AMON USERS MANUAL

Amon Version 2.0

15 August 2016

Martin Eberhard

Revision History

Manual Revision	Firmware Revision	Author	Revision
15 Aug 2016	2.00	M. Eberhard	First released version

TABLE OF CONTENTS

AMON

Full Featured ROM Monitor
For an Altair 8800 with an 88-2SIOJP or an 88-2SIO

INTRODUCTION

Amon is a full-featured ROM-based monitor for an Altair 8800 with my own 88-2SIOJP, or with an Altair 88-2SIO or an Altair 88-UIO. (The 88-UIO does not have a second serial port, so the default Transfer Port will not work.)

Amon provides commands for manipulating memory, transferring (uploading and downloading) memory in Altair Absolute Binary format and Intel Hex format, as well as booting from any Altair boot device (paper tape, cassette tape, 8" floppy disks, minidisks, or from an Altair Datakeeper hard disk).

Amon can also be used to program EPROMs, using a memory-based EPROM programmer such as any of the Cromemco Bytesavers.

MEMORY REQUIREMENTS

Amon requires the highest 256-byte page of contiguous RAM for stack space, variables, buffers, and for relocated code. During initialization, Amon will search and find this page. The address of Amon's RAM page is printed immediately following the sign-on banner.

I/O PORTS

Amon uses the 88-2SIOJP's (or 88-2SIO's) Port 0 as its console. It can use this board's Port 1 for its "Transfer Port", as well as any of the other standard Altair serial or parallel ports. The Transfer Port is initialized to be the 88-2SIOJP's (or 88-2SIO's) Port 1.

The Transfer Port is used as the source or destination for any of the five Transfer Commands.

AMON COMMANDS

Amon commands may be typed at the Amon prompt, '>'. Commands are executed once you type the Return key. You can correct typing mistakes with the DEL or the BACKSPACE key.

All parameters are 4 hex digits (16-bits), unless otherwise noted. Additional upper hex digits are ignored and leading zeros are assumed.

MEMORY COMMANDS

CO <SOURCE> <DEST> <COUNT> [<REPEAT>] (COPY MEMORY)

Copies <Count> bytes memory starting at address <SOURCE> to memory starting at address <DEST>. Optionally, repeats the copy <Repeat> times. (Max value for <Repeat> is FF for 255 passes.)

A period is printed on the Console for each completed pass through the copy, unless <Repeat>=1 (the default).

The CO command verifies the copy when done, using the VE command.

Press CONTROL-C to abort a Copy.

This command can be used to program an EPROM with (for example) a Cromemco Bytesaver board. See example below.

DU [<ADDRESS> [<COUNT>]] (DUMP MEMORY)

Dumps <Count> bytes memory on the Console in hexadecimal, starting at <Address>, which defaults to 0. If no <Count> is specified, then dump all 65K bytes of memory.

Press the space bar to pause and restart the dump, and press CONTROL-C to abort the memory Dump.

EN [<ADDRESS>] (ENTER MEMORY DATA)

Allows you to enter 2-digit hex data into memory starting at <Address>, using a space or Return as a separator between bytes. Type Return on a blank line to exit. If no address is provided, then the starting address will be 0.

CONTROL-C aborts without saving the current line of data.

EX [<ADDRESS> [<OPTION>]] (EXECUTE)

Calls <Address>, which defaults to 0. A RET instruction will return to the monitor, if the stack remains intact and the PROM has not been disabled (with an IN FFh instruction).

If <Option> = 1 (or any odd number), then Amon will input from port FFh prior to executing the requested code. This

will disable the Amon PROM on an 88-2SIOJP board (with its ED switch closed) and enable any other memory that occupies the top 2K-bytes of memory (starting at F800h).

For example, if your Altair has both an 88-2SIOJP and MITS's ROM Basic Module (88-RMB) board installed, then the top 2K-bytes of the 88-RMB will be disabled, and Amon will occupy this memory space, until an IN from port FFh is executed. Start ROM Basic (at address C000) from Amon this way, to disable the Amon PROM and enable the top 2K-bytes of ROM Basic:

```
EX C000 1
```

FI [<VALUE> [<ADDRESS> [<COUNT>]]] (FILL MEMORY)

Fills <Count> bytes of memory, starting at <Address>, with <Value>, which is a 2-digit hex value. <Value> and <Address> default to 0. <Count> defaults to all of memory, wrapping around if necessary. The fill stops after either <Count> bytes have been filled or the fill reaches the RAM pages used by Amon.

Note that FI with no arguments will clear all memory.

SE <ADDRESS> <ELEMENT1> [<ELEMENT2> [..<ELEMENTN>]] (SEARCH)

Search memory, starting at the specified address, for the specified sequence of elements, where each element is either a 2-digit hexadecimal number or a text string within single-quotes. Example: (Try this to find a string in the Amon PROM.)

```
SE 0 'M. Eberhard' 0D 0A 'RAM:'
```

This will print the address of the beginning of the sequence if it is found. You will be given a chance to continue searching for another instance of the sequence, if the sequence is found.

VE <ADDRESS1> <ADDRESS2> <COUNT> (VERIFY MEMORY)

Compares a block of memory starting at <Address1> that is <Count> bytes long, to an equal-sized block of memory starting at <Address2>. Differences are reported on the Console, with the address and data from the first data block, followed by the data found in the second block.

Press CONTROL-C to abort a verify operation.

Transfer Commands

All of these commands use the specified Transfer Port as either the source or destination for data.

AD <Address> <Count>(Dump Memory as Altair Binary File)

Dumps <Count> bytes of memory to the Transfer Port, starting at <Address>, in Altair Absolute Binary format. Note that AD does not append a GO Record to the file.

AL (Load and Execute Altair Binary File)

Loads an Altair Absolute Binary file via the Transfer Port and jumps to its execution address. Amon will input from port FFh prior to executing the loaded code, to disable Amon's PROM, freeing all 65K of memory space for RAM (assuming the ED switch on the 88-2SIOJP is closed).

You can abort a load with CONTROL-C. If your Altair Binary File does not end with a GO Record, then you will need to type CONTROL-C to return to Amon when the load is done.

HD <Address> <Count> [<Offset>](Dump Memory as Intel Hex)

Dumps <Count> bytes of memory to the Transfer Port, starting at <Address>, in Intel Hex format. Add optional <Offset> to the address of each record.

HL [<Offset>] (Load Intel Hex into memory)

Loads an Intel Hex file from the Transfer Port into memory at the addresses specified in the hex file. If <Offset> is specified, then it is added to the record addresses.

A period is printed on the Console for each record.

Any hex record data that would overwrite Amon will cause an address error. Amon will report hex errors and checksum errors as well. Any such error will abort the load.

Loading terminates with any record that has 0 data bytes. You can also abort the load by typing CONTROL-C.

TE [<ExitChr>] (Terminal Mode)

Enters Terminal Mode: console keyboard data goes to Transfer Port, and Transfer Port data goes to the Console. (Use this command to verify a Transfer Port connection.)

<ExitChr> specifies the Exit Character, a control character that defaults to CONTROL-C. Control characters may be entered without the CONTROL. For example, you may type Z instead of CONTROL-Z. (Note: if you type CONTROL-C as as <ExitChr>, the TE command will immediately abort.)

Type the Exit Character to exit Terminal Mode.

TP [<0-7>] (Set Transfer Port)

The Transfer Port is the port used for transferring Intel hex files with the AD, AL, HD, HL, and TE commands.

TP Value	Port	Port Address
0	88-2SIOJP Port 0 (2 stop bits)	10h,11h
1	88-2SIOJP Port 0 (2 stop bits)	10h,11h
2	88-SIO	00h,01h
3	88-ACR	06h,07h
4	88-4PIO Port 0	20h,21h
5	88-PIO	04h,05h
6	88-2SIOJP Port 1 (2 stop bits)	12h,13h
7	88-2SIOJP Port 0 (2 stop bits)	10h,11h

(TP 7 is a spare location that can be used for a custom port, with reassembly of Amon.)

Other Commands

BO (Boot from Altair Floppy Diskette)

This will boot from either an Altair 88-DCDD 8” diskette or from an Altair 88-MDS Minidisk, automatically determining which type of floppy drive is installed. Amon will input from port FFh prior to executing the loaded code, to disable Amon’s PROM, freeing all 65K of memory space for RAM (assuming the ED switch on the 88-2SIOJP is closed).

HB [<0/1>] (Boot from hard disk)

Boot from Altair Datakeeper hard disk subsystem. ‘HB 0’ boots from the removable cartridge (default), and ‘HB 1’ boots from the fixed platter. Amon will input from port FFh prior to executing the loaded code, to disable Amon’s PROM, freeing all 65K of memory space for RAM (assuming the ED switch on the 88-2SIOJP is closed).

TT <0/1> (Set Terminal Type)

TT 0 (or just TT) specifies a terminal that can backspace. TT 1 specifies a terminal (such as a Teletype) that cannot backspace. This command just affects how backspaces that you type are presented.

Entry Points

Amon has four different entry points. You can set up the 88-2SIOJP to jump to any of these at reset, using SW1 and the JS switch. (See the 88-2SIOJP manual.)

F800h: Amon Monitor

Entry at F800h invokes the monitor, as described in the previous section.

FC00h: Boot from Altair Hard Disk (HDBL)

Entry at FC00h boots from the removable cartridge of an Altair Datakeeper hard disk subsystem. The HDBL code will input from port FFh prior to executing the loaded code, to disable Amon's PROM, freeing all 65K of memory space for RAM (assuming the ED switch on the 88-2SIOJP is closed).

FE00h: Boot from Altair Tape (MBL)

Entry at FD00h boots from either an Altair paper tape or cassette tape. This is exactly the same as invoking MITS's MBL loader PROM. The MBL code will input from port FFh prior to executing the loaded code, to disable Amon's PROM, freeing all 65K of memory space for RAM (assuming the ED switch on the 88-2SIOJP is closed). The boot device is specified by three switches on the Altair front panel, as follows:

A10	A9	A8	Boot Device
0	0	0	88-2SIO Port 0
0	0	1	88-2SIO Port 0
0	1	0	88-SIO
0	1	1	88-ACR
1	0	0	88-4PIO Port 0
1	0	1	88-PIO
1	1	0	88-2SIO Port 1
1	1	1	88-2SIO Port 0 (Custom port)

FF00h: Boot Altair Floppy Disk (DBL & MDBL)

Entry at FF00h boots from either an Altair 88-DCDD 8″ floppy disk or from an Altair 88-MDS minidisk. This is equivalent to MITS's DBL and MDBL boot PROMs, with the added functionality of automatically detecting which kind of drive is attached. (This is exactly the same as my own CDBL Combo-Disk Boot Loader.) The CDBL code will input from port FFh prior to executing the loaded code, to disable Amon's PROM, freeing all 65K of memory space for RAM (assuming the ED switch on the 88-2SIOJP is closed).

EXAMPLE: PROGRAMMING AN EPROM WITH A CROMEMCO BYTESAVER

As an example, suppose:

1. We have a Cromemco 8K Bytesaver board, which occupies addresses E000h through FFFFh[1]
2. We have assembled code whose target address is E400h (which is Socket 1 in this 8K Bytesaver)
3. We actually use the Bytesaver's Socket 2 (starting at E800h) for programming EPROMS.[2]
4. We will use 400h bytes of RAM, starting at 1000h, as a buffer
5. We plan to load the hex file via Port 1 of an 88-2SIOJP

Step 1: Select 88-2SIOJP's Port 1 as the Transfer Port

```
>TP 6
```

(You can verify that the Transfer Port is working by using the TE command.)

Step 2: Load the Intel Hex file into the RAM buffer:

The Intel Hex file that was generated by our assembler has address fields starting at E400. The address offset to our buffer is calculated as follows:

1000h - E400h = -D400h

To create a negative hex number, compliment, and add one:

-D400h = 2BFFh+1 = 2C00h

Load the Intel Hex file with this offset:

```
>HL 2C00
```

{Send the Intel Hex file to the Transfer port}

The file should now be in RAM, starting at 1000h. You can see it using the Memory Dump command:

```
>DU 1000 400
```

Step 3: Program the EPROM

The 8K Bytesaver uses 2708 EPROMs, which have 400h bytes of data, and require 60 (3Ch) programming passes on a Cromemco 8K Bytesaver.

Note that Cromemco recommends removing the Programming Diodes on the 8K Bytesaver, for any EPROM sockets that contain code that you don't want to overwrite accidentally. Make sure that

[1] An 8K Bytesaver has eight sockets, each of which can read or program a 2708 EPROM.

[2] We might do this because we have a ZIF socket installed in the 8K Bytesaver's Socket 2 (or any other socket).

the socket that you plan to use for programming has its Programming Diode installed. (These diodes are just above the sockets, near pin 24 - see the 8K Bytesaver manual.)

To program and verify our EPROM:

1. Insert a blank EPROM in 8K Bytesaver Socket 2
2. Turn on the red programming switch on the 8K Bytesaver
3. Issue a Copy command:

```
>CO 1000 E800 400 3C
```

Programming will take about 35 seconds. When done, the EPROM will be verified, and any mismatches will be reported on the Console.

4. Turn off the red programming switch on the 8K Bytesaver.

Step 4: Move the EPROM to its target socket

Remove the EPROM from Socket 2 and insert it in Socket 1.

Alternatively, we could have just put the EPROM in the 8K Bytesaver’s Socket 1 in the first place (assuming that Socket 1 has its Programming Diode installed), and programmed it there:

```
>CO 1000 E400 400 32
```

ALTAIR ABSOLUTE BINARY FILE FORMAT

An Altair 'Absolute Binary file' has Four sections, which may be separated by any number of nulls. These sections are:

1. The Leader, which comprises 2 or more identical bytes, the value of which is the length of the checksum loader.
2. The Checksum Loader, which is a program that is normally used to load the subsequent sections. This Loader is written backwards in the file.
3. Zero or more Load Records, each structured as follows:

   byte 0: Sync byte = 3Ch (identifies a Load Record)

   byte 1: NN = number of data bytes in the Load Record

   byte 2: LL = load address low byte

   byte 3: HH = load address high byte

   bytes 4-NN+3: NN data bytes to store at HHLL, NN>0

   byte NN+4: CC = checksum of bytes 2 through NN+3

4. The GO record, structured as follows

   byte 0: Sync byte = 78H (identifies the GO record)

   byte 1: LL = low byte of go address

   byte 2: HH = high byte of go address

Altair file Leaders and Checksum Loaders are specific to both the version of the particular software and the memory size. For example, the Checksum Loader for 4K Basic 3.2 is different than the Checksum Loader for 8K Basic 3.2, and both the Leader and Checksum Loader for 8K Basic 3.2 are different than those for 8K Basic 4.0.

Amon avoids problems with the different Checksum Loaders by ignoring the Checksum Loader in the file, and loading the Load Records directly.

Amon Source Code Listing

```
;=============================================================
; AMON
;
; ROM-based monitor for an 8080- based system, supporting the
; 88-2SIOJP and the Altair 88-2SIO.
;
; AMON assumes the console is on port 0 of the 88-2SIO/JP,
; and that the console terminal optionally may be a printing
; terminal (e.g. a Teletype) that has no backspace capability.
;
; AMON defines a "transfer port" for uploads, downloads, and
; terminal mode. This can be set to any of the standard Altair
; ports. You can also set up a custom port prior to assembly,
; which will be port 7 in the TP command. (If your custom port
; requires initialization, then you must add code for this.)
;
; Formatted to assemble with digital Research's ASM.
;
;=============================================================
; Entry Points:
; F800h: Cold-start AMON, enter command loop
; FC00h: Boot from MITS 88-HDSK Altair Hard Disk
;        (equivalent to my HDBL)
; FE00h: Boot from Altair paper or cassette tape
;        (equivalent to MITS's MBL)
; FF00h: Boot from MITS 88-DCDD 8" floppy or 88-MDS minidisk
;        (equivalent to my CDBL, and MITS's DBL and MDBL)
;=============================================================
; Commands (all values are in hex):
;
; AD <ADR> <BCNT>
;     Write <BCNT> bytes of memory starting at <ADR> in Altair
;     Absolute Binary format, to the current Transfer Port.
;
; AL Load and execute an Altair Absolute Binary file from the
;    current Transfer Port. (This is MBL.)
;
; BO  Boot from Altair floppy disk. (This is CDBL.)
;
; CO <SRC> <DST> <BCNT> [<RPT>]
;    Copy <BCNT> bytes of memory from address <SRC> to address
;    <DST>. optionally repeat <RPT> times (For programming
;    EPROMS with e.g. a Cromemco Bytesaver).
;
; DU [<ADR> [<BCNT>]]
;    Dump <BCNT> (which defaults to 1) bytes of memory starting
;    at address <ADR> (which defaults to 0).
;
; EN [<ADR>]
;    Enter hex data into memory at <ADR>, which defaults to 0.
;    values are separated with spaces or CR'S. Quit EN command
;    with a blank line.
;
; EX [<ADR> [<OPT>]]
;    Execute at <ADR>, which defaults to 0. Programs can ret
;    to AMON's MAIN loop. If <OPT>=1 then an IN from port
;    FF is executed first, to disable this PROM.
;
; FI [<VAL> [<ADR> [<BCNT>]]]
;    Fill <BCNT> bytes of memory starting at <ADR> with <VAL>
;    <VAL> and <ADR> default to 0. <BCNT> defaults to all of
```

```
;    memory, stopping (after wrap-around if necessary) when
;    the fill reaches AMON's RAM page.
;
; HB [<PLTR>] Boot from hard disk platter <PLTR> (0 or 1)
;
; HD <ADR> <BCNT> [<OFST>]
;    Intel hex dump <BCNT> bytes of memory starting at <ADR>,
;    to the Transfer Port. Add <OFST> to each address.
;
; HL [<OFST>]
;    Load Intel hex file to memory from the Transfer Port. Add
;    optional address offset <OFST> to each record address.
;    Prints a pacifier dot on the console for each record.
;
; SE <ADR> <BYTE1> [<BYTE2> [<BYTE3> [..<BYTEn>]]]
;     or
; SE <ADR> 'text string'
;    Search for string of bytes in memory, starting at <ADR>
;    can also mix forms, e.g.
; SE 100 'hello world' 0D 0A 'second line'
;
; TE [<EXCHR>]
;    Terminal Mode: console keyboard data goes to the Transfer
;    port, and Transfer Port data goes to the console.
;    ^C to exit, unless you specified a different exit chr.
;
; TP [<port>]
;    Set the Transfer Port:
;        port   device
;         0    88-2SIO port 0, 2 stop bits
;         1    88-2SIO port 0, 2 stop bits
;         2    88-SIO
;         3    88-ACR
;         4    88-4PIO port 0
;         5    88-PIO
;         6    88-2SIO port 1, 2 stop bits
;         7    Custom port (set up for 88-2SIO Port 0)
;
; TT [0/1]
;    TT 1 specifies a Teletype (or other non-backspacing
;    device) as the console. TT or TT 0 specifies a device
;    (such as a terminal) that can backspace. This controls how
;    a backspace is displayed.
;
; VE <SRC> <DST> <BCNT>
;    Verify (compare) <BCNT> bytes of memory, starting at <SRC>
;    and <DST>
;
;=================================================================
; RAM USAGE
; This program finds and uses the highest contiguous 256-byte
; page of RAM for its stack, buffers, and serial I/O routines.
; This page is organized as follows. Note that the 88-2SIOJP
; may be configured to disable the PROM once an "IN 0FFh"
; (input from the front panel switch register) is executed.
; When MBL is executed directly (not via a call from AMON),
; it reads the switch register to determine the boot port.
; xx00: Transfer Port I/O routines
;       RSETP: set the Transfer Port according to register a
;              (see TP command below.)
;       RTPIS: get Transfer Port input status. Z clear if
;              data is available.
;       RTPIN: wait for and get one chr from the Transfer Port
```

```
                ;       RTIIF: read immediately from the Transfer Port (flush)
                ;       RTPOUT: write a to the Transfer Port
                ; xx6B-xx7A: Stack (room for 8 pushes)
                ; xx7B-xxFF: Sector buffer (for BO command)
                ; xx7B-xxFF: MBL RAM code for AL command, especially for direct
                ;           execution from FE00
                ; xx7B-xxCA: Command line buffer for monitor
                ;=============================================================
                ; REVISION HISTORY
                ; Vers. 1.00-1.06
                ;   Development
                ; Vers. 2.0  M. Eberhard  26 July 2016
                ;   First released version
                ;=============================================================
0000 =          FALSE   equ     0
FFFF =          TRUE    equ     not FALSE

                ;=============================================
                ; Custom Port Definition
                ; Change these values for a custom transfer port
                ; The custom port's data port address must be
                ; immediately after its ctrl/stat port.
                ;=============================================
                ;default is 88-2SIO
0010 =          CPRCTL  equ     10h             ;Rx ctrl/stat port
0011 =          CPRDAT  equ     CPRCTL+1        ;Rx data must be CPRCTL+1
0001 =          CPRRDY  equ     01h             ;Receiver ready flag

0010 =          CPTCTL  equ     10h             ;Tx ctrl/stat port
0011 =          CPTDAT  equ     CPTCTL+1        ;Tx data must be CPTCTL+1
0002 =          CPTRDY  equ     02h             ;transmitter ready flag

0000 =          CPSPOL  equ     0               ;0 for active-high flags
                                                ;1 for ative-low flags
                ;*****
                ;ASCII
                ;*****
0003 =          CTRLC   equ     03H             ;control-C
0008 =          BS      equ     08H             ;backspace
000D =          CR      equ     0DH
000A =          LF      equ     0AH
0027 =          QUOTE   equ     27h             ;single-quote
007F =          DEL     equ     7Fh             ;delete

                ;---------------
                ;program Equates
                ;---------------
003E =          PROMPT  equ     '>'     ;Prompt character
0003 =          CABKEY  equ     CTRLC   ;command abort character
0003 =          DTEXIT  equ     CTRLC   ;default Terminal Mode exit CHR
0020 =          PAUKEY  equ     ' '     ;pauses dumping
002E =          PCFIER  equ     '.'     ;console pacifier character

0050 =          LBSIZE  equ     80      ;input line buffer size
0010 =          HRLEN   equ     16      ;Intel hex record length for HD

0006 =          DTPORT  equ     6       ;default transfer port

                ;-------------------------------
                ;Single-Character 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

                ;-----------------------------------
                ;Altair Absolute Binary file Equates
                ;-----------------------------------
003C =          ALTPLR  equ     3CH     ;program load record
0078 =          ALTEOF  equ     78H     ;EOF/GO address record
0055 =          ALTBNR  equ     55H     ;begin/program name (not supported)
000D =          ALTBND  equ     0DH     ;end-of-name mark (not supported)
003C =          LBSYNC  equ     3CH     ;Altair file Load block synch chr
003C =          LDRLEN  equ     60      ;Leader/trailer length

                ;--------------------
                ;Sense Switch Equates
                ;--------------------
00FF =          SSWTCH  equ     0FFh    ;front panel switch register
0007 =          LDMASK  equ     007H    ;load device mask

                ;--------------
                ;88-SIO Equates
                ;--------------
                ;88-SIO registers

0000 =          SIOCTL  equ     00              ;control port
0000 =          SIOSTA  equ     00              ;status
0001 =          SIOTXD  equ     01              ;transmit data
0001 =          SIORXD  equ     01              ;receive data

                ;status register bits

0001 =          SIOIDR  equ     00000001B       ;input dev rdy (RX BUF full)
0004 =          SIOPE   equ     00000100B       ;parity error
0008 =          SIOFE   equ     00001000B       ;framing error
0010 =          SIODOV  equ     00010000B       ;data overflow
0080 =          SIOODR  equ     10000000B       ;output dev rdy (TX BUF empty)

                ;-----------------------------------------------------
                ;88-ACR (Audio Cassette recorder) Equates
                ;NOTE: the Altair 88-ACR is built around an Altair 88-SIO
                ;-----------------------------------------------------
                ;88-ACR registers

0006 =          ACRCTL  equ     06              ;control port
0006 =          ACRSTA  equ     06              ;status
0007 =          ACRTXD  equ     07              ;transmit data
0007 =          ACRRXD  equ     07              ;receive data

                ;status register bits

0001 =          ACRIDR  equ     00000001B       ;input dev rdy (RX BUF full)
0004 =          ACRPE   equ     00000100B       ;parity error
0008 =          ACRFE   equ     00001000B       ;framing error
0010 =          ACRDOV  equ     00010000B       ;data overflow
0080 =          ACRODR  equ     10000000B       ;output dev rdy (TX BUF empty)

                ;-------------------------------------------
                ;88-4PIO Equates
                ;NOTE: the 88-HSR uses port 1 of the 88-4PIO
                ;-------------------------------------------
                ;88-4PIO registers

0020 =          P4CA0   equ     20h             ;port 0 section A ctrl/stat
0021 =          P4DA0   equ     21H             ;port 0 section A data
```

```
0022 =          P4CB0   equ     22H             ;port 0 section B ctrl/stat
0023 =          P4DB0   equ     23H             ;port 0 section B data
0024 =          P4CA1   equ     24H             ;port 1 section A ctrl/stat
0025 =          P4DA1   equ     25H             ;port 1 section A data
0026 =          P4CB1   equ     26H             ;port 1 section B ctrl/stat
0027 =          P4DB1   equ     27H             ;port 1 section B data

                ;control register bits

0001 =          P4C1C0  equ     00000001B       ;C1 control bit 0
0002 =          P4C1C1  equ     00000010B       ;C1 control bit 1
0004 =          P4DDR   equ     00000100B       ;data direction register
0008 =          P4C2C3  equ     00001000B       ;C2 control bit 3
0010 =          P4C2C4  equ     00010000B       ;C2 control bit 4
0020 =          P4C2C5  equ     00100000B       ;C2 control bit 5
0040 =          P4IC2   equ     01000000B       ;C2 interrupt control bit
0080 =          P4IC1   equ     10000000B       ;C1 interrupt control bit

                ;status register bits

0080 =          P4RDF   equ     10000000B       ;RX data register full
0080 =          P4TDE   equ     10000000B       ;TX data register empty
0040 =          HSRRDF  equ     01000000B       ;RX data register full for HSR

                ;--------------
                ;88-PIO Equates
                ;--------------
                ;88-PIO registers

0004 =          PIOCTL  equ     04              ;control port
0004 =          PIOSTA  equ     04              ;status
0005 =          PIOTXD  equ     05              ;transmit data
0005 =          PIORXD  equ     05              ;receive data

                ;status register bits

0002 =          PIORDF  equ     00000010B       ;RX data register full
0001 =          PIOTDE  equ     00000001B       ;TX data register empty

                ;----------------
                ; 88-2SIO Equates
                ;----------------
                ; 88-2SIO registers

0010 =          SIOBAS  equ     10h
0010 =          S2CTLA  EQU     SIOBAS          ;ACIA A control output port
0010 =          S2STAA  EQU     SIOBAS          ;ACIA A status input port
0011 =          S2TXDA  EQU     SIOBAS+1        ;ACIA A Tx data register
0011 =          S2RXDA  EQU     SIOBAS+1        ;ACIA A Rx data register
0012 =          S2CTLB  EQU     SIOBAS+2        ;ACIA B control output port
0012 =          S2STAB  EQU     SIOBAS+2        ;ACIA B status input port
0013 =          S2TXDB  EQU     SIOBAS+3        ;ACIA B Tx data register
0013 =          S2RXDB  EQU     SIOBAS+3        ;ACIA B Rx data register

                ;MOTOROLA 6850 ACIA ctrl/stat values

0001 =          S2RDF   EQU     00000001B       ;Rx data register full
0002 =          S2TBE   equ     00000010B       ;Tx data register empty

0003 =          S2RST   equ     00000011B       ;Master reset
0011 =          S22STP  equ     00010001B       ;2 stop bits, /16
0015 =          S21stP  equ     00010101B       ;1 stop bit, /16
```

```
                        ;--------------------------------------------------------
                        ; 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 =          SRTPOUT   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)

                        ;------------------------
                        ; Floppy 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 Allocation
                        ;******************
0000 =          DMAADR  equ     00000h          ;Disk load/execution address
                                                ;(Code assumes low byte=0)
F800 =          MONADR  equ     0F800h          ;Address of monitor
FC00 =          HDBADR  equ     0FC00h          ;Beginning of HDBL PROM
FE00 =          MBLADR  equ     0FE00h          ;MBL Subsystem address
FF00 =          DBLADR  equ     0FF00h          ;CDBL Subsystem address

                        ;-----------------------------------------
                        ;Addresses Offsets of components in High RAM
                        ;-----------------------------------------
0000 =          RAMCOD  equ     0               ;Relocated code at bottom
007B =          RAMBUF  equ     100h-SECSIZ     ;Exactly room for 1 complete sector
```

```
007B =          STACK   equ     RAMBUF          ;Stack grows down from here
0010 =          MINSTK  equ     10h             ;minimum stack size

                ;-----------------------------------------------
                ; Addresses offsets of sector components within
                ; the page that contains RAMBUF
                ;-----------------------------------------------
007C =          SFSIZE  equ     RAMBUF+1        ;address offset to file size
007E =          SDATA   equ     RAMBUF+HDRSIZ   ;address offset to sector data
00FE =          SMARKR  equ     SDATA+BPS       ;address offset to marker byte
00FF =          SCKSUM  equ     SMARKR+1        ;address offset to checksum byte

                ;============================
                ;= Cold-start Initialization =
                ;============================
F800                    org     MONADR          ;Monitor ROM start

F800 01C2F8             lxi     b,INIT2         ;return address

                ;Fall into INIT

                ;***Special Subroutine************************
                ; Initialization
                ;     find RAM for the stack and sector buffer
                ;     Install RAM code
                ;     Initialize I/O ports
                ; On Entry:
                ;   bc = return address
                ; On Exit:
                ;   bc = 0
                ;   sp = address of new stack
                ;   All standard Altair I/O ports initialized
                ;   interrupts disabled
                ;*********************************************
F803 F3         INIT:   di                      ;no interrupts please

                ;-----------------------------------------------
                ; Hunt for the highest RAM page
                ; This assumes at least one 256-byte page of RAM
                ;-----------------------------------------------
F804 217BFF             lxi     h,0FF00h+RAMBUF

F807 24         CSLOOP: inr     h               ;next RAM page

F808 7E                 mov     a,m             ;Original RAM data
F809 5F                 mov     e,a             ;remember for a moment
F80A 2F                 cma
F80B 77                 mov     m,a             ;write inverted
F80C BE                 cmp     m               ;Correct?
F80D 73                 mov     m,e             ;put original data back
F80E CA07F8             jz      CSLOOP          ;keep looking if RAM write OK

                ;-------------
                ; Create stack
                ;-------------
F811 25                 dcr     h               ;point to last good RAM
F812 F9                 sphl                    ;stack just below buffer
F813 C5                 push    b               ;stack return address

                ;----------------------------------
                ; Relocate  and Install RAM code
                ; h = destination address high byte
                ;----------------------------------
```

```
 F814 0159F8            lxi     b,RIOCOD                ;source
 F817 1E69              mvi     e,RCEND-RIOCOD          ;count
 F819 2E00              mvi     l,RAMCOD                ;destination

 F81B 0A        RCLOOP: ldax    b
 F81C B8                cmp     b                       ;need to relocate an
address?
 F81D C227F8            jnz     RCL1

 F820 2B                dcx     h                       ;back up to fix low address
byte
 F821 7E                mov     a,m
 F822 D659              sui     (RIOCOD-RAMCOD) and 0FFh ;low byte of offset
 F824 77                mov     m,a
 F825 23                inx     h

 F826 7C                mov     a,h                     ;high byte

 F827 77        RCL1:   mov     m,a

 F828 03                inx     b
 F829 23                inx     h
 F82A 1D                dcr     e
 F82B C21BF8            jnz     RCLOOP

                ;---------------------------------------
                ; Reset all standard Altair I/O devices
                ; the way that MBL does
                ;---------------------------------------
                ;make 4PIO 'A' channels inputs and 'B' channels outputs

 F82E AF                xra     a
 F82F D320              out     P4CA0           ;access 4PIO port 0A DDR
 F831 D321              out     P4DA0           ;set 4PIO port 0A as input

 F833 D322              out     P4CB0           ;access 4PIO port 0B DDR
 F835 2F                cma                     ;0FFh
 F836 D323              out     P4DB0           ;set 4PIO port 0B as output

                ;Set up the other 3 4PIO ports all the same

 F838 3E2C              mvi     a,2Ch   ;bits 0,1: C1 input active low, int off
                                        ;bit 2: access data reg
                                        ;bits 3-5: C2 output handshake

 F83A D320              out     P4CA0           ;4PIO port 0A control
 F83C D322              out     P4CB0           ;4PIO port 0B control

                ;Send reset command to both 2SIO ports

 F83E 3E03              mvi     a,S2RST         ;2SIO reset
 F840 D310              out     S2CTLA          ;2SIO port 0
 F842 D312              out     S2CTLB          ;2SIO port 1

                ;Set up both 2SIO ports: 8 data bits, 2 stop bits, no parity,
                ;clock divide by 16

 F844 3E11              mvi     a,S22STP        ;8N2, /16
 F846 D310              out     S2CTLA          ;2SIO port 0 control
 F848 D312              out     S2CTLB          ;2SIO port 1 control

                ;-------------------------------------------------
                ; Set the trasfer port to the default, and "Return"
```

```
                ; to the address provided in bc on entry.
                ;-----------------------------------------------

F84A 2E06               mvi     l,DTPORT

                ;Fall into SETTP to set the default transfer port

                ;***Command Routine******************
                ; TP [<port>] Set Transfer Port
                ;  Port   Device
                ;   0    88-2SIO port 0, 2 stop bits
                ;   1    88-2SIO port 0, 2 stop bits
                ;   2    88-SIO
                ;   3    88-ACR
                ;   4    88-4PIO port 0
                ;   5    88-PIO
                ;   6    88-2SIO port 1, 2 stop bits
                ;   7    Custom Port
                ;
                ; On Entry:
                ;   l=port number
                ;************************************
F84C 7D         SETTP:  mov     a,l             ;get port
F84D FE08               cpi     8
F84F D267FE             jnc     CMDERR

F852 210000             lxi     h,0             ;find address of RSETP
F855 39                 dad     sp              ;...located in RAM
F856 2E00               mvi     l,RSETP-RIOCOD+RAMCOD
F858 E9                 pchl                    ;run RSETP (with value in a)

                ;=============================================================
                ; AMON RAM I/O Code
                ; This code must be in RAM either because it gets modified or
                ; because it may get called after an IN from port FF (which may
                ; disable the PROM). All of RIOCOD must be in the same page.
                ;
                ; The ROM version of some of these routines also double as the
                ; console I/O routines, when called in ROM.
                ;=============================================================
                RIOCOD:

                ;---RAM Subroutine-----------------------------------------
                ; Patch the Transfer Port routines with the correct parameters
                ; for the load port that is specified in a.
                ; On Entry:
                ;    a = transfer port value (values compatible with MITS
                ;        loaders from rev 3.0 onward.). A < 8
                ; Trashes psw,b,de,hl
                ;---------------------------------------------------------
F859 11A1F8     RSETP:  lxi     d,PTABLE        ;lookup table

F85C 87                 add     a               ;4 bytes/entry
F85D 87                 add     a
F85E 83                 add     e               ;look up in PTABLE (clears carry)
F85F 5F                 mov     e,a             ;de=PTABLE(port value)

                ;Set up the input port routine

F860 1A                 ldax    d               ;input data port & CMA flag
F861 1F                 rar                     ;move CMA flag into Carry
F862 3292F8             sta     TPIDP+1         ;install data port address
```

```
                ;hl will get the status port (in l) and either NOP or CMA (in h)

F865 2600               mvi     h,0             ;NOP instruction
F867 D26CF8             jnc     RSETP1
F86A 262F               mvi     h,CMA           ;CMA instruction
                RSETP1:

F86C 3D                 dcr     a               ;status port = data port-1
F86D 6F                 mov     l,a             ;install status port address

                ;Set the status port and either NOP or CMA instruction

F86E 2286F8             shld    TPISP+1         ;status port and NOP/CMA

F871 1C                 inr     e               ;next table entry is
F872 1A                 ldax    d               ;..the data available mask
F873 3289F8             sta     TPIMSK+1        ;install mask

                ;Set up the output port routine

F876 1C                 inr     e               ;next table entry is
F877 1A                 ldax    d               ;..the data output port address
F878 329FF8             sta     TPODP+1         ;install data port address

F87B 3D                 dcr     a               ;status port = data port-1
F87C 6F                 mov     l,a             ;install stat port address
F87D 2296F8             shld    TPOSP+1         ;status port and NOP/CMA

F880 1C                 inr     e               ;next table entry is
F881 1A                 ldax    d               ;..the transmitter ready mask
F882 3299F8             sta     TPOMSK+1        ;install ready mask

                ;       ret

                ;Fall into RTPIS to return, saving one byte

                ;===Subroutine================
                ; Get Console keyboard Status
                ; On Exit:
                ;   Z clear if data available
                ;=============================
                KSTAT:

                ;Fall into the ROM version of Transfer Port Input Status

                ;---RAM Subroutine-------------------------------------
                ; Get Transfer Port input status
                ; This code gets modified by RSETP
                ; On Exit:
                ;   Z clear if data available
                ;------------------------------------------------------
                RTPIS:
F885 DB10       TPISP:  in      S2STAA          ;(status port address)read status
F887 00         TPINOP: nop                     ;(may get modified to CMA)
F888 E601       TPIMSK: ani     S2RDF           ;(port mask)
F88A C9                 ret

                ;---RAM Subroutine-------------------------------------
                ; Wait for and get a byte from the Transfer Port
                ; This code gets modified by RSETP
                ; On Exit:
                ;   a = input character
                ;   Z cleared
```

```
                ;------------------------------------------------------
F88B CD85F8     RTPIN:  call    RTPIS
F88E CA8BF8             jz      RTPIN           ;wait for data

                ;Fall into RTPIF

                ;---RAM Subroutine-----------------------------------------
                ; Get/Flush a byte from the Transfer Port immediately
                ; This code gets modified by RSETP
                ; On Entry:
                ;   Transfer port Rx data is ready
                ; On Exit:
                ;   a = input character
                ;   Z cleared
                ;------------------------------------------------------
                RTPIF:                          ;call here to flush port
F891 DB11       TPIDP:  in      S2RXDA          ;(data port place)get data byte

F893 C9                 ret                     ;result in a

                ;===Subroutine================
                ; Send byte to Console
                ; On Entry:
                ;    a = byte to send
                ; On Exit:
                ;    All registers preserved
                ;=============================
                PRINTA:

                ;Fall into the ROM version of Transfer Port Tx Data

                ;---RAM Subroutine-----------------------------------------
                ; Send a byte to the Transfer Port
                ; This code gets modified by RSETP
                ; On Entry:
                ;    a = byte to send
                ;------------------------------------------------------
F894 F5         RTPOUT: push    psw

                WAITPO:
F895 DB10       TPOSP:  in      S2STAA          ;(status port address)read status
F897 00         TPONOP: nop                     ;(may get modified to CMA)
F898 E602       TPOMSK: ani     S2TBE           ;(Tx port mask)
F89A CA95F8             jz      WAITPO

F89D F1                 pop     psw
F89E D311       TPODP:  out     S2TXDA          ;(data port place)
F8A0 C9                 ret

                ;---RAM Table---------------------------------------------------
                ;Port parameters: One 4-byte entry for each port:
                ; byte 1 = Rx data port address * 2 + cma flag
                ; byte 2 = ready mask for data input
                ; byte 3 = Tx data port address
                ; byte 4 = ready mask for data output
                ; Assumptions:
                ;  the control port for TX or Rx immediately precede the data
                ;    port.
                ;  the polarity of the Tx ready status bit is the same as the
                ;    rx empty status bit.
                ;  Rx port addresses are all < 80h
                ;------------------------------------------------------------
F8A1 22011102   PTABLE: db      S2RXDA*2,S2RDF,S2TXDA,S2TBE     ;0:2SIO A
```

```
F8A5 22011102          db      S2RXDA*2,S2RDF,S2TXDA,S2TBE     ;1:2SIO A
F8A9 03010180          db      SIORXD*2+1,SIOIDR,SIOTXD,SIOODR ;2:SIO
F8AD 0F010780          db      ACRRXD*2+1,ACRIDR,ACRTXD,ACRODR ;3:ACR
F8B1 42802380          db      P4DA0*2,P4RDF,P4DB0,P4TDE       ;4:4PIO port 0
F8B5 0A020501          db      PIORXD*2,PIORDF,PIOTXD,PIOTDE   ;5:PIO
F8B9 26011302          db      S2RXDB*2,S2RDF,S2TXDB,S2TBE     ;6:2SIO B

               ;8th entry is a custom port, defined above

F8BD 22011102          db      CPRDAT*2+CPSPOL,CPRRDY,CPTDAT,CPTRDY

               ;=============================================================
               ; RAM Variables
               ;=============================================================
F8C1 00        TTYPE:  db      0       ;0 means terminal (with backspace)
                                       ;1 means Teletype (without backspace)

               ;=============================================================
               ; Check for assembly problem: all of RIOCOD must be in the same
               ; 256-byte page of PROM
               ;=============================================================
F8C2 =         RCEND   equ     $

                if (RCEND-1)/256-(RIOCOD/256)
                        ERROR: RAM I/O code is not all in one page
                endif

               ;=============================================================
               ; Check for assembly problem: all of RIOCOD must fit in RAM
               ; together with the stack and the RAM buffer
               ;=============================================================
                if (((RCEND-1)-RIOCOD)+MINSTK+SECSIZ)/256
                        ERROR: RAM I/O code is too large
                endif

               ;=========================================
               ;= Cool-start Initialization             =
               ;= Repair stack, print banner, go to MAIN =
               ;= On Entry:                             =
               ;=   sp points to a valid stack address   =
               ;=========================================
F8C2 CD45FD    INIT2:  call    CILPRT          ;print banner
F8C5 414D4F4E20        db      'AMON  2.0 by M. Eberhard',CR,LF
F8DF 52414D3AA0        db      'RAM:',' '+80h

               ;Announce address of the first byte of RAM page

F8E4 AF                xra     a               ;set Z flag

               ;Fall into CABORT with Z set and a=0

               ;*************************************************
               ; Command abort: fix stack, go to MAIN
               ; On Entry:
               ;   sp points to a valid stack address
               ;   Z set and a=0 if stack address should be printed
               ;*************************************************
F8E5 210000    CABORT: lxi     h,0             ;find the stack
F8E8 39                dad     sp

F8E9 6F                mov     l,a             ;address lsb always 0
F8EA CC2AFE            cz      PHLCHX          ;Print hl on console
```

```
F8ED 2E7B               mvi     l,STACK         ;point to bottom of stack
F8EF F9                 sphl                    ;fix stack

                ;Fall into MAIN

                ;**********************************
                ; Command Processor Main entry point
                ; Get and process commands
                ;**********************************
                ;Print the prompt, and get a line of keyboard input

F8F0 01F0F8     MAIN:   lxi     b,MAIN          ;create command-return
F8F3 C5                 push    b               ;..address on the stack

F8F4 CD45FD             call    CILPRT          ;print CR,LF, prompt
F8F7 BE                 db      PROMPT+80h      ;will also enable ints

F8F8 CDE8FB             call    GETLIN          ;get user input line
                                                ;de=beginning of line
                                                ;Z set if no character found
                                                ;0 at end of line

F8FB C8                 rz                      ;No command? just ignore.

                ;Check command list, and execute the command if found

F8FC EB                 xchg                    ;command address to hl
F8FD 11B7FD             lxi     d,COMTAB-2      ;point to command table

F900 4E                 mov     c,m             ;1st command chr in c
F901 23                 inx     h               ;2nd command chr in m

                ;Search through table at de for a 2-character match of c,m
                ;allowing uppercase or lowercase letters.

F902 13         NXTCOM: inx     d               ;skip over address offset
F903 13                 inx     d
F904 1A                 ldax    d
F905 B7                 ora     a               ;test for table end
F906 CA67FE             jz      CMDERR          ;not in table

F909 A9                 xra     c               ;test first character
F90A 47                 mov     b,a             ;temp save result
F90B 13                 inx     d               ;2nd table character
F90C 1A                 ldax    d
F90D AE                 xra     m               ;test 2nd character

F90E 13                 inx     d               ;point to address offset

F90F B0                 ora     b                       ;both characters match?
F910 E6DF               ani     ('a'-'A') XOR 0FFh      ;lowercase is ok
F912 C202F9             jnz     NXTCOM                  ;NO match: keep looking

F915 23                 inx     h               ;skip past 2-letter command

                ;Got a match. Get command routine address and put it on the stack

F916 EB                 xchg                    ;(hl)=address of command routine
                                                ;de=input pointer

F917 4E                 mov     c,m             ;address low byte
F918 23                 inx     h
```

```
F919 7E                 mov     a,m             ;address high byte
F91A F680               ori     80h             ;clear non-hex flag bit
F91C 47                 mov     b,a             ;..to make legit address

F91D C5                 push    b               ;command routine address

                ;If the msb of the routine address was zero (this bit used as
                ;a flag), then any parameters are not hex - so go directly to
                ;the command execution routine.

F91E BE                 cmp     m               ;Non-hex?
F91F C0                 rnz                     ;y: go directly to routine

                ;Get the following hex parameter (if any) and put it in hl.
                ;Set the Carry flag if no parameter present.
                ;Leave de pointing to the 1st character after the 1st parameter.
                ;'return' to the Command Routine on the stack.

                ;Fall into FNDHEX

                ;***Subroutine*****************************************
                ; Scan past blanks and get a hex value
                ; On Entry:
                ;   de=address of next item in the input line buffer
                ; On Exit:
                ;   hl=value
                ;   de advanced past character
                ;   Z set, Carry clear if value
                ;   Carry set and a=hl=0 if no value found
                ;*****************************************************
F920 210000     FNDHEX: lxi     h,0             ;default value
F923 CDF6FB             call    SKIPB           ;skip spaces to find 1st digit
F926 37                 stc                     ;Carry set if no digits
F927 C8                 rz

F928 1A         FHEXLP: ldax    d               ;get digit
F929 B7                 ora     a               ;end of line?
F92A C8                 rz                      ;y: ret with carry clear

F92B FE20               cpi     ' '             ;value separator?
F92D C8                 rz                      ;y: ret with carry clear

F92E FE41               cpi     'A'             ;convert letters to uppercase
F930 DA35F9             jc      FHNUM
F933 E6DF               ani     ('a'-'A') XOR 0FFh
                FHNUM:

F935 29                 dad     h               ;make room for the new digit
F936 29                 dad     h
F937 29                 dad     h
F938 29                 dad     h

F939 CD6EFE             call    HEXCON          ;Do the conversion
F93C D267FE             jnc     CMDERR          ;not valid hexidecimal value?

F93F 85                 add     l
F940 6F                 mov     l,a             ;move new digit in
F941 13                 inx     d               ;bump the pointer
F942 C328F9             jmp     FHEXLP

                ;***Command Routine***********************************
                ; AD <SRC> <BCNT> (Dump memory in Altair binary format)
                ; Note that this does not punch a GO record at the end.
```

```
                ; On Entry:
                ;   hl=<SRC>
                ;   Carry set if none entered
                ;   de points to <BCNT>
                ;   TP command has set up the Transfer Port
                ;*****************************************************
F945 E5         ADUMP:  push    h               ;save source address
F946 CD63FE             call    GETHEX          ;get byte count
F949 D1                 pop     d

                ;de = source address
                ;hl = byte count

                ;Punch a pre-leader so that MITS's MBL can load this file

F94A 3E20               mvi     a,20h           ;punch 20h as the pre-leader
F94C CD7AF9             call    LEADER          ;returns b=0

                ;Punch null leader

F94F AF                 xra     a               ;Punch null leader
F950 CD7AF9             call    LEADER          ;returns with b=0

                ;Loop to punch all the requested data
                ;(b=0 here, both on initial entry and upon looping)

                ;Compute b=data byte count of the next block, max=255

F953 05         NXTBLK: dcr     b               ;b=FFh=255

F954 7C                 mov     a,h             ;>256 bytes left?
F955 B7                 ora     a
F956 C25AF9             jnz     BLKSIZ
F959 45                 mov     b,l             ;N: do what's left
                BLKSIZ:

                ;Punch the the block header info:
                ; sync chr, byte count, & 2-byte load address
                ;  b = block size
                ; de = starting memory address for block data
                ; hl = remaining bytes to punch

F95A D5                 push    d               ;save load address

F95B 1E3C               mvi     e,LBSYNC        ;Punch load-block sync chr
F95D 50                 mov     d,b             ;and block byte count
F95E CDA3FB             call    TPOED

F961 D1                 pop     d               ;restore load address
F962 CDA3FB             call    TPOED           ;Punch de=load address
                                                ;ends with a=d
F965 83                 add     e               ;a=checksum of the address

                ;Punch b bytes of block data, computing checksum as we go
                ;  a = checksum so far
                ;  b = block size
                ; de = starting memory address for block data
                ; hl = remaining bytes to punch

F966 4F         BDATLP: mov     c,a             ;temp save checksum
F967 1A                 ldax    d               ;get memory data
F968 CDA8FB             call    TPOUT           ;...and punch it
F96B 2B                 dcx     h               ;one fewer to punch
```

```
F96C 81                 add     c               ;update checksum

F96D 13                 inx     d               ;Next address
F96E 05                 dcr     b               ;Loop 'til done with block data
F96F C266F9             jnz     BDATLP          ;ends with b=0

                ;a = block checksum
                ;b = 0

F972 CDA8FB             call    TPOUT           ;Punch the block checksum

                ;Continue until all the data has been punched
                ;  b = 0
                ; de = next address to punch
                ; hl = remaining bytes to punch
                ; Test for hl=0, meaning there are more bytes to punch

F975 7D                 mov     a,l
F976 B4                 ora     h
F977 C253F9             jnz     NXTBLK          ;Y: Do another block

                ;Fall into LEADER (with a=0) to punch the trailer

                ;---Subroutine-------------------
                ; Punch a leader
                ; On Entry:
                ;   a = leader character
                ; On Exit:
                ;   b=0
                ;   all other registers preserved
                ;--------------------------------
F97A 063C       LEADER: mvi     b,LDRLEN        ;leader length

F97C CDA8FB     LEADLP: call    TPOUT
F97F 05                 dcr     b
F980 C27CF9             jnz     LEADLP          ;ends with b=0

F983 C9                 ret                     ;done with command

                ;***Command Routine*********************************
                ; CO <SRC> <DST> <BCNT> [<RPT>] (Copy Memory)
                ;
                ; copy <BCNT> bytes of memory from <SRC> to <DST>.
                ; Repeat <RPT> times (FOR EPROM programming). Verify
                ; result when done.
                ; On Entry:
                ;  hl=<SRC>
                ;  de points to <DST>, <BCNT>, <RPT> follow
                ;************************************************
F984 E5         MCOPY:  push    h               ;save source address

F985 CD63FE             call    GETHEX          ;get destination
F988 E5                 push    h               ;save destination

F989 CD63FE             call    GETHEX          ;get byte count
F98C E5                 push    h               ;save byte count

F98D CD45FD             call    CILPRT
F990 436F707969         db      'Copyin','g'+80h

F997 CD20F9             call    FNDHEX          ;get repeat count
F99A 7D                 mov     a,l             ;No value means 1
```

```
F99B CE00               aci     0               ;Carry if no value given

                ;Repeat copy the specified number of times (in a)

F99D C1         MCRLP:  pop     b               ;bc=count
F99E D1                 pop     d               ;de=destination
F99F E1                 pop     h               ;hl=source

F9A0 E5                 push    h               ;save source
F9A1 D5                 push    d               ;save Dest
F9A2 C5                 push    b               ;save count

F9A3 F5                 push    psw             ;save a=repeat count

                ;Loop to copy bc bytes from (hl) to (de)

F9A4 7E         MCLOOP: mov     a,m
F9A5 12                 stax    d
F9A6 23                 inx     h
F9A7 13                 inx     d
F9A8 0B                 dcx     b
F9A9 78                 mov     a,b
F9AA B1                 ora     c
F9AB C2A4F9             jnz     MCLOOP

                ;Repeat the copy as requested by the user

F9AE F1                 pop     psw             ;recover repeat count

F9AF 3D                 dcr     a               ;repeat as requested

                ;Pacifier dot for all but the last pass
                ;(This eliminates the dot for a single-pass copy)
F9B0 47                 mov     b,a             ;temp save repeat count
F9B1 3E2E               mvi     a,PCFIER
F9B3 C494F8             cnz     PRINTA          ;preserves all regs
F9B6 78                 mov     a,b             ;repeat count

F9B7 C29DF9             jnz     MCRLP

F9BA C3C6F9             jmp     VERIFY          ;good copy?

                ;***Command Routine*********************************
                ; VE <SRC> <DST> <BCNT> (Verify Memory)
                ;
                ; Compare <BCNT> bytes of memory from <SRC> to <DST>
                ; and report pass/fail
                ; On Entry:
                ;  hl=<SRC>
                ;  Carry set if none entered
                ;  de points to <DST>, <BCNT> follows
                ;************************************************
F9BD E5         VERCMD: push    h               ;save source address

F9BE CD63FE             call    GETHEX          ;get destination
F9C1 E5                 push    h               ;save destination

F9C2 CD63FE             call    GETHEX          ;get byte count
F9C5 E5                 push    h               ;save byte count

                ;Fall into VERIFY to actually verify

                ;***Subroutine*****************************
```

```
                ; Verify memory. Report errors to console.
                ; On Entry:
                ;   Top of stack=byte count
                ;   next on stack = destination address
                ;   next on stack - source address
                ;   next on stack=return address (TO MAIN)
                ;****************************************
F9C6 C1         VERIFY: pop     b               ;byte count
F9C7 E1                 pop     h               ;hl=destination
F9C8 D1                 pop     d               ;de=source

F9C9 CD45FD             call    CILPRT
F9CC 436865636B         db      'Checkin','g'+80h

                ;Loop to compare memory, reporting mismatches

F9D4 1A         VLOOP:  ldax    d               ;get expected data
F9D5 BE                 cmp     m               ;match?
F9D6 C4B1FB             cnz     MERROR          ;N: error

F9D9 23                 inx     h
F9DA 13                 inx     d
F9DB 0B                 dcx     b
F9DC 78                 mov     a,b
F9DD B1                 ora     c
F9DE C2D4F9             jnz     VLOOP

F9E1 C9                 ret

                ;***Command Routine*********************************
                ; SE <ADR> <BYTE1> [<BYTE2> [<BYTEn>]] (Search)
                ;    <BYTEn> can be either hex byte or 'text string'
                ;    Search for string of bytes, starting at <ADR>
                ; On Entry:
                ;  hl=<ADR>
                ;  Carry set if none entered
                ;  de points to <BYTEs>
                ;************************************************
                SEARCH:

                ;Get search string from input buffer, convert each byte
                ;to binary, and save result in buffer

F9E2 CDEFFF             call    FNDBUF          ;push hl, find RAM buffer

F9E5 E5                 push    h               ;binary string address
F9E6 012700             lxi     b,QUOTE         ;b=byte count, c=QUOTE

                ;-----------------------------------------------
                ;loop to get either a 2-digit hex value or a text
                ;string (in quotes) each pass
                ;-----------------------------------------------
F9E9 CDF6FB     SCHLUP: call    SKIPB           ;returns a=found chr, 0 if none

F9EC B9                 cmp     c               ;is 1st chr a quote?
F9ED CC51FA             cz      SSTRNG          ;y:search for a string
F9F0 C45EFA             cnz     SCHHEX          ;n: search for hex
                                                ;returns carry set if end
F9F3 D2E9F9             jnc     SCHLUP          ;loop to get all input

                ;---------------------------------------
                ;Search RAM for the requested string
                ; b = string length
```

```
                ; top-of-stack = binary string address
                ; next-on-stack = starting search address
                ;---------------------------------------
F9F6 D1                 pop     d               ;binary string address
F9F7 E1                 pop     h               ;search start address

F9F8 78                 mov     a,b             ;anything to search for?
F9F9 B7                 ora     a
F9FA CA67FE             jz      CMDERR          ;error if not

F9FD E5         SLOOP1: push    h               ;search start address
F9FE D5                 push    d               ;binary string address

F9FF 48                 mov     c,b             ;string byte count

                ;Loop through all bytes of the requested string
                ;until either all bytes match or 1st non-matching byte

FA00 7A         SLOOP2: mov     a,d
FA01 BC                 cmp     h               ;don't search our own RAM page
FA02 CA3CFA             jz      NOMTCH

FA05 1A                 ldax    d               ;search string
FA06 BE                 cmp     m               ;current RAM
FA07 C23CFA             jnz     NOMTCH

FA0A 23                 inx     h               ;test next byte
FA0B 13                 inx     d
FA0C 0D                 dcr     c               ;tested all bytes yet?
FA0D C200FA             jnz     SLOOP2

                ;String match found. Print address, ask to continue search

FA10 D1                 pop     d               ;binary string address
FA11 E1                 pop     h               ;search start address

FA12 CD45FD             call    CILPRT
FA15 466F756E64         db      'Found',' '+80h
FA1B C5                 push    b
FA1C CD2AFE             call    PHLCHX          ;print match address, trash bc
FA1F C1                 pop     b

FA20 CD45FD             call    CILPRT
FA23 4D6F726520         db      'More (Y/N)?',' '+80h

FA2F CDE1FB             call    GETKBD          ;user response
FA32 CD94F8             call    PRINTA          ;echo

FA35 F620               ori     ('y'-'Y')       ;make it lowercase
FA37 FE79               cpi     'y'
FA39 C0                 rnz                     ;anything but y ends

FA3A E5                 push    h               ;search start address
FA3B D5                 push    d               ;binary string address

                ;Try again, starting at the next byte

FA3C D1         NOMTCH: pop     d               ;binary string address
FA3D E1                 pop     h               ;search start address

FA3E 23                 inx     h               ;next RAM
FA3F 7C                 mov     a,h
FA40 B5                 ora     l               ;End of memory?
```

```
FA41 C2FDF9            jnz     SLOOP1

FA44 CD45FD            call    CILPRT
FA47 4E6F742066        db      'Not foun','d'+80h

FA50 C9                ret

                ;---Local Subroutine-----------------------
                ;Get a text string from user input at (de),
                ;store string at (hl), bump count in b
                ; On Entry:
                ;   b = byte count
                ;   c=QUOTE
                ;   de points to initial quote
                ; On Exit:
                ;   Z flag set
                ;-----------------------------------------
FA51 13         SSTRNG: inx     d               ;skip over quote

FA52 1A         STLOOP: ldax    d
FA53 B7                 ora     a               ;end quote is not required
FA54 C8                 rz

FA55 13                 inx     d               ;point past this input chr

FA56 B9                 cmp     c               ;end of string?
FA57 C8                 rz

FA58 77                 mov     m,a             ;store a string byte
FA59 23                 inx     h
FA5A 04                 inr     b

FA5B C352FA             jmp     STLOOP          ;get more of this string

                ;---Local Subroutine-------------------------------
                ;Get one hex value from user input at (de), convert
                ;it to binary, store it at (hl), bump count in b
                ; On Exit:
                ;    Carry set if no hex digit found
                ;-------------------------------------------------
FA5E E5         SCHHEX: push    h               ;next string address byte

FA5F CD20F9             call    FNDHEX          ;get a value. hl=0 & carry set if
none

FA62 24                 inr     h               ;no high byte allowed
FA63 25                 dcr     h               ;does not change carry
FA64 C267FE             jnz     CMDERR

FA67 7D                 mov     a,l             ;binary value
FA68 E1                 pop     h               ;next string address byte

FA69 D8                 rc                      ;carry set means end of input

FA6A 77                 mov     m,a             ;store the hex digit
FA6B 23                 inx     h
FA6C 04                 inr     b

FA6D C9                 ret

                ;***Command Routine***********************
                ; TT [0/1] Set Terminal Type
                ;   0 means backspacing works
```

```
                ;   1 means no backspacing (e.g. Teletype)
                ;***************************************
FA6E 7D         SETTT:  mov     a,l             ;get terminal type
FA6F E601               ani     1               ;odd or even :-)

FA71 210000             lxi     h,0             ;find address of TTYPE
FA74 39                 dad     sp              ;...located in RAM
FA75 2E68               mvi     l,TTYPE-RIOCOD+RAMCOD
FA77 77                 mov     m,a             ;remember terminal type
FA78 C9                 ret

                ;***Command Routine********************
                ; EX [<ADR> [<OPT>]] (execute)
                ;
                ; JUMP to <ADR>. If <OPT>=1 the execute
                ; an "IN FF" first, to disable this PROM
                ; On Entry:
                ;  hl = address, default to 0
                ;  de points to <OPT>
                ;  Carry set if none entered
                ;  TOP-of-stack has MAIN address
                ;***************************************
FA79 E5         EXEC:   push    h               ;remember exec address

FA7A CD20F9             call    FNDHEX          ;get l=<OPT>
FA7D 2D                 dcr     l               ;anything but 1
FA7E C0                 rnz                     ;..just executes at <ADR>


                ; Fall into EXECDP

                ;***Exit*******************************
                ;Execite "IN FF" and then jump to code
                ;(This disables PROM)
                ; On Entry:
                ;   l = 0
                ;   execution address is on stack
                ;***************************************
FA7F 65         EXECDP: mov     h,l             ;make sure h is clear too

FA80 1EC9               mvi     e,RET           ;RET opcode, <don't care>
FA82 D5                 push    d               ;..onto stack

FA83 11DBFF             lxi     d,0FFDBh        ;IN FF opcode
FA86 D5                 push    d               ;..onto stack

FA87 39                 dad     sp              ;point hl to our code

FA88 D1                 pop     d               ;point sp to <ADR>
FA89 D1                 pop     d
FA8A E9                 pchl                    ;execute: IN  FF
                                                ;         RET

                ;***Command Routine***********************************
                ; DU [<ADR>] [<BCNT>] (dump memory to console)
                ;
                ; Print <BCNT> bytes of memory contents from <ADR> on
                ; the console in hex. If no count is specified, then
                ; then print the contents of all memory, 10000h bytes.
                ; Pause with the space bar, abort with control-C.
                ; On Entry:
                ;    hl=<ADR>
                ;    de points to <BCNT>, if any
```

```
                ;*****************************************************
FA8B E5         DUMP:   push    h               ;save 1st address

FA8C CD20F9             call    FNDHEX          ;get hl=byte count or 0
FA8F 7D                 mov     a,l             ;low byte
FA90 CE00               aci     0               ;default to 1
FA92 6F                 mov     l,a

FA93 EB                 xchg                    ;de has byte count
FA94 E1                 pop     h               ;recover start address

                ;Print the address at the beginning of each line

FA95 CD54FE     DLINE:  call    PHLADR          ;print hl as an address
                                                ;Sets b=0, trashes c

                ;Print 16 bytes of hex data. separated by spaces

FA98 E5                 push    h               ;save for ASCII dump
FA99 D5                 push    d

FA9A 7E         DLOOP:  mov     a,m             ;get the character
FA9B CD38FE             call    PAHEX           ;TO console in hex (b=0)

FA9E CD4AFD             call    ILPRNT          ;print a space
FAA1 A0                 db      ' '+80h

FAA2 23                 inx     h               ;next address

FAA3 1B                 dcx     d               ;all done?
FAA4 7A                 mov     a,d
FAA5 B3                 ora     e
FAA6 CAAFFA             jz      DLDONE          ;Y: done with command

FAA9 3E0F               mvi     a,0Fh           ;new line every XXX0 hex
FAAB A5                 ana     l

FAAC C29AFA             jnz     DLOOP           ;not zero if more for this line

FAAF D1         DLDONE: pop     d               ;recover count and address
FAB0 E1                 pop     h

                ;Print up to 16 ASCII characters, or '.' if unprintable

FAB1 CD4AFD             call    ILPRNT          ;pretty space
FAB4 A0                 db      ' '+80h

FAB5 7E         ADLOOP: mov     a,m             ;get the character
FAB6 3C                 inr     a               ;del (7F) is also nonprinting
FAB7 E67F               ani     7fh             ;clear parity
FAB9 FE21               cpi     ' '+1           ;everything below space
FABB D2C0FA             jnc     PRNTBL          ;..is nonprinting

FABE 3E2F               mvi     a,'.'+1         ;dot for non-printing

FAC0 3D         PRNTBL: dcr     a               ;undo inc

FAC1 CD94F8             call    PRINTA          ;Print ASCII or dot

FAC4 1B                 dcx     d               ;all done?
FAC5 7A                 mov     a,d
FAC6 B3                 ora     e
FAC7 C8                 rz                      ;done with command
```

```
FAC8 23                 inx     h               ;next line?
FAC9 3E0F               mvi     a,0Fh           ;new line every XXX0 hex
FACB A5                 ana     l

FACC C2B5FA             jnz     ADLOOP          ;not zero if more for this line

                ;Give the user a chance to break in at the end of each line

FACF CD61FD             call    CHKKBD          ;see if a character waiting
                                                ;..and abort if it's control-C
FAD2 FE20               cpi     PAUKEY          ;is it a pause?
FAD4 CCE1FB             cz      GETKBD          ;Y: go wait for another chr
                                                ;..abort if control-C

FAD7 C395FA             jmp     DLINE           ;next line

                ;***Command Routine*****************************************
                ; EN [<ADR>] (enter data into memory)
                ;
                ; Get hex values from the keyboard and enter them
                ; sequentially into memory, starting at <ADR>. a blank
                ; line ends the routine and returns control to the
                ; command Mode. values may be separated by spaces or CR'S.
                ; Print the current address at the beginning of each line.
                ; On Entry:
                ;  hl = address, defaulting to 0
                ;  Carry set if none entered
                ;*********************************************************
FADA CD54FE     ENTER:  call    PHLADR          ;print hl as an address

FADD CDE8FB             call    GETLIN          ;get a line of user input
FAE0 C8                 rz                      ;z=blank line terminates

                ;Get hex data from the user input line and write it to memory

FAE1 E5         ENLOOP: push    h               ;save memory address
FAE2 CD20F9             call    FNDHEX          ;Get/convert value

FAE5 7D                 mov     a,l             ;get low byte as converted
FAE6 E1                 pop     h               ;recover memory address

FAE7 77                 mov     m,a             ;put in the value
FAE8 23                 inx     h               ;next address

FAE9 CDF6FB             call    SKIPB           ;Scan to next input value
FAEC C2E1FA             jnz     ENLOOP          ;not end of line: continue

FAEF C3DAFA             jmp     ENTER           ;end of line: start new line

                ;***Command Routine*********************************
                ; FI [<VAL> [<ADR> [<BCNT>]]] (fill memory)
                ;
                ; Fill <BCNT> bytes of memory with <VAL> from <ADR>.
                ; if <VAL> is not provided, then fill the specified
                ; range with 00. <ADR> defaults to 0. If <BCNT> is not
                ; provided, fill until we reach AMON's RAM page.
                ; On Entry:
                ;  hl=<ADR>
                ;  Carry set if none entered
                ;  de points to <BCNT>, <VAL> follows, if any
                ;***************************************************
FAF2 4D         FILMEM: mov     c,l             ;fill data in c
```

```
FAF3 CD20F9            call    FNDHEX          ;get address, default 0
FAF6 E5                push    h               ;save address

FAF7 CD20F9            call    FNDHEX          ;get hl=byte count
FAFA EB                xchg                    ;de has byte count

FAFB 210000            lxi     h,0             ;get stack address
FAFE 39                dad     sp              ;to find RAM page
FAFF 44                mov     b,h             ;b remembers RAM page

FB00 E1                pop     h               ;hl has start address

                ;Loop to fill memory, quitting if RAM page

FB01 7C         FMLOOP: mov     a,h
FB02 B8                cmp     b               ;Filling RAM page?
FB03 C8                rz                      ;y: done

FB04 71                mov     m,c
FB05 23                inx     h

FB06 1B                dcx     d               ;done yet?
FB07 7A                mov     a,d
FB08 B3                ora     e
FB09 C201FB            jnz     FMLOOP

FB0C C9                ret

                ;***Command Routine*********************************
                ; TE [<EXCHR>] (simple Terminal Mode)
                ;
                ; Send all console keyboard data to Transfer Port,
                ; and send send all Transfer Port data to the console.
                ; If the Transfer Port is the console, then just echo
                ; the keyboard to the console.
                ; <EXCHR> on the keyboard to exit
                ;  (defaults to DTEXIT)
                ;***********************************************
FB0D CD45FD     TERMNL: call    CILPRT          ;announce exit character
FB10 457869743A         db      'Exit: ','^'+80h

FB17 CDF6FB             call    SKIPB           ;get optional exit character
FB1A C21FFB             jnz     TMNL1           ;Got an exit value in a
FB1D 3E03               mvi     a,DTEXIT        ;default abort

                ;Convert exit character to uppercase, non-control, and
                ;print exit character message

FB1F E61F       TMNL1:  ani     1Fh             ;make it a control chr
FB21 6F                 mov     l,a             ;remember exit character

FB22 F640               ori     'C'-CTRLC       ;make it printable
FB24 CD94F8             call    PRINTA

FB27 CD45FD             call    CILPRT          ;CR,LF,LF to be pretty
FB2A 8A                 db      LF+80h

                ;Be a terminal until we get an exit character=l.
                ;Just echo if Transfer Port = console

FB2B CD85F8     TLOOP:  call    KSTAT           ;anything typed?
FB2E C458FD             cnz     KDATA           ;Y:get the keyboard data
```

```
FB31 CA39FB             jz      TMNL2

FB34 BD                 cmp     l               ;exit character?
FB35 C8                 rz                      ;Y: done

FB36 CDA8FB             call    TPOUT           ;KBD data to Transfer Port

FB39 CD7AFD     TMNL2:  call    TESTTP          ;Transfer Port = console?
                                                ;Z set if so

FB3C C46FFD             cnz     TPISTA          ;Any Transfer Port data?
                                                ;NZ if so

FB3F C473FD             cnz     TPIN            ;get Transfer Port data
                                                ;always returns W/ nz
FB42 C494F8             cnz     PRINTA          ;and send it to console
FB45 C32BFB             jmp     TLOOP

                ;***Command Routine*****************************************
                ; HD <ADR> <BCNT> [<OFST>] (Intel hex dump to transfer port)
                ;
                ; Dump the specified memory range to the Transfer
                ; Port as an Intel hex file
                ; On Entry:
                ;    hl=ADR
                ;    de points to subsequent parameters
                ;***********************************************************
FB48 E5         HEXDMP: push    h               ;save start address

FB49 CD63FE             call    GETHEX          ;get hl=byte count
FB4C E5                 push    h               ;save byte count

FB4D CD20F9             call    FNDHEX          ;get hl=offset, allow default
FB50 44                 mov     b,h             ;bc=offset
FB51 4D                 mov     c,l

FB52 E1                 pop     h               ;hl= byte count
FB53 D1                 pop     d               ;de= start address
FB54 C5                 push    b               ;address offset onto stack

                ;Loop to send requested data in HRLEN-byte records

                ;send record-start

FB55 D5         HDLINE: push    d               ;print CRLF
FB56 CDA0FB             call    TPCRLF
FB59 D1                 pop     d

FB5A 3E3A               mvi     a,':'
FB5C CDA8FB             call    TPOUT

                ;compute this record byte count

FB5F 0610               mvi     b,HRLEN         ;default bytes/line

FB61 7D                 mov     a,l             ;short last line?
FB62 90                 sub     b               ;normal bytes/line
FB63 7C                 mov     a,h
FB64 DE00               sbi     0
FB66 D26AFB             jnc     HDLIN1          ;N: full line

FB69 45                 mov     b,l             ;Y:short line
                HDLIN1:
```

```
                ;If byte count is 0 then go finish EOF record

FB6A 78                 mov     a,b
FB6B B7                 ora     a
FB6C CA95FB             jz      HDEOF

                ;Send record byte count=a to Transfer Port (b<>0)

FB6F CD34FE             call    PAHEXC          ;send byte count

FB72 48                 mov     c,b             ;initiate checksum

                ;Compute the address by adding the RAM address to the
                ;address offset. Send the address at the beginning of
                ;each address, computing checksum in c (b<>0)

FB73 E3                 xthl                    ;hl=address offset
                                                ;remaining byte count on stack
FB74 E5                 push    h               ;save address offset

FB75 19                 dad     d               ;compute address with offset
FB76 CD2CFE             call    PHLHEX          ;send address with offset

FB79 E1                 pop     h               ;recover address offset
FB7A E3                 xthl                    ;offset on stack,
                                                ;remaining byte count to hl

                ;Send the record type (00)

FB7B AF                 xra     a
FB7C CD34FE             call    PAHEXC

                ;Send b bytes of hex data on each line, computing
                ;the checksum in c

FB7F 1A         HDLOOP: ldax    d               ;get the character
FB80 CD34FE             call    PAHEXC          ;send to Transfer Port
                                                ;(b<>0)
FB83 2B                 dcx     h
FB84 13                 inx     d
FB85 05                 dcr     b               ;next
FB86 C27FFB             jnz     HDLOOP

                ;Send the checksum (with b<>0)

FB89 AF                 xra     a
FB8A 91                 sub     c
FB8B 04                 inr     b               ;b<>0 means Transfer Port
FB8C CD34FE             call    PAHEXC

                ;Give the user a chance to break in at the end of each line

FB8F CD61FD             call    CHKKBD          ;see if a character waiting
                                                ;..and abort if it's control-C

                ;Next record

FB92 C355FB             jmp     HDLINE          ;next record

                ;-----------------------------------------------
                ; Finish end-OF-file Intel hex record
                ; On Entry:
```

```
                ;   The CR LF and colon have already been sent
                ;   The address offset is still on the stack
                ;-----------------------------------------
FB95 C1         HDEOF:  pop     b               ;chuck address offset
FB96 0605               mvi     b,5             ;5 bytes for EOF

FB98 AF         HDELP:  xra     a
FB99 CD38FE             call    PAHEX           ;b<>0 for Transfer Port
FB9C 05                 dcr     b
FB9D C298FB             jnz     HDELP

                ;Fall into TPCRLF

                ;===============
                ;= subroutines =
                ;===============

                ;***Subroutine******************
                ; Send CRLF to the transfer port
                ; Trashes de
                ;*******************************
FBA0 110D0A     TPCRLF: lxi     d,LF*256+CR     ;terminating CRLF

                ;Fall into TPOED

                ;***Subroutine**********************
                ; Send e then d to the transfer port
                ; On Exit:
                ;   a=d
                ;***********************************
FBA3 7B         TPOED:  mov     a,e
FBA4 CDA8FB             call    TPOUT
FBA7 7A                 mov     a,d

                ;Fall into TPOUT

                ;***Subroutine**********************
                ; Send a to the Transfer Port
                ; On Entry:
                ;   a = data to send
                ;   SP points into the RAM page
                ;   Transfer Port is already set up
                ; all registers preserved, Z cleared
                ;***********************************
FBA8 E5         TPOUT:  push    h
FBA9 210000             lxi     h,0                     ;find RAM page
FBAC 39                 dad     SP
FBAD 2E3B               mvi     l,RTPOUT-RIOCOD+RAMCOD
FBAF E3                 xthl                            ;restore hl, put
                                                        ;..address on stack
FBB0 C9                 ret                             ;go to RTPOUT with a

                ;***Subroutine************************
                ; Print memory error details, and give
                ; user a chance to pause or abort
                ; On Entry:
                ;   a=Expected (Source) data
                ;  hl=Destination Address
                ;(hl)=Found data
                ; trashes psw
                ;*************************************
FBB1 C5         MERROR: push    b
FBB2 F5                 push    psw
```

```
FBB3 CD45FD            call    CILPRT
FBB6 3FBA              db      '?',':'+80H
FBB8 CD2AFE            call    PHLCHX          ;Print hl on console, trash c
                                               ;set b=0

FBBB CD4AFD            CALL    ILPRNT
FBBE 2045787065        db      ' Expected',' '+80H

FBC8 F1                pop     psw
FBC9 CD38FE            call    PAHEX

FBCC CD4AFD            call    ILPRNT
FBCF 2C20726561        db      ', read',' '+80H
FBD6 7E                mov     a,m
FBD7 CD38FE            call    PAHEX

FBDA C1                pop     b

FBDB CD61FD            call    CHKKBD          ;Abort or pause?

FBDE FE20              cpi     PAUKEY          ;Pause?
FBE0 C0                rnz

                ;Fall into GETKBD and wait for any key to end pause

                ;***Subroutine*********************************
                ; Get a keyboard character, abort if control-C
                ; On Exit:
                ;    a=keyboard character, Z cleared
                ;***********************************************
FBE1 CD61FD     GETKBD: call    CHKKBD          ;get KBD character, test for ^C
FBE4 CAE1FB             jz      GETKBD          ;wait for character

FBE7 C9                 ret

                ;***Subroutine*************************************************
                ; Read a command line from the keyboard, echoing and saving
                ; it in the input line buffer
                ;
                ; CR input endS the sequence. the CR is not saved in the
                ; input line buffer. instead, the line is terminated with 0.
                ;
                ; On Exit:
                ;   complete command line is in the input line buffer
                ;   de=address of the first non-blank character on the line
                ;   a = first non-blank value found
                ;   Z set if nothing but blanks found
                ;***************************************************************
FBE8 CDEFFF     GETLIN: call    FNDBUF          ;find buffer, push hl

FBEB E5                 push    h               ;save input line buffer'S
                                                ;..start address

                ;Get & echo characters, stashing them in the input line buffer at
                ;hl, until a CR is encountered

FBEC CD07FD     GLLOOP: call    LBCHR           ;get kbd chr into line buffer
FBEF D2ECFB             jnc     GLLOOP          ;Carry means CR detected

FBF2 3600               mvi     m,0             ; overwrite CR with null

FBF4 D1                 pop     d               ;input line buffer address
```

```
FBF5 E1                 pop     h               ;Restore

                ;Fall into SKIPB to skip initial spaces

                ;***Subroutine*****************************
                ; Scan past blank positions looking
                ; for the first non-blank character
                ;
                ; On Entry:
                ;   de=address withIN the input line buffer
                ; On Exit:
                ;   a=0 and Z set if none found
                ;   a=character value if found
                ;*****************************************
FBF6 1A         SKIPB:  ldax    d               ;get next character
FBF7 B7                 ora     a               ;terminating null?
FBF8 C8                 rz

FBF9 FE20               cpi     ' '
FBFB C0                 rnz                     ;we're past them

FBFC 13                 inx     d               ;next scan address
FBFD C3F6FB             jmp     SKIPB           ;keep skipping


                ;================================================================
                ; Check for assembly problem on Hole 0: The above code must not
                ; overrun the next section
                ;================================================================
FC00 =          H0END   equ     $

                 if ((HDBADR/256) - ((H0END-1)/256+1))/256
                        ERROR: HDBL is overwriting prior code
                 endif

                ;================================================================
                ; Hard Disk Boot Loader Subsystem(HDBL)
                ; By Martin Eberhard
                ;
                ; The standard 88-HDSK system uses a Pertec D3422 disk drive,
                ; which contains 2 platters - one is in a removable cartridge,
                ; the other is a fixed platter. However, The 88-HDSK controller
                ; can actually support up to 4 platters, supporting the Pertec
                ; D3462 disk drive, which has one removable platter, and 3
                ; fixed platters.
                ;
                ; There are 24 256-byte sectors per track, and these are
                ; numbered 0 through 23 on each track. Each platter has 2
                ; sides, numbered 0 and 1. Data on each platter is organized as
                ; a sequence of Disk Pages, where each Page is one sector.
                ; Pages are numbered sequentially starting at 0 (on track 0,
                ; side 0), through the 24 sectors on track 0, side 0, and then
                ; on to track 0, side 1, where sector 0 is page 24.  Page 47 is
                ; the first sector on track 1, side 0, and page numbering
                ; continues this way through all the tracks.
                ;
                ; Page 0 (which is track 0, side 0, sector 0) is the Pack
                ; Descriptor Page, containing various information about the
                ; particular disk platter. Bytes 40-43 of this Page are the
                ; "Opsys Pointers." Bytes 40 & 41 are the Page number of the
                ; starting boot Page, Bytes 42 & 43 are the number of Pages to
                ; load during boot. HDBL assumes that the boot file is to be
                ; loaded into memory starting at address 0000, and executed
```

```
                ; there.
                ;
                ; During loading, the INTE (Interrupt Enabled) LED on the front
                ; panel will be off. Any error during loading will cause the
                ; INTE LED to light and a "LOAD ERR" message to be printed
                ; on the Terminal. The error code is stored in memory at
                ; address 0. HDBL will then hwng in a loop until Reset.
                ;
                ; This code is written to assemble with ASM by Digital Research
                ;
                ;=============================================================

                ; 88-HDSK ports (The interface board is actually an 88-4PIO.)

00A0 =          CREADY  equ     0A0h    ;IN: Ctlr ready for command (bit7)
00A1 =          CSTAT   equ     0A1h    ;IN: error flags, reset CREADY
00A2 =          ACSTA   equ     0A2h    ;IN: Command Ack (bit 7)
00A3 =          ACMD    equ     0A3h    ;IN: reset Command Ack
                                        ;OUT: Command high byte/initiate
00A4 =          CDSTA   equ     0A4h    ;IN: data/stat availablr at CDATA
00A5 =          CDATA   equ     0A5h    ;IN: Disk data or status from Ctlr
00A6 =          ADSTA   equ     0A6h    ;IN: ADATA Port Available (bit 7)
00A7 =          ADATA   equ     0A7h    ;OUT: Command low byte

                ; 88-HDSK ACMD:ADATA Commands

0000 =          CSEEK   equ     00h     ;Bits 15:12 = 0000b
                                        ;Bits 11:10 = Unit #
                                        ;Bits  9:0  = Cylinder #

0030 =          CRDSEC  equ     30h     ;Bits 15:12 = 0011b
                                        ;Bits 11:10 = Unit #
                                        ;Bits  9:8  = Buffer #
                                        ;Bit   7:6  = Platter #
                                        ;Bits    5  = Side #
                                        ;Bits  4:0  = Sector #

0020 =          CSIDE    equ      020h    ;Side select for CRDSEC
00C0 =          CFPLTR   equ      0C0h    ;platter mask for CRDSEC
000C =          CUNIT    equ      00Ch    ;Unit mask for CSEEK & CRDSEC

0050 =          CRDBUF  equ     50h     ;Bits 15:12 = 0101b
                                        ;Bits 11:10 = not used
                                        ;Bits  9:8  = buffer #
                                        ;Bits  7:0  = # bytes to transfer
                                        ;(00 means 256)

                ; 88-HDSK CSTAT error bits

0001 =          ERDNR   equ     01h     ;drive not ready
0002 =          ERBADS  equ     02h     ;illegal sector
0004 =          ERSCRC  equ     04h     ;CRC error during sector read
0008 =          ERHCRC  equ     08h     ;CRC error during header read
0010 =          ERSWRG  equ     10h     ;header has wrong sector
0020 =          ERCWRG  equ     20h     ;header has wrong cylinder
0040 =          ERHWRG  equ     40h     ;header has wrong head
0080 =          WPROT   equ     80h     ;Write Protect
007F =          ERMASK  equ     7Fh     ;all the actual error bits

                ; 88-HDSK Constants

0028 =          OSOFF   equ     40      ;Page 0 offset to opsys pointers
0018 =          HDSPT   equ     24      ;Sectors per track
```

```
0000 =          DBUFR   equ     0       ;Default controller buffer: 0-3
                                        ;Code gets longer if <>0

                ;===============================================================
                ; Start of HDBL Subsystem
FC00                    org     HDBADR
                ;===============================================================

                ;===============================================================
                ; Entry here to execute HDBL directly, to boot from a hard disk.
                ; This is the same address where my HDBL starts.
                ;===============================================================
FC00 0106FC     HDBL:   lxi     b,HDBRET        ;return address
FC03 C303F8             jmp     INIT            ;go find a real stack
                                                ;and initialize ACIAs

FC06 2E00       HDBRET: mvi     l,0             ;boot from platter 0

                ;Fall into HBOOT

                ;***Command Routine******
                ; HB  Boot from hard disk
                ; On Entry:
                ;   l<0> = platter
                ;************************
FC08 7D         HBOOT:  mov     a,l
FC09 E601               ani     1               ;just the lsb
FC0B 0F                 rrc                     ;Platter goes in bits <7:6>
FC0C 0F                 rrc                     ;..which is CFPLTR
FC0D 47                 mov     b,a             ;b<7:6>=platter bits

                ;---------------------------------------
                ; Initialize 88-HDSK interface board
                ; (Actually ports 0 and 1 of an 88-4PIO)
                ; On Exit:
                ;   b = platter in bits <7:6>
                ;  de = 0
                ;---------------------------------------
FC0E AF                 xra     a
FC0F 57                 mov     d,a             ;set load initial page
FC10 5F                 mov     e,a

FC11 D3A0               out     0A0h            ;Select port 0Ah DDR
FC13 D3A2               out     0A2h            ;Select port 0Bh DDR
FC15 D3A4               out     0A4h            ;Select port 1Ah DDR
FC17 D3A6               out     0A6h            ;Select port 1Bh DDR
FC19 D3A1               out     0A1h            ;Port 0Ah is an input port
FC1B D3A5               out     0A5h            ;Port 1Ah is an input port

FC1D 2F                 cma
FC1E D3A3               out     0A3h            ;Port 0Bh is an output port
FC20 D3A7               out     0A7h            ;Port 1Bh is an output port

FC22 3E2C               mvi     a,2Ch           ;set up input port handshakes
FC24 D3A0               out     0A0h
FC26 D3A4               out     0A4h
FC28 D3A6               out     0a6h            ;output port 1Bh handshakes

FC2A 3E24               mvi     a,24h           ;set up port 0Bh handshakes
FC2C D3A2               out     0A2h

FC2E DBA1               in      CSTAT           ;clear Controller Ready bit
```

```
                        ;-------------------------------------------------
                        ; Read the Pack Descriptor Page (Disk Page 0)
                        ; to get the Opsys Pointers:
                        ;   Bytes 41:40 = Initial Disk Page number
                        ;   Bytes 43:42 = Disk Page count (Byte 43=MSB=0)
                        ; On Entry:
                        ;   b = platter in bits <7:6>
                        ;  de = 0 = load address
                        ;-------------------------------------------------
FC30 062B                       mvi     b,OSOFF+3       ;byte count to end of pointers
FC32 CD53FC                     call    GETPAG          ;Seek, read page hl into buffer
                                                        ;set up to read b buffer bytes

FC35 D5                         push    d               ;execution address on stack

                        ; Read from the controller buffer and discard everything until
                        ; we get to the opsys pointers. Load the opsys pointers into
                        ; c & hl. Note: no testing any handshake here - just assume
                        ; the controller can keep up. (The controller can send a data
                        ; byte every 2.5 uS.) This only reads the low byte of the
                        ; page count, since the high byte must be 0 anyway.

FC36 DBA5               PTRLUP: in      CDATA           ;read byte from controller

FC38 6C                         mov     l,h             ;shift everybody over...
FC39 61                         mov     h,c
FC3A 4F                         mov     c,a             ;...and put it away

FC3B 05                         dcr     b
FC3C C236FC                     jnz     PTRLUP

                        ;-------------------------------------------------
                        ; Read c Pages from disk, starting at Page hl, into
                        ; memory starting at the address on the stack
                        ; On Entry:
                        ;   b = platter in bits <7:6>
                        ;   c = page count
                        ;  de = LDADDR (e=0)
                        ;  hl = initial Disk page number
                        ;-------------------------------------------------
FC3F CD53FC             PAGELP: call    GETPAG          ;Seek, read page hl into buffer
                                                        ;set up to read b buffer bytes
                                                        ;b=0 here always.

                        ; Load 256 bytes of buffer data into memory at de (b=0 here)
                        ; Note: no testing any handshake here - just assume the
                        ; controller can keep up. (The controller can send a data byte
                        ; every 2.5 uS.)

FC42 DBA5               BYTELP: in      CDATA           ;get a data byte
FC44 12                         stax    d               ;write it to RAM
FC45 1C                         inr     e               ;write entire page
FC46 C242FC                     jnz     BYTELP          ;until done

                        ; Next Disk Page

FC49 14                         inr     d               ;next RAM page
FC4A 23                         inx     h               ;Next Disk Page
FC4B 0D                         dcr     c               ;bump Disk Page count
FC4C C23FFC                     jnz     PAGELP

                        ;-------------------------------------------------
                        ; Go execute loaded code, at the address on the stack
```

```
                ;-----------------------------------------------
FC4F 69                 mov     l,c             ;l=0
FC50 C37FFA             jmp     EXECDP          ;disable PROM, execute loaded code

                ;***Subroutine*****************************************
                ; Seek and read disk Page hl into 88-HDSK buffer 0
                ; On Entry:
                ;   b = platter in bits <7:6>
                ; On Exit:
                ;   a,flags trashed, all others preserved
                ;   Controller has specified sector data in its buffer
                ;***************************************************
FC53 E5         GETPAG: push    h               ;Save requested Page
FC54 D5                 push    d               ;Save regs
FC55 C5                 push    b               ;save byte count

                ;-------------------------------------------------------------
                ; Compute cylinder and sectorX2  from Disk Page number in hl
                ;  hl := hl / (2*HDSPT) (Quotient=cylinder)
                ;   h := hl MOD (2*HDSPT) (Remainder=sectorx2)
                ; This is fast only if the cylinder number is low. MITS
                ; usually put the boot image starting at cylinder 0, side 1,
                ; so this will be faster and shorter than the 'fast'
                ; division of previous HDBL rev. This will become slower
                ; if the boot image is above cylinder 20 or so. But we will
                ; always miss the next sector anyway, so each sector will
                ; require a full disk rev (25 mS), plenty of time
                ;-------------------------------------------------------------
FC56 01D0FF             lxi     b,-2*HDSPT
FC59 50                 mov     d,b             ;de=FFFF=-1
FC5A 58                 mov     e,b             ;since loop goes 1 extra

FC5B 13         DIV1:   inx     d               ;compute quotient=cylinder
FC5C 09                 dad     b               ;hl gets remainder
FC5D DA5BFC             jc      DIV1

FC60 7D                 mov     a,l             ;fix remainder, since
FC61 91                 sub     c               ;..loop went 1 extra

FC62 EB                 xchg                    ;cylinder number to hl

                ;-----------------------------------------------
                ; Compute Sector & Side
                ; If sectorX2 > sectors/track then set CSIDE
                ; bit, and reduce sector number by sectors/track
                ;   hl= Quotient (cylinder)
                ;   a = Remainder (sectorX2, either for head 0 or 1)
                ;-----------------------------------------------
FC63 FE18               cpi     HDSPT           ;past end of side 0?
FC65 DA6AFC             jc      SIDEOK          ;N: sector number is good

FC68 C608               adi     CSIDE-HDSPT     ;Compute sector mod HDSPT,
                                                ;..and set side 1 bit

FC6A B0         SIDEOK: ora     b               ;combine platter bits
FC6B 4F                 mov     c,a             ;save sector # with side

                ;-------------------------------------------
                ; Seek Cylinder
                ;   b = platter in bits <7:6>
                ;   c = sector number, with side and platter
                ;   hl = cylinder number<9:0>
                ;-------------------------------------------
```

```
                 if CSEEK+DBUFR                 ;these are actually 00
                        mov     a,h             ;h<1:0>=cylinder<9:8>
                        ori     CSEEK+DBUFR     ;combine with SEEK cmd
                        mov     h,a
                 endif

FC6C CD82FC             call    HDCMD           ;hl=SEEK command with cyl #

                ;-------------------------------------------------------
                ; Read Sector from current track into controller's buffer 0
                ;   c<7:6> = platter
                ;     c<5> = side
                ;   c<4:0> = sector number
                ;-------------------------------------------------------
FC6F 79                 mov     a,c
FC70 CD83FC             call    HDCMDA          ;low command byte is in a

                ;-----------------------------------------
                ; Issue CRDBUF command to kick off read of 256
                ; bytes from the controller's buffer
                ; Note: this assumes the controller is ready.
                ; (and it is, because HDCMD left it that way.)
                ;-----------------------------------------
FC73 DBA5               in      CDATA           ;reset CDA in CDSTA
FC75 DBA3               in      ACMD            ;clear CMDACK in ACSTA

FC77 AF                 xra     a               ;256 bytes to load
FC78 D3A7               out     ADATA           ;..to controller

FC7A 3E50               mvi     a,CRDBUF+DBUFR  ;issue Read Buffer command
FC7C D3A3               out     ACMD            ;..to controller

FC7E C1                 pop     b               ;(10)
FC7F D1                 pop     d               ;(10)
FC80 E1                 pop     h               ;(10) 15 uS total from 'out'

                ; The 8x300 is ready to transmit data in 8 uS. This code takes
                ; 40 cycles (including the 'ret'), or 20 uS min to get around
                ; to reading the data - so there is no need to wait on CDSTA

                 if FALSE
                DATAWT: in      CDSTA           ;Wait for data port to be ready
                        rlc                     ;msb=CDA
                        jnc     DATAWT
                 endif

                ;---------------------------------
                ; Controller is ready to transfer
                ; 256 bytes of data from its buffer
                ;---------------------------------
FC81 C9                 ret                     ;(10)done with GETPAG

                ;***Subroutine************************************************
                ; Issue a disk command, and then wait for the controller
                ; to complete it
                ;
                ; Note: this just assumes the controller is ready, which is OK
                ; since the last command was either a seek (where HDCMD waited
                ; for the controller to become ready) or it was a CRDBUF, which
                ; ended with all bytes transferred - and the controller becomes
                ; ready very soon (1.5 uS) after the last byte is transferred.
                ; On Entry at HDCMD:
                ;   hl = complete command
```

```
                        ; On Entry at HDCMDA:
                        ;   a=low byte of command
                        ;   h=high byte of command
                        ; On Exit:
                        ;   a,flags trashed, all others preserved.
                        ;   The command is completed and the controller is ready.
                        ;   Any errors will terminate the load, and print an error
                        ;   message on the Terminal
                        ;*************************************************************
FC82 7D                 HDCMD:  mov     a,l             ;low byte of command

FC83 D3A7               HDCMDA: out     ADATA           ;..to data port

FC85 DBA1                       in      CSTAT           ;reset CRDY flag just in case
FC87 DBA3                       in      ACMD            ;clear CMDACK in ACSTA

FC89 7C                         mov     a,h             ;command high byte
FC8A D3A3                       out     ACMD            ;issue command

FC8C DBA0               HDWAIT: in      CREADY          ;Is the controller done?
FC8E 07                         rlc                     ;look at msb=CRDY
FC8F D28CFC                     jnc     HDWAIT          ;N: keep waiting

FC92 DBA1                       in      CSTAT           ;reset CRDY flag
FC94 E67F                       ani     ERMASK          ;and get A=error code
FC96 C8                         rz                      ;No errors: happy return

                        ;       Fall into error exit

                        ;***Error Exit*********************************
                        ; Report a load error and go to AMON's main loop
                        ; On Entry:
                        ;   a = error flag bits
                        ;  hl = disk command
                        ;*************************************************
FC97 CD32FE                     call    PCAHEX          ;print error code in hex
FC9A 37                         stc                     ;it's an error
FC9B C3E4FF                     jmp     LDDONE          ;finish the error message

                        ;***Command Routine*****************************************
                        ; HL [<OFST>] (Intel hex load from transfer port)
                        ;
                        ; Load an Intel hex file from the Transfer Port into memory
                        ; at the addresses specified in the hex file, with optional
                        ; address offset <OFST>. done when any record with 0 data
                        ; bytes is encountered, or if control-C is typed.
                        ;
                        ; On Entry:
                        ;  hl= address offset from user (defaults to 0)
                        ;
                        ; register usage during hex load:
                        ;   b: Scratch
                        ;   c: record byte counter
                        ;   d: record checksum
                        ;   e: record byte count for EOF test
                        ;   hl: memory address
                        ;   Top of stack: address offset
                        ;   Next on stack: record count
                        ;*************************************************************
FC9E E5                 HEXLOD: push    h               ;address offset onto stack

FC9F 210000                     lxi     h,0             ;initialize record count
FCA2 E5                         push    h               ;onto stack too
```

```
                ;Eat all characters until we get record-start colon

FCA3 CD84FD     GETCOL: call    GETTPD
FCA6 D63A               sui     ':'
FCA8 C2A3FC             jnz     GETCOL

FCAB 57                 mov     d,a             ;d=0:Init checksum

FCAC CD4AFD             call    ILPRNT          ;pacifier per record
FCAF AE                 db      PCFIER+80h

                ;Restart checksum, then get 4-byte record header: (a=0 here)
                ; c gets 1st byte = data byte count
                ; h gets 2nd byte = address high byte
                ; l gets 3rd byte = address low byte
                ; b gets 4th byte = record type (ignored)

FCB0 1E04               mvi     e,4             ;get 4 header bytes

                ;Shift in the four header bytes: c <- h <- l <- b

FCB2 4C         HEDRLP: mov     c,h             ;c=byte 1: byte count
FCB3 65                 mov     h,l             ;h=byte 2: address MSB
FCB4 68                 mov     l,b             ;l=byte 3: address LSB
FCB5 CD99FD             call    TPBYTE          ;get header byte, do checksum
FCB8 1D                 dcr     e
FCB9 C2B2FC             jnz     HEDRLP

                ;Offset the address by the value on top of the stack
                ;and bump the record count. a=checksum so far here

FCBC D1                 pop     d               ;get offset
FCBD 19                 dad     d               ;offset the address in hl

FCBE E3                 xthl                    ;bump record count
FCBF 23                 inx     h
FCC0 E3                 xthl                    ;..leaving it on the stack

FCC1 D5                 push    d               ;save offset

FCC2 57                 mov     d,a             ;d=checksum so far
FCC3 59                 mov     e,c             ;remember count for EOF test

                ; c = e = record byte count
                ; hl = RAM address for this record=record address+offset

FCC4 79                 mov     a,c             ;c=record byte count
FCC5 B7                 ora     a               ;0-byte record?
FCC6 CADBFC             jz      GETCSM

                ;Loop to get data into memory at hl.

FCC9 CD99FD     DATALP: call    TPBYTE          ;data byte in b, cksm in d

                ;See if this byte will overwrite our RAM area. This blocks
                ;out a 256-byte region of memory wherever this program found
                ;RAM for its stack.

FCCC CDF1FF             call    RAMPAG          ;(stuffs hl on stack)
FCCF 7C                 mov     a,h             ;a=RAM page address
FCD0 E1                 pop     h               ;restore RAM address
FCD1 BC                 cmp     h               ;same as AMON's RAM page?
```

```
FCD2 CAFDFC             jz      ADRERR          ;y:address error

FCD5 70                 mov     m,b             ;store data in RAM
FCD6 23                 inx     h

FCD7 0D                 dcr     c
FCD8 C2C9FC             jnz     DATALP

FCDB CD99FD     GETCSM: call    TPBYTE          ;get checksum in a & Z flag
FCDE C2F3FC             jnz     CSMERR          ;should be zero

                ;All done with this record. Check for EOF (byte count=0)

FCE1 B3                 ora     e               ;zero-byte record?
FCE2 C2A3FC             jnz     GETCOL          ;N: go get another record

                ;-----------------------------------------
                ;Done. Print record count and return to MAIN
                ;-----------------------------------------
FCE5 E1         HLDONE: pop     h               ;remove offset from stack
FCE6 E1                 pop     h               ;record count

FCE7 CD45FD             call    CILPRT
FCEA 526563733A         db      'Recs:',' '+80h
FCF0 C32AFE             jmp     PHLCHX

                ;--------------
                ;Checksum error
                ;--------------
FCF3 CD45FD     CSMERR: call    CILPRT
FCF6 43736DBF           db      'Csm','?'+80h

FCFA C3E5FC             jmp     HLDONE

                ;-----------------------------------------------
                ;Overwrite error (attempt to write on our RAM space)
                ;-----------------------------------------------
FCFD CD45FD     ADRERR: call    CILPRT
FD00 416472BF           db      'Adr','?'+80h

FD04 C3E5FC             jmp     HLDONE

                ;===============
                ;= subroutines =
                ;===============

                ;***Subroutine*************************************************
                ; Get, echo, and store a console character in the input
                ; line buffer. Handle deletes and backspaces.
                ;
                ; On Entry:
                ;  hl = next free spot in the input line buffer
                ;  LBSIZE is max characters allowed in the input line buffer
                ;On Exit (not full, no deletes):
                ;      a=character
                ;     hl = hl+1
                ; (hl-1) = character
                ;     Carry set and hl not advanced if CR
                ;*************************************************************
FD07 CDE1FB     LBCHR:  call    GETKBD          ;get a character

FD0A FE0D               cpi     CR              ;don't echo CR
```

```
FD0C 37                 stc                     ;Carry set for CR
FD0D C8                 rz                      ;Z set for CR

FD0E 77                 mov     m,a             ;store character in buffer

FD0F FE7F               cpi     DEL             ;DEL character?
FD11 CA16FD             jz      GCDEL
FD14 FE08               cpi     BS              ;BS is same as DEL
FD16 7D         GCDEL:  mov     a,l             ;buffer address low byte
FD17 CA22FD             jz      GDELET

FD1A EECB               xri     RAMBUF+LBSIZE   ;input buffer full?
FD1C C8                 rz                      ;full: ignore it (carry clear)

FD1D 7E                 mov     a,m             ;recover chr for echo ret
FD1E 23                 inx     h               ;bump line buffer pointer

FD1F C394F8             jmp     PRINTA          ;Echo & ret with Carry clear

                ;-------------------------------------------------------------
                ;Backspace or delete found. Delete if there is anything to
                ; delete, and echo to the user the right way, based on TTYPE.
                ;-------------------------------------------------------------
FD22 D67B       GDELET: sui     RAMBUF          ;back up if we can
FD24 C8                 rz                      ;done if not. (carry clear)

                ;backspace either by erasing onscreen or Teletype-style

FD25 2D                 dcr     l               ;back up

FD26 3E68               mvi     a,TTYPE-RIOCOD+RAMCOD
FD28 CDF1FF             call    RAMPAG
FD2B 7E                 mov     a,m             ;get TTYPE variable
FD2C E1                 pop     h
FD2D B7                 ora     a               ;0 means backspacing terminal
FD2E CA3EFD             jz      GCBKUP          ;(carry clear)

FD31 CD4AFD             call    ILPRNT          ;print deleted character
FD34 AF                 db      '/'+80h         ;..between slashes
FD35 7E                 mov     a,m
FD36 CD94F8             call    PRINTA
FD39 CD4AFD             call    ILPRNT
FD3C AF                 db      '/'+80h
FD3D C9                 ret                     ;(carry clear)

FD3E CD4AFD     GCBKUP: call    ILPRNT          ;back up on screen
FD41 082088             db      BS,' ',BS+80h   ;Erase old character & back up

FD44 C9                 ret

                ;***Subroutine*****************************
                ; Print CR LF then inline string at (sp)
                ; Calls to CILPRT are followed by the string
                ; the last string byte has its MSB set
                ; a trashed, all other registers preserved
                ;*******************************************
FD45 CD4AFD     CILPRT: call    ILPRNT
FD48 0D8A               db      CR,LF+80h

                ;Fall into ILPRNT

                ;***Subroutine*****************************
                ; Print inline string at (SP)
```

```
                ; calls to ILPRNT are followed by the string
                ; the last string byte has its MSB set
                ; On Exit:
                ;    a = 80h
                ;    Z cleared
                ;    all other registers preserved
                ;*****************************************
FD4A E3         ILPRNT: xthl                    ;save hl, get string address

FD4B 7E         IPLOOP: mov     a,m             ;loop through message
FD4C E67F               ani     7Fh             ;strip end-marker
FD4E CD94F8             call    PRINTA
FD51 AE                 xra     m               ;end? (clears Carry too)
FD52 23                 inx     h
FD53 CA4BFD             jz      IPLOOP

FD56 E3                 xthl                    ;restore hl
                                                ;..get ret address
FD57 C9                 ret

                ;***Subroutine*****************************
                ; Get console keyboard data
                ; On Entry:
                ;   A keyboard character is already waiting
                ; On Exit:
                ;   a=keyboard character, parity stripped
                ;   Z clear (unless null typed)
                ;*****************************************
FD58 DB11       KDATA:  in      S2RXDA          ;get keyboard character
FD5A E67F               ani     7Fh             ;strip parity
FD5C C9                 ret

                ;***Subroutine*************************************
                ; Get keyboard status unless the transfer port is the
                ; console. Abort if CABKEY (control-C).
                ; On Exit:
                ;   if a character is waiting, then character is in a
                ;   if no character waiting, Z set, a=0
                ;*************************************************
FD5D CD7AFD     CKABRT: call    TESTTP          ;Transfer Port = console?
FD60 C8                 rz

                ;Fall into CHKKBD

                ;***Subroutine*************************************
                ; Get keyboard status. If a character is waiting,
                ; then return it in a with parity stripped. Abort
                ; if CABKEY (control-C).
                ; On Exit:
                ;   if a character is waiting, then character is in a
                ;   if no character waiting, Z set, a=0
                ;*************************************************
FD61 CD85F8     CHKKBD: call    KSTAT           ;anything typed?
FD64 C8                 rz                      ;N: return w/ Z set

FD65 CD58FD             call    KDATA           ;Y: get the data
FD68 FE03               cpi     CABKEY          ;abort character typed?
FD6A C0                 rnz

FD6B B7                 ora     a               ;clear Z flag
FD6C C3E5F8             jmp     CABORT          ;with Z flag cleared

                ;***Subroutine***************
```

```
                        ; Get Transfer Port Rx status
                        ; On Exit:
                        ;   a=0 & Z set if no data
                        ;****************************
FD6F 3E2C               TPISTA: mvi     a,RTPIS-RIOCOD+RAMCOD
FD71 B7                         ora     a               ;clear carry

FD72 DA                         db      JC              ;jc opcode skips mvi a below

                        ;Fall into TPIN, skipping mvi a instruction

                        ;***Subroutine***************
                        ; Get Transfer Port data
                        ; On Exit:
                        ;   a=byte from Transfer Port
                        ;   Z cleared
                        ;****************************
FD73 3E32               TPIN:   mvi     a,RTPIN-RIOCOD+RAMCOD

FD75 CDF1FF                     call    RAMPAG

FD78 E3                         xthl                    ;fix hl, put address on stack
FD79 C9                         ret                     ;'call' RTPIN

                        ;***Subroutine*************************
                        ;Test to see if Transfer Port = console
                        ; On Exit:
                        ;   Z set if console = Transfer Port
                        ; Trashes a
                        ;************************************
FD7A 3E2D               TESTTP: mvi     a,TPISP+1-RIOCOD+RAMCOD ;status register is here
FD7C CDF1FF                     call    RAMPAG

FD7F 7E                         mov     a,m
FD80 E1                         pop     h

FD81 FE10                       cpi     S2STAA                  ;Console's status port?
FD83 C9                         ret

                        ;***Subroutine*********************************************
                        ; Get a printable ASCII byte from the Transfer Port,
                        ; strip parity,  check for abort from the console,
                        ; unless the console is also the Transfer Port
                        ; On Entry:
                        ;   SP points into the RAM page
                        ;   RAM page byte FE = 1 for Transfer Port, 0 for console
                        ; On Exit:
                        ;   character in a, with parity stripped
                        ;*****************************************************
FD84 CD7AFD             GETTPD: call    TESTTP          ;Transfer Port = console?
FD87 CAE1FB                     jz      GETKBD          ;Y: get and test keyboard chr

FD8A CD61FD             GTPLUP: call    CHKKBD          ;user abort?
FD8D CD6FFD                     call    TPISTA          ;Transfer Port character?
FD90 CA8AFD                     jz      GTPLUP          ;n: keep waiting

FD93 CD73FD                     call    TPIN            ;get Transfer Port character
FD96 E67F                       ani     7Fh             ;strip parity

FD98 C9                         ret

                        ;***Subroutine****************************************
                        ; Get 2 hex digits from the Transfer Port, combine them
```

```
                ; into 1 byte, and add the result to the checksum in d
                ; On Entry:
                ;   d = checksum so far
                ; On Exit:
                ;   b=byte of data
                ;   a=d=new checksum value
                ;   Z flag set if checksum is now 0
                ;   all other registers preserved, unless error abort
                ;*****************************************************
FD99 CDA9FD     TPBYTE: call    GTPHEX          ;get high nibble
FD9C 87                 add     a               ;Shift high nibble in place
FD9D 87                 add     a
FD9E 87                 add     a
FD9F 87                 add     a
FDA0 47                 mov     b,a
FDA1 CDA9FD             call    GTPHEX          ;get low nibble

FDA4 B0                 ora     b               ;combine nibbleS
FDA5 47                 mov     b,a             ;save result for return
FDA6 82                 add     d               ;compute checksum
FDA7 57                 mov     d,a             ;ret with checksum in a & d
FDA8 C9                 ret

                ;---Local subroutine--------------------
                ; get a hex digit from the Transfer Port,
                ; validate it, and return it in A<3:0>
                ;--------------------------------------
FDA9 CD84FD     GTPHEX: call    GETTPD
FDAC CD6EFE             call    HEXCON
FDAF D8                 rc                      ;Carry means OK

                ;Abort: ASCII character error - not a valid hex digit

FDB0 CD45FD             call    CILPRT
FDB3 4368F2             db      'Ch','r'+80h

FDB6 C367FE             jmp     CMDERR          ;error handler

                ;=================================================
                ; Command Table
                ; Each entry:
                ;   Byte 0 = 1st command character
                ;   Byte 1 = 2nd command character
                ;   Byte 2 = command execution address low byte
                ;   Byte 3<6:0> = command execution address<14:8>
                ;                   (address<15> is assumed to be 1)
                ;   Byte 3<7> = 0 if the command's parameters are
                ;               not hexidecimal values
                ;
                ; The table is terminated by a null in Byte 0
                ;=================================================
FDB9 4144       COMTAB: db      'AD'            ;Dump in Altair format
FDBB 45F9               dw      ADUMP
FDBD 414C               db      'AL'            ;Load Altair format
FDBF 06FE               dw      ALOAD

FDC1 424F               db      'BO'            ;Boot from FLOPPY
FDC3 06FF               dw      FBOOT
FDC5 434F               db      'CO'            ;Copy memory
FDC7 84F9               dw      MCOPY
FDC9 4455               db      'DU'            ;Dump to console
FDCB 8BFA               dw      DUMP
FDCD 454E               db      'EN'            ;Enter
```

```
FDCF DAFA               dw      ENTER
FDD1 4558               db      'EX'            ;Execute
FDD3 79FA               dw      EXEC
FDD5 4649               db      'FI'            ;Fill memory
FDD7 F2FA               dw      FILMEM

FDD9 4842               db      'HB'            ;Boot from hard disk
FDDB 08FC               dw      HBOOT

FDDD 4844               db      'HD'            ;Intel hex dump
FDDF 48FB               dw      HEXDMP
FDE1 484C               db      'HL'            ;Intel hex load
FDE3 9EFC               dw      HEXLOD

FDE5 5345               db      'SE'            ;Search
FDE7 E2F9               dw      SEARCH

FDE9 5445               db      'TE'            ;Terminal Mode
FDEB 0D7B               dw      TERMNL and 7FFFh ;non-hex parameter

FDED 5450               db      'TP'            ;Set Transfer Port
FDEF 4CF8               dw      SETTP
FDF1 5454               dw      'TT'            ;Terminal Type
FDF3 6EFA               dw      SETTT

FDF5 5645               db      'VE'            ;Verify
FDF7 BDF9               dw      VERCMD

FDF9 00                 db      0               ;end of table mark

                ;================================================================
                ; Check for assembly problem: can't overwrite CDBL
                ;================================================================
                ;================================================================
                ; Check for assembly problem in Hole 1: all of Monitor must not
                ; overrun the next section
                ;================================================================
FDFA =          H1END   equ     $

                 if ((MBLADR/256) - ((H1END-1)/256+1))/256
                        ERROR: MBL is overwriting prior code
                 endif

                ;================================================================
                ;               Multi Boot Loader Subsystem
                ;
                ; Loads and runs an Altair 'Absolute Binary file' from input
                ; Transfer Port specified by the Sense switch settings.
                ;
                ; This code may be entered either by a call from the AMON main
                ; loop or directly from reset (either via the front panel or via
                ; Jump-Start hardware). If entered from AMON, then AMON will
                ; pass the selected load port, as requested by the user. If
                ; executed directly, then this code will look at the front panel
                ; switch register to determine the load port.
                ;
                ;** Differences between MITS MBL and this code **
                ;
                ; 1) The code starts off by relocating itself to the highest
                ;    page of RAM that is found, so that it will still work
                ;    if the PROM is Phantomed by an IN instruction from port
                ;    FF (the switch register).
                ; 2) All HSR support is eliminated, including 88-4PIO port 1
```

```
;    initialization and code for starting the HSR transport.
; 3) The second 88-2SIOJP port (port 1) is initialized.
; 4) The switch setting that was assigned to the HSR has been
;    reassigned to the 88-2SIOJP's second port.
; 5) PTABLE has an 8th entry, which is the same as the 1st
;    (2SIO port 0). Testing for illegal sense switch setting
;    is thereby eliminated.
; 6) An initial read is performed for both the 88-PIO and the
;    88-4PIO port 0, to clear data handshake latches in
;    external devices such as the OP-80 paper tape reader
; 7) If the tape leader character is 0, then no checksum
;    loader will be skipped.
; 9) Sense switch A11 is ignored when getting the load device,
;    rather than generating an I error.
;=============================================================
; program Notes
; Since the 88-2SIOJP may optionally disable PROMS whenever an IN
; instruction accesses port FFh (like some version sof the MITS
; 8800b Turnkey Module), this code cannot execute from
; PROM - at least not from the point where the Sense switches
; are read onwards.
;
; StrategY:
; 1) search the memory space for the highest actual RAM, as
;    MITS's MBL did. this page of memory will be used not only
;    for the stack, but also for the relocated code.
; 2) copy code into the high RAM page that was found in step 1.
;    (The code lands in the RAM buffer.) the high byte
;    of addresses are relocated to the RAM execution address
;    as the bytes are copied.
; 3) Jump to the RAM code, and run from there - never to
;    return to PROM.
;
; the RAM page is laid out as follows:
;  * the high portion (RAMBUF) contains the relocated MBL code.
;  * Immediately below this is the stack, growing downward.
;    (Note that a push decrements the stack pointer before
;     writing to the stack.)
;  * Immediately below this are the modifiable transfer port
;    routines, which get modified at the beginning for the
;    particular load port selected.
;
;=============================================================
; An Altair 'Absolute Binary file' has 4 sections, which may be
; separated by any number of nulls. these sections are:
;
; 1) the Leader, which comprises 2 or more identical bytes, the
;    value of which is the length of the checksum loader.
;
; 2) the checksum loader, which is a program that is normally
;    used to load the subsequent sections
;
; 3) zero or more load records, each structured as follows:
;       byte 0: Sync byte = 3Ch (identifies a load record)
;       byte 1: NN = number of data bytes in record
;       byte 2: LL = load address low byte
;       byte 3: HH = load address high byte
; bytes 4-NN+3: NN data bytes to store at HHLl, NN>0
;    byte NN+4: CC = checksum of bytes 2 through NN+3
;
; 4) the Go record, structured as follows
;       byte 0: Sync byte = 78H (identifies the Go record)
;       byte 1: LL = low byte of go address
```

```
                ;       byte 2: HH = high byte of go address
                ;
                ; Altair file Leaders and checksum loaders are specific to
                ; both the version of the particular software and the memory
                ; size. for example, the checksum loader for 4K Basic 3.2 is
                ; different than the checksum loader for 8K Basic 3.2. and
                ; both the Leader and checksum loader for 8K Basic 3.2 are
                ; different than those for 8K Basic 4.0.
                ;
                ; The MBL code is able to read any such Altair file by simply
                ; skipping over the Leader and checksum loader, and loading
                ; the load and Go records directly.
                ;
                ; When executed at the MBL address, MBL chooses its input
                ; Port based on the front panel Sense switches <2:0>, using
                ; the conventions set up in Basic 4.X, more or less.
                ;
                ;  device                    bits 2:0
                ;  88-2SIO port 0 (2 stops)   000b
                ;  88-2SIO port 0 (2 stops)   001b
                ;  88-SIO                     010b
                ;  88-ACR                     011b
                ;  88-4PIO                    100b
                ;  88-PIO                     101b
                ;  88-2SIO Port 1 (2 stops)   110b
                ;  88-2SIO port 0 (2 stops)   000b (spare)
                ;
                ; Prior to Basic 4.0, MITS used different Sense switch settings
                ; to specify the console device. You can load an older tape
                ; by setting the switches according to the above table and
                ; starting the load. after the checksum loader on the tape
                ; has been skipped, and load records are loading (but before
                ; the load completes) change the Sense switch settings as
                ; required by the earlier version of Basic (or other program)
                ; that you are loading.
                ;================================================================
FE00                    org     MBLADR
                ;================================================================

                ;----------------------------------------------------------------
                ;find RAM
                ;----------------------------------------------------------------
FE00 010CFE     MBL:    lxi     b,GOMBL         ;return address
FE03 C303F8             jmp     INIT            ;go find a real stack, install
                                                ;self-modifying I/O routines,
                                                ;and initialize all known ports
                                                ;returns with bc=0


                ;***Command Routine*************************
                ; AL (Boot from tape)
                ;
                ; On Entry:
                ;    TP command has set up the Transfer Port
                ;*******************************************
FE06 CD45FD     ALOAD:  call    CILPRT          ;be pretty
FE09 8A                 db      LF+80h

FE0A 0E01               mvi     c,1             ;entry from amon

                ;----------------------------------------------------------------
                ; Entry here from cold-start at MBLADR:
                ;    c=0
```

```
                ;    nothing on stack
                ; Entry here from monitor call to ALOAD:
                ;    c=1
                ;    bottom of stack = address of MAIN
                ;-----------------------------------------------------------
FE0C 210000     GOMBL:  lxi     h,0             ;find RAM page
FE0F 39                 dad     sp
FE10 2E7B               mvi     l,RAMBUF        ;sector buffer at end of page

                ;-----------------------------------------------------------
                ;  Relocate PROM image to the sector buffer in RAM.
                ;    Run-time relocation of addresses is done by replacing any
                ;    byte that matches the MSB of the org address with the MSB
                ;    of the destination RAM address. this requires the value
                ;    of the org MSB never to appear in the assembled code other
                ;    than as the MSB of an address. (F800 works for this.)
                ; On Entry:
                ;    hl = RAMBUF address (where we will put code)
                ;    c = 0 for cold-start, 1 for entry from monitor
                ; On Exit:
                ;    c = unchanged
                ;-----------------------------------------------------------
FE12 117BFE             lxi     d,MIOCOD

FE15 E5                 push    h               ;RAM code execution address
FE16 C5                 push    b

FE17 0685               mvi     b,MRCEND-MIOCOD ;byte count

FE19 1A         RELOOP: ldax    d
FE1A BA                 cmp     d               ;relocatable address byte?
FE1B C21FFE             jnz     NOTADR
FE1E 7C                 mov     a,h             ;relocate this address
FE1F 77         NOTADR: mov     m,a
FE20 23                 inx     h
FE21 13                 inx     d
FE22 05                 dcr     b
FE23 C219FE             jnz     RELOOP

FE26 C1                 pop     b

                ;The code image has been copied to RAM and addresses relocated,
                ;and its address is on the stack.

FE27 59                 mov     e,c                     ;e=1 if entry from amon
FE28 0D                 dcr     c                       ;use switches?
FE29 C9                 ret

                ;===========================================================
                ; AMON Subroutines, occupying a hole in the PROM space
                ;===========================================================

                ;***Subroutine*************************
                ; Print hl as 4 hex digits on the console
                ; On Entry:
                ;   c=checksum so far
                ;  hl=2 bytes to print
                ; On Exit:
                ;   b=0
                ;   c=updated checksum
                ; Trashes a
                ;*****************************************
FE2A 0600       PHLCHX: mvi     b,0     ;print on console
```

```
                ;Fall into PHLHEX

                ;***Subroutine***************
                ; Print hl as 4 hex digits
                ; On Entry:
                ;   b=0 for the console
                ;   b<>0 for the Transfer Port
                ;   c=checksum so far
                ;  hl=2 bytes to print
                ; On Exit:
                ;   c=updated checksum
                ; Trashes a
                ;*****************************
FE2C 7C         PHLHEX: mov     a,h             ;h first
FE2D CD34FE             call    PAHEXC          ;returns with Carry clear
FE30 7D                 mov     a,l             ;then l

FE31 FE                 db      CPI             ;CPI opcode skips PCAHEX
                                                ;executing a NOP, and then
                                                ;..falling into PAHEX

                ;***Subroutine**********************
                ; Print a on console as 2 hex digits
                ; On Entry:
                ;   a=byte to print
                ; On Exit:
                ;   b=0
                ; Trashes a,c
                ;***********************************
FE32 0600       PCAHEX: mvi     b,0             ;print to console

                ;Fall into PAHEX

                ;***Subroutine*********************************
                ; Print a as 2 hex digits and update checksum
                ; On Entry:
                ;   a=byte to print
                ;   b=0 for the console
                ;   b<>0 for the Transfer Port
                ;   c=checksum so far
                ; On Exit:
                ;   c=updated checksum
                ; Trashes a
                ;******************************************
FE34 F5         PAHEXC: push    psw
FE35 81                 add     c               ;compute checksum
FE36 4F                 mov     c,a
FE37 F1                 pop     psw             ;recover character

                ;Fall into PAHEX

                ;***Subroutine***************
                ; Print a as 2 hex digits
                ; On Entry:
                ;   a=byte to print
                ;   b=0 for the console
                ;   b<>0 for the Transfer Port
                ; On Exit:
                ; Trashes a
                ;*****************************
FE38 F5         PAHEX:  push    psw             ;save for low digit
```

```
FE39 0F               rrc                     ;move the high four down
FE3A 0F               rrc
FE3B 0F               rrc
FE3C 0F               rrc
FE3D CD41FE           call    PNIBLE          ;put them out
FE40 F1               pop     psw             ;this time the low four

                ;Fall into PNIBLE

                ;---Local subroutine----------
                ; Print low nibble of a in hex
                ; On Entry:
                ;   b=0 for the console
                ;   b<>0 for the Transfer Port
                ; On Exit:
                ;   a trashed
                ;-----------------------------
FE41 E60F       PNIBLE: ani     0Fh             ;four on the floor
FE43 C630               adi     '0'             ;We work with ASCII here
FE45 FE3A               cpi     '9'+1           ;0-9?
FE47 DA4CFE             jc      PNIB1           ;YUP: print & return

FE4A C607               adi     'A'-'9'-1       ;make it a letter

FE4C 04         PNIB1:  inr     b               ;which port?
FE4D 05                 dcr     b
FE4E C2A8FB             jnz     TPOUT           ;print on Transfer Port

FE51 C394F8             jmp     PRINTA          ;exit from there

                ;***Subroutine*********************************
                ; Print hl in hex on the console,
                ; preceeded by CR,LF,space, and followed by ': '
                ; On Exit:
                ;    b=0
                ; Trashes a,c
                ;***********************************************
FE54 CD45FD     PHLADR: call    CILPRT          ;CR LF space begins line
FE57 A0                 db      ' '+80h

FE58 0600               mvi     b,0             ;output address to console
FE5A CD2AFE             call    PHLCHX          ;hl=address, b=0, trash c

FE5D CD4AFD             call    ILPRNT          ;print colon space
FE60 3AA0               db      ':',' '+80h
FE62 C9                 ret

                ;***Subroutine*****************************************
                ; Get a hex value from the line buffer
                ; Abort to CMDERR if none provided
                ; On Entry:
                ;   de=address of next item in the input line buffer
                ; On Exit:
                ;   hl=value
                ;   de advanced past character
                ;   abort to CMDERR if no value found
                ;*******************************************************
FE63 CD20F9     GETHEX: call    FNDHEX          ;get hl=hex value
FE66 D0                 rnc

                ;Fall into CMDERR if no address

                ;**********************
```

```
                        ; Command error handler
                        ;***********************
 FE67 CD4AFD    CMDERR: call    ILPRNT          ;returns Z flag cleared
 FE6A BF                db      '?'+80h

 FE6B C3E5F8            jmp     CABORT          ;Repair stack, go to MAIN

                        ;***Subroutine*******************************
                        ; Convert ASCII hex digit to binary
                        ; On Entry:
                        ;   a=character to convert
                        ; On Exit:
                        ;   a=binary result
                        ;   Carry set if OK, clear if bogus character
                        ;*****************************************
 FE6E D630      HEXCON: sui     '0'             ;Remove ASCII bias
 FE70 FE0A              cpi     10
 FE72 D8                rc                      ;If 0-9 then we're done

 FE73 D611              sui     9+('A'-'9')     ;Should be 0-5 now
 FE75 FE06              cpi     6               ;Gap chr or too high?
 FE77 D0                rnc                     ;Error if so

 FE78 D6F6              sui     0F6h            ;Add 0AH, Set carry
 FE7A C9                ret                     ;Ret with carry set


                        ;================================================================
                        ; Check for assembly problem in Hole 2: the above code must not
                        ; overrun the next section
                        ;================================================================
 FE7B =         H2END   equ     $

                 if (MIOCOD - (H2END-1))/256
                        ERROR: Code in Hole 2 is too big
                 endif

                        ;================================================================
                        ; MBL RAM Execution Code
                        ; All of the following code gets copied into the RAM Buffer
                        ; (which is in the highest page of RAM that was discovered
                        ; during initialization). this is in RAM because an IN from
                        ; port FF (the front panel switch register) optionally disables
                        ; the PROM.
                        ; On Entry:
                        ;    e = 0 and Z clear if cold entry
                        ;    e = 1 and Z set if entry from monitor
                        ;          (Transfer Port already set up)
                        ;    h = RAM Execution page
                        ;================================================================
 FE7B                   org     MBLADR+RAMBUF           ;force low address byte the
same

 FE7B CA85FE    MIOCOD: jz      SKPSW                   ;Amon active?
 FE7E DBFF              in      SSWTCH                  ;N: get Transfer Port
 FE80 E607              ani     LDMASK                  ;bits that specify load
device


                        ;       call    RSETP           ;set up Transfer Port
 FE82 CD                db      CALL                    ;call opcode
 FE83 00                db      RSETP-RIOCOD+RAMCOD     ;low address byte
 FE84 FE                db      MIOCOD/256              ;high byte (gets relocated)
```

```
                SKPSW:


                ;-----------------------------------------------
                ; Flush external data latches for e.g. the OP-80
                ; or flush garbage from UARTs
                ; On Entry & exit:
                ;    d = RAM execution page
                ;    e = 0 if PROM may be disabled
                ;-----------------------------------------------
                ;       call    RTPIF
FE85 CD                 db      CALL                    ;call opcode
FE86 38                 db      RTPIF-RIOCOD+RAMCOD     ;low address byte
FE87 FE                 db      MIOCOD/256              ;high byte (gets relocated)

                ;-------------------------------------------------------------
                ; Skip over leader - a sequence of identical bytes, the value
                ; of which is the length of the checksum loader. If the value
                ; is  0, then there is no loader to skip, so go get records.
                ; On Entry:
                ;    d = RAM Execution page
                ; On Exit:
                ;    c = checksum loader length
                ;    d = RAM execution page
                ;    e = 0 if PROM may be disabled
                ;    the 1st byte of the checksum loader has already been read
                ;-------------------------------------------------------------
FE88 CDBCFE             call    GETBYT          ;get 1st byte

FE8B 4F                 mov     c,a             ;number of bytes in loader

FE8C B7                 ora     a               ;null leader?
FE8D CA9FFE             jz      RCHUNT          ;Y: skip leader

FE90 CDBCFE     LDSKIP: call    GETBYT          ;get another byte

FE93 B9                 cmp     c
FE94 CA90FE             jz      LDSKIP          ;loop until different

                ;-------------------------------------------------------------
                ; Skip over checksum loader
                ;
                ; On Entry:
                ;    the 1st byte of the checksum loader has already been read
                ;    c=checksum loader length
                ;    d = RAM execution page
                ;    e = 0 if PROM may be disabled
                ; On Exit:
                ;    d = RAM execution page
                ;    e = 0 if PROM may be disabled
                ;    the checksum loader has been skipped
                ;-------------------------------------------------------------
FE97 0D                 dcr     c               ;since we got a byte already

FE98 CDBCFE     CLSKIP: call    GETBYT          ;get a loader byte
FE9B 0D                 dcr     c
FE9C C298FE             jnz     CLSKIP

                ;-------------------------------------------------------------
                ; Main record-loading loop
                ;
                ; Hunt for a sync character - either for another load record
                ; or for the Go record. ignore all else.
```

```
                ; On Entry:
                ;    c = 0
                ;    d = RAM execution page
                ;    e = 0 if PROM may be disabled
                ; on jmp tp LDREC:
                ;    c = 0
                ;    d = RAM execution page
                ;    e = 0 if PROM may be disabled
                ;    RCHUNT address is on the stack
                ;-------------------------------------------------------
FE9F 219FFE     RCHUNT: lxi     h,RCHUNT                ;create return address
FEA2 E5                 push    h

FEA3 CDBCFE             call    GETBYT                  ;hunt for sync character

                ;Note: can't use cpi opcode here because it is FEh

FEA6 EE3C               xri     ALTPLR                  ;load record sync byte?
FEA8 CACBFE             jz      LDREC                   ;Y: go load the record

FEAB EE44               xri     ALTEOF XOR ALTPLR       ;EOF record sync byte?
FEAD C0                 rnz                             ;N: ignore

                ;Fall into GO record execution

                ;-----------------------------------------------
                ; Go Record: get the GO address and go there
                ;
                ; On Entry:
                ;    d = RAM execution page
                ;    e = 0 if PROM may be disabled
                ;    GO-record sync byte has already been read
                ;-----------------------------------------------
FEAE CDB3FE             call    GETWRD          ;get a,l=address
FEB1 67                 mov     h,a             ;high byte
FEB2 E9                 pchl                    ;go there

                ;---subroutine----------------------
                ; get 2-byte word from Transfer Port
                ; On Entry:
                ;   b=checksum so far
                ; On Exit:
                ;   l = next byte
                ;   a = subsequent byte
                ;   b := b+a+l
                ;-----------------------------------
FEB3 CDBCFE     GETWRD: call    GETBYT
FEB6 6F                 mov     l,a

                ;Fall into GETBYT (entering at GB1 is ok)

                ;---subroutine-------------------------------------
                ; get a byte of data from the Transfer Port
                ; On Entry:
                ;   b = checksum so far
                ;   e =1 if called from AMON
                ; On Exit:
                ;   a = received character
                ;   b = updated checksum
                ;   Z set if received chr matched original checksum
                ; -->MAin entry is iat GETBYT<---
                ;-------------------------------------------------
FEB7 7B         GB1:    mov     a,e
```

```
 FEB8 B7                ora     a                       ;AMON active?
 FEB9 C45DFD            cnz     CKABRT                  ;Y: user abort?

                GETBYT:
                ;       call    RTPIS                   ;get transfer port status
 FEBC CD                db      CALL                    ;call opcode
 FEBD 2C                db      RTPIS-RIOCOD+RAMCOD     ;low address byte
 FEBE FE                db      MIOCOD/256              ;high byte (gets relocated)

 FEBF CAB7FE            jz      GB1                     ;Wait for character

                ;       call    RTPIF                   ;get transfer port  data
byte
 FEC2 CD                db      CALL                    ;call opcode
 FEC3 38                db      RTPIF-RIOCOD+RAMCOD     ;low address byte
 FEC4 FE                db      MIOCOD/256              ;high byte (gets relocated)

 FEC5 B8                cmp     b                       ;set Z if this byte matched
cksum
 FEC6 F5                push    psw                     ;temp save
 FEC7 80                add     b                       ;update checksum in B
 FEC8 47                mov     b,a
 FEC9 F1                pop     psw                     ;recover data byte
 FECA C9                ret

                ;-----------------------------------------------------------
                ; Load Record: Read and store data from a load record
                ;
                ; On Entry:
                ;    the load record sync byte has already been read
                ;    c = 0
                ;    d = RAM execution page
                ;    e = 0 if PROM may be disabled
                ;    RCHUNT's address is on the stack
                ; on return (to RCHUNT):
                ;    c = 0
                ;    d = RAM execution page
                ;    e = 0 if PROM may be disabled
                ;    a complete load record's data has been loaded into RAM
                ;-----------------------------------------------------------
 FECB CDBCFE    LDREC:  call    GETBYT          ;get record byte count

 FECE 41                mov     b,c             ;c=0: initialize checksum
 FECF 4F                mov     c,a             ;c counts data bytes

 FED0 CDB3FE            call    GETWRD          ;get load address into a,l

 FED3 67                mov     h,a             ;hl = record load address

                ;Loop to read c data bytes into memory at hl.
                ;Make sure data won't overwrite RAM Execution page.

 FED4 7A        LRLOOP: mov     a,d             ;d=RAM Execution page
 FED5 BC                cmp     h               ;error if same page as load address
 FED6 3E4F              mvi     a,OERMSG        ;overwrite error message
 FED8 CAF0FE            jz      ERDONE          ;error exit if overwrite

 FEDB CDBCFE            call    GETBYT          ;get a data byte

 FEDE 77                mov     m,a             ;store data byte
 FEDF BE                cmp     m               ;did it store correctly?
```

```
FEE0 3E4D               mvi     a,MERMSG        ;memory error message
FEE2 C2F0FE             jnz     ERDONE          ;error exit if mismatch

FEE5 23                 inx     h               ;bump dest pointer
FEE6 0D                 dcr     c               ;bump byte count
FEE7 C2D4FE             jnz     LRLOOP          ;loop through all bytes

                ;Validate checksum, fail if it doesn't match
                ; c = 0 here

FEEA CDBCFE             call    GETBYT          ;test record's checksum

FEED C8                 rz                      ;match: get another record

FEEE 3E43               mvi     a,CERMSG        ;checksum error message

                ;Fall into ERDONE

                ;---end-----------------------------------------------------
                ;Error Handler:
                ; If the PROM has not been disabled (by a read from port FF),
                ; then report the error and return to the AMON monitor. If
                ; port FF has been read (to determine the load port), then
                ; save the error code and address at beginning of memory, and
                ; hang writing the error code forever to the console.
                ; On Entry:
                ;    a = error code
                ;    e = 0 if PROM may be disabled
                ;    e = 1 if PROM not disabled
                ;   hl = offending address
                ;-----------------------------------------------------------
FEF0 1D         ERDONE: dcr     e               ;PROM disabled?
FEF1 CACAFF             jz      RPTERR          ;N: report, return to monitor
                                                ;this routine not relocated

                ;PROM is possibly disabled. Report error the old way.

FEF4 320000             sta     00000h          ;PROM disabled: store error code
FEF7 220100             shld    00001H
FEFA FB                 ei

FEFB D311       ERHANG: out     S2TXDA          ;2SIO
FEFD C3FBFE             jmp     ERHANG

                ;===============================================================
                ; End of MBL code copied into the RAM buffer
                MRCEND:
                ;===============================================================

FF00 =          SUBEND  equ     $

                 if ((DBLADR/256) - ((SUBEND-1)/256+1))/256
                        ERROR: CDBL is overwriting prior code

                 endif

                ;===============================================================
                ;=         CDBL - Combo Disk boot loader Subsystem             =
                ;=        for the Altair 88-DCDD 8" disk system and            =
                ;=            the Altair 88-MDS Minidisk system                =
                ;=                                                             =
                ;= 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                               =
                ;=                                                              =
                ;= 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 (RAMBUF). 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 the CDBL subsystem =
                ;= print an error message with the offending address, and      =
                ;= jump to the AMON main loop.                                 =
                ;================================================================

                ;================================================================
FF00                    org     DBLADR
                ;================================================================
                ; Entry here to execute CDBL directly, to boot from a floppy.
                ; This is the same address where MITS's DBL and MDBL start.
                ;================================================================

FF00 0106FF     CDBL:   lxi     b,FBOOT         ;return address
FF03 C303F8             jmp     INIT            ;go find a real stack
                                                ;and initialize ACIAs
```

```
                        ;***Command Routine*********
                        ; BO (Boot from floppy disk)
                        ;***************************
FF06 CD4AFD     FBOOT:  call    ILPRNT          ;CRLF to look pretty
FF09 0D8A               db      CR,LF+80h

FF0B CDEFFF             call    FNDBUF          ;find buffer, push hl
FF0E E3                 xthl                    ;buffer address onto stack

                        ;-------------------------------------------------------
                        ; 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.
                        ;-------------------------------------------------------
FF0F CD61FD     WAITEN: call    CHKKBD          ;abort from user?
FF12 AF                 xra     a               ;boot from disk 0
FF13 D308               out     DENABL          ;enable disk 0
FF15 DB08               in      DSTAT           ;Read drive status
FF17 E608               ani     DRVRDY          ;Diskette in drive?
FF19 C20FFF             jnz     WAITEN          ;no: wait for drive ready

FF1C 3E04               mvi     a,HDLOAD        ;load 8" disk head, or enable
FF1E D309               out     DCTRL           ;..minidisk for 6.4 Sec

                        ;-------------------------------------------------------
                        ; Step in once, then step out until track 0 is detected
                        ; On Exit: b=0
                        ;-------------------------------------------------------
FF20 018206             lxi     b,20000/12      ;20 mS delay 1st time thru
FF23 3E01               mvi     a,STEPIN        ;step in once first

FF25 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.

FF27 0B         DELAY:  dcx     b               ;(5)
FF28 78                 mov     a,b             ;(5)
FF29 B1                 ora     c               ;(4)
FF2A C227FF             jnz     DELAY           ;(10)12 uS/pass

FF2D 0C                 inr     c               ;from now on, the above loop
                                                ;goes 1 time.

FF2E DB08       WSTEP:  in      DSTAT           ;wait for step to complete
FF30 0F                 rrc                     ;put MVHEAD bit in Carry
FF31 0F                 rrc                     ;is the servo stable?
FF32 DA2EFF             jc      WSTEP           ;no: wait for servo to settle

FF35 E610               ani     TRACK0/4        ;Are we at track 00?
FF37 3E02               mvi     a,SRTPOUT       ;STEP-out command
FF39 C225FF             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

FF3C DB09               CKDSK1: in      DSECTR          ;Read the sector position

FF3E E63F                       ani     SECMSK+SVALID   ;mask sector bits, and hunt
FF40 FE1E                       cpi     (MDSPT-1)*2     ;..for minidisk last sector
FF42 C23CFF                     jnz     CKDSK1          ;..only while SVALID is 0

                        ; wait for this sector to pass

FF45 DB09               CKDSK2: in      DSECTR          ;Read the sector position
FF47 0F                         rrc                     ;wait for invalid sector
FF48 D245FF                     jnc     CKDSK2

                        ; wait for and get the next sector number

FF4B DB09               CKDSK3: in      DSECTR          ;Read the sector position
FF4D 0F                         rrc                     ;put SVALID in Carry
FF4E DA4BFF                     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).

FF51 E61F                       ani     SECMSK/2        ;mask sector bits
FF53 C610                       adi     MDSPT           ;compute SPT
FF55 4F                         mov     c,a             ;..and save SPT in c

                        ;-----------------------------------------
                        ; Set up to load
                        ; On Entry:
                        ;   b = 0 (initial sector number)
                        ;   c = SPT (for either minidisk or 8" disk)
                        ;-----------------------------------------
FF56 210000                     lxi     h,DMAADR        ;initial DMA address

                        ;-------------------------------------------------------
                        ; 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
                        ;-------------------------------------------------------
FF59 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
                        ;  top-of-stack = RAMBUF address (for retries)
                        ;-----------------------------------------
FF5B D1                 RDSECT: pop     d               ;(10)get RAMBUF address
FF5C D5                         push    d               ;(11)keep it on the stack
FF5D 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.
                ;-------------------------------------------------------
FF5E DB09       FNDSEC: in      DSECTR          ;(10)Read the sector position

FF60 E63F               ani     SECMSK+SVALID   ;(7)yes: mask sector bits
                                                ;..along with -SVALID bit
FF62 0F                 rrc                     ;(4)sector bits to bits <4:0>
FF63 B8                 cmp     b               ;(4)found the desired sector
                                                ;..with -SVALID low?
FF64 C25EFF             jnz     FNDSEC          ;(10)no: wait for it

                ;-------------------------------------------------------
                ; Test for DMA address that would overwrite the sector buffer
                ; or the stack. Do this here, while we have some time.
                ;-------------------------------------------------------
FF67 7C                 mov     a,h             ;(5)high byte of DMA address
FF68 AA                 xra     d               ;(4)high byte of RAM code addr
FF69 3E4F               mvi     a,OERMSG        ;(7)overlay error message
FF6B CACAFF             jz      RPTERR          ;(10)report overlay error

                ;---------------------------------------
                ; Set up for the upcoming data move
                ; Do this here, while we have some time.
                ;---------------------------------------
FF6E E5                 push    h               ;(11)DMA address for retry
FF6F C5                 push    b               ;(11)Current sector & SPT
FF70 018000             lxi     b,BPS           ;(10)b= init checksum,
                                                ;c= byte count for movLUP

                ;-------------------------------------------------------
                ; Sector Read: Step 2. Read sector data into RAMBUF at de.
                ;              RAMBUF 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.
                ;-------------------------------------------------------
FF73 DB08       DATLUP: in      DSTAT           ;(10)Read the drive status
FF75 07                 rlc                     ;(4)new Read data Available?
FF76 DA73FF             jc      DATLUP          ;(10)no: wait for data

FF79 DB0A               in      DDATA           ;(10)Read data byte
FF7B 12                 stax    d               ;(7)store it in sector buffer
FF7C 1C                 inr     e               ;(5)Move to next buffer address
                                                ;..and test for end
FF7D C273FF             jnz     DATLUP          ;(10)loop if more data

                ;-------------------------------------------------------
                ; Sector Read: Step 3. Move sector data from RAMBUF into
                ;              memory at hl. compute checksum as we go.
                ;
                ; 8327 cycles for this section
                ;-------------------------------------------------------
FF80 1E7E               mvi     e,SDATA          ;(7)de= address of sector data
                                                 ;..within the sector buffer

FF82 1A         movLUP: ldax    d               ;(7)get sector buffer byte
FF83 77                 mov     m,a             ;(7)store it at the destination
FF84 BE                 cmp     m               ;(7)Did it store correctly?
FF85 C2C8FF             jnz     MEMERR          ;(10)no: abort w/ memory error

FF88 80                 add     b               ;(4)update checksum
```

```
FF89 47                 mov     b,a             ;(5)save the updated checksum

FF8A 13                 inx     d               ;(5)bump sector buffer pointer
FF8B 23                 inx     h               ;(5)bump DMA pointer
FF8C 0D                 dcr     c               ;(5)more data bytes to copy?
FF8D C282FF             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
                ; 134 cycles for for this section
                ;-------------------------------------------------------------
FF90 EB                 xchg                    ;(4)hl=1st trailer byte address
                                                ;de=DMA address
FF91 4E                 mov     c,m             ;(7)get marker, should be FFh
FF92 0C                 inr     c               ;(5)c should be 0 now

FF93 23                 inx     h               ;(5)(hl)=checksum byte
FF94 AE                 xra     m               ;(7)compare to computed cksum
FF95 B1                 ora     c               ;(4)..and test marker=ff

FF96 C1                 pop     b               ;(10)Current sector & SPT
FF97 C2BBFF             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.

FF9A 2E7C               mvi     l,SFSIZE        ;(7)hl=address of file size
FF9C 7E                 mov     a,m             ;(7)low byte
FF9D 23                 inx     h               ;(5)point to high byte
FF9E 66                 mov     h,m             ;(7)high byte
FF9F 6F                 mov     l,a             ;(5)hl=SFSIZE

FFA0 EB                 xchg                    ;(4)put DMA address back in hl
                                                ;..and file size into de

FFA1 7D                 mov     a,l             ;(4)16-bit subtraction
FFA2 93                 sub     e               ;(4)
FFA3 7C                 mov     a,h             ;(5)..throw away the result
FFA4 9A                 sbb     d               ;(4)..but keep Carry (borrow)

FFA5 D1                 pop     d               ;(10)chuck old DMA address
FFA6 D1                 pop     d               ;(10)chuck old retry count

FFA7 D2E4FF             jnc     LDDONE          ;(10)done loading if hl >= de

                ;-------------------------------------------------------------
                ; Next Sector: the sectors are interleaved by two. Read all
                ;              the even sectors first, then the odd sectors.
                ;
                ; 44 cycles for the next even or next odd sector
                ;-------------------------------------------------------------
FFAA 1159FF             lxi     d,NXTSEC        ;(10)for compact jumps
FFAD D5                 push    d               ;(10)

FFAE 04                 inr     b               ;(5)sector = sector + 2
FFAF 04                 inr     b               ;(5)

FFB0 78                 mov     a,b             ;(5)even or odd sectors done?
FFB1 B9                 cmp     c               ;(4)c=SPT
```

```
FFB2 D8                 rc                      ;(5/11)no: go read next sector
                                                ;..at NXTSEC

                ; Total sector-to-sector = 28+8327+134+44=8533 cycles=4266.5 uS
                ; one 8" sector time = 5208 uS, so with 2:1 interleave, we will
                ; make the next sector, no problem.

FFB3 0601               mvi     b,01H           ;1st odd sector number
FFB5 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.
                ;-------------------------------------------------------------
FFB6 78                 mov     a,b             ;STEPIN happens to be 01h
FFB7 D309               out     DCTRL

FFB9 05                 dcr     b               ;start with b=0 for sector 0
FFBA 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
                ;*************************************************************
FFBB 3E04       BADSEC: mvi     a,HDLOAD        ;Restart Minidisk 6.4 uS timer
FFBD D309               out     DCTRL

FFBF E1                 pop     h               ;Restore DMA address
FFC0 F1                 pop     psw             ;get retry count
FFC1 3D                 dcr     a               ;Any more retries left?
FFC2 C25BFF             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
                ;-------------------------------------------------
FFC5 3E43               mvi     a,CERMSG        ;checksum error message
FFC7 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
                ;*************************************
FFC8 3E4D       MEMERR: mvi     a,MERMSG        ;memory error message

                ;Fall into RPTERR
```

```
                ;***CDBL Termination*************************************
                ;   Report an error: turn the disk controller off, turn the
                ;   INTE light on, report the error on the console, jump
                ;   to the console loop.
                ;   On Entry:
                ;      a = error code
                ;     hl = offending RAM address
                ;********************************************************
FFCA CD94F8     RPTERR: call    PRINTA          ;print the error code

FFCD 3E80               mvi     a,DDISBL        ;Disable the disk controller
FFCF D308               out     DENABL

FFD1 CD4AFD             call    ILPRNT
FFD4 206572726F         db      ' error at',' '+80h
FFDE CD2AFE             call    PHLCHX          ;print address on console

                ; Cool-start AMON code

FFE1 C3C2F8             jmp     INIT2           ;go to monitor

                ;***CDBL Termination*************************************
                ; Successful load: Disable PROM and execute loaded code
                ; Note that EXECDP assumes low byte of DMAADR=0
                ;********************************************************
FFE4 3E80       LDDONE: mvi     a,DDISBL        ;Disable the disk controller
FFE6 D308               out     DENABL

FFE8 210000             lxi     h,DMAADR
FFEB E5                 push    h

FFEC C37FFA             jmp     EXECDP          ;disable PROM and execute code

                ;***Subroutine*******************
                ; Find the RAMBUF address
                ; On Exit:
                ;   hl = RAM page item address
                ;   prior hl value is on the stack
                ;   Carry is clear
                ; trashes a
                ;*********************************
FFEF 3E7B       FNDBUF: mvi     a,RAMBUF

                ; Fall into RAMPAG

                ;***Subroutine*********************
                ; Set hl to location within RAM page
                ; On Entry:
                ;   a = address offset into RAM page
                ; On Exit:
                ;   hl = RAM page item address
                ;   prior hl value is on the stack
                ;   Carry is clear
                ;**********************************
FFF1 E3         RAMPAG: xthl                            ;save hl, get return address
FFF2 E5                 push    h                       ;restore return address
FFF3 210000             lxi     h,0
FFF6 39                 dad     sp                      ;get RAM page, clear carry
FFF7 6F                 mov     l,a                     ;requested RAM address
FFF8 C9                 ret

                ;============================================================
                ; Check for assembly problem: all of CDBL must fit in one page
```

```
                ;=================================================================
FFF9 =          DBLEND  equ     $

                if ((DBLEND-1)/256)-(DBLADR/256)
                        ERROR: CDBL does not fit in a single page

                endif
FFF9                    end
```