CDBL

Combo Disk Boot Loader

User's Guide

Version 3.00
Martin Eberhard
Mike Douglas
16 January 2016

Revision History

Revision	Date	Author	Notes
1.00	1 May 2014	M. Eberhard	Created from DBLe and MDBL, to produce exactly either of these loaders, selected by an assembly option.
2.00	7 May 2014	M. Eberhard	General code rewrite. Automatic detection of disk drive type. Select disk unit via sense switches. Improve restore. Add ‘O’ error. Verify after code copy.
2.01	15 May 2014	M. Eberhard	Step in once before seeking track 0, in case the head is out past track 0. Restart 6.4 Sec motor shutoff timer on retries.
2.02	4 June 2014	M. Eberhard	Eliminate booting from other than drive 0, because most Altair software just loads a 2-sector loader, and that loader loads the rest of the code (always from drive 0).
2.03	17 Jan 2015	M. Douglas	Add 40 mS delay prior to changing seek direction, to meet Pertec FD400 spec
2.04	12 Mar 2015	M. Douglas	Fix 88-2SIO initialization value
2.05	11 Jan 2016	M. Eberhard	No IN or OUT instructions until the code is moved into RAM, for compatibility with some Turnkey modules. Also fix bug when reporting an overlay error.
3.00	16 Jan 2016	M. Douglas	Make PROM position independent

CONTENTS

1. INTRODUCTION..... 3
2. INSTALLATION..... 4
3. MEMORY REQUIREMENTS..... 4
4. SENSE SWITCH SETTINGS..... 4
5. OPERATING PROCEDURES..... 5
6. ERROR INDICATIONS..... 5
APPENDEX A - ALTAIR DISK BOOT-TRACK FORMAT..... 7
APPENDIX B - SOURCE CODE LISTING..... 8

1. INTRODUCTION

CDBL is a boot loader PROM program for booting from either the Altair 88-DCDD 8″ floppy disk system or the Altair 88-MDS Minidisk system.

CDBL works exactly the same as the Altair DBL boot PROM (for the 88-DCDD 8″ disk controller) and the Altair MDBL boot PROM (for the 88-MDS Minidisk controller), with the following improvements:

1. Automatic Disk Drive Type Detection

   Before booting, CDBL determines whether the disk is an 8″ disk or a Minidisk by identifying the sector that follows sector 15. Sector 16 means it is an 8″ disk (which has 32 sectors per track), while sector 0 means it a Minidisk (which has only 16 sectors per track).

2. RAM Usage

   Like DBL and MDBL, CDBL requires some zero wait-state RAM for fast code, a buffer, and the stack. DBL uses about 400 bytes starting at address 026000 octal (which limits the loaded program to no more than 11K bytes). MDBL uses about 400 bytes starting at 046000 octal (which limits the loaded program to no more than 19K bytes). CDBL uses 512 bytes starting at address 046000 octal, which limits the loaded program to 19K bytes.

3. Position Independence

   CDBL version 3.00 and later will run from any PROM socket at a 256-byte page boundary, except page 0.

4. Memory Overlay Detection

   CDBL checks to see if the disk data will overwrite CDBL’s own RAM (at 046000 octal), and will abort with an overlay error if so.

5. Track 0 Overshoot Correction

   CDBL steps inward once, before seeking track 0. This will allow booting even when the track 0 end-stop is maladjusted. (On a maladjusted drive, it is possible to seek outward past track 0.)

6. Restart Shutoff Timer on Retries

   The Altair Minidisk controller has a timer that gets restarted whenever a Step command is issued, or when an explicit Timer Reset command is issued. 6.4 seconds later, this timer shuts off the disk controller, unless it gets restarted first.

   MDBL never explicitly restarts this timer, relying on the Step commands to restart the timer at the beginning of each track. However, each sector-read retry will add 200 mS to the load time of the current track. This means that 32 retries on one track will cause the MDBL load to fail because the controller will shut off.

Unlike MDBL, CDBL restarts this timer whenever it retries a sector read, so that it can complete booting even from a disk that requires many retries.

7. Correct 88-2SIO Initialization

   The 88-2SIO initialization bug in the Altair DBL PROM has been corrected. This means that error codes will be printed correctly on an 88-2SIO Terminal. (This bug does not exist in the MDBL PROM.)

8. 88-2SIO Initialization Reset Delay

   CDBL delays before resetting the 88-2SIO long enough that the transmission of any character in its output buffer (e.g. the 'B' command from UBMON) will complete.

2. INSTALLATION

Like the DBL and MDBL PROMs, the CDBL PROM is normally installed at address 177400 octal. It belongs in slot H on an 88-PMC memory card (which is jumpered to start at address 174000 octal), or in slot H1 on a Turnkey Module.

However, CDBL (version 3.00 or later) will run from any PROM socket. Note that UBMON expects it to be at address 174000 octal. Examples in this manual assume 174000 octal - substitute the start address of your PROM socket if your CDBL is installed elsewhere.

3. MEMORY REQUIREMENTS

CDBL requires two 256-byte pages of zero wait-state RAM starting at address 046000 octal. The code that is loaded from the disk gets written to RAM, starting at address 0. For practical purposes, this means that the Altair must have at least 20K bytes of RAM starting at address 0, and at least the last 4K block of this RAM must be zero wait-state RAM. (This zero wait-state RAM may be dynamic RAM, as the timing has sufficient margin to handle the occasional refresh cycle.)

Because of the location of CDBL's RAM page, the boot file that is loaded from the disk cannot be larger than 19K bytes in size. (This is the same boot file size limitation as MDBL. DBL limits boot files to 11K bytes.)

4. SENSE SWITCH SETTINGS

Loaded software, such as Altair Disk BASIC, will normally read the four sense switches <A15:A12> to determine the Terminal device. See the manual for the software that you are loading.

5. OPERATING PROCEDURES

5.1 ACCESSING CDBL

If you do not have TURMON or UBMON, then use the front panel to STOP and RESET the Altair, and then EXAMINE address 177400 octal.

5.2 SETTING THE SENSE SWITCHES

1. Set Sense Switches <A15:A12> according to the manual for the program you are loading (e.g. the Altair BASIC Reference manual, Appendix B). For reference, here are the standard Altair Terminal Device Sense Switch settings for many Altair programs:

Terminal Device	A15	A14	A13	A12
88-2SIO Port 0 (2 stop bits)	0	0	0	0
88-2SIO Port 0 (2 stop bits)	0	0	0	1
88-SIO	0	0	1	0
4PIO	0	1	0	0
PIO	0	1	0	0

5.3 INITIATING THE BOOT

1. Insert the boot disk into the disk drive. If you are using a Minidisk, check that the disk has a write-protect tab installed.
2. Start CDBL
    a. If you are using TURMON, type “J177400” on the Terminal.
    b. If you are using UBMON, type “B” on the Terminal.
    c. If you are starting with a front panel, press the RUN switch.

5.4 BOOTING

CDBL will read the Altair disk boot file one sector at a time, starting with track 0, sector 0, and write the file into RAM, starting at address 0. Within each track, the sectors are interleaved 2:1 - the even sectors are loaded first, followed by the odd sectors.

Loading will continue until either an irrecoverable error is encountered, or until the complete boot file has been loaded into RAM. If the file load completes, then CDBL will jump to address 0 to execute the loaded code. If an irrecoverable error occurs, then CDBL will report that error and hang - see the next section.

Note that many programs (including later versions of Altair Disk Basic) load just a few sectors from disk, and then jump to that little loader program to load the rest of the software.

6. ERROR INDICATIONS

The front panel's Interrupt Enable light remains off while loading is proceeding properly. If CDBL encounters an irrecoverable error, it will turn on the Interrupt Enable light, store the ASCII error code in memory location 0, and store the 16-bit offending address in memory locations 1 and 2. CDBL will then send the error code to all standard Altair Terminal devices continuously until you STOP or RESET the Altair from the front panel.

The CDBL error codes are:

C - Checksum error-------A computed checksum and the checksum on the disk did not match, or a Marker Byte was not 377 octal. CDBL will retry any sector that has either of these errors 15 times before giving up and indicating a Checksum error.

M - Memory error---------Defective memory, read-only memory, or protected memory was encountered when attempting to write to RAM; the address of the offending RAM location is stored in memory locations 1 and 2.

O - Overlay error--------An attempt was made to load disk data beyond the first 19K bytes of memory, which would overlay the memory page that contains CDBL's stack, buffer, and disk routines.

APPENDEX A - ALTAIR DISK BOOT-TRACK FORMAT

Altair disks are hard-sectored disks. 8” disks (for the 88-DCDD) have 32 sectors per track, while Minidisks (for the 88-MDS system) have 16 sectors per track. Sectors are numbered sequentially on each track, starting at sector 0. Sectors each contain 128 bytes of actual data.

Tracks are numbered sequentially from track 0, starting at the outside disk diameter. 8” disks have 77 tracks, and Minidisks have 35 tracks. Loading begins with track 0, and continues sequentially through the tracks until the required amount of data (which is specified in the File Byte Count portion of each sector’s header) has been read. The first several tracks are reserved for the boot code, and usually have a slightly different format than the rest of the disk.

For at least the boot portion of an Altair disk, sectors are interleaved with a 2:1 pattern: on each track, the even sectors (starting with sector 0) are all loaded first, followed by the odd sectors (starting with sector 1). (CDBL requires less than one sector time to process a sector’s data once it has been read into the buffer. So with 2:1 interleave, each track can be read with just two disk revolutions, once the track step is complete and sector 0 has been found.)

Each sector contains 137 bytes. On the boot tracks, these sectors are formatted as follows:

Byte 0	Track Number, with MSB set (the sync bit)
Bytes 1-2	File Byte Count (16 bits)
Bytes 3-130	Sector Data (128 bytes)
Byte 131	Marker Byte, must be 377 octal
Byte 132	Sector Data Checksum (8-bit sum of all data bytes)
Bytes 133-136	Spare bytes (not read by CDBL)

APPENDIX B - SOURCE CODE LISTING

The following pages list the source code for CDBL. This code was assembled using Digital Research’s ASM assembler. As such, all values are in hexadecimal, rather than in octal as is normal for MITS software.

```
;=================================================================
;=            CDBL - Combo Disk Boot Loader ROM                  =
;=        For the Altair 88-DCDD 8" disk system and              =
;=            the Altair 88-MDS Minidisk system                  =
;=                  By Martin Eberhard                           =
;=                                                               =
;= CDBL loads software (e.g. Altair Disk BASIC) from an         =
;= Altair 88-DCDD 8" disk or an 88-MDS 5-1/4" minidisk,         =
;= automatically detecting which kind of drive is attached.     =
;=================================================================
;=                            NOTES                              =
;=                                                               =
;= Like DBL and MDBL, CDBL fits in one (256-byte) 1702A         =
;= EPROM and execution begins at address FF00h. However,        =
;= since version 3.00, the PROM is position independent         =
;= and can run at most any 256 byte boundary.                   =
;=                                                               =
;= Because of the slow 1702A EPROM access time, and because     =
;= some versions of MITS's 8800b Turnkey Module disable PROM    =
;= when any IN instruction is executed, CDBL copies itself      =
;= into RAM at 4C00h (RAMADR), and executes there. CDBL         =
;= therefore requres 512 bytes of 0-wait state RAM, starting    =
;= at RAMADR.                                                   =
;=                                                               =
;= Minidisks have 16 sectors/track, numbered 0 through 15.      =
;= 8" disks have 32 sectors/track, numbered 0 through 31.       =
;= CDBL figures out which kind of disk drive is attached,       =
;= based on the existance of sector number 16.                  =
;=                                                               =
;=       ALTAIR DISK SECTOR FORMAT (FOR BOOT SECTORS)           =
;=                                                               =
;=   BYTE(s)     FUNCTION                  BUFFER ADDRESS       =
;=     0       Track number+80h (sync)     RAMADR+7Bh          =
;=     1       File size low byte          RAMADR+7Ch          =
;=     2       File size High Byte         RAMADR+7Dh          =
;=   3-130     Sector data          RAMADR+7Eh to RAMADR+FDh    =
;=    131      Marker Byte (0FFh)          RAMADR+FEh          =
;=    132      Checksum                    RAMADR+FFh          =
;=    133-136  Spare                        not read           =
;=                                                               =
;= Each sector header contains a 16-bit file-size value:        =
;= this many bytes (rounded up to an exact sector) are read     =
;= from the disk and written to RAM, starting at address 0.     =
;= When done (assuming no errors), CDBL then jumps to           =
;= address 0 (DMAADR) to execute the loaded code.               =
;=                                                               =
;= Sectors are interleaved 2:1. CDBL reads the even sectors     =
;= on each track first (starting with track 0, sector 0)        =
;= followed by the odd sectors (starting with sector 1),        =
;= continuing through the interleaved sectors of each track     =
;= until the specified number of bytes have been read.          =
;=                                                               =
;= CDBL first reads each sector (including the actual data      =
;= payload, as well as the 3 header and the first 2 trailer     =
;= bytes) from disk into the RAM buffer (SECBUF). Next, CDBL    =
;= checks to see if this sector would overwrite the RAM         =
;= portion of CDBL, and aborts with an 'O' error if so. It      =
;= then copies the data payload portion from the buffer to      =
;= its final RAM location, calculating the checksum along the   =
;= way. During the copy, each byte is read back, to verify      =
;= correct writes. Any write-verify failure will immediately    =
```

```
                ;= abort the load with an 'M' error.                          =
                ;=                                                            =
                ;= Any disk read error (a checksum error or an incorrect      =
                ;= marker byte) will cause a retry of that sector read. After =
                ;= 16 retries on the same sector, CDBL will abort the load    =
                ;= with a 'C' error.                                          =
                ;=                                                            =
                ;= If the load aborts with any error, then CDBL will turn on  =
                ;= the INTE LED on the front panel (as an indicator), write   =
                ;= the error code ('C', 'M', or 'O') to RAM address 0, write  =
                ;= the offending memory address to RAM addresses 1 and 2, and =
                ;= then hang forever in a loop, printing the error code to    =
                ;= all known Altair Terminal output ports.                    =
                ;================================================================
                ;=                       REVISION HISTORY                       =
                ;=                                                              =
                ;=  1.00  05May2014  M.Eberhard                                =
                ;=    Combined MDBL and DBLme code, with assembly options to    =
                ;=    create exactly both of these boot loaders                 =
                ;=  2.00  08May2014  M.Eberhard                                =
                ;=    Restructure and squeeze the code. Automatic 8"/Minidisk   =
                ;=    detection. Select boot disk from the sense switches.      =
                ;=    Improve track 0 seek by waiting for -MVHEAD before        =
                ;=    testing TRACK0. Detect memory overrun errors. Verify      =
                ;=    copy of CDBL code into RAM.                               =
                ;=  2.01  15May2014  M.Eberhard                                =
                ;=    Step in once before seeking track 0. Restart 6.4 Sec      =
                ;=    motor shutoff timer on retries.                           =
                ;=  2.02  04Jun2014  M. Eberhard                               =
                ;=    Eliminate booting from other than drive 0 because Basic   =
                ;=    and Burcon CP/M just load a 2-sector boot loader, and     =
                ;=    that boot loader loads the rest, always from drive 0.     =
                ;=  2.03  17Jan2015  M. Douglas                                =
                ;=    Force 43ms minimum delay when changing seek direction     =
                ;=    to meet/exceed 8" drive requirements.                     =
                ;=  2.04  12Mar2015  M. Douglas                                =
                ;=    Change 2SIO init constant (ACINIT) from 21h (7E2, xmit    =
                ;=    interrupts on) to 11h (8N2)                               =
                ;=  2.05  11Jan2016  M. Eberhard                               =
                ;=    No IN or OUT instructions until code is moved to RAM      =
                ;=    (for compatibility with some Turnkey Modules). This       =
                ;=    increased the RAM footprint from 256 to 512 bytes. Also   =
                ;=    fixed a bug when reporting overlay errors.                =
                ;=  3.00 12Jan2016  M. Douglas                                 =
                ;=    Make the PROM position independent by making the          =
                ;=    routine that copies PROM to RAM position independent.     =
                ;================================================================

                ;-----------------
                ; Disk Parameters
                ;-----------------
0080 =          BPS     equ     128             ;data bytes/sector
0010 =          MDSPT   equ     16              ;Minidisk sectors/track
                                                ;this code assumes SPT for 8"
                                                ;disks = MDSPT * 2.

0003 =          HDRSIZ  equ     3               ;header bytes before data
0002 =          TLRSIZ  equ     2               ;trailer bytes read after data

0085 =          SECSIZ  equ     BPS+HDRSIZ+TLRSIZ ;total bytes/sector

0010 =          RETRYS  equ     16              ;max retries per sector
```

```
                ;-------------------------------------------------------------
                ; Memory Parameters. To keep code short, several assumptions
                ; about these values are embedded in the code:
                ;  1) RAMADR and PROM low address byte = 0
                ;  2) The address of the last byte of SECBUF (the SECSIZ-sized
                ;     sector buffer) must be XXFF.
                ;  3) The ls bit of the high byte of RAMADR must be 0
                ;  4) The value of DMAADR is assumed to be 0
                ;-------------------------------------------------------------
4C00 =          RAMADR  equ     4C00H           ;Address for code copied to RAM
4D7B =          SECBUF  equ     RAMADR+512-SECSIZ
4D7B =          STACK   equ     SECBUF          ;grows down from here
0000 =          DMAADR  equ     0               ;Disk load/execution address

                ;-----------------------------------------
                ; Addresses of sector components within SECBUF
                ;-----------------------------------------
4D7C =          SFSIZE  equ     SECBUF+1        ;address of file size
4D7E =          SDATA   equ     SECBUF+HDRSIZ   ;address of sector data
4DFE =          SMARKR  equ     SDATA+BPS       ;address of marker byte
4DFF =          SCKSUM  equ     SMARKR+1        ;address of checksum byte

                ;----------------
                ; 88-SIO Equates
                ;----------------
0000 =          SIOCTL  EQU     00              ;Control
0000 =          SIOSTA  EQU     00              ;Status
0001 =          SIODAT  EQU     01              ;Rx/Tx Data

                ;-----------------------------------------------
                ; 88-2SIO's port 0, Turnkey Module, and 88-UIO
                ; Equates (all based on the Motorola 6850 ACIA)
                ;-----------------------------------------------
0010 =          ACCTRL  equ     10h             ;ACIA Control output port
0010 =          ACSTAT  equ     10h             ;ACIA Status input port
0011 =          ACDATA  equ     11h             ;ACIA Tx/Rx Data register

0003 =          ACRST   equ     03h             ;Master reset
0011 =          ACINIT  equ     11h             ;/16, 8bit, No Parity, 2Stops

                ;----------------
                ; 88-PIO Equates
                ;----------------
0004 =          PIOCTL  EQU     04              ;Control
0004 =          PIOSTA  EQU     04              ;Status
0005 =          PIODAT  EQU     05              ;Tx/Rx Data

                ;-----------------
                ; 88-4PIO equates
                ;-----------------
0020 =          P4CA0   equ     20h             ;Port 0 Section A Ctrl/Status
0021 =          P4DA0   equ     21h             ;Port 0 Section A Data
0022 =          P4CB0   equ     22h             ;Port 0 Section B Ctrl/Status
0023 =          P4DB0   equ     23h             ;Port 0 Section B Data

002C =          P4CINI  equ     2Ch             ;Control/status initialization

                ;-------------------------------------------------------
                ; Altair 8800 Disk Controller Equates (These are the same
                ; for the 88-DCDD controller and the 88-MDS controller.)
                ;-------------------------------------------------------
0008 =          DENABL  equ     08H             ;Drive Enable output
0080 =          DDISBL    equ     80h             ;disable disk controller
```

```
0008 =          DSTAT   equ     08H             ;Status input (active low)
0001 =          ENWDAT    equ     01h             ;-Enter Write Data
0002 =          MVHEAD    equ     02h             ;-Move Head OK
0004 =          HDSTAT    equ     04h             ;-Head Status
0008 =          DRVRDY    equ     08h             ;-Drive Ready
0020 =          INTSTA    equ     20h             ;-Interrupts Enabled
0040 =          TRACK0    equ     40h             ;-Track 0 detected
0080 =          NRDA      equ     80h             ;-New Read Data Available

0009 =          DCTRL   equ     09h             ;Drive Control output
0001 =          STEPIN    equ     01H             ;Step-In
0002 =          STPOUT    equ     02H             ;Step-Out
0004 =          HDLOAD    equ     04H             ;8" disk: load head
                                                  ;Minidisk: restart 6.4 S timer
0008 =          HDUNLD    equ     08h             ;unload head (8" only)
0010 =          IENABL    equ     10h             ;Enable sector interrupt
0020 =          IDSABL    equ     20h             ;Disable interrupts
0080 =          WENABL    equ     80h             ;Enable drive write circuits

0009 =          DSECTR  equ     09h             ;Sector Position input
0001 =          SVALID    equ     01h             ;Sector Valid (1st 30 uS
                                                  ;..of sector pulse)
003E =          SECMSK    equ     3Eh             ;Sector mask for MDSEC

000A =          DDATA   equ     0Ah             ;Disk Data (input/output)

                ;-----------------------------
                ; Single-byte error messages
                ;-----------------------------
0043 =          CERMSG  equ     'C'             ;checksum/marker byte error
004D =          MERMSG  equ     'M'             ;memory write verify error
004F =          OERMSG  equ     'O'             ;Memory overlay error message

                ;================================================================
4C00                    ORG     RAMADR          ;assemble at dest RAM address
                ;================================================================

4C00 F3                 di                      ;turn off INTE (no error yet)

                ;----------------------------------------------------------------
                ; Copy the PROM content to RAM for execution. This copy routine
                ;   is position independent so the boot PROM can be at most any
                ;   address. The LSB of the PROM address and RAMADR must be 0.
                ;----------------------------------------------------------------
4C01 110E4C             lxi     d,MLOOP         ;DE->MLOOP in RAM

4C04 317B4D             lxi     sp,STACK
4C07 21E1E9             lxi     h,0E9E1h        ;H=PCHL,L=POP H
4C0A E5                 push    h               ;POP H, PCHL at STACK-2, STACK-1
4C0B CD794D             call    STACK-2         ;addr of MLOOP in HL and stack RAM

4C0E 3B         MLOOP:  dcx     sp              ;point SP to MLOOP address
4C0F 3B                 dcx     sp              ;    in stack memory

4C10 7E                 mov     a,m             ;get next EPROM byte
4C11 12                 stax    d               ;store it in RAM

4C12 1C                 inr     e               ;bump pointers
4C13 2C                 inr     l
4C14 C0                 rnz                     ;copy to end of 256 byte page

4C15 C3184C             jmp     RAMIMG          ;jump to code now in RAM
```

```
                ; e=l=0

                ;===============================================================
                ; RAM Code Image
                ; All of the following code gets copied into RAM at RAMADR,
                ; and run there.
                ;===============================================================
                RAMIMG:

                ;---------------------------------------------------------------
                ; Wait for user to insert a diskette into the drive 0, and
                ; then load that drive's head. Do this first so that the disk
                ; has plenty of time to settle. Note that a minidisk will
                ; always report that it is ready. Minidisks will hang (later
                ; on) waiting for sector 0F, until a few seconds after the
                ; user inserts a disk.
                ;
                ; On Entry:
                ;   l = 0
                ;---------------------------------------------------------------
4C18 AF         WAITEN: xra     a               ;boot from disk 0
4C19 D308               out     DENABL          ;enable disk 0
4C1B DB08               in      DSTAT           ;Read drive status
4C1D E608               ani     DRVRDY          ;Diskette in drive?
4C1F C2184C             jnz     WAITEN          ;no: wait for drive ready

4C22 3E04               mvi     a,HDLOAD        ;Load 8" disk head, or enable
4C24 D309               out     DCTRL           ;..minidisk for 6.4 Sec

                ;---------------------------------------------------------
                ; Step in once, then step out until track 0 is detected
                ; On Exit: b=0
                ;---------------------------------------------------------
4C26 018206             lxi     b,20000/12      ;20 mS delay 1st time thru
4C29 3E01               mvi     a,STEPIN        ;step in once first

4C2B D309       SKTRK0: out     DCTRL           ;issue step command

                ; The first time through, delay at least 20ms to force a
                ; minimum 43 ms step wait instead of 10ms. This meets
                ; the 8" spec for changing seek direction. The minidisk
                ; step time is always 50ms.

4C2D 0B         DELAY:  dcx     b               ;(5)
4C2E 78                 mov     a,b             ;(5)
4C2F B1                 ora     c               ;(4)
4C30 C22D4C             jnz     DELAY           ;(10)12 uS/pass

4C33 0C                 inr     c               ;from now on, the above loop
                                                ;goes 1 time.

4C34 DB08       WSTEP:  in      DSTAT           ;wait for step to complete
4C36 0F                 rrc                     ;put MVHEAD bit in carry
4C37 0F                 rrc                     ;is the servo stable?
4C38 DA344C             jc      WSTEP           ;no: wait for servo to settle

4C3B E610               ani     TRACK0/4        ;Are we at track 00?
4C3D 3E02               mvi     a,STPOUT        ;STEP-OUT command
4C3F C22B4C             jnz     SKTRK0          ;no: step out another track

                ;Exit with b=0
```

```
                        ;--------------------------------------------------------
                        ; Determine if this is an 8" disk or a minidisk, and set
                        ; c to the correct sectors/track for the detected disk.
                        ; An 8" disk has 20h sectors, numbered 0-1Fh. A minidisk
                        ; has 10h sectors, numbered 0-0Fh.
                        ;--------------------------------------------------------

                        ; wait for the highest minidisk sector, sector number 0Fh

4C42 DB09               CKDSK1: in      DSECTR          ;Read the sector position

4C44 E63F                       ani     SECMSK+SVALID   ;Mask sector bits, and hunt
4C46 FE1E                       cpi     (MDSPT-1)*2     ;..for minidisk last sector
4C48 C2424C                     jnz     CKDSK1          ;..only while SVALID is 0

                        ; wait for this sector to pass

4C4B DB09               CKDSK2: in      DSECTR          ;Read the sector position
4C4D 0F                         rrc                     ;wait for invalid sector
4C4E D24B4C                     jnc     CKDSK2

                        ; wait for and get the next sector number

4C51 DB09               CKDSK3: in      DSECTR          ;Read the sector position
4C53 0F                         rrc                     ;put SVALID in carry
4C54 DA514C                     jc      CKDSK3          ;wait for sector to be valid

                        ; The next sector after sector 0Fh will be 0 for a minidisk,
                        ; and 10h for an 8" disk. Adding MDSPT (10h) to that value
                        ; will compute c=10h (for minidisks) or c=20h (for 8" disks).

4C57 E61F                       ani     SECMSK/2        ;Mask sector bits
4C59 C610                       adi     MDSPT           ;compute SPT
4C5B 4F                         mov     c,a             ;..and save SPT in c

                        ;--------------------------------------------------------------
                        ; Initialize the ACIA (2SIO port 0/Turnkey/UIO). Do this
                        ; late in the initialization, so that e.g. the 'B' character
                        ; from UBMON won't get eaten by resetting the ACIA.
                        ;--------------------------------------------------------------
4C5C 3E03                       mvi     a,ACRST         ;reset first
4C5E D310                       out     ACCTRL

4C60 3E11                       mvi     a,ACINIT        ;then initialize
4C62 D310                       out     ACCTRL

                        ;---------------------
                        ; Initialize the 4PIO
                        ;---------------------
4C64 AF                         xra     a
4C65 D322                       out     P4CB0           ;Port 0 section B is output
4C67 2F                         cma                     ;All output bits high
4C68 D323                       out     P4DB0
4C6A 3E2C                       mvi     a,P4CINI        ;set up handshake bits
4C6C D322                       out     P4CB0

                        ;-------------------------------------------
                        ; Set up to load
                        ; On Entry:
                        ;   b = 0 (initial sector number)
                        ;   c = SPT (for either minidisk or 8" disk)
                        ;   l = 0 (part of DMA address)
                        ;-------------------------------------------
```

```
4C6E 65                 mov     h,l             ;initial DMA address=0000

                ;---------------------------------------------------------
                ; Read current sector over and over, until either the
                ; checksum is right, or there have been too many retries
                ;   b = current sector number
                ;   c = sectors/track for this kind of disk
                ;  hl = current DMA address
                ;---------------------------------------------------------
4C6F 3E10       NXTSEC: mvi     a,RETRYS        ;(7)Initialize sector retries

                ;---------------------------------------------------------
                ; Begin Sector Read
                ;   a = Remaining retries for this sector
                ;   b = Current sector number
                ;   c = Sectors/track for this kind of disk
                ;  hl = current DMA address
                ;---------------------------------------------------------
4C71 317B4D     RDSECT: lxi     sp,STACK        ;(10)(re)initialize the stack
4C74 F5                 push    psw             ;(11)Remaining retry count

                ;---------------------------------------------------------
                ; Sector Read: Step 1. Hunt for sector specified in b. Data
                ;              will become avaiable 250 uS after -SVALID
                ;              goes low. -SVALID is low for 30 uS.
                ;---------------------------------------------------------
4C75 DB09       FNDSEC: in      DSECTR          ;(10)Read the sector position

4C77 E63F               ani     SECMSK+SVALID   ;(7)yes: Mask sector bits
                                                ;..along with -SVALID bit
4C79 0F                 rrc                     ;(4)sector bits to bits <4:0>
4C7A B8                 cmp     b               ;(4)Found the desired sector
                                                ;..with -SVALID low?
4C7B C2754C             jnz     FNDSEC          ;(10)no: wait for it

                ;---------------------------------------------------------
                ; Test for DMA address that would overwrite this RAM code
                ; or the next page (which contains the sector buffer stack)
                ; Do this here, while we have some time.
                ;---------------------------------------------------------
4C7E 117B4D             lxi     d,SECBUF        ;(10)Sector buffer address

4C81 7C                 mov     a,h             ;(5)high byte of DMA address
4C82 AA                 xra     d               ;(4)high byte of RAM code addr
4C83 E6FE               ani     0FEh            ;(7)ignore lsb
4C85 3E4F               mvi     a,OERMSG        ;(7)overlay error message
4C87 CAE14C             jz      RPTERR          ;(10)report overlay error

                ;---------------------------------------
                ; Set up for the upcoming data move
                ; Do this here, while we have some time.
                ;---------------------------------------
4C8A E5                 push    h               ;(11)Current DMA address
4C8B C5                 push    b               ;(11)Current sector & SPT
4C8C 018000             lxi     b,BPS           ;(10)b= init checksum,
                                                ;c= byte count for MOVLUP

                ;---------------------------------------------------------
                ; Sector Read: Step 2. Read sector data into SECBUF at de.
                ;              SECBUF is positioned in memory such that e
                ;              overflows at the end of the buffer. Read data
                ;              becomes available 250 uS after -SVALID becomes
                ;              true (0).This loop must be << 32 uS per pass.
```

```
                ;-------------------------------------------------------------
4C8F DB08       DATLUP: in      DSTAT           ;(10)Read the drive status
4C91 07                 rlc                     ;(4)New Read Data Available?
4C92 DA8F4C             jc      DATLUP          ;(10)no: wait for data

4C95 DB0A               in      DDATA           ;(10)Read data byte
4C97 12                 stax    d               ;(7)Store it in sector buffer
4C98 1C                 inr     e               ;(5)Move to next buffer address
                                                ;..and test for end
4C99 C28F4C             jnz     DATLUP          ;(10)Loop if more data

                ;-------------------------------------------------------
                ; Sector Read: Step 3. Move sector data from SECBUF into
                ;              memory at hl. Compute checksum as we go.
                ;
                ; 8327 cycles for this section
                ;-------------------------------------------------------
4C9C 1E7E               mvi     e,SDATA and 0FFh ;(7)de= address of sector data
                                                 ;..within the sector buffer

4C9E 1A         MOVLUP: ldax    d               ;(7)Get sector buffer byte
4C9F 77                 mov     m,a             ;(7)Store it at the destination
4CA0 BE                 cmp     m               ;(7)Did it store correctly?
4CA1 C2DF4C             jnz     MEMERR          ;(10)no: abort w/ memory error

4CA4 80                 add     b               ;(4)update checksum
4CA5 47                 mov     b,a             ;(5)Save the updated checksum

4CA6 13                 inx     d               ;(5)Bump sector buffer pointer
4CA7 23                 inx     h               ;(5)Bump DMA pointer
4CA8 0D                 dcr     c               ;(5)More data bytes to copy?
4CA9 C29E4C             jnz     MOVLUP          ;(10)yes: loop

                ;-------------------------------------------------------------
                ; Sector Read: Step 4. Check Marker byte and compare computed
                ;              checksum against sector's checksum. Retry/abort
                ;              if wrong Marker byte or checksum mismatch.
                ;
                ; a=computed checksum
                ; 98 cycles for for this section
                ;-------------------------------------------------------------
4CAC EB                 xchg                    ;(4)hl=1st trailer byte address
                                                ;de=DMA address
4CAD 4E                 mov     c,m             ;(7)get marker, should be FFh
4CAE 0C                 inr     c               ;(5)c should be 0 now

4CAF 23                 inx     h               ;(5)(hl)=checksum byte
4CB0 AE                 xra     m               ;(7)compare to computed cksum
4CB1 B1                 ora     c               ;(4)..and test marker=ff

4CB2 C1                 pop     b               ;(10)Current sector & SPT
4CB3 C2D24C             jnz     BADSEC          ;(10)NZ: checksum error

                ; Compare next DMA address to the file byte count that came
                ; from the sector header. Done of DMA address is greater.

4CB6 2A7C4D             lhld    SFSIZE          ;(16)hl gets file size
4CB9 EB                 xchg                    ;(4)put DMA address back in hl
                                                ;..and file size into de

4CBA 7D                 mov     a,l             ;(4)16-bit subtraction
4CBB 93                 sub     e               ;(4)
4CBC 7C                 mov     a,h             ;(4)..throw away the result
```

```
4CBD 9A                 sbb     d               ;(4)..but keep carry (borrow)

4CBE D2E34C             jnc     LDDONE          ;(10)done loading if hl >= de
                                                ;carry will be clear at LDDONE

                ;-----------------------------------------------------------
                ; Next Sector: The sectors are interleaved by two. Read all
                ;              the even sectors first, then the odd sectors.
                ;              Note that NXTSEC will repair the stack.
                ;
                ; 44 cycles for the next even or next odd sector
                ;-----------------------------------------------------------
4CC1 116F4C             lxi     d,NXTSEC        ;(10)for compact jumps
4CC4 D5                 push    d               ;(10)

4CC5 04                 inr     b               ;(5)sector = sector + 2
4CC6 04                 inr     b               ;(5)

4CC7 78                 mov     a,b             ;(5)even or odd sectors done?
4CC8 B9                 cmp     c               ;(4)c=SPT
4CC9 D8                 rc                      ;(5/11)no: go read next sector
                                                ;..at NXTSEC

                ; Total sector-to-sector = 28+8327+98+44=8497 cycles=4248.5 uS
                ; one 8" sector time = 5208 uS, so with 2:1 interleave, we will
                ; make the next sector, no problem.

4CCA 0601               mvi     b,01H           ;1st odd sector number
4CCC C8                 rz                      ;Z: must read odd sectors now
                                                ;..at NXTSEC

                ;-----------------------------------------------------------
                ; Next Track: Step in, and read again.
                ;             Don't wait for the head to be ready (-MVHEAD),
                ;             since we just read the entire previous track.
                ;             Don't need to wait for this step-in to complete
                ;             either, because we will definitely blow a
                ;             revolution going from the track's last sector to
                ;             sector 0. (One revolution takes 167 mS, and one
                ;             step takes a a maximum of 40 uS.)
                ;             Note that NXTRAC will repair the stack.
                ;-----------------------------------------------------------
4CCD 78                 mov     a,b             ;STEPIN happens to be 01h
4CCE D309               out     DCTRL

4CD0 05                 dcr     b               ;start with b=0 for sector 0
4CD1 C9                 ret                     ;go to NXTSEC

                ;***Error Routine*********************************************
                ; Checksum error: attempt retry if not too many retries
                ; already. Otherwise, abort, reporting the error
                ; On Entry:
                ;   Top of stack = adress for first byte of the failing sector
                ;   Next on stack = retry count
                ;*************************************************************
4CD2 3E04       BADSEC: mvi     a,HDLOAD        ;Restart Minidisk 6.4 uS timer
4CD4 D309               out     DCTRL

4CD6 E1                 pop     h               ;Restore DMA address
4CD7 F1                 pop     psw             ;Get retry count
4CD8 3D                 dcr     a               ;Any more retries left?
4CD9 C2714C             jnz     RDSECT          ;yes: try reading it again
```

```
                ;-----------------------------------------------------
                ; Irrecoverable error in one sector: too many retries.
                ; These errors may be either incorrect marker bytes,
                ; wrong checksums, or a combination of both.
                ; On Entry:
                ;   hl=RAM adress for first byte of the failing sector
                ;-----------------------------------------------------
4CDC 3E43               mvi     a,CERMSG        ;Checksum error message
4CDE 11                 db      11H             ;'lxi d' opcode to skip
                                                ;..MEMERR and go to RPTERR

                ;***Error Routine*********************
                ; Memory error: memory readback failed
                ; On Entry:
                ;   hl = offending RAM address
                ;*************************************
4CDF 3E4D       MEMERR: mvi     a,MERMSG        ;Memory Error message

                ; Fall into RPTERR

                ;***CDBL Termination*******************************************
                ; Entry at RPTERR:
                ;   Report an error: turn the disk controller off, turn the
                ;   INTE light on, record the error in RAM at 0000h-0002h, and
                ;   then loop forever writing the error code (in register a)
                ;   to all known Terminal ports.
                ;   On Entry:
                ;      a = error code
                ;     hl = offending RAM address
                ;
                ; Entry at LDDONE:
                ;   Normal exit: Disable the disk controller and go execute
                ;   the loaded code at DMAADR.
                ;   On Entry:
                ;     Carry bit is cleared
                ;*************************************************************
4CE1 47         RPTERR: mov     b,a             ;error code
4CE2 37                 stc                     ;remember we had an error

4CE3 3E80       LDDONE: mvi     a,DDISBL        ;Disable the disk controller
4CE5 D308               out     DENABL

4CE7 D20000             jnc     DMAADR          ;normal exit: go execute the
                                                ;..loaded program

4CEA FB                 ei                      ;Signal error on the INTE LED

4CEB 220100             shld    1               ;Store the bad address
4CEE 78                 mov     a,b             ;recover the error code
4CEF 320000             sta     0               ;Store the error code

4CF2 D301       ERHANG: out     SIODAT          ;SIO
4CF4 D311               out     ACDATA          ;2SIO port 0/Turnkey/UIO
4CF6 D305               out     PIODAT          ;PIO
4CF8 D323               out     P4DB0           ;4PIO
4CFA C3F24C             jmp     ERHANG          ;Keep printing error code

4CFD                    end
```