AMON USERS MANUAL

Amon Version 2.4

13 October 2016

Martin Eberhard

Revision History

Manual Revision	Firmware Revision	Author	Revision
28 Aug 2016	2.1	M. Eberhard	First released version
3 Oct 2016	2.2	M. Eberhard	Add IN and OT commands
10 Oct 2016	2.3	M. Eberhard	Better error messages, etc.
13 Oct 2016	2.4	M. Eberhard	Add GO record to AD command. Option to not GO on AL command.

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 FOR TRANSFERRING DATA

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 [<ADDR> [<COUNT>]] (DUMP MEMORY)

Dumps <Count> bytes memory on the Console in hexadecimal, starting at <Addr>, 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 [<ADDR>] (ENTER MEMORY DATA)

Allows you to enter 2-digit hex data into memory starting at <Addr>, 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 [<ADDR> [<OPTION>]] (EXECUTE)

Calls <Addr>, 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> [<ADDR> [<COUNT>]]] (FILL MEMORY)

Fills <Count> bytes of memory, starting at <Addr>, with <Value>, which is a 2-digit hex value. <Value> and <Addr> 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 <ADDR> <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 <ADDR1> <ADDR2> <COUNT> (VERIFY MEMORY)

Compares a block of memory starting at <Addr1> that is <Count> bytes long, to an equal-sized block of memory starting at <Addr2>. 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 <Addr> <Count> [<Go>](Dump Memory as Altair Binary File)

Dumps <Count> bytes of memory to the Transfer Port, starting at <Addr>, in Altair Absolute Binary format.

If a Go Address <GO> is provided, then the dump will be terminated with a GO record that contains this address. Otherwise, no GO record will be included.

AL [<0/1>] (Load and Execute Altair Binary File)

Loads an Altair Absolute Binary file via the Transfer Port and optionally jumps to its GO 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.

If no parameter is typed (or the parameter is not 0) then a GO record will cause execution at the GO address. If the optional parameter is 0 then a GO record will cause the GO address to be printed on the console and control returned to the monitor.

The Altair binary loader will terminate and print an error message for any of the following reasons. The error message will include a single-character error code and the 16-bit (4 hex digit) memory address associated with the error.

Error Code	Error Type	Explanation
C	Checksum Error	Record checksum is incorrect
M	Memory Error	Write to memory failed
O	Overwrite Error	Attempt to overwrite AMON's RAM

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

Dumps <Count> bytes of memory to the Transfer Port, starting at <Addr>, 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.

The HL command will terminate and print an error message for any of the following reasons. The error message will include a single-character error code and the 16-bit (4 hex digit) memory address associated with the error.

Error Code	**Error Type**	**Explanation**
C	Checksum Error	Hex record checksum is incorrect
H	Hex Error	Bad hex digit in input
M	Memory Error	Write to memory failed
O	Overwrite Error	Attempt to overwrite Amon’s RAM

Note that if a Hex Error is detected during the first four bytes of a hex record then the address in the error message will be meaningless. Otherwise, the address printed is the memory address associated with the failure (which includes the offset, if one was provided).

Loading terminates normally with any record that has 0 data bytes. You can also abort the load by typing CONTROL-C. If the load terminates normally then the total number of records loaded will be displayed on the console.

The maximum baud rate when loading a file with HL is 19200 baud.

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.)

I/O Commands

IN <Port> (Input from port)

The specified input port is read, and the result printed on the console. Note that if the ED switch is closed on the 88-2SIOJP then an “IN FF” will disable the AMON EPROM, and the software will most likely crash.

OT <Port> <Data> (Output to port)

The specified data is written to the specified output port.

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).

The floppy disk boot code will retry any sector with a checksum error up to 16 times before giving up. Booting will terminate and print an error message for any of the following reasons. The error message will include a single-character error code and the 16-bit (4 hex digit) memory address associated with the error.

Error Code	Error Type	Explanation
C	Checksum Error	Sector checksum is incorrect after 16 retries
M	Memory Error	Write to memory failed
O	Overwrite Error	Attempt to overwrite Amon’s RAM

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).

The hard disk boot code will terminate and print an error message if it gets an error from the Datakeeper disk controller. The error message will contain the 8-bit (2 hex digit) error code from the Datakeeper controller, and the 16-bit (4 hex digit) disk command that caused the error. See the Datakeeper documentation for interpretation of these error components.

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. When TT 0 is selected (the default), a backspace or delete key will cause the cursor to back up, erasing the previous character typed. When TT 1 is selected, a backspace or delete key will cause the previous character to be displayed between two slashes, indicating that this character has been deleted.

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. Upon successful load, HDBL code will input from the Altair's sense switches (port FFh) prior to executing the loaded code, to disable Amon's PROM, freeing all 65K of memory space for RAM (if the ED switch on the 88-2SIOJP is closed).

If loading from the hard disk fails, an error message will be printed, and control will pass to the monitor. See the HB command for a description of the error messages.

FE00H: BOOT FROM ALTAIR TAPE (MBL)

Entry at FE00h 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 the Altair's sense switches (port FFh) to determine the boot device. This input will also disable Amon's PROM, freeing all 65K of memory space for RAM (if the ED switch on the 88-2SIOJP is closed). The boot device is specified by three of the Altair's sense switches, 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)

Because the MBL code reads from the Altair's sense switches prior to loading, the monitor may be disabled when an error is detected. For this reason, if an error is detected then the error code will be printed out continuously on the console and also stored at address 0000. The memory address associated with the error will be stored at addresses 0001 and 0002.

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 the Altair's sense switches (port FFh) prior to executing the loaded code, to disable Amon's PROM, freeing all 65K of memory space for RAM (if the ED switch on the 88-2SIOJP is closed).

If an error occurs during loading from the floppy disk, then an error message will be printed, and control will pass to the monitor. See the BO command for a description of the error messages.

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' on tape has up to 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. If there is no Checksum loader then the Leader will be nulls.
2. The (optional) Checksum Loader, which is a program that is normally used to load the subsequent sections. This Loader is written backwards on the tape.
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’s AL command avoids problems with the different Checksum Loaders by ignoring the Checksum Loader on the tape, 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
;
; 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)
;===============================================================
;
; AMON assumes the console is on port 0 of the 88-2SIO/JP,
; and that the console terminal may optionally 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.)
; Commands (all values are in hex):
;
; AD <ADR> <BCNT> [<GO>]
;     Write <BCNT> bytes of memory starting at <ADR> in Altair
;     Absolute Binary format, to the current Transfer Port.
;     Optional GO record appended if <GO> provided.
;
; AL  [<0/1>]
;    Load and execute an Altair Absolute Binary file from the
;    current Transfer Port. (This is MBL.) If the optional
;    parameter is 0 then the GO record in the file will be
;    ignored, and control returns to the monitor, after
;    printing the GO address on the console. Parameter defaults
;    to 1 (meaning a Go record is executed).
;
; 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.
;
; IN <PORT>
;    Read from <PORT> and print the result on the console
;
; OT <PORT> <VAL>
;    Write the specified value to the specified output port
;
; 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
;
```

```
               ; Amon finds and uses the highest contiguous 256-byte page of
               ; RAM for its stack, buffers, and code that gets modified (such
               ; as the serial I/O routines).
               ;
               ; When MBL is executed directly (not via a call from AMON), it
               ; reads the switch register to determine the boot port. Note
               ; that the 88-2SIOJP may be configured (by closing the ED
               ; switch) to disable the PROM once an "IN OFFh" (input from the
               ; front panel switch register) is executed.
               ;
               ; The sector buffer is positioned within the RAM page such that
               ; its last byte is the last byte of the RAM page. This makes
               ; the timing work in the critical byte-read loop, when booting
               ; from an 8" floppy diskette.
               ;
               ; Organization:
               ;
               ; 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
               ; Vers. 2.1  M. Eberhard  27 August 2016
               ;   Fix bug when executing at F800
               ; Vers. 2.2  M. Eberhard 4 October 2016
               ;   Squeeze code a bit, add IN and OT commands
               ; Vers. 2.3 M. Eberhard  10 October 2016
               ;   Squeeze code, improve error reporting, improve comments
               ; Vers. 2.4 M. Eberhard 13 October 2016
               ;   Unify error messages, add GO record to AD command, add
               ;   option to ignore GO record on AL command, verify memory
               ;   write on HL command
               ;==============================================================
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
0048 =         HERMSG  equ     'H'     ;Illegal hex digit
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)

               ;---------------
               ;88-2SIO 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

               ;--------------
               ;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

                ;--------------------------------------------
                ;88-4PIO 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

                ;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

                ;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

                ;4PIO Initialization

002C =          P4INIT  equ     P4C2C5+P4C2C3+P4DDR     ;2Ch
                                        ;bits 0,1: C1 input active low, int off
                                        ;bit 2: access data reg
                                        ;bits 3-5: C2 output handshake

                ;------------------------------------------------------------
                ;Altair 8800 Floppy 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 =          STEPOT    equ     02H             ;Step-Out
0004 =          HEDLOD    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

               ;------------------------------------
               ;88-HDSK Datakeeper Hard Disk Equates
               ;------------------------------------

               ;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 available 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

0024 =         BINIT   equ     24h     ;bits 0,1: C1 input active low, int off
                                       ;bit 2: access data reg
                                       ;bits 3-5: C2 input handshake


002C =         CINIT   equ     2Ch     ;bits 0,1: C1 input active low, int off
                                       ;bit 2: access data reg
                                       ;bits 3-5: C2 output handshake

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

               ;*****************
               ;Memory Allocation
               ;*****************
0000 =         DMAADR  equ     00000h          ;Disk load/execution address
                                               ;(Code assumes DMAADR=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 AMON's RAM page
               ;---------------------------------------------------
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

               ;Floppy disk sector buffer component offsets

007C =         SFSIZE  equ     RAMBUF+1        ;file size
007E =         SDATA   equ     RAMBUF+HDRSIZ   ;sector data
00FE =         SMARKR  equ     SDATA+BPS       ;marker byte
00FF =         SCKSUM  equ     SMARKR+1        ;checksum byte

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

F800 01BCF8            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:
               ;    e = 0
               ;   sp = address of new stack
               ;   All standard Altair I/O ports initialized
               ;   interrupts disabled
               ; Trashes psw,d,hl
               ;*********************************************
F803 F3        INIT:   di                      ;no interrupts please

               ;----------------------------------------------
               ;Hunt for the highest RAM page
               ;This assumes at least one 256-byte page of RAM
               ;and that if one byte within each page is RAM
               ;then the other 255 bytes are RAM too.
               ;----------------------------------------------
F804 2100FF            lxi     h,0FF00h

F807 24        CSLOOP: inr     h               ;next RAM page

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

F811 25                dcr     h               ;point to last good RAM page

               ;-----------------------------------------
               ;Relocate  and Install RAM code
               ;This loop moves more bytes than necessary
               ;to install the actual RAM code. The extra
               ;bytes land in the (uninitialized) stack
               ;space and buffer space.
               ;On Entry:
               ;   h = destination address high byte
               ; On Exit:
               ;   e = 0
               ;-----------------------------------------
F812 1153F8            lxi     d,RIOCOD        ;RAM code source

F815 1A        RCLOOP: ldax    d
F816 BA                cmp     d               ;need to relocate an address?
F817 C221F8            jnz     RCL1

F81A 2B                dcx     h               ;back up to fix low address byte
F81B 7E                mov     a,m
F81C D653              sui     (RIOCOD-RAMCOD) and 0FFh ;low byte of offset
F81E 77                mov     m,a
F81F 23                inx     h

F820 7C                mov     a,h             ;relocate high byte

F821 77        RCL1:   mov     m,a
```

```
F822 2C                inr     l
F823 1C                inr     e               ;end with e=0 for INIT exit
F824 C215F8            jnz     RCLOOP

               ;-----------------------------------------------
               ;Create stack, and push the given return address
               ;-----------------------------------------------
F827 2E7B              mvi     l,STACK         ;put stack in RAM page
F829 F9                sphl
F82A C5                push    b               ;push our return address

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

F82B AF                xra     a
F82C D320              out     P4CA0           ;access 4PIO port 0A DDR
F82E D321              out     P4DA0           ;set 4PIO port 0A as input

F830 D322              out     P4CB0           ;access 4PIO port 0B DDR
F832 2F                cma                     ;0FFh
F833 D323              out     P4DB0           ;set 4PIO port 0B as output

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

F835 3E2C              mvi     a,P4INIT
F837 D320              out     P4CA0           ;4PIO port 0A control
F839 D322              out     P4CB0           ;4PIO port 0B control

               ;Send reset command to both 2SIO ports

F83B 3E03              mvi     a,S2RST         ;2SIO reset
F83D D310              out     S2CTLA          ;2SIO port 0
F83F D312              out     S2CTLB          ;2SIO port 1

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

F841 3E11              mvi     a,S22STP        ;8N2, /16
F843 D310              out     S2CTLA          ;2SIO port 0 control
F845 D312              out     S2CTLB          ;2SIO port 1 control

               ;----------------------------------------------------
               ;Fall into SETTP to set the default transfer port and
               ;"Return" to the address provided in bc on entry.
               ;----------------------------------------------------
F847 2E06              mvi     l,DTPORT        ;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 (upper digit ignored)
               ; Trashes psw,bc,hl
               ;**************************************
F849 7D        SETTP:  mov     a,l                     ;get port
F84A E607              ani     7                       ;make it a legal value

F84C 210000            lxi     h,0                     ;find address of RSETP
F84F 39                dad     sp                      ;...located in RAM
F850 2E00              mvi     l,RSETP-RIOCOD+RAMCOD
F852 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 versions 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,bc,hl
               ;-------------------------------------------------------------
F853 019BF8    RSETP:  lxi     b,PTABLE        ;lookup table

F856 87                add     a               ;4 bytes/entry
F857 87                add     a
F858 81                add     c               ;look up in PTABLE (clr carry)
F859 4F                mov     c,a             ;bc=PTABLE(port value)

               ;Set up the input port routine

F85A 0A                ldax    b               ;input data port & CMA flag
F85B 1F                rar                     ;move CMA flag into Carry
F85C 328CF8            sta     TPIDP+1         ;install data port address

               ;hl gets the status port (in l) and either NOP or CMA (in h)

F85F 2600              mvi     h,NOP           ;NOP instruction
F861 D266F8            jnc     RSETP1
F864 262F              mvi     h,CMA           ;CMA instruction
               RSETP1:

F866 3D                dcr     a               ;status port = data port-1
F867 6F                mov     l,a             ;install status port address

               ;Set the status port and either NOP or CMA instruction

F868 2280F8            shld    TPISP+1         ;status port and NOP/CMA

F86B 0C                inr     c               ;next table entry is
F86C 0A                ldax    b               ;..the data available mask
F86D 3283F8            sta     TPIMSK+1        ;install mask

               ;Set up the output port routine
```

```
F870 0C                 inr     c                   ;next table entry is
F871 0A                 ldax    b                   ;..the data output port address
F872 3299F8             sta     TPODP+1             ;install data port address

F875 3D                 dcr     a                   ;status port = data port-1
F876 6F                 mov     l,a                 ;install stat port address
F877 2290F8             shld    TPOSP+1             ;status port and NOP/CMA

F87A 0C                 inr     c                   ;next table entry is
F87B 0A                 ldax    b                   ;..the transmitter ready mask
F87C 3293F8             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
                ;   a=0 and Z set if not
                ;---------------------------------
                RTPIS:
F87F DB10       TPISP:  in      S2STAA              ;(status port address)read status
F881 00         TPINOP: nop                         ;(may get modified to CMA)
F882 E601       TPIMSK: ani     S2RDF               ;(port mask)
F884 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
                ;----------------------------------------------
F885 CD7FF8     RTPIN:  call    RTPIS
F888 CA85F8             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
F88B DB11       TPIDP:  in      S2RXDA              ;(data port place)get data byte

F88D 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
               ;---------------------------------
F88E F5        RTPOUT: push    psw

               WAITPO:
F88F DB10      TPOSP:  in      S2STAA          ;(status port address)read status
F891 00        TPONOP: nop                     ;(may get modified to CMA)
F892 E602      TPOMSK: ani     S2TBE           ;(Tx port mask)
F894 CA8FF8            jz      WAITPO

F897 F1                pop     psw
F898 D311      TPODP:  out     S2TXDA          ;(data port place)
F89A 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
               ;------------------------------------------------------------
F89B 22011102  PTABLE: db      S2RXDA*2,S2RDF,S2TXDA,S2TBE     ;0:2SIO A
F89F 22011102          db      S2RXDA*2,S2RDF,S2TXDA,S2TBE     ;1:2SIO A
F8A3 03010180          db      SIORXD*2+1,SIOIDR,SIOTXD,SIOODR ;2:SIO
F8A7 0F010780          db      ACRRXD*2+1,ACRIDR,ACRTXD,ACRODR ;3:ACR
F8AB 42802380          db      P4DA0*2,P4RDF,P4DB0,P4TDE       ;4:4PIO port 0
F8AF 0A020501          db      PIORXD*2,PIORDF,PIOTXD,PIOTDE   ;5:PIO
F8B3 26011302          db      S2RXDB*2,S2RDF,S2TXDB,S2TBE     ;6:2SIO B

               ;8th entry is a custom port, defined above

F8B7 22011102          db      CPRDAT*2+CPSPOL,CPRRDY,CPTDAT,CPTRDY

               ;==============================================================
               ; RAM Variables
               ;==============================================================
F8BB 00        TTYPE:  db      0       ;0 (even) means terminal (backspacing)
                                       ;1 (odd) means Teletype (no backspace)

               ;===Assembly Check=======================================
               ; All of RIOCOD must be in the same 256-byte page of PROM
               ;========================================================
F8BC =         RCEND   equ     $

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

               ;===Assembly Check======================
               ; 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   =
               ;==========================================
F8BC CD47FD    INIT2:  call    CILPRT          ;print banner
F8BF 414D4F4E20        db      'AMON  2.4 by M. Eberhard',CR,LF
F8D9 52414D3AA0        db      'RAM:',' '+80h

               ;CILPRT returns with Z flag cleared
               ;Announce address of the first byte of RAM page

F8DE 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
               ;***************************************************
F8DF CDE9FF    CABORT: call    RAMPAG          ;find stack (a=0 if Z set)
F8E2 CCF2FC            cz      PHLCHX          ;Perhaps print hl on console

F8E5 2E7B              mvi     l,STACK         ;point to bottom of stack
F8E7 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

F8E8 01E8F8    MAIN:   lxi     b,MAIN          ;create command-return
F8EB C5                push    b               ;..address on the stack

F8EC CD47FD            call    CILPRT          ;print CR,LF, prompt
F8EF BE                db      PROMPT+80h

F8F0 CDD9FB            call    GETLIN          ;get user input line
                                               ;de=beginning of line
                                               ;Z set if no character found
                                               ;0 at end of line
```

```
F8F3 F3                di                      ;INTE light off (cancel error)
F8F4 C8                rz                      ;No command? just ignore.

               ;Check command list, and execute the command if found

F8F5 EB                xchg                    ;command address to hl
F8F6 11B5FD            lxi     d,COMTAB-2      ;point to command table

F8F9 4E                mov     c,m             ;1st command chr in c
F8FA 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.

F8FB 13        NXTCOM: inx     d               ;skip over address offset
F8FC 13                inx     d
F8FD 1A                ldax    d
F8FE B7                ora     a               ;test for table end
F8FF CAF4FF            jz      CMDERR          ;not in table

F902 A9                xra     c               ;test first character
F903 47                mov     b,a             ;temp save result
F904 13                inx     d               ;2nd table character
F905 1A                ldax    d
F906 AE                xra     m               ;test 2nd character

F907 13                inx     d               ;point to address offset

F908 B0                ora     b                       ;both characters match?
F909 E6DF              ani     ('a'-'A') XOR 0FFh      ;lowercase is ok
F90B C2FBF8            jnz     NXTCOM                  ;NO match: keep looking

F90E 23                inx     h               ;skip past 2-letter command

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

F90F EB                xchg                    ;(hl)=address of cmd routine
                                               ;de=input pointer

F910 4E                mov     c,m             ;address low byte
F911 23                inx     h

F912 7E                mov     a,m             ;address high byte
F913 F680              ori     80h             ;clear non-hex flag bit
F915 47                mov     b,a             ;..to make legit address

F916 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.

F917 BE                cmp     m               ;Non-hex?
F918 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 chr after the 1st parameter.
               ;'return' to the Command Routine on the stack.

               ;skip into FNDHEX

F919 21                db      21h             ;'lxi h' opcode skips 2
```

```
               ;***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
               ;   top-of-stack = prior hl value
               ;   Z set, Carry clear if value
               ;   Carry set and a=hl=0 if no value found
               ;****************************************************
F91A E3        PHFHEX: xthl                     ;push hl
F91B E5                push    h                ;..beneath return address

               ;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:
               ;   de advanced past character
               ;   top-of-stack = prior hl value
               ;   Z set, Carry clear if value
               ;   Carry set and a=hl=0 if no value found
               ;****************************************************
F91C 210000    FNDHEX: lxi     h,0              ;default value
F91F CDEDFB            call    SKIPB            ;skip spaces to find 1st digit
F922 37                stc                      ;Carry set if no digits
F923 C8                rz

F924 1A        FHEXLP: ldax    d                ;get digit
F925 B7                ora     a                ;end of line?
F926 C8                rz                       ;y: ret with carry clear

F927 FE20              cpi     ' '              ;value separator?
F929 C8                rz                       ;y: ret with carry clear

F92A FE41              cpi     'A'              ;convert letters to uppercase
F92C DA31F9            jc      FHNUM
F92F E6DF              ani     ('a'-'A') XOR 0FFh
               FHNUM:

F931 29                dad     h                ;make room for the new digit
F932 29                dad     h
F933 29                dad     h
F934 29                dad     h

F935 CD6EFE            call    HEXCON           ;Do the conversion
F938 D2F4FF            jnc     CMDERR           ;not valid hexidecimal value?

F93B 85                add     l
F93C 6F                mov     l,a              ;move new digit in
F93D 13                inx     d                ;bump the pointer
F93E C324F9            jmp     FHEXLP

               ;***Command Routine*************************************
               ; AD <SRC> <BCNT> [<GO>]
               ; (Dump memory in Altair binary format)
               ; On Entry:
               ;   hl=<SRC>
               ;   Carry set if none entered
               ;   de points to <BCNT>
               ;   TP command has set up the Transfer Port
               ;******************************************************
F941 CDEFFF    ADUMP:  call    GETHEX          ;save <SRC>, get <BCNT>
F944 CD1AF9            call    PHFHEX          ;save <BCNT>, get <GO>

F947 D1                pop     d               ;get de=<BCNT>
F948 E3                xthl                    ;save <GO>, get <SRC>
F949 F5                push    psw             ;Carry set if no <GO> provided

F94A EB                xchg                    ;de= <SRC>, hl=<BCNT>

               ;de = source address
               ;hl = byte count

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

F94B 3E20              mvi     a,20h           ;punch 20h as the pre-leader
F94D CD88F9            call    LEADER

               ;Punch null leader

F950 CD87F9            call    LEADR0          ;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 CD96FB            call    TPOED

F961 D1                pop     d               ;restore load address
F962 CD96FB            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 CD9BFB            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 CD9BFB             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

                ;Punch a GO record, if the user asked for one
F97A F1                 pop     psw             ;carry set if no <GO> provided
F97B D1                 pop     d               ;Go address
F97C DA87F9             jc      LEADR0          ;no go record?

F97F 3E78               mvi     a,ALTEOF        ;Go record sync chr
F981 CD9BFB             call    TPOUT           ;punch it

F984 CD96FB             call    TPOED           ;Punch de=go address

                ;Fall into LEADR0 to punch a null trailer and return to MAIN

                ;---Local Subroutine-------------
                ; Punch a null leader
                ; On Exit:
                ;   a=0
                ;   b=0
                ;   all other registers preserved
                ;--------------------------------
F987 AF         LEADR0: xra     a               ;leader chr

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

                ;---Local Subroutine-------------
                ; Punch a leader
                ; On Entry:
                ;   a = leader character
                ; On Exit:
                ;   b=0
                ;   all other registers preserved
                ;--------------------------------
F988 063C       LEADER: mvi     b,LDRLEN        ;leader length

F98A CD9BFB     LEADLP: call    TPOUT
F98D 05                 dcr     b
F98E C28AF9             jnz     LEADLP          ;ends with b=0

F991 C9                 ret

                ;***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
               ;***************************************************
F992 CDEFFF    MCOPY:  call    GETHEX          ;save source, get destination
F995 CDEFFF            call    GETHEX          ;save dest, get byte count

F998 CD47FD            call    CILPRT
F99B 436F707969        db      'Copyin','g'+80h

F9A2 CD1AF9            call    PHFHEX          ;save <BCNT>, get <RPT>
F9A5 7D                mov     a,l             ;default to 1
F9A6 CE00              aci     0               ;Carry if no value given

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

F9A8 C1        MCRLP:  pop     b               ;bc=count
F9A9 D1                pop     d               ;de=destination
F9AA E1                pop     h               ;hl=source

F9AB E5                push    h               ;save source
F9AC D5                push    d               ;save Dest
F9AD C5                push    b               ;save count

F9AE F5                push    psw             ;save a=repeat count

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

F9AF 7E        MCLOOP: mov     a,m
F9B0 12                stax    d
F9B1 23                inx     h
F9B2 13                inx     d
F9B3 0B                dcx     b
F9B4 78                mov     a,b
F9B5 B1                ora     c
F9B6 C2AFF9            jnz     MCLOOP

               ;Repeat the copy as requested by the user

F9B9 F1                pop     psw             ;recover repeat count
F9BA 3D                dcr     a               ;repeat as requested

               ;Print a pacifier dot for all but the last pass
               ;(This eliminates the dot for a single-pass copy)
F9BB 47                mov     b,a             ;temp save repeat count
F9BC 3E2E              mvi     a,PCFIER
F9BE C48EF8            cnz     PRINTA          ;preserves all regs
F9C1 78                mov     a,b             ;repeat count

F9C2 C2A8F9            jnz     MCRLP

F9C5 C3CFF9            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
               ;***************************************************
F9C8 CDEFFF    VERCMD: call    GETHEX          ;save <SRC>, get <DST>
F9CB CDEFFF            call    GETHEX          ;save <DST>, get <BCNT>
F9CE E5                push    h               ;save <BCNT>

               ;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)
               ;*****************************************
F9CF C1        VERIFY: pop     b               ;byte count
F9D0 E1                pop     h               ;hl=destination
F9D1 D1                pop     d               ;de=source

F9D2 CD47FD            call    CILPRT
F9D5 436865636B        db      'Checkin','g'+80h

               ;Loop to compare memory, reporting mismatches

F9DD 1A        VLOOP:  ldax    d               ;get expected data
F9DE BE                cmp     m               ;match?
F9DF C4A2FB            cnz     MERROR          ;N: error

F9E2 23                inx     h
F9E3 13                inx     d
F9E4 0B                dcx     b
F9E5 78                mov     a,b
F9E6 B1                ora     c
F9E7 C2DDF9            jnz     VLOOP

F9EA C9                ret

               ;***Command Routine*********************************
               ; SE <ADR> <BYTE1> [<BYTE2> [<BYTEn>]]
               ;    Search for string of bytes, starting at <ADR>
               ;    <BYTEn> can be either hex byte or 'text string'
               ; 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 the RAM buffer

F9EB CDE5FF            call    FNDBUF          ;push hl, find RAM buffer

F9EE E5                push    h               ;binary string address
F9EF 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
               ;------------------------------------------------
F9F2 CDEDFB    SCHLUP: call    SKIPB           ;returns a=found chr, 0 if none

F9F5 B9                cmp     c               ;is 1st chr a quote?
F9F6 CC5AFA            cz      SSTRNG          ;y:search for a string
F9F9 C467FA            cnz     SCHHEX          ;n: search for hex
                                               ;returns carry set if end
F9FC D2F2F9            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
               ;----------------------------------------
F9FF D1                pop     d               ;binary string address
FA00 E1                pop     h               ;search start address

FA01 78                mov     a,b             ;anything to search for?
FA02 B7                ora     a
FA03 CAF4FF            jz      CMDERR          ;error if not

FA06 E5        SLOOP1: push    h               ;search start address
FA07 D5                push    d               ;binary string address

FA08 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

FA09 7A        SLOOP2: mov     a,d
FA0A BC                cmp     h               ;don't search our own RAM page
FA0B CA45FA            jz      NOMTCH

FA0E 1A                ldax    d               ;search string
FA0F BE                cmp     m               ;current RAM
FA10 C245FA            jnz     NOMTCH

FA13 23                inx     h               ;test next byte
FA14 13                inx     d
FA15 0D                dcr     c               ;tested all bytes yet?
FA16 C209FA            jnz     SLOOP2

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

FA19 D1                pop     d               ;binary string address
FA1A E1                pop     h               ;search start address

FA1B CD47FD            call    CILPRT
FA1E 466F756E64        db      'Found',' '+80h
FA24 C5                push    b
FA25 CDF2FC            call    PHLCHX          ;print match address, trash bc
FA28 C1                pop     b

FA29 CD47FD            call    CILPRT
FA2C 4D6F726520        db      'More (Y/N)?',' '+80h

FA38 CDD2FB            call    GETKBD          ;user response
FA3B CD8EF8            call    PRINTA          ;echo

FA3E F620              ori     ('y'-'Y')       ;make it lowercase
FA40 FE79              cpi     'y'
FA42 C0                rnz                     ;anything but y ends
```

```
FA43 E5                 push    h               ;search start address
FA44 D5                 push    d               ;binary string address

               ;Search again, starting at the next byte after hl.
               ;Quit if we've reached the end of memory, FFFFh

FA45 D1        NOMTCH: pop     d               ;binary string address
FA46 E1                 pop     h               ;search start address

FA47 23                 inx     h               ;next RAM
FA48 7C                 mov     a,h
FA49 B5                 ora     l               ;End of memory?
FA4A C206FA             jnz     SLOOP1

FA4D CD47FD             call    CILPRT
FA50 4E6F742066         db      'Not foun','d'+80h

FA59 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
               ;-------------------------------------------
FA5A 13        SSTRNG: inx     d               ;skip over quote

FA5B 1A        STLOOP: ldax    d
FA5C B7                 ora     a               ;end quote is not required
FA5D C8                 rz

FA5E 13                 inx     d               ;point past this input chr

FA5F B9                 cmp     c               ;end of string?
FA60 C8                 rz

FA61 77                 mov     m,a             ;store a string byte
FA62 23                 inx     h
FA63 04                 inr     b

FA64 C35BFA             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
               ;---------------------------------------------------
FA67 CD1AF9    SCHHEX: call    PHFHEX          ;save next string addr byte,
                                                ;get a value.
                                                ;hl=0 & carry set if none

FA6A 24                 inr     h               ;no high byte allowed
FA6B 25                 dcr     h               ;does not change carry
FA6C C2F4FF             jnz     CMDERR

FA6F 7D                 mov     a,l             ;binary value
FA70 E1                 pop     h               ;next string address byte
```

```
FA71 D8                rc                      ;carry set means end of input

FA72 77                mov     m,a             ;store the hex digit
FA73 23                inx     h
FA74 04                inr     b

FA75 C9                ret

               ;***Command Routine*****************************
               ; TT [0/1] Set Terminal Type
               ;   0 (even) means backspacing works
               ;   1 (odd) means no backspacing (e.g. Teletype)
               ; On Entry:
               ;   l = 0 or 1 (odd or even really)
               ;***********************************************
FA76 4D        SETTT:  mov     c,l             ;user value

FA77 3E68              mvi     a,TTYPE-RIOCOD+RAMCOD
FA79 CDE9FF            call    RAMPAG

FA7C 71                mov     m,c             ;remember terminal type
FA7D 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
               ;***************************************
FA7E CD1AF9    EXEC:   call    PHFHEX          ;save <ADR>, get l=<OPT>
FA81 2D                dcr     l               ;anything but 1
FA82 C0                rnz                     ;..just executes at <ADR>

               ; Fall into EXECDP

               ;***Exit******************************
               ;Execute "IN FF" and then jump to code
               ;(This disables PROM)
               ; On Entry:
               ;   execution address is on stack
               ;*************************************
FA83 1EC9      EXECDP: mvi     e,RET           ;RET opcode, <don't care>
FA85 D5                push    d               ;..onto stack

FA86 11DBFF            lxi     d,0FFDBh        ;IN FF opcode
FA89 D5                push    d               ;..onto stack

FA8A 210000            lxi     h,0
FA8D 39                dad     sp              ;point hl to our code

FA8E D1                pop     d               ;point sp to <ADR>
FA8F D1                pop     d
FA90 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
               ;*****************************************************
FA91 CD1AF9    DUMP:   call    PHFHEX          ;save <ADR>, get hl=<BCNT>
FA94 7D                mov     a,l             ;low byte
FA95 CE00              aci     0               ;default to 1
FA97 6F                mov     l,a

FA98 EB                xchg                    ;de has byte count
FA99 E1                pop     h               ;recover start address

               ;Print the address at the beginning of each line

FA9A CD5FFE    DLINE:  call    PHLADR          ;print hl as an address
                                               ;Sets b=0, trashes c

               ;Print 16 bytes of hex data separated by spaces

FA9D E5                push    h               ;save for ASCII dump
FA9E D5                push    d

FA9F 7E        DLOOP:  mov     a,m             ;get the character
FAA0 CD43FE            call    PAHEX           ;TO console in hex (b=0)

FAA3 CD4CFD            call    ILPRNT          ;print a space
FAA6 A0                db      ' '+80h

FAA7 23                inx     h               ;next address

FAA8 1B                dcx     d               ;all done?
FAA9 7A                mov     a,d
FAAA B3                ora     e
FAAB CAB4FA            jz      DLDONE          ;Y: done with command

FAAE 3E0F              mvi     a,0Fh           ;new line every XXX0 hex
FAB0 A5                ana     l

FAB1 C29FFA            jnz     DLOOP           ;not zero if more for this line

FAB4 D1        DLDONE: pop     d               ;recover count and address
FAB5 E1                pop     h

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

FAB6 CD4CFD            call    ILPRNT          ;pretty space
FAB9 A0                db      ' '+80h

FABA 7E        ADLOOP: mov     a,m             ;get the character
FABB 3C                inr     a               ;del (7F) is also nonprinting
FABC E67F              ani     7Fh             ;clear parity
FABE FE21              cpi     ' '+1           ;everything below space
FAC0 D2C5FA            jnc     PRNTBL          ;..is nonprinting

FAC3 3E2F              mvi     a,'.'+1         ;dot for non-printing

FAC5 3D        PRNTBL: dcr     a               ;undo inc
```

```
FAC6 CD8EF8            call    PRINTA          ;Print ASCII or dot

FAC9 1B                dcx     d               ;all done?
FACA 7A                mov     a,d
FACB B3                ora     e
FACC C8                rz                      ;done with command

FACD 23                inx     h               ;next line?
FACE 3E0F              mvi     a,0Fh           ;new line every XXX0 hex
FAD0 A5                ana     l

FAD1 C2BAFA            jnz     ADLOOP          ;not zero if more for this line

               ;Give the user a chance to pause or quit at the end of each line

FAD4 CDCCFB            call    CKPAUS          ;Pause or abort?
FAD7 C39AFA            jmp     DLINE           ;next 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
               ;*****************************************************
FADA 4D        FILMEM: mov     c,l             ;<VAL> c

FADB CD1CF9            call    FNDHEX          ;get <ADR>, default 0
FADE CD1AF9            call    PHFHEX          ;save <ADR>, get hl=<BCNT>
FAE1 EB                xchg                    ;de has byte count

FAE2 CDE9FF            call    RAMPAG          ;find our RAM
FAE5 44                mov     b,h             ;b remembers RAM page

FAE6 E1                pop     h               ;hl has start address

               ;Loop to fill memory, quitting if RAM page

FAE7 7C        FMLOOP: mov     a,h
FAE8 B8                cmp     b               ;Filling RAM page?
FAE9 C8                rz                      ;y: done

FAEA 71                mov     m,c
FAEB 23                inx     h

FAEC 1B                dcx     d               ;done yet?
FAED 7A                mov     a,d
FAEE B3                ora     e
FAEF C2E7FA            jnz     FMLOOP

FAF2 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. Nulls from the keyboard
               ; are ignored.
               ; <EXCHR> on the keyboard to exit
               ;  (defaults to DTEXIT)
               ;*****************************************************
FAF3 CD4CFD    TERMNL: call    ILPRNT          ;announce exit character
FAF6 457869743A        db      'Exit: ','^'+80h

FAFD CDEDFB            call    SKIPB           ;get optional exit character
FB00 C205FB            jnz     TMNL1           ;Got an exit value in a
FB03 3E03              mvi     a,DTEXIT        ;default abort

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

FB05 E61F      TMNL1:  ani     1Fh             ;make it a control chr
FB07 6F                mov     l,a             ;remember exit character

FB08 F640              ori     'C'-CTRLC       ;make it printable
FB0A CD8EF8            call    PRINTA

FB0D CD47FD            call    CILPRT          ;CR,LF,LF to be pretty
FB10 8A                db      LF+80h

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

FB11 CD7FF8    TLOOP:  call    KSTAT           ;anything typed?
FB14 C45AFD            cnz     KDATA           ;Y:get the keyboard data

FB17 BD                cmp     l               ;exit character?
FB18 C8                rz                      ;Y: done

FB19 B7                ora     a               ;anything typed? (ignore nulls)
FB1A C49BFB            cnz     TPOUT           ;KBD data to Transfer Port

FB1D CD7CFD            call    TESTTP          ;Transfer Port = console?
                                               ;Z set if so

FB20 C471FD            cnz     TPISTA          ;Any Transfer Port data?
                                               ;NZ if so

FB23 C475FD            cnz     TPIN            ;get Transfer Port data
                                               ;always returns W/ nz
FB26 C48EF8            cnz     PRINTA          ;and send it to console
FB29 C311FB            jmp     TLOOP

               ;***Command Routine**********************************
               ; OT <PORT> <DATA> (Output to port)
               ;
               ; On Entry:
               ;    l=PORT
               ;    de points to DATA
               ;
               ; Creates this routine on the stack, then executes it
               ;
               ;       NOP
               ;       MVI  a,<DATA>
               ;       OUT  <PORT>
               ;       RET
               ;****************************************************
FB2C 26C9      OPORT:  mvi     h,RET           ;opcode
FB2E CD1AF9            call    PHFHEX          ;push <PORT>, RET opcode
```

```
                                               ;Get l=<DATA>

FB31 26D3              mvi     h,OUT           ;opcode
FB33 E5                push    h               ;data, OUT opcode

FB34 21003E            lxi     h,3E00h         ;NOP, MVI A, opcodes
FB37 E5                push    h

FB38 65                mov     h,l             ;hl=0
FB39 39                dad     sp              ;hl points to routine

FB3A D1                pop     d               ;fix stack
FB3B D1                pop     d
FB3C D1                pop     d
FB3D E9                pchl                    ;execute RAM routine

               ;***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
               ;***********************************************************
FB3E CDEFFF    HEXDMP: call    GETHEX          ;save <ADR>, get hl=<BCNT>
FB41 CD1AF9            call    PHFHEX          ;save <BCNT>, get hl=<OFST>

FB44 E3                xthl                    ;hl=byte count
FB45 C1                pop     b               ;bc=offset
FB46 D1                pop     d               ;de= start address
FB47 C5                push    b               ;address offset onto stack

               ;Loop to send requested data in HRLEN-byte records

               ;send record-start

FB48 D5        HDLINE: push    d               ;print CRLF
FB49 CD93FB            call    TPCRLF
FB4C D1                pop     d

FB4D 3E3A              mvi     a,':'
FB4F CD9BFB            call    TPOUT

               ;Compute this record byte count

FB52 0610              mvi     b,HRLEN         ;default bytes/line

FB54 7D                mov     a,l             ;short last line?
FB55 90                sub     b               ;normal bytes/line
FB56 7C                mov     a,h
FB57 DE00              sbi     0
FB59 D25DFB            jnc     HDLIN1          ;N: full line

FB5C 45                mov     b,l             ;Y:short line
               HDLIN1:

               ;If byte count is 0 then go finish EOF record

FB5D 78                mov     a,b
FB5E B7                ora     a
FB5F CA88FB            jz      HDEOF
```

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

FB62 CD3FFE             call    PAHEXC          ;send byte count

FB65 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)

FB66 E3                 xthl                    ;hl=address offset
                                                ;remaining byte count on stack
FB67 E5                 push    h               ;save address offset

FB68 19                 dad     d               ;compute address with offset
FB69 CD37FE             call    PHLHEX          ;send address with offset

FB6C E1                 pop     h               ;recover address offset
FB6D E3                 xthl                    ;offset on stack,
                                                ;remaining byte count to hl

                ;Send the record type (00)

FB6E AF                 xra     a
FB6F CD3FFE             call    PAHEXC

                ;Send b bytes of hex data on each line, computing
                ;the checksum in c. b>0 here.

FB72 1A         HDLOOP: ldax    d               ;get the character
FB73 CD3FFE             call    PAHEXC          ;send to Transfer Port
                                                ;(b<>0)
FB76 2B                 dcx     h
FB77 13                 inx     d
FB78 05                 dcr     b               ;next
FB79 C272FB             jnz     HDLOOP

                ;Send the checksum (with b<>0)

FB7C AF                 xra     a
FB7D 91                 sub     c
FB7E 04                 inr     b               ;b<>0 means Transfer Port
FB7F CD3FFE             call    PAHEXC

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

FB82 CD63FD             call    CHKKBD          ;abort if user says so

                ;Next record

FB85 C348FB             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
                ;--------------------------------------------
FB88 C1         HDEOF:  pop     b               ;chuck address offset
FB89 0605               mvi     b,5             ;5 bytes for EOF

FB8B AF         HDELP:  xra     a
FB8C CD43FE             call    PAHEX           ;b<>0 for Transfer Port
```

```
FB8F 05                 dcr     b
FB90 C28BFB             jnz     HDELP

                ;Fall into TPCRLF

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

                ;***Subroutine*****************
                ; Send CRLF to the transfer port
                ; Trashes de
                ;******************************
FB93 110D0A     TPCRLF: lxi     d,LF*256+CR

                ;Fall into TPOED

                ;***Subroutine*********************
                ; Send e then d to the transfer port
                ; On Exit:
                ;   a=d
                ;**********************************
FB96 7B         TPOED:  mov     a,e
FB97 CD9BFB             call    TPOUT
FB9A 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
                ;**********************************
FB9B CDE7FF     TPOUT:  call    HRMPAG                  ;don't mess up a
FB9E 2E3B               mvi     l,RTPOUT-RIOCOD+RAMCOD  ;hl points to RTPOUT

FBA0 E3                 xthl                            ;restore hl, put
                                                        ;..address on stack
FBA1 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
                ;************************************
FBA2 C5         MERROR: push    b
FBA3 F5                 push    psw             ;save source data

FBA4 CD47FD             call    CILPRT
FBA7 3FBA               db      '?',':'+80H
FBA9 CDF2FC             call    PHLCHX          ;Print address in hl on console,
                                                ;..trash c, set b=0

FBAC CD4CFD             CALL    ILPRNT
FBAF 2045787065         db      ' Expected',' '+80H
```

```
FBB9 F1                 pop     psw             ;recover source data
FBBA CD43FE             call    PAHEX

FBBD CD4CFD             call    ILPRNT
FBC0 2C20726561         db      ', read',' '+80H
FBC7 7E                 mov     a,m             ;Get destination data
FBC8 CD43FE             call    PAHEX

FBCB C1                 pop     b

                ;Fall into CKPAUS

                ;***Subroutine*******************************
                ; Get a keyboard character, abort if control-C
                ; pause (until anything else typed) if space
                ; On Exit:
                ;    a=keyboard character, Z cleared
                ;********************************************
FBCC CD63FD     CKPAUS: call    CHKKBD          ;Abort or pause?

FBCF FE20               cpi     PAUKEY          ;Pause?
FBD1 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
                ;********************************************
FBD2 CD63FD     GETKBD: call    CHKKBD          ;get KBD character, test for ^C
FBD5 CAD2FB             jz      GETKBD          ;wait for character

FBD8 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
                ;********************************************************
FBD9 CDE5FF     GETLIN: call    FNDBUF          ;find buffer, push hl

FBDC 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

FBDD CD0EFD     GLLOOP: call    LBCHR           ;get kbd chr into line buffer
                                                ;with echo

FBE0 D60D               sui     CR              ;end of line from user?
FBE2 C2DDFB             jnz     GLLOOP          ;n: get another chr

FBE5 2B                 dcx     h               ;back up to CR
```

```
FBE6 77                mov     m,a             ; overwrite CR with null

FBE7 CD4CFD            call    ILPRNT          ;linefeed to follow CR
FBEA 8A                db      LF+80h

FBEB D1                pop     d               ;input line buffer address
FBEC E1                pop     h               ;Restore original hl

               ;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 and Z clear if found
               ;******************************************
FBED 1A        SKIPB:  ldax    d               ;get next character
FBEE B7                ora     a               ;terminating null?
FBEF C8                rz

FBF0 FE20              cpi     ' '
FBF2 C0                rnz                     ;we're past them

FBF3 13                inx     d               ;next scan address
FBF4 C3EDFB            jmp     SKIPB           ;keep skipping

               ;===Assembly Check================================
               ; The above code must not overrun the next section
               ;=================================================
FBF7 =         H0END   equ     $

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

               ;==============================================================
               ; Hard Disk Boot Loader Subsystem (HDBL)
               ;
               ; 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.
               ;==============================================================

FC00                   org     HDBADR

               ;==============================================================
               ; Entry here to execute HDBL directly, to boot from a hard
               ; disk. This is the same address where my HDBL PROM starts.
               ;==============================================================
FC00 0106FC    HDBL:   lxi     b,HDBRET        ;return address
FC03 C303F8            jmp     INIT            ;go find a real stack
                                               ;and initialize ACIAs
                                               ;returns with e=0

FC06 6B        HDBRET: mov     l,e             ;boot from platter 0

               ;Fall into HBOOT

               ;***Command Routine******
               ; HB  Boot from hard disk
               ; On Entry:
               ;   l<0> = platter
               ;************************
FC07 7D        HBOOT:  mov     a,l
FC08 E601              ani     1               ;just the lsb
FC0A 0F                rrc                     ;Platter goes in bits <7:6>
FC0B 0F                rrc                     ;..which is CFPLTR
FC0C 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
               ;--------------------------------------
FC0D AF                xra     a
FC0E 57                mov     d,a             ;set load initial page
FC0F 5F                mov     e,a

FC10 D3A0              out     0A0h            ;Select port 0Ah DDR
FC12 D3A2              out     0A2h            ;Select port 0Bh DDR
FC14 D3A4              out     0A4h            ;Select port 1Ah DDR
FC16 D3A6              out     0A6h            ;Select port 1Bh DDR
FC18 D3A1              out     0A1h            ;Port 0Ah is an input port
FC1A D3A5              out     0A5h            ;Port 1Ah is an input port

FC1C 2F                cma
FC1D D3A3              out     0A3h            ;Port 0Bh is an output port
FC1F D3A7              out     0A7h            ;Port 1Bh is an output port

FC21 3E2C              mvi     a,CINIT         ;set up input port handshakes
FC23 D3A0              out     0A0h
FC25 D3A4              out     0A4h
FC27 D3A6              out     0a6h            ;output port 1Bh handshakes

FC29 3E24              mvi     a,BINIT         ;set up port 0Bh handshakes
FC2B D3A2              out     0A2h

FC2D 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
               ;-----------------------------------------------
FC2F 062B              mvi     b,OSOFF+3       ;byte count to end of pointers
FC31 CD51FC            call    GETPAG          ;Seek, read page hl into buffer
                                               ;set up to read b buffer bytes

FC34 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.

FC35 DBA5      PTRLUP: in      CDATA           ;read byte from controller

FC37 6C                mov     l,h             ;shift everybody over...
FC38 61                mov     h,c
FC39 4F                mov     c,a             ;...and put it away

FC3A 05                dcr     b
FC3B C235FC            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
               ;-------------------------------------------------
FC3E CD51FC    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.)

FC41 DBA5      BYTELP: in      CDATA           ;get a data byte
FC43 12                stax    d               ;write it to RAM
FC44 1C                inr     e               ;write entire page
FC45 C241FC            jnz     BYTELP          ;until done

               ; Next Disk Page

FC48 14                inr     d               ;next RAM page
FC49 23                inx     h               ;Next Disk Page
FC4A 0D                dcr     c               ;bump Disk Page count
FC4B C23EFC            jnz     PAGELP

               ;---------------------------------------------------
               ;Go execute loaded code, at the address on the stack
               ;On Entry: c=0
               ;---------------------------------------------------
FC4E C383FA            jmp     EXECDP          ;disable PROM,
                                               ;go 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
               ;*****************************************************
FC51 E5        GETPAG: push    h               ;Save requested Page
FC52 D5                push    d               ;Save regs
FC53 C5                push    b               ;save byte count

               ;----------------------------------------------------------
               ;Compute cylinder and sectorX2  from Disk Page number in hl
               ; hl := hl / (2*HDSPT) (Quotient=cylinder)
               ;  a := 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.
               ;But we will always miss the next sector anyway, so each
               ;sector will require a full disk rev (25 mS), lots of time.
               ;----------------------------------------------------------
FC54 01D0FF            lxi     b,-2*HDSPT
FC57 50                mov     d,b             ;de=FFFF=-1
FC58 58                mov     e,b             ;since loop goes 1 extra

FC59 13        DIV1:   inx     d               ;compute quotient=cylinder
FC5A 09                dad     b               ;hl gets remainder
FC5B DA59FC            jc      DIV1

FC5E 7D                mov     a,l             ;fix remainder, since
FC5F 91                sub     c               ;..loop went 1 extra

FC60 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)
               ;--------------------------------------------------
FC61 FE18              cpi     HDSPT           ;past end of side 0?
FC63 DA68FC            jc      SIDEOK          ;N: sector number is good

FC66 C608              adi     CSIDE-HDSPT     ;Compute sector mod HDSPT,
                                               ;..and set side 1 bit

FC68 B0        SIDEOK: ora     b               ;combine platter bits
FC69 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

FC6A CD80FC            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
               ;---------------------------------------------------------
FC6D 79                mov     a,c
FC6E CD81FC            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.)
               ;--------------------------------------------
FC71 DBA5              in      CDATA           ;reset CDA in CDSTA
FC73 DBA3              in      ACMD            ;clear CMDACK in ACSTA

FC75 AF                xra     a               ;256 bytes to load
FC76 D3A7              out     ADATA           ;..to controller

FC78 3E50              mvi     a,CRDBUF+DBUFR  ;issue Read Buffer command
FC7A D3A3              out     ACMD            ;..to controller

FC7C C1                pop     b               ;(10)
FC7D D1                pop     d               ;(10)
FC7E 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
               ;---------------------------------
FC7F 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
               ;**************************************************************
FC80 7D        HDCMD:  mov     a,l             ;low byte of command

FC81 D3A7      HDCMDA: out     ADATA           ;..to data port

FC83 DBA1              in      CSTAT           ;reset CRDY flag just in case
FC85 DBA3              in      ACMD            ;clear CMDACK in ACSTA

FC87 7C                mov     a,h             ;command high byte
FC88 D3A3              out     ACMD            ;issue command

FC8A DBA0      HDWAIT: in      CREADY          ;Is the controller done?
FC8C 07                rlc                     ;look at msb=CRDY
FC8D D28AFC            jnc     HDWAIT          ;N: keep waiting

FC90 DBA1              in      CSTAT           ;reset CRDY flag
FC92 E67F              ani     ERMASK          ;and get A=error code
FC94 C8                rz                      ;No errors: happy return

               ; Report a load error and go to AMON's main loop
               ; On Entry:
               ;   a = error flag bits
               ;  hl = disk command

FC95 CD3DFE            call    PCAHEX          ;print error code in hex
FC98 C3D3FF            jmp     HDERR           ;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
               ;**********************************************************
FC9B E5        HEXLOD: push    h               ;address offset onto stack

FC9C 210000            lxi     h,0             ;initialize record count
FC9F E5                push    h               ;onto stack too

               ;Eat all characters until we get record-start colon

FCA0 CD86FD    GETCOL: call    GETTPD
FCA3 D63A              sui     ':'
```

```
FCA5 C2A0FC             jnz     GETCOL

FCA8 57                 mov     d,a             ;d=0: Init checksum

FCA9 CD4CFD             call    ILPRNT          ;print a pacifier per record
FCAC 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)

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

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

FCAF 4C        HEDRLP:  mov     c,h             ;c=byte 1: byte count
FCB0 65                 mov     h,l             ;h=byte 2: address MSB
FCB1 68                 mov     l,b             ;l=byte 3: address LSB
FCB2 CD9BFD             call    GETTPH          ;get header byte, do checksum
FCB5 1D                 dcr     e
FCB6 C2AFFC             jnz     HEDRLP

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

FCB9 D1                 pop     d               ;get offset
FCBA 19                 dad     d               ;offset the address in hl

FCBB E3                 xthl                    ;bump record count
FCBC 23                 inx     h
FCBD E3                 xthl                    ;..leaving it on the stack

FCBE D5                 push    d               ;save offset

FCBF 57                 mov     d,a             ;d=checksum so far
FCC0 59                 mov     e,c             ;remember count for EOF test

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

FCC1 79                 mov     a,c             ;c=record byte count
FCC2 B7                 ora     a               ;0-byte record?
FCC3 CADDFC             jz      GETCSM

               ;Loop to get data into memory at hl.

FCC6 CD9BFD    DATALP:  call    GETTPH          ;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.

FCC9 CDE7FF             call    HRMPAG          ;(stuffs hl on stack)
FCCC 7C                 mov     a,h             ;a=RAM page address
FCCD E1                 pop     h               ;restore RAM address
FCCE BC                 cmp     h               ;same as AMON's RAM page?
FCCF CA63FF             jz      OVRERR          ;y:abort with overwrite error

               ;Write to memory, and verify the write
FCD2 70                 mov     m,b             ;store data in RAM
FCD3 7E                 mov     a,m
```

```
FCD4 B8                 cmp     b
FCD5 C2CAFF             jnz     MEMERR          ;successful write?
FCD8 23                 inx     h

FCD9 0D                 dcr     c
FCDA C2C6FC             jnz     DATALP

FCDD CD9BFD    GETCSM:  call    GETTPH          ;get checksum in a & Z flag
FCE0 C2C7FF             jnz     CSMERR          ;should be zero

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

FCE3 B3                 ora     e               ;zero-byte record?
FCE4 C2A0FC             jnz     GETCOL          ;N: go get another record

               ;-------------------------------------------
               ;Done. Print record count and return to MAIN
               ;-------------------------------------------
FCE7 E1        HLDONE:  pop     h               ;remove offset from stack
FCE8 E1                 pop     h               ;record count

FCE9 CD47FD             call    CILPRT
FCEC 526563733A         db      'Recs:',' '+80h

               ;Fall into PHLCHX

               ;***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
               ;*****************************************
FCF2 0600      PHLCHX:  mvi     b,0     ;print on console
FCF4 C337FE             jmp     PHLHEX

               ;***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 = <ADR>, defaulting to 0
               ;  Carry set if none entered
               ;*********************************************************
FCF7 CD5FFE    ENTER:   call    PHLADR          ;print hl as an address

FCFA CDD9FB             call    GETLIN          ;get a line of user input
FCFD C8                 rz                      ;z=blank line terminates

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

FCFE CD1AF9    ENLOOP:  call    PHFHEX          ;save memory address,
                                                ;Get/convert value

FD01 7D                 mov     a,l             ;get low byte as converted
FD02 E1                 pop     h               ;recover memory address
```

```
FD03 77                 mov     m,a             ;put in the value
FD04 23                 inx     h               ;next address

FD05 CDEDFB             call    SKIPB           ;Scan to next input value
FD08 C2FEFC             jnz     ENLOOP          ;not end of line: continue

FD0B C3F7FC             jmp     ENTER           ;end of line: start new line

               ;===============
               ;= 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
               ;***********************************************************
FD0E CDD2FB    LBCHR:   call    GETKBD          ;get a character

FD11 77                 mov     m,a             ;store character in buffer

FD12 FE7F               cpi     DEL             ;DEL character?
FD14 CA19FD             jz      GCDEL
FD17 FE08               cpi     BS              ;BS is same as DEL
FD19 7D        GCDEL:   mov     a,l             ;buffer address low byte
FD1A CA25FD             jz      GDELET

FD1D EECB               xri     RAMBUF+LBSIZE   ;input buffer full?
FD1F 7E                 mov     a,m             ;recover chr for echo
FD20 C8                 rz                      ;full: ignore it

FD21 23                 inx     h               ;bump line buffer pointer

FD22 C38EF8             jmp     PRINTA          ;echo & ret

               ;-----------------------------------------------------------
               ;Backspace or delete found. Delete if there is anything to
               ;delete, and echo to the user the right way, based on TTYPE.
               ;-----------------------------------------------------------
FD25 D67B      GDELET:  sui     RAMBUF          ;anything on the line?
FD27 C8                 rz                      ;done if not.

               ;backspace either by erasing onscreen or Teletype-style

FD28 2D                 dcr     l               ;back up in buffer

FD29 3E68               mvi     a,TTYPE-RIOCOD+RAMCOD
FD2B CDE7FF             call    HRMPAG          ;pushes hl too
FD2E 7E                 mov     a,m             ;get TTYPE variable
FD2F E1                 pop     h
FD30 1F                 rar                     ;0 (even): backspacing terminal
FD31 D240FD             jnc     GCBKUP

               ;Teletype-style delete
```

```
FD34 CD3BFD            call    PSLASH          ;print deleted character

FD37 7E                mov     a,m             ;..between slashes
FD38 CD8EF8            call    PRINTA

               ;Fall into PSLASH

               ;---Local Subroutine---
               ;Print slash
               ;----------------------
FD3B CD4CFD    PSLASH: call    ILPRNT
FD3E AF                db      '/'+80h
FD3F C9                ret

               ;Terminal-style delete

FD40 CD4CFD    GCBKUP: call    ILPRNT          ;back up on screen
FD43 082088            db      BS,' ',BS+80h   ;Erase old character & back up

FD46 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
               ; On Exit:
               ;    a = 80h
               ;    Z & Carry cleared
               ;    all other registers preserved
               ;*******************************************
FD47 CD4CFD    CILPRT: call    ILPRNT
FD4A 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 & Carry cleared
               ;    all other registers preserved
               ;*******************************************
FD4C E3        ILPRNT: xthl                    ;save hl, get string address

FD4D 7E        IPLOOP: mov     a,m             ;loop through message
FD4E E67F              ani     7Fh             ;strip end-marker
FD50 CD8EF8            call    PRINTA
FD53 AE                xra     m               ;end? (clears Carry too)
FD54 23                inx     h
FD55 CA4DFD            jz      IPLOOP

FD58 E3                xthl                    ;restore hl
                                               ;..get ret address
FD59 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)
               ;******************************************
FD5A DB11      KDATA:  in      S2RXDA          ;get keyboard character
FD5C E67F              ani     7Fh             ;strip parity
FD5E 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
               ;****************************************************
FD5F CD7CFD    CKABRT: call    TESTTP          ;Transfer Port = console?
FD62 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
               ;****************************************************
FD63 CD7FF8    CHKKBD: call    KSTAT           ;anything typed?
FD66 C8                rz                      ;N: return w/ Z set

FD67 CD5AFD            call    KDATA           ;Y: get the data
FD6A FE03              cpi     CABKEY          ;abort character typed?
FD6C C0                rnz

FD6D B7                ora     a               ;clear Z flag to prevent
FD6E C3DFF8            jmp     CABORT          ;..address from being printed

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

FD73 B7                ora     a               ;clear carry
FD74 DA                db      JC              ;jc opcode skips mvi a below

               ;Skip into TPIN, skipping mvi a instruction

               ;***Subroutine***************
               ; Get Transfer Port data
               ; On Exit:
               ;   a=byte from Transfer Port
               ;   Z cleared
               ;****************************
FD75 3E32      TPIN:   mvi     a,RTPIN-RIOCOD+RAMCOD

FD77 CDE7FF            call    HRMPAG          ;pushes h

FD7A E3                xthl                    ;fix hl, put address on stack
FD7B C9                ret                     ;'call' RTPIN

               ;***Subroutine**************************
               ; Test to see if Transfer Port = console
               ; On Exit:
               ;   Z set if console = Transfer Port
               ; Trashes a
               ;***************************************
FD7C 3E2D      TESTTP: mvi     a,TPISP+1-RIOCOD+RAMCOD ;status register addr
FD7E CDE7FF            call    HRMPAG                  ;pushes h

FD81 7E                mov     a,m
FD82 E1                pop     h

FD83 FE10              cpi     S2STAA                  ;Console's status port?
FD85 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
               ;*******************************************************
FD86 CD7CFD    GETTPD: call    TESTTP          ;Transfer Port = console?
FD89 CAD2FB            jz      GETKBD          ;Y: get and test keyboard chr

FD8C CD63FD    GTPLUP: call    CHKKBD          ;user abort?
FD8F CD71FD            call    TPISTA          ;Transfer Port character?
FD92 CA8CFD            jz      GTPLUP          ;n: keep waiting

FD95 CD75FD            call    TPIN            ;get Transfer Port character
FD98 E67F              ani     7Fh             ;strip parity

FD9A 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
               ;*****************************************************
FD9B CDABFD    GETTPH: call    GETTPN          ;get high nibble
FD9E 87                add     a               ;Shift high nibble in place
FD9F 87                add     a
FDA0 87                add     a
FDA1 87                add     a
FDA2 47                mov     b,a
FDA3 CDABFD            call    GETTPN          ;get low nibble

FDA6 B0                ora     b               ;combine nibbleS
FDA7 47                mov     b,a             ;save result for return
FDA8 82                add     d               ;compute checksum
FDA9 57                mov     d,a             ;ret with checksum in a & d
FDAA C9                ret

               ;---Local subroutine---------------------
               ; Get a hex digit from the Transfer Port,
               ; validate it, and return it in A<3:0>
               ;----------------------------------------
FDAB CD86FD    GETTPN: call    GETTPD
FDAE CD6EFE            call    HEXCON
FDB1 D8                rc                      ;Carry means OK

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

FDB2 3E48              mvi     a,HERMSG
FDB4 C3CCFF            jmp     RPTERR

               ;==================================================
               ; 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
               ;==================================================
FDB7 4144      COMTAB: db      'AD'            ;Dump in Altair format
FDB9 41F9              dw      ADUMP
FDBB 414C              db      'AL'            ;Load Altair format
FDBD 06FE              dw      ALOAD

FDBF 424F              db      'BO'            ;Boot from FLOPPY
FDC1 06FF              dw      FBOOT
FDC3 434F              db      'CO'            ;Copy memory
FDC5 92F9              dw      MCOPY
FDC7 4455              db      'DU'            ;Dump to console
FDC9 91FA              dw      DUMP
FDCB 454E              db      'EN'            ;Enter
FDCD F7FC              dw      ENTER
FDCF 4558              db      'EX'            ;Execute
FDD1 7EFA              dw      EXEC
FDD3 4649              db      'FI'            ;Fill memory
FDD5 DAFA              dw      FILMEM

FDD7 4842              db      'HB'            ;Boot from hard disk
FDD9 07FC              dw      HBOOT

FDDB 4844              db      'HD'            ;Intel hex dump
FDDD 3EFB              dw      HEXDMP
FDDF 484C              db      'HL'            ;Intel hex load
FDE1 9BFC              dw      HEXLOD

FDE3 494E              db      'IN'            ;Input from port
FDE5 27FE              dw      IPORT
FDE7 4F54              db      'OT'            ;Output to port
FDE9 2CFB              dw      OPORT

FDEB 5345              db      'SE'            ;Search
FDED EBF9              dw      SEARCH

FDEF 5445              db      'TE'            ;Terminal Mode
FDF1 F37A              dw      TERMNL and 7FFFh ;non-hex parameter

FDF3 5450              db      'TP'            ;Set Transfer Port
FDF5 49F8              dw      SETTP
FDF7 5454              dw      'TT'            ;Terminal Type
```

```
FDF9 76FA              dw      SETTT

FDFB 5645              db      'VE'            ;Verify
FDFD C8F9              dw      VERCMD

FDFF 00                db      0               ;end of table mark

               ;===Assembly Check================================
               ; All of Monitor must not overrun the next section
               ;=================================================
FE00 =         H1END   equ     $

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

               ;==============================================================
               ;              Multi Boot Loader Subsystem (MBL)
               ;
               ; 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.
               ;
               ; Since the 88-2SIOJP may optionally disable PROMS when an IN
               ; instruction accesses port FFh (like some versions of the MITS
               ; 8800b Turnkey Module), this code cannot execute from
               ; PROM - at least not from the point where the Sense switches
               ; are read onwards.
               ;
               ;==============================================================
               ; 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 010AFE    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 e=0
               ;***Command Routine**************************************
               ; AL <0/1> (Boot from paper or cassette tape)
               ;     Go record ignored if parameter=0. Default to 1.
               ; Note: parameter is not bounds-checked, but
               ; nothing bad will happen with bogus values
               ;
               ; On Entry:
               ;    TP command has set up the Transfer Port
               ;    carry set if no parameter typed
               ;    l = 0 and carry clear if GO record should be ignored
               ;********************************************************
FE06 3E01      ALOAD:  mvi     a,1
FE08 8D                adc     l               ;catch carry bit
FE09 5F                mov     e,a             ;e = 1 or 2

               ;Fall into GOMBL

               ;---------------------------------------------------
               ;Entry here from cold-start at MBL:
               ;   e=0
               ;   nothing on stack
               ;Entry here from monitor call to ALOAD (AL command):
               ;  e = 1 if Go record should be ignored
               ;  e = 2 if Go record should be executed
               ;  bottom of stack = address of MAIN
               ;---------------------------------------------------
FE0A 3E7B      GOMBL:  mvi     a,RAMBUF        ;sector buffer at end of page
FE0C CDE9FF            call    RAMPAG          ;hl=address of buffer

               ;---------------------------------------------------------
               ;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 to move code and execute it)
               ;  e = 0 if PROM may be disabled (cold-start at MBL)
               ;  e = 1 if Go record should be ignored
               ;  e = 2 if Go record should be executed
               ;On 'ret' to the RAM code:
               ;  d = RAM execution page
               ;  e = unchanged
               ;  Z set if sense switches determine transfer port
               ;---------------------------------------------------------
FE0F E5                push    h               ;RAM code execution address

FE10 017BFE            lxi     b,MRCODE        ;source address

FE13 0A        RELOOP: ldax    b
FE14 B8                cmp     b               ;relocatable address byte?
FE15 C219FE            jnz     NOTADR
FE18 7C                mov     a,h             ;Y: relocate this address
FE19 77        NOTADR: mov     m,a
FE1A 03                inx     b
FE1B 2C                inr     l               ;don't let h change at the end
FE1C C213FE            jnz     RELOOP          ;run to the end of the page

               ;Set d=RAM execution page for overwrite detection during load

FE1F 54                mov     d,h
```

```
               ;Test sense switches if cold-start. Otherwise use transfer port
               ;as set up by Amon.

FE20 1C                inr     e               ;e=1 if entry from amon
FE21 1D                dcr     e               ;use switches? Z set if so
FE22 C8                rz                      ;execute the loaded code

FE23 E1                pop     h
FE24 2E82              mvi     l,(SKPSW-MRCODE)+RAMBUF
FE26 E9                pchl                    ;skip sense switch test

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

               ;***Command Routine*************************************
               ; IN <PORT> (Input from port)
               ; On Entry:
               ;    l=PORT
               ; Creates this routine on the stack, then executes it,
               ; then returns through PAHEX to print the value
               ;
               ;       NOP
               ;       IN   <PORT>
               ;       RET
               ;*******************************************************
FE27 113DFE    IPORT:  lxi     d,PCAHEX        ;create return address
FE2A D5                push    d               ;ret through PCAHEX

FE2B 26C9              mvi     h,RET           ;Opcode
FE2D E5                push    h               ;L=<PORT>
FE2E 2100DB            lxi     h,IN*256        ;NOP,IN opcode
FE31 E5                push    h
FE32 65                mov     h,l             ;hl=0
FE33 39                dad     sp              ;hl points to routine

FE34 D1                pop     d               ;fix stack
FE35 D1                pop     d
FE36 E9                pchl                    ;execute RAM routine

               ;***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
               ;******************************
FE37 7C        PHLHEX: mov     a,h             ;h first
FE38 CD3FFE            call    PAHEXC          ;returns with Carry clear
FE3B 7D                mov     a,l             ;then l

FE3C 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
               ;************************************
FE3D 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
               ;*********************************************
FE3F F5        PAHEXC: push    psw
FE40 81                add     c               ;compute checksum
FE41 4F                mov     c,a
FE42 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
               ;*****************************
FE43 F5        PAHEX:  push    psw             ;save for low digit

FE44 0F                rrc                     ;move the high four down
FE45 0F                rrc
FE46 0F                rrc
FE47 0F                rrc
FE48 CD4CFE            call    PNIBLE          ;put them out
FE4B 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
               ;-----------------------------
FE4C E60F      PNIBLE: ani     0Fh             ;four on the floor
FE4E C630              adi     '0'             ;We work with ASCII here
FE50 FE3A              cpi     '9'+1           ;0-9?
FE52 DA57FE            jc      PNIB1           ;YUP: print & return

FE55 C607              adi     'A'-'9'-1       ;make it a letter

FE57 04        PNIB1:  inr     b               ;which port?
FE58 05                dcr     b
```

```
FE59 C29BFB            jnz     TPOUT           ;print on Transfer Port

FE5C C38EF8            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
               ;***********************************************
FE5F CD47FD    PHLADR: call    CILPRT          ;CR LF space begins line
FE62 A0                db      ' '+80h

FE63 0600              mvi     b,0             ;output address to console
FE65 CDF2FC            call    PHLCHX          ;hl=address, b=0, trash c

FE68 CD4CFD            call    ILPRNT          ;print colon space
FE6B 3AA0              db      ':',' '+80h
FE6D C9                ret

               ;***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


               ;===Assembly Check================================
               ; The above code must not overrun the next section
               ;=================================================
FE7B =         H2END   equ     $

                if (MRCODE - H2END)/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 sense switches) optionally disables
               ; the PROM.
               ; On Entry:
               ;    d = RAM Execution page
               ; Entry at MRCODE:
               ;    e = 0 (PROM will be disabled by the upcoming IN FF)
               ; Entry at SKPSW:
               ;   Transfer Port already set up
               ;   PROM is still enabled
               ;   e = 1 if Go record should be ignored
               ;   e > 1 if Go record should be executed
               ;=============================================================
FE7B                   org     MBLADR+RAMBUF   ;force low address byte
                                               ;..to be the same

FE7B DBFF      MRCODE: in      SSWTCH          ;N: read sense switches
FE7D E607              ani     LDMASK          ;bits specify load device


               ;       call    RSETP           ;set up Transfer Port
FE7F CD                db      CALL            ;call opcode
FE80 00                db      RSETP-RIOCOD+RAMCOD ;low address byte
FE81 FE                db      MRCODE/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
               ;  e = 1 if Go record should be ignored
               ;  e > 1 if Go record should be executed
               ;-----------------------------------------------
               ;       call    RTPIF
FE82 CD                db      CALL            ;call opcode
FE83 38                db      RTPIF-RIOCOD+RAMCOD ;low address byte
FE84 FE                db      MRCODE/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
               ;  e = 1 if Go record should be ignored
               ;  e > 1 if Go record should be executed
               ;  The 1st byte of the checksum loader has already been read
               ;-----------------------------------------------------------
FE85 CDF2FE            call    GETBYT          ;get 1st byte

FE88 4F                mov     c,a             ;number of bytes in loader

FE89 B7                ora     a               ;null leader?
FE8A CA9CFE            jz      RCHUNT          ;Y: skip leader

FE8D CDF2FE    LDSKIP: call    GETBYT          ;get another byte

FE90 B9                cmp     c
FE91 CA8DFE            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
                ;  e = 1 if Go record should be ignored
                ;  e > 1 if Go record should be executed
                ;-----------------------------------------------------------
FE94 0D                 dcr     c               ;since we got a byte already

FE95 CDF2FE     CLSKIP: call    GETBYT          ;get a loader byte
FE98 0D                 dcr     c
FE99 C295FE             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:
                ;  d = RAM execution page
                ;  e = 0 if PROM may be disabled
                ;  e = 1 if Go record should be ignored
                ;  e > 1 if Go record should be executed
                ;----------------------------------------------------------
FE9C CDF2FE     RCHUNT: call    GETBYT          ;hunt for sync character

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

FE9F EE3C               xri     ALTPLR          ;load record sync byte?
FEA1 C2E0FE             jnz     CHEKGO          ;n: go see if it's a GO

                ;--------------------------------------------------------
                ;Load Record: Read and store data from a load record
                ;
                ;On Entry:
                ;  the load record sync byte has already been read
                ;  d = RAM execution page
                ;  e = 0 if PROM may be disabled
                ;  e = 1 if Go record should be ignored
                ;  e > 1 if Go record should be executed
                ;  RCHUNT's address is on the stack
                ;--------------------------------------------------------
FEA4 CDF2FE             call    GETBYT          ;get record byte count
FEA7 4F                 mov     c,a             ;c counts data bytes

FEA8 CDEEFE             call    GETWRD          ;get load address into a,l
FEAB 67                 mov     h,a             ;hl = record load address

FEAC 85                 add     l               ;initialize checksum
FEAD 47                 mov     b,a

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

FEAE 7A         LRLOOP: mov     a,d             ;d=RAM Execution page
FEAF BC                 cmp     h               ;same page as load address?
FEB0 3E4F               mvi     a,OERMSG        ;overwrite error message
FEB2 CAD0FE             jz      ERDONE          ;error exit if overwrite

FEB5 CDF2FE             call    GETBYT          ;get a data byte

FEB8 77                 mov     m,a             ;store data byte
FEB9 BE                 cmp     m               ;did it store correctly?
FEBA C2CEFE             jnz     MERDON          ;error exit if mismatch
```

```
FEBD 80                 add     b               ;compute checksum
FEBE 47                 mov     b,a

FEBF 23                 inx     h               ;bump dest pointer
FEC0 0D                 dcr     c               ;bump byte count
FEC1 C2AEFE             jnz     LRLOOP          ;loop through all bytes

                ;Validate checksum, fail if it doesn't match

FEC4 CDF2FE             call    GETBYT          ;test record's checksum
FEC7 B8                 cmp     b
FEC8 CA9CFE             jz      RCHUNT          ;match: get another record

FECB 3E43               mvi     a,CERMSG        ;checksum error message
FECD CA                 db      JZ              ;skips 2 bytes

                ;Skip into ERDONE


FECE 3E4D       MERDON: mvi     a,MERMSG        ;memory error message

                ;Fall into ERDONE

                ;-----------------------------------------------------------
                ;Load Error:
                ; Turn the INTE light on as an error indicator. 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
                ;   hl = offending address
                ;-----------------------------------------------------------
FED0 1D         ERDONE: dcr     e               ;PROM disabled?
FED1 F2CCFF             jp      RPTERR          ;N: report, return to monitor
                                                ;this routine not relocated

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

FED4 320000             sta     00000h          ;PROM disabled: store error code
FED7 220100             shld    00001H          ;Store offending address

FEDA FB                 ei                      ;INTE light as error indicator

FEDB D311       ERHANG: out     S2TXDA          ;Console output
FEDD C3DBFE             jmp     ERHANG

                ;--------------------
                ; Test for GO record
                ;--------------------
FEE0 EE44       CHEKGO: xri     ALTEOF XOR ALTPLR ;EOF record sync byte?
FEE2 C29CFE             jnz     RCHUNT          ;N: ignore

                ;Fall into GO record execution

                ;-----------------------------------------
                ;Go Record: get the GO address and go there
                ;
                ;On Entry:
```

```
               ;  e = 0 if PROM may be disabled
               ;  e = 1 if Go record should be ignored
               ;  e > 1 if Go record should be executed
               ;  GO-record sync byte has already been read
               ;--------------------------------------------
FEE5 CDEEFE            call    GETWRD          ;get a,l=address
FEE8 67                mov     h,a             ;high byte

FEE9 1D                dcr     e               ;execute go record?
FEEA CAF2FC            jz      PHLCHX          ;n:print go address and quit

FEED E9                pchl                    ;go to go address

               ;---Local 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
               ;-----------------------------------
FEEE CDF2FE    GETWRD: call    GETBYT
FEF1 6F                mov     l,a

               ;Fall into GETBYT to get the high byte

               ;---Local Subroutine------------------------------
               ; Get a byte of data from the Transfer Port
               ; with user-abort opportunity
               ; On Entry:
               ;   e = 0 if AMON PROM may be disabled
               ; On Exit:
               ;   a = received character
               ;-------------------------------------------------
               GETBYT:
               ;       call    RTPIS           ;get transfer port status
FEF2 CD                db      CALL            ;call opcode
FEF3 2C                db      RTPIS-RIOCOD+RAMCOD ;low address byte
FEF4 FE                db      MRCODE/256      ;high byte (gets relocated)

               ;       jnz     RTPIF           ;go get transfer port data byte
FEF5 C2                db      JNZ             ;call opcode
FEF6 38                db      RTPIF-RIOCOD+RAMCOD ;low address byte
FEF7 FE                db      MRCODE/256      ;high byte (gets relocated)

FEF8 7B                mov     a,e
FEF9 B7                ora     a               ;PROM certainlty enabled?
FEFA C45FFD            cnz     CKABRT          ;Y: user abort?

FEFD C3F2FE            jmp     GETBYT          ;Wait for character

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

               ;===Assembly Check=============================
               ; MBL code must not overwrite the CDBL code below
               ;==============================================
FF00 =         SUBEND  equ     $

                if (DBLADR - SUBEND)/256
```

```
        ERROR: CDBL is overwriting prior code

 endif

;==============================================================
;=         Combo Disk boot loader Subsystem (CDBL)            =
;=        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)
               ;***************************
               FBOOT:

               ;------------------------------------------------------------
               ;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.
               ;------------------------------------------------------------
FF06 AF        WAITEN: xra     a               ;boot from disk 0
FF07 D308              out     DENABL          ;..so enable disk 0

FF09 CD63FD            call    CHKKBD          ;abort from user?

FF0C DB08              in      DSTAT           ;Read drive status
FF0E E608              ani     DRVRDY          ;Diskette in drive?
FF10 C206FF            jnz     WAITEN          ;no: wait for drive ready

FF13 3E04              mvi     a,HEDLOD        ;load 8" disk head, or enable
FF15 D309              out     DCTRL           ;..minidisk for 6.4 Sec

               ;----------------------------------------------------------------
               ;Step in once, then step out until track 0 is detected.
               ;The first time through, delay at least 25 ms to force a minimum
               ;43 ms step wait instead of 10ms. This meets the 8" spec for
               ;changing seek direction. (Minidisk step time is always 50ms,
               ;enforced by the mninidsk conrtoller hardware.) See the 88-DCDD
               ;documentation for details. This loop ends with hl=0.
               ;----------------------------------------------------------------
FF17 212308            lxi     h,25000/12      ;25 mS delay 1st time thru
FF1A 3E01              mvi     a,STEPIN        ;step in once first

FF1C D309      SEEKT0: out     DCTRL           ;issue step command

FF1E 2C                inr     l               ;After the 1st time, the following
                                               ;..loop goes 1 time.

FF1F 2B        T0DELY: dcx     h               ;(5)
FF20 7C                mov     a,h             ;(5)
FF21 B5                ora     l               ;(4)
FF22 C21FFF            jnz     T0DELY          ;(10)12 uS/pass

FF25 DB08      WSTEP:  in      DSTAT           ;wait for step to complete
FF27 0F                rrc                     ;put MVHEAD bit in Carry
FF28 0F                rrc                     ;is the servo stable?
FF29 DA25FF            jc      WSTEP           ;no: wait for servo to settle

FF2C E610              ani     TRACK0/4        ;Are we at track 00?
```

```
FF2E 3E02              mvi     a,STEPOT        ;Step-out command
FF30 C21CFF            jnz     SEEKT0          ;no: step out another track

               ;------------------------------------------------------
               ;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

FF33 DB09      CKDSK1: in      DSECTR          ;Read the sector position

FF35 E63F              ani     SECMSK+SVALID   ;mask sector bits, and hunt
FF37 FE1E              cpi     (MDSPT-1)*2     ;..for minidisk last sector
FF39 C233FF            jnz     CKDSK1          ;..only while SVALID is 0

               ;wait for this sector to pass

FF3C DB09      CKDSK2: in      DSECTR          ;Read the sector position
FF3E 0F                rrc                     ;wait for invalid sector
FF3F D23CFF            jnc     CKDSK2

               ;wait for and get the next sector number

FF42 DB09      CKDSK3: in      DSECTR          ;Read the sector position
FF44 0F                rrc                     ;put SVALID in Carry
FF45 DA42FF            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).

FF48 E61F              ani     SECMSK/2        ;mask sector bits
FF4A C610              adi     MDSPT           ;compute SPT
FF4C 4F                mov     c,a             ;..and save SPT in c

               ;------------------------------------------
               ;Set up to load
               ;On Entry:
               ;  hl = 0 (DMA address & execution address)
               ;  c = SPT (for either minidisk or 8" disk)
               ;------------------------------------------
FF4D E5                push    h               ;execution address = 0 onto stack

FF4E CDE5FF            call    FNDBUF          ;find hl=buffer address,
                                               ;push DMA address = 0
FF51 E3                xthl                    ;push buffer address, recover
                                               ;DMA address = 0

FF52 45                mov     b,l             ;initial sector number = 0

               ;------------------------------------------------------
               ;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
               ; top-of-stack = buffer address
               ; next on stack = execution address
               ;------------------------------------------------------
FF53 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
               ; next on stack = execution address = 0
               ;-----------------------------------------
FF55 D1        RDSECT: pop     d               ;(10)get RAMBUF address
FF56 D5                push    d               ;(11)keep it on the stack
FF57 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.
               ;---------------------------------------------------
FF58 DB09      FNDSEC: in      DSECTR          ;(10)Read the sector position

FF5A E63F              ani     SECMSK+SVALID   ;(7)yes: mask sector bits
                                               ;..along with -SVALID bit
FF5C 0F                rrc                     ;(4)sector bits to bits <4:0>
FF5D B8                cmp     b               ;(4)found the desired sector
                                               ;..with -SVALID low?
FF5E C258FF            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.
               ;-----------------------------------------------------------
FF61 7C                mov     a,h             ;(5)high byte of DMA address
FF62 BA                cmp     d               ;(4)high byte of RAM code addr

               ;Entry point for reporting an overrun error from HL command
               ;(Z flag is set on entry from HL.)

FF63 3E4F      OVRERR: mvi     a,OERMSG        ;(7)overlay error message
FF65 CACCFF            jz      RPTERR          ;(10)report overlay error

               ;---------------------------------------
               ;Set up for the upcoming data move
               ;Do this here, while we have some time.
               ;---------------------------------------
FF68 E5                push    h               ;(11)DMA address for retry
FF69 C5                push    b               ;(11)Current sector & SPT
FF6A 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
               ;exactly 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.
               ;-------------------------------------------------------
FF6D DB08      DATLUP: in      DSTAT           ;(10)Read the drive status
FF6F 07                rlc                     ;(4)new Read data Available?
FF70 DA6DFF            jc      DATLUP          ;(10)no: wait for data

FF73 DB0A              in      DDATA           ;(10)Read data byte
```

```
FF75 12                stax    d               ;(7)store it in sector buffer
FF76 1C                inr     e               ;(5)Move to next buffer address
                                               ;..and test for end
FF77 C26DFF            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
               ;------------------------------------------------
FF7A 1E7E              mvi     e,SDATA          ;(7)de= address of sector data
                                                ;..within the sector buffer

FF7C 1A        MOVLUP: ldax    d               ;(7)get sector buffer byte
FF7D 77                mov     m,a             ;(7)store it at the destination
FF7E BE                cmp     m               ;(7)Did it store correctly?
FF7F C2CAFF            jnz     MEMERR          ;(10)no: abort w/ memory error

FF82 80                add     b               ;(4)update checksum
FF83 47                mov     b,a             ;(5)save the updated checksum

FF84 13                inx     d               ;(5)bump sector buffer pointer
FF85 23                inx     h               ;(5)bump DMA pointer
FF86 0D                dcr     c               ;(5)more data bytes to copy?
FF87 C27CFF            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.
               ;On Entry and Exit:
               ;   a=computed checksum
               ;134 cycles for for this section
               ;----------------------------------------------------
FF8A EB                xchg                    ;(4)hl=1st trailer byte address
                                               ;de=DMA address
FF8B 4E                mov     c,m             ;(7)get marker, should be FFh
FF8C 0C                inr     c               ;(5)c should be 0 now

FF8D 23                inx     h               ;(5)(hl)=checksum byte
FF8E AE                xra     m               ;(7)compare to computed cksum
FF8F B1                ora     c               ;(4)..and test marker=ff

FF90 C1                pop     b               ;(10)Current sector & SPT
FF91 C2BDFF            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.

FF94 2E7C              mvi     l,SFSIZE        ;(7)hl=address of file size
FF96 7E                mov     a,m             ;(7)low byte
FF97 23                inx     h               ;(5)point to high byte
FF98 66                mov     h,m             ;(7)high byte
FF99 6F                mov     l,a             ;(5)hl=SFSIZE

FF9A EB                xchg                    ;(4)put DMA address back in hl
                                               ;..and file size into de

FF9B 7D                mov     a,l             ;(4)16-bit subtraction
FF9C 93                sub     e               ;(4)
FF9D 7C                mov     a,h             ;(5)..throw away the result
FF9E 9A                sbb     d               ;(4)..but keep Carry (borrow)
```

```
FF9F D1                pop     d               ;(10)chuck old DMA address
FFA0 D1                pop     d               ;(10)chuck old retry count

FFA1 D2B5FF            jnc     FDEXEC          ;(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
               ;-----------------------------------------------------
FFA4 1153FF            lxi     d,NXTSEC        ;(10)for compact jumps
FFA7 D5                push    d               ;(10)

FFA8 04                inr     b               ;(5)sector = sector + 2
FFA9 04                inr     b               ;(5)

FFAA 78                mov     a,b             ;(5)even or odd sectors done?
FFAB B9                cmp     c               ;(4)c=SPT
FFAC 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.

FFAD 0601              mvi     b,01H           ;1st odd sector number
FFAF 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 maximum
               ;of 40 uS.) Note that NXTRAC will repair the stack.
               ;------------------------------------------------------------
FFB0 78                mov     a,b             ;STEPIN happens to be 01h
FFB1 D309              out     DCTRL

FFB3 05                dcr     b               ;start with b=0 for sector 0
FFB4 C9                ret                     ;go to NXTSEC

               ;------------------------------------------------
               ;Execute successfully loaded code, after disabling
               ;the floppy drive and disabling the PROM
               ;On Entry:
               ;  Top of stack = RAMBUF address
               ;  Next on stack = execution address
               ;------------------------------------------------
FFB5 3E80      FDEXEC: mvi     a,DDISBL        ;Disable floppy controller
FFB7 D308              out     DENABL

FFB9 D1                pop     d               ;chuck RAMBUF address
                                               ;..to expose exec address

FFBA C383FA            jmp     EXECDP          ;disable PROM and execute code

               ;***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
               ;*************************************************************
FFBD 3E04      BADSEC: mvi     a,HEDLOD        ;Restart Minidisk 6.4 uS timer
FFBF D309              out     DCTRL

FFC1 E1                pop     h               ;Restore DMA address
FFC2 F1                pop     psw             ;get retry count
FFC3 3D                dcr     a               ;Any more retries left?
FFC4 C255FF            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
               ;  sp = valid address in RAM page
               ;----------------------------------------------------
FFC7 3E43      CSMERR: mvi     a,CERMSG        ;checksum error message
FFC9 11                db      11h             ;'lxi d' opcode to skip
                                               ;..MEMERR and go to RPTERR

               ;Skip into RPTERR

               ;***Error Routine*********************
               ; memory error: memory readback failed
               ; On Entry:
               ;   hl = offending RAM address
               ;   sp = valid address in RAM page
               ;*************************************
FFCA 3E4D      MEMERR: mvi     a,MERMSG        ;memory error message

               ;Fall into RPTERR

               ;***CDBL (and MBL) Termination**************************
               ; Report an error: turn the disk controller off, report
               ; the error on the console, Turn on the INTE light, jump
               ; to the console loop.
               ; On Entry:
               ;    a = ASCII error code
               ;   hl = offending RAM address
               ;   sp = valid address in RAM page
               ;*******************************************************
FFCC CD8EF8    RPTERR: call    PRINTA          ;print the ASCII error code

FFCF 3E80              mvi     a,DDISBL        ;Disable floppy controller
FFD1 D308              out     DENABL

               ;Fall into HDERR

               ;***HDBL Termination************************************
               ; Report an error: report the error on the console, Turn
               ; on the INTE light, jump to the console loop.
               ; On Entry:
               ;    a = error code
               ;   hl = offending RAM address or HDSK command
               ;   sp = valid address in RAM page
               ;*******************************************************
FFD3 CD4CFD    HDERR:  call    ILPRNT
FFD6 206572726F        db      ' error:',' '+80h
```

```
FFDE CDF2FC            call    PHLCHX          ;print hl in hex on console

               ; Cool-start AMON code

FFE1 FB                ei                      ;INTE light on (indicate error)
FFE2 C3BCF8            jmp     INIT2           ;go to monitor

               ;***Subroutine********************
               ; Find the RAMBUF address
               ; On Entry:
               ;   sp = valid address in RAM page
               ; On Exit:
               ;   hl = RAM page item address
               ;   prior hl value is on the stack
               ;   Carry is clear
               ; trashes a
               ;*********************************
FFE5 3E7B      FNDBUF: mvi     a,RAMBUF

               ; Fall into HRMPAG

               ;***Subroutine**********************
               ; Set hl to location within RAM page
               ; On Entry:
               ;    a = address offset into RAM page
               ;   sp = valid address in RAM page
               ; On Exit:
               ;   hl = RAM page item address
               ;   prior hl value is on the stack
               ;   Carry is clear
               ;   other flags unafffected
               ;***********************************
FFE7 E3        HRMPAG: xthl                    ;save hl, get return address
FFE8 E5                push    h               ;restore return address

               ;Fall into RAMPAG

               ;***Subroutine**********************
               ; Set hl to location within RAM page
               ; On Entry:
               ;    a = address offset into RAM page
               ;   sp = valid address in RAM page
               ; On Exit:
               ;   hl = RAM page item address
               ;   Carry is clear
               ;   other flags unafffected
               ;***********************************
FFE9 210000    RAMPAG: lxi     h,0
FFEC 39                dad     sp              ;get RAM page, clear carry
FFED 6F                mov     l,a             ;requested RAM address
FFEE 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
               ;   top-of-stack = prior hl value
               ;   abort to CMDERR if no value found
               ;***************************************************
```

```
FFEF E3        GETHEX: xthl                    ;save hl, put ret address in hl
FFF0 CD1AF9            call    PHFHEX          ;save hl, get hl=hex value
FFF3 D0                rnc

               ;Fall into CMDERR if no value

               ;**********************
               ; Command error handler
               ;**********************
FFF4 CD47FD    CMDERR: call    CILPRT          ;returns Z flag cleared
FFF7 BF                db      '?'+80h

FFF8 C3DFF8            jmp     CABORT          ;Repair stack, go to MAIN

               ;===Assembly Check================
               ; All of CDBL and the subsequent
               ; subroutines must fit in one page
               ;=================================
FFFB =         DBLEND  equ     $

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

                endif
FFFB                   end
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
```