                      ;Any non-zero byte will do
        db    0
        dw    1                           ;One byte
        dw    13f5h                       ;Magical place in monitor
        db    senabl                       ;Enable serial input
        db    1

endif

*****
*
* Initialize the North Star Mother board, left serial port, right
* serial port, and North Star RAM parity.
*
*****

if      contyp eq 6                       ;North Star drivers
ttyset:                                     ;Set up the parallel port + motherboard
        xra   a                           ;Initialize mother board
        out   6
        out   6
        out   6
        out   6

        mvi   a,30h                       ;Reset the parallel port input flag
        out   nspsta
        mvi   a,60h                       ;Set the parallel port output flag
        out   nspsta
        mvi   a,acr                       ;Force a CR out the parallel port
        call  nspout

        mvi   a,nslin1                     ;Initialize the left serial port
        out   nslsta                       ;See the equates for bit definations
        mvi   a,nslin2
        out   nslsta
        xra   a                           ;Clear the input/output buffers
        out   nsldat
        in    nsldat
        in    nsldat

```

```

;Initialize the right serial port
;See the equates for bit definations
mvi    a,nsrin1
out    nsrsta
mvi    a,nsrin2
out    nsrsta
xra    a                ;Clear the input/output buffers
out    nsrdat
in     nsrdat
in     nsrdat

if     nsram ne 0       ;Reset parity on North Star RAMs
mvi    a,40h           ;Disable parity logic
out    nsram
lxi    h,0             ;Starting address

nset0: mov    a,m        ;Get a byte
      mov    m,a        ;Rewrite, set proper parity
      inr   l           ;Bump the address pointer
      jnz   nset0

nset1: inr   h           ;Skip to the next memory page
      jz    nset2       ;Skip if all done
      mvi   a,(high $) + 1 ;Is the pointer above us?
      cmp   h           ;Set carry if pointer is <= our page+1
      jc   nset0       ;Reset the next pages parity
      mov   a,m         ;Test for a PROM or no memory
      mov   b,a         ;Save the original byte
      cma                ;See if this location will change
      mov   m,a
      cmp   m           ;Test for a change
      mov   m,b         ;Restore the original value
      jz    nset0       ;Value complemented, must be RAM
      ora   a           ;Test for no memory present
      jz    nset1       ;Skip to the next page if no memory
      lxi   d,700h      ;Skip 2K bytes of 'PROM'
      dad   d
      jnc   nset1       ;Do a page check if no overflow

nset2: mvi    a,41h      ;Re-enable parity on the memory boards
      out    nsram
      endif

crtset:                ;Null routines
ptrset:
ptpset:
uclset:
urlset:
ur2set:
uplset:
up2set:
lptset:
ullset:

      ret
      endif                ;North Star drivers

if     (lsttyp ge 2) and (lsttyp le 5) ;Serial Multi I/O list drivers

lstset: call   sellst    ;Select printer group
      mvi   a,dlab      ;Access divisor latch
      out   lcr
      lhld deflst      ;Get LST: baud rate divisor
      mov   a,h
      out   dlm         ;Set upper baud rate
      mov   a,l
      out   dll
      mvi   a,stb+wls0+wls1 ;2 stop bits + 8 bit word

```

```

    out    lcr
    mvi    a,dtrenb+rtsenb      ;DTR + RTS enabled
    out    mcr
    in     rbr                  ;Clear input buffer
    xra    a
    out    ier                  ;No interrupts
    ret

endif

    db     0,0ffh,0

codelen equ    ($-bios)        ;Length of Cbios code

    if '   codelen gt 1000h      ;Test for SYSGEN problems
'FATAL ERROR, system is too big for SYSGEN rev. 4.X'
dbgtmp set    codelen          ;Cbios code length ! <debug>
endif

    if     debug
dbgtmp set    codelen          ;Cbios code length ! <debug>
endif

    ds     512-($-buffer)      ;Buffer for 512 byte sectors

    if     (maxfd ne 0) or (maxdm ne 0) or (maxmw ne 0)
    ds     512                  ;Additional space for lk sector devices
    endif

*****
*
* Each bad map entry consists of 9 bytes:
*   Logical drive number (1 byte)
*   Track number of bad sector (2 bytes)
*   Sector number of bad sector (2 bytes)
*   Track number of alternate sector (2 bytes)
*   Sector number of alternate sector (2 bytes)
*
*****

badmap: ds     badsiz*9+1      ;32 entries + end marker

dirbuf: ds     128             ;Directory buffer

tempb:  ds     16              ;A little temporary buffer

*****
*
* Allocation and checked directory table area
*
*****

    if     maxhd ne 0
    if     hdpart ne 0          ;Use non-standard partitioning

    tracks set    hptrks/hdlog   ;Number of tracks per partition
    dsm    set    hdsectp/8*tracks/4-1 ;Number of groups per partition
    alv    set    (dsm/8)+1

    dn     set     0
    rept   maxhd*hdlog          ;Generate CKS and ALV tables
    alloc  hd,%dn,%alv,0

    dn     set     dn+1
    endm

    else
;Standard partitioning

```

```

dn      set      0
      rept     maxhd
      if       m26 ne 0
dn      alloc    hd,%dn,252,0
      set      dn+1
dn      alloc    hd,%dn,252,0
      set      dn+1
dn      alloc    hd,%dn,256,0
      set      dn+1
      endif

      if       m10 ne 0
dn      alloc    hd,%dn,159,0
      set      dn+1
dn      alloc    hd,%dn,161,0
      set      dn+1
      endif

      if       m20 ne 0
dn      alloc    hd,%dn,255,0
      set      dn+1
dn      alloc    hd,%dn,255,0
      set      dn+1
dn      alloc    hd,%dn,129,0
      set      dn+1
      endif
endm
endif

endif

dn      if       maxfd ne 0
      set      0

      rept     maxfd
dn      alloc    fd,%dn,75,64
      set      dn+1
      endm
      endif

dn      if       maxdm ne 0
      set      0

      rept     maxdm
dn      alloc    dm,%dn,75,64
      set      dn+1
      endm
      endif

dn      if       maxmf ne 0
      set      0
      rept     maxmf
dn      alloc    mf,%dn,22,16
      set      dn+1
      endm
      endif

      if       maxmw ne 0
      if       mwpart ne 0                ;Use non-standard partitioning

tracks set      mwtrks/mwlog            ;Number of tracks per partition
dsm    set      mwsectp/8*tracks/4-1    ;Number of groups per partition
alv    set      (dsm/8)+1
dn     set      0

```

```

rept maxmw*mwlog ;Generate CKS and ALV tables
alloc mw,%dn,%alv,0
dn set dn+1
endm

else ;Use standard partitioning

dn set 0
trkoff set 8192/(mwsecpt/8)+1
psize set trkoff*(mwsecpt/8)

rept maxmw

blocks set mwsecpt/8*mwtrks

rept blocks/8192 ;Generate some 8 megabyte ALV's
alloc mw,%dn,256,0
blocks set blocks-psize
dn set dn+1
endm

blocks set blocks/4

if blocks gt 256 ;Use the remainder
blocks set blocks-1
alv set (blocks/8)+1
alloc mw,%dn,%alv,0
dn set dn+1
endif
endm

endif
endif

bioslen equ (high ($-bios))+1 ;BIOS length in pages

if bioslen gt biosln ;Test for overflow
'FATAL ERROR, system overflow. BIOSLN must be at least'
dbgtmp set bioslen ;BIOSLN! <debug>
endif

if debug
dbgtmp set biosln ;Current BIOSLN! <debug>
if biosln gt bioslen
dbgtmp set bioslen ;Optimal BIOSLN! <debug>
endif
endif

end

```