MBLe
Enhanced Altair Multi Boot Loader
User's Guide

Martin Eberhard

16 January 2016

Revision History

Revision	Date	Author	Notes
1.00	1 SEP 2013	M. Eberhard	Created from disassembled TURMON
1.01	1 SEP 2013	M. Eberhard	Replace HSR support with 88-2SIO 2nd port support
1.02	3 SEP 2013	M. Eberhard	Parallel port read during initialization, to clear OP-80
1.03	9 SEP 2013	M. Eberhard	Don’t skip checksum loader if leader character is 0. Also, change name from MBLme to MBLe
1.04	12 MAR 2014	M. Eberhard	Check for RAM end before initializing I/O ports, so that incoming echo transmission will complete
1.05	5 JUN 2014	M. Eberhard	Ignore Sense Switch A11
2.00	12 AUG 2014	M. Eberhard	Turnkey Module compatibility: Relocate most code to RAM and execute from there.
2.01	24 AUG 2014	M. Eberhard	88-SYS-CLG compatibility: relocate ALL I/O instructions to RAM before executing any.
3.00	16 JAN 2016	M. Douglas	Make ROM code position independent, and speed up initialization

ABSTRACT

MBLe is an improved version of the Altair MBL PROM, a checksum loader for loading files that are “punched” in the Altair Binary Absolute Load Format. Most software distributed by MITS, such as Altair BASIC (on paper tape) ands Altair Cassette Tape BASIC, are “punched” in this format.

Files punched using MITS’s TURNMON PROM (or the improved UBMON PROM) “D” command are punched in this format, as are files created by MAKEALT (which is a CP/M utility that converts .HEX files to Altair-compatible .TAP files).

CONTENTS

1. MBLe IMPROVEMENTS

MBLe works exactly the same as the Altair MBL PROM, except for the following changes:

1. Turnkey Module Compatibility

   MITS's MBL PROM will not work with any of their Turnkey Modules except the oldest revision of these boards, without MITS's 88-SYS-CLG rework, because the Turnkey Module will disable the PROMs upon the first IN instruction from the sense switch port. (Some versions of the Turnkey Module will disable the PROMs upon any IN or OUT instruction. See Section 7.)

   To fix this, MBLe relocates itself to the same 256-byte page of RAM where its stack is located, and runs from there - never executing any IN or OUT instructions while running in PROM. SO, unlike MITS's MBL PROM, MBLe works correctly will all Turnkey Module versions.

2. Position Independence

   MBLe 3.0 and later PROMs will run from any 256-byte page of memory, except page 0.

3. High-Speed Serial Tape Reader Support

   MBL's 88-HSR load device has been replaced with support for loading from the second port on the 88-2SIO. Thus, the front panel sense switch setting that was originally assigned to the 88-HSR (see below) now selects the 88-2SIO's second serial port as the load device. This makes it possible to load Altair BASIC and other files using a high-speed RS-232 tape reader connected to the 88-2SIO's second port, and still use the 88-PIO's first port for the Terminal.

4. OAE OP-80 Support

   An initial read is performed from both the 88-PIO and the 88-4PIO Port 0, to clear data handshake latches in external devices such as the Oliver Audio Engineering OP-80 paper tape reader. This makes loading files (e.g. Altair BASIC) with the OP-80 work correctly.

5. MAKEALT and UBMON Punched File Support

   Both MBL and MBLe skip over the checksum loader that is normally punched on an Altair BASIC tape between the tape leader and the actual file to be loaded. (MBL contains its own checksum loader that is independent of the particular file being loaded.) On a tape that does contain a checksum loader, the binary value of the leader character is also the length (in bytes) of the checksum loader. MBL and MBLe both use this leader value to count bytes as it skips over the checksum loader.

   However, tapes that were punched using MAKEALT, or with older versions of UBMON "D" command, may not contain a checksum loader. Such tapes just has a long string of leader nulls before the file to

```
;=================================================================
; MBLe - Enhanced Multi Boot Loader for the Altair 8800
;
; Loads and runs an Altair 'Absolute Binary File' from input
; transfer port specified by the Sense Switch settings.
; Normally run in PROM at address 0FE00h. However, since
; version 3.00 the PROM is position independent and can run
; at most any 256 byte boundary.
;
; Vers.   Date      Author    Comments
; 1.00  01Sep2013  M. Eberhard
;    Disassembled from MITS EPROMS
; 1.01  01Sep2013  M. Eberhard
;    Modified to support e.g. a fast reader on the 2SIO's 2nd
;    port, instead of a MITS HSR
; 1.02  03Sep2013  M. Eberhard
;    initial read from parallel ports to clear latches
;    (fixes the OP-80)
; 1.03  09Sep2013  M. Eberhard
;    fix for no checksum loader (see 7 below.)
; 1.04  12Mar2014  M. Eberhard
;    Search for end of RAM before initializing ports, so the
;    UART has time to finish echoing, upon entry
; 1.05  05Jun2014  M. Eberhard
;    Ignore sense switch A11 (see 9 below)
; 2.00  11Aug2014  M. Eberhard
;    Major rewrite: copy to RAM and run from there, so it works
;    with an 8800b Turnkey Module too
; 2.01 24Aug2014  M. Eberhard
;   Also move I/O port setup to RAM code, so that MBLe will
;   work with older Turnkey modules, with just the 88-SYS-CLG
;   rework (Which may disable PROM on any IN or OUT
;   instruction). Also tidy up comments.
; 3.00 16Jan2016 M. Douglas
;   Make the PROM position independent by making the RAM
;   sizing and relocation routines position independent.
;   Change the address relocation technique to free up the
;   space required for the position independent code.
;   Eliminate the two second start-up delay by sizing RAM with
;   a page-by-page search instead of a byte-by-byte search.
;
; Written to assemble with ASM by Digital Research.
;
; Thanks to Geoff Harrison for his MBL disassembly, which I
; plagerized freely.
;
;** Differences between MITS MBL and MBLe **
;
; 1) The code starts off by relocating itself to the highest
;    page of RAM that is found, so that it will still work on
;    a Turnkey Module that phantoms the PROMs upon a IN from
;    port FFh (the Sense Switches), or any IN or OUT
;    instruction (i.e. 88-SYS-CLG Turnkey Modules)
; 2) All HSR support is eliminated, including 88-4PIO Port 1
;    initialization and code for starting the HSR transport.
; 3) The second 88-2SIO port (port 1) is initialized.
; 4) The 88-HSR entry in PTABLE is replaced with an entry for
;    the 8-2SIO port 1. See sense switch table below.
; 5) PTABLE has an 8th entry, which is the same as the 7th
;    (2SIO port 1). Testing for illegal sense switch setting
;    is 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 leader character is 0, then no checksum loader
;    will be skipped.
; 8) The end-of-memory hunt to find the end of RAM occurs before
;    the I/O ports get initialized, instead of afterwards. This
;    gives time for the UART to finish transmitting the echo of
;    e.g. the 'T' character, when entering MBLe from UBMON.
; 9) Sense switch A11 is ignored when getting the load device,
;    rather than generating an I error. This allows A11 to be
;    used for something else - e.g. selecting a boot disk.
;=================================================================
; Program Notes
; Since the 8800b Turnkey Module disables PROMS whenever an IN
; instruction accesses port FFh, this code cannot execute from
; PROM - at least not from the point where the Sense Switches
; are read onwards. Additionally, some versions of the Turnkey
; Module are broken, and will disable PROM when *any* IN
; instruction is executed. This means that MITS's MBL, as well
; as versions 1.xx of MBLe, will not work in a Turnkey Module.
;
; MBLe 3.00 Strategy:
; 1) Search the memory space for the highest actual RAM, as
;    MITS's MBL did. This page of memory will be used not only
;    for the stack, but also for the relocated MBLe code.
; 2) Copy code into the high RAM page that was found in step 1.
;    (This is called the RAM Execution Page.) The high byte
;    of addresses are relocated to the RAM execution address
;    as the bytes are copied.
; 3) Jump to the RAM code, and run from there - never to
;    return to PROM.
;
; The RAM page is laid out as follows:
;  * The high portion (From RAMPAG up to FFh) contains the
;    relocated MBLe code
;  * Immediately below this is the stack, initialized to RAMPAG
;    and growing downward. (Note that a PUSH decrements the
;    stack pointer before writing to the stack.)
; The stack therefore has as much space as is occupied by the
; bit of code that executes from PROM: plenty of room.
;
; Although this uses more memory in the highest page of RAM,
; it will behave the same as MITS's MBL, because both programs
; still abort with an "M" error if a Load Record attempts to
; write anywhere into the page that contains the stack.
;
;=================================================================
; 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.
                ;
                ; MBL and MBLe are able to read any such Altair file by simply
                ; skipping over the Leader and Checksum Loader, and loading
                ; the Load and Go Records directly.
                ;
                ; MBLe 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 A (2 stops)   000b
                ;  88-2SIO Port A (2 stops)   001b
                ;  88-SIO                     010b
                ;  88-ACR                     011b
                ;  88-4PIO                    100b
                ;  88-PIO                     101b
                ;  88-2SIO Port B (2 stops)   110b
                ;  88-2SIO Port B (2 stops)   111b
                ;
                ; Prior to Basic 4.0, MITS used different Sense Switch settings
                ; to specify the console device. You can load an older tape
                ; with MBLe by setting the switches according to the above
                ; table and starting the load. After MBL has skipped over the
                ; Checksum Loader on the tape and has begun to load the Load
                ; Records (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.
                ;-----------------------------------------------------------
                ; 8080 EQUATES
                ;-----------------------------------------------------------
00CA =          JZOP    EQU     0CAH    ;JZ OPCODE
00C2 =          JNZOP   EQU     0C2H    ;JNZ OPCODE
                ;-----------------------------------------------------------
                ; 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)
                ;-----------------------------------------------------------
                ;SENSE SWITCH EQUATES
                ;-----------------------------------------------------------
00FF =          SSWTCH  EQU     0FFH    ;FRONT PANEL SWITCH REGISTER
0007 =          LDMASK  EQU     007H    ;LOAD DEVICE MASK <-ME (WAS 00FH)
                ;-----------------------------------------------------------
                ;88-2SIO EQUATES
                ;-----------------------------------------------------------
                ;88-2SIO REGISTERS

0010 =          S2CTLA  EQU     10H     ;ACIA A CONTROL OUTPUT PORT
```

```
0010 =          S2STAA  EQU     10H     ;ACIA A STATUS INPUT PORT
0011 =          S2TXDA  EQU     11H     ;ACIA A TX DATA REGISTER
0011 =          S2RXDA  EQU     11H     ;ACIA A RX DATA REGISTER
0012 =          S2CTLB  EQU     12H     ;ACIA B CONTROL OUTPUT PORT
0012 =          S2STAB  EQU     12H     ;ACIA B STATUS INPUT PORT
0013 =          S2TXDB  EQU     13H     ;ACIA B TX DATA REGISTER
0013 =          S2RXDB  EQU     13H     ;ACIA B RX DATA REGISTER

                ;ACIA CONTROL REGISTER BITS

0001 =          S2DS1   EQU     00000001B       ;COUNTER DIVIDE SEL 1
0002 =          S2DS2   EQU     00000010B       ;COUNTER DIVIDE SEL 2
0004 =          S2WS1   EQU     00000100B       ;WORD SELECT 1
0008 =          S2WS2   EQU     00001000B       ;WORD SELECT 2
0010 =          S2WS3   EQU     00010000B       ;WORD SELECT 3
0020 =          S2TC1   EQU     00100000B       ;TX CONTROL 1
0040 =          S2TC2   EQU     01000000B       ;TX CONTROL 2
0080 =          S2RIE   EQU     10000000B       ;RX INT ENABLE

0003 =          S2RST   EQU     00000011B       ;MASTER RESET

                ;ACIA STATUS REGISTER BITS

0001 =          S2RDF   EQU     00000001B       ;RX DATA REG FULL
0002 =          S2TDE   EQU     00000010B       ;TX DATA REG EMPTY
0004 =          S2DCD   EQU     00000100B       ;DATA CARRIER DETECT
0008 =          S2CTS   EQU     00001000B       ;CLEAR TO SEND
0010 =          S2FE    EQU     00010000B       ;FRAMING ERROR
0020 =          S2ORE   EQU     00100000B       ;RX OVERRUN ERROR
0040 =          S2PE    EQU     01000000B       ;PARITY ERROR
0080 =          S2IRQ   EQU     10000000B       ;INTERRUPT REQUEST
                ;---------------------------------------------------------------
                ;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              ;RECEUVE DATA

                ;STATUS REGISTER BITS

0001 =          ACRIDR  EQU     00000001B       ;INPUT DEV RDY (RX BUF FULL)
0004 =          ACRPE   EQU     00000100B       ;PARITY ERROR
```

```
0008 =          ACRFE   EQU     00001000B       ;FRAMING ERROR
0010 =          ACRDOV  EQU     00010000B       ;DATA OVERFLOW
0080 =          ACRODR  EQU     10000000B       ;OUTPUT DEV RDY (TX BUF EMPTY)
                ;---------------------------------------------------------------
                ;88-4PIO EQUATES
                ;NOTE: THE 88-HSR USES PORT 1 OF THE 88-4PIO
                ;---------------------------------------------------------------
                ;88-4PIO REGISTERS

0020 =          P4CA0   EQU     20H             ;PORT 0 SECTION A CTRL/STATUS
0021 =          P4DA0   EQU     21H             ;PORT 0 SECTION A DATA
0022 =          P4CB0   EQU     22H             ;PORT 0 SECTION B CTRL/STATUS
0023 =          P4DB0   EQU     23H             ;PORT 0 SECTION B DATA
0024 =          P4CA1   EQU     24H             ;PORT 1 SECTION A CTRL/STATUS
0025 =          P4DA1   EQU     25H             ;PORT 1 SECTION A DATA
0026 =          P4CB1   EQU     26H             ;PORT 1 SECTION B CTRL/STATUS
0027 =          P4DB1   EQU     27H             ;PORT 1 SECTION B DATA

                ;CONTROL REGISTER BITS

0001 =          P4C1C0  EQU     00000001B       ;C1 CONTROL BIT 0
0002 =          P4C1C1  EQU     00000010B       ;C1 CONTROL BIT 1
0004 =          P4DDR   EQU     00000100B       ;DATA DIRECTION REGISTER
0008 =          P4C2C3  EQU     00001000B       ;C2 CONTROL BIT 3
0010 =          P4C2C4  EQU     00010000B       ;C2 CONTROL BIT 4
0020 =          P4C2C5  EQU     00100000B       ;C2 CONTROL BIT 5
0040 =          P4IC2   EQU     01000000B       ;C2 INTERRUPT CONTROL BIT
0080 =          P4IC1   EQU     10000000B       ;C1 INTERRUPT CONTROL BIT

                ;STATUS REGISTER BITS

0080 =          P4RDF   EQU     10000000B       ;RX DATA REG FULL
0040 =          HSRRDF  EQU     01000000B       ;RX DATA REG FULL for HSR
                ;---------------------------------------------------------------
                ;88-PIO EQUATES
                ;---------------------------------------------------------------
                ;88-PIO REGISTERS

0004 =          PIOCTL  EQU     04              ;CONTROL PORT
0004 =          PIOSTA  EQU     04              ;STATUS
0005 =          PIOTXD  EQU     05              ;TRANSMIT DATA
0005 =          PIORXD  EQU     05              ;RECEIVE DATA

                ;STATUS REGISTER BITS

0002 =          PIORDF  EQU     00000010B       ;RX DATA REG FULL
                ;---------------------------------------------------------------
                ;SINGLE-BYTE ERROR MESSAGES
                ;---------------------------------------------------------------
0043 =          CERMSG  equ     'C'     ;checksum error
004D =          MERMSG  equ     'M'     ;memory error
004F =          OERMSG  equ     'O'     ;overwrite error

                ;---------------------------------------------------------------
                ;RELOCATION EQUATES
                ;---------------------------------------------------------------

                ;  ORG statement
                ;     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 that the value
                ;     of the ORG MSB never appears in the assembled code other
                ;     than as the MSB of an address. FD00 works well for this.
```

```
FD00                    org     0FD00h
00FD =          ADRMARK equ     ($ SHR 8)       ;address mark value
0080 =          STACK0  equ     80h             ;page zero stack


                ;===============================================================
                ; Start of Code
                ;===============================================================

FD00 F3                 di                      ;interrupts off during load

                ;---------------------------------------------------------------
                ;  Size RAM (position independent). Assumes page 0 exists,
                ;     then checks byte C9 in each page until a page is found
                ;     that is not RAM.
                ;---------------------------------------------------------------
FD01 318000             lxi     sp,STACK0       ;setup a stack on page zero
FD04 21C900             lxi     h,RET           ;H=0, L=RET instruction
FD07 E5                 push    h               ;RET at STACK0-2
FD08 CD7E00             call    STACK0-2        ;puts addr of SZLOOP in stack RAM

                ; SZLOOP - check a byte in each page until RAM not found

FD0B 3B         SZLOOP: dcx     sp              ;point SP to SZLOOP address
FD0C 3B                 dcx     sp              ;    in stack memory

FD0D 24                 inr     h               ;move to next 256 byte page
FD0E 7E                 mov     a,m             ;read from address in hl
FD0F 47                 mov     b,a             ;save original value in b
FD10 2F                 cma                     ;form and write inverted value
FD11 77                 mov     m,a
FD12 BE                 cmp     m               ;read and compare
FD13 70                 mov     m,b             ;restore original value
FD14 C8                 rz      ;jz SZLOOP

                ;---------------------------------------------------------------
                ;  Move PROM image to high RAM (position independent).
                ;     On entry, HL is within the 1st page AFTER the end of
                ;     RAM. At the current stack pointer is the address of
                ;     SZLOOP from the RAM sizing loop above.
                ;---------------------------------------------------------------
FD15 C1                 pop     b               ;BC->SZLOOP in PROM
FD16 0E2C               mvi     c,NOTREL AND 0ffH       ;BC->NOTREL in PROM

FD18 25                 dcr     h               ;point HL inside last page of RAM
FD19 EB                 xchg                    ;form RAM destination in DE
FD1A 1E23               mvi     e,MOVELP AND 0ffH       ;DE->MOVELP in RAM

FD1C 21E1E9             lxi     h,0E9E1h        ;H=PCHL,L=POP H
FD1F E5                 push    h               ;POP H, PCHL at STACK-4, STACK-3
FD20 CD7C00             call    STACK0-4        ;PROM MOVELP in stack mem and HL

                ; Move code from PROM to RAM. If a byte matches the MSB of the
                ;     assembled ORG address (i.e., it is an address MSB), then
                ;     it is replaced with the destination RAM MSB.

FD23 3B         MOVELP: dcx     sp              ;point SP to MLOOP address
FD24 3B                 dcx     sp              ;    in stack memory

FD25 7E                 mov     a,m             ;Get next EPROM byte

FD26 FEFD               cpi     ADRMARK         ;relocatable address byte?
```

```
 FD28 C5                push    b               ;put jump address on stack
 FD29 C0                rnz     ;jnz NOTREL
 FD2A C1                pop     b               ;remove unused jump address
 FD2B 7A                mov     a,d             ;a=MSB of code image in RAM

 FD2C 12        NOTREL: stax    d               ;move byte to RAM

 FD2D 1C                inr     e               ;bump pointers
 FD2E 2C                inr     l
 FD2F C0                rnz     ;jnz MOVELP     copy to end of 256 byte page

                ; code image has been copied to high RAM and addresses relocated.
Init
                ;   the stack pointer just below the code, then jump to the code.

 FD30 62                mov     h,d             ;HL and DE in RAM execution page
 FD31 2E35              mvi     l,RAMCOD AND 0ffh       ;HL->entry address in RAM
 FD33 F9                sphl                    ;stack grows down from start
 FD34 E9                pchl                    ;jump to start

                ;===============================================================
                ; RAM Execution Code
                ; All of the following code gets copied into the RAM Execution
                ; Page (which is the highest page of RAM that was discovered
                ; during initialization).
                ; On Entry:
                ;    d = h = RAM Execution Page
                ;===============================================================

                ; Delay 1/10s to allow time for a 110 baud character to finish
                ;    transmission. A character may have been typed in a monitor
                ;    (e.g., UBMON) just prior to entering MBL. The RAM size and
                ;    copy loops above provide a max delay of about 10ms.

 FD35 018D20    RAMCOD: lxi     b,8333          ;1/10s, 24 cycle loop @2mhz

 FD38 0B        DELAY:  dcx     b               ;5
 FD39 78                mov     a,b             ;5
 FD3A B1                ora     c               ;4
 FD3B C238FD            jnz     DELAY           ;10

                ;---------------------------------------------------------------
                ; Reset all known load devices
                ; Note that a bug in the 88-SYS-CLG rework to older Turnkey
                ; Modules will cause any IN or OUT instruction to occasionally
                ; cause the PROMs to become disabled. For this reason, this
                ; initiaization is done after the code relocates to RAM.
                ; On Entry:
                ;    a = 0
                ;    h = d = RAM Execution Page
                ; On Exit:
                ;    h = d = RAM Execution Page
                ;---------------------------------------------------------------
                ;Make 4PIO 'A' channels inputs and 'B' channels outputs

 FD3E D320              out     P4CA0           ;access 4PIO Port 0A DDR
 FD40 D321              out     P4DA0           ;set 4PIO Port 0A as input

 FD42 D322              out     P4CB0           ;access 4PIO Port 0B DDR
 FD44 2F                cma                     ;0FFH
 FD45 D323              out     P4DB0           ;set 4PIO Port 0B as output

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

```
FD47 3E2C               mvi     a,2cH   ;bits 0,1: C1 input active low, int off
                                        ;bit 2: access data reg
                                        ;bits 3-5: C2 output handshake

FD49 D320               out     P4CA0           ;4PIO Port 0A control
FD4B D322               out     P4CB0           ;4PIO Port 0B control

                ;Send reset command to both 2SIO ports

FD4D 3E03               mvi     a,S2RST         ;2SIO reset
FD4F D310               out     S2CTLA          ;2SIO Port A
FD51 D312               out     S2CTLB          ;2SIO Port B

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

FD53 3E11               mvi     a,11H           ;8N2, /16
FD55 D310               out     S2CTLA          ;2SIO Port 0 control
FD57 D312               out     S2CTLB          ;2SIO Port 1 control

                ;---------------------------------------------------------------
                ; Patch the GETBYT routine with the correct parameters for the
                ; load port that is specified by Sense Switches 2:0
                ; On Entry & Exit:
                ;    h = d = RAM Execution Page
                ;---------------------------------------------------------------
FD59 DBFF               in      SSWTCH                  ;read sense switches
                                                        ;This also disables PROMS...
FD5B E607               ani     LDMASK                  ;bits specifies load device

FD5D 87                 add     a                       ;2 bytes/entry
FD5E C6ED               adi     PTABLE and 0FFh         ;Look up in PTABLE
FD60 5F                 mov     e,a             ;de=PTABLE((SWITCHS) <2:0>)

FD61 1A                 ldax    d
FD62 2EE6               mvi     l,(GBDP+1) and 0FFH     ;Data port addr place
FD64 1F                 rar                             ;move jnz flag int carry
FD65 77                 mov     m,a                     ;install data port addr

FD66 2EDF               mvi     l,(GBSP+1) and 0FFh     ;Status port addr place
FD68 3D                 dcr     a                       ;stat port = data port-1
FD69 77                 mov     m,a                     ;install stat port addr

FD6A 1C                 inr     e                       ;next table entry is
FD6B 1A                 ldax    d                       ;..the status port mask
FD6C 2EE1               mvi     l,(GBMASK+1) and 0FFh   ;status mask place
FD6E 77                 mov     m,a                     ;install stat port mask

FD6F D275FD             jnc     ITSJZ                   ;test jnz flag
FD72 2C                 inr     l                       ;jnz right after mask
FD73 36C2               mvi     m,JNZOP                 ;install jnz opcode

                ;-----------------------------------------------
                ; Flush external data latches for e.g. the OP-80
                ; or flush garbage from UARTs
                ; On Entry & Exit:
                ;    d = RAM Execution Page
                ;-----------------------------------------------
FD75 CDE5FD     ITSJZ:  call    GETNOW

                ;---------------------------------------------------------------
                ; 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
                ;    The 1st byte of the checksum loader has already been read
                ;-----------------------------------------------------------------
FD78 CDDEFD             call    GETBYT          ;get 1st byte
FD7B 4F                 mov     c,a             ;number of bytes in loader
FD7C B7                 ora     a               ;Null leader?
FD7D CA8FFD             jz      RCHUNT          ;y: skip leader


FD80 CDDEFD     LDSKIP: call    GETBYT          ;get another byte
FD83 B9                 cmp     c
FD84 CA80FD             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
                ; On Exit:
                ;    d = RAM Execution Page
                ;    The checksum loader has been skipped
                ;-----------------------------------------------------------------
FD87 0D                 dcr     c               ;since we got a byte already

FD88 CDDEFD     CLSKIP: call    GETBYT          ;get a loader byte
FD8B 0D                 dcr     c
FD8C C288FD             jnz     CLSKIP

                ;-----------------------------------------------------------------
                ; Main Record-Loading Loop
                ;
                ; Hunt for a sync character - either for another Load Record
                ; or for the Go Record. Ignore all else.
                ; On Entry:
                ;    c = 0
                ;    d = RAM Execution Page
                ; On jmp tp LDREC:
                ;    c = 0
                ;    d = RAM Execution Page
                ;    RCHUNT address is on the stack
                ;-----------------------------------------------------------------
FD8F 62         RCHUNT: mov     h,d             ;restore page address
FD90 2E8F               mvi     l,RCHUNT and 0FFh
FD92 E5                 push    h               ;create return address

FD93 CDDEFD             call    GETBYT          ;hunt for sync character
FD96 FE3C               cpi     ALTPLR          ;load record sync byte?
FD98 CAA3FD             jz      LDREC           ;Y: go load the record

FD9B FE78               cpi     ALTEOF          ;EOF record sync byte?
FD9D C0                 rnz                     ;N: ignore

                ; Fall into GO record execution

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

```
                ; Go Record: Get the Go Address and go there
                ;
                ; On Entry:
                ;    Go-Record sync byte has already been read
                ;-------------------------------------------------------------
FD9E CDDAFD             call    GETWRD          ;get a,l=address
FDA1 67                 mov     h,a             ;high byte
FDA2 E9                 pchl                    ;go there

                ;-------------------------------------------------------------
                ; Load Record: Read and store data from a Load Record
                ;
                ; On Entry:
                ;    The Load Record sync byte has already been read
                ;    c = 0
                ;    d = RAM Execution Page
                ;    RCHUNT's address is on the stack
                ; On Return (to RCHUNT):
                ;    c = 0
                ;    d = RAM Execution Page
                ;    A complete Load Record's data has been loaded into RAM
                ;-------------------------------------------------------------
FDA3 CDDEFD     LDREC:  call    GETBYT          ;get record byte count
FDA6 41                 mov     b,c             ;c=0: initialize checksum
FDA7 4F                 mov     c,a             ;c counts data bytes

FDA8 CDDAFD             call    GETWRD          ;get load address into a,l
FDAB 67                 mov     h,a             ;hl = record load address

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

FDAC 7A         LRLOOP: mov     a,d             ;d=RAM Execution Page
FDAD BC                 cmp     h               ;error if same page as load address
FDAE 3E4F               mvi     a,OERMSG        ;overwrite error message
FDB0 CAC8FD             jz      ERDONE          ;error exit if overwrite

FDB3 CDDEFD             call    GETBYT          ;get a data byte
FDB6 77                 mov     m,a             ;store data byte
FDB7 BE                 cmp     m               ;did it store correctly?

                ;Entry at MERR from end-of-RAM search (while running in ROM)

FDB8 3E4D       MERR:   mvi     a,MERMSG        ;Memory Error message
FDBA C2C8FD             jnz     ERDONE          ;error exit if mismatch

FDBD 23                 inx     h               ;bump dest pointer
FDBE 0D                 dcr     c               ;bump byte count
FDBF C2ACFD             jnz     LRLOOP          ;loop through all bytes

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

FDC2 CDDEFD             call    GETBYT          ;test record's checksum
FDC5 C8                 rz                      ;match: get another record

FDC6 3E43               mvi     a,CERMSG        ;Checksum Error message

                ; Fall into ERDONE

                ;---End--------------------------------------------------------
                ;Error handler:
                ; Save error code and address at beginning of memory
```

```
                ; Hang writing the error code forever, to all known consoles.
                ;-------------------------------------------------------------
FDC8 320000     ERDONE: sta     00000H
FDCB 220100             shld    00001H
FDCE FB                 ei

FDCF D301       ERHANG: out     SIOTXD          ;SIO
FDD1 D311               out     S2TXDA          ;2SIO
FDD3 D305               out     PIOTXD          ;PIO
FDD5 D323               out     P4DB0           ;4PIO port 0
FDD7 C3CFFD             jmp     ERHANG

                ;---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
                ;-----------------------------------
FDDA CDDEFD     GETWRD: call    GETBYT
FDDD 6F                 mov     l,a

                ; Fall into GETBYT

                ;---Subroutine-----------------------------------------
                ; Wait for and get a byte from the transfer port
                ; This code gets modified once the input port is known
                ; On Entry:
                ;   b=checksum so far
                ; On Exit:
                ;   a = input character
                ;   Z set if received byte matched previous checksum
                ;   b := b+a
                ;-----------------------------------------------------
                GETBYT:
FDDE DB00       GBSP:   in      0       ;(Status Port Address)read status
FDE0 E600       GBMASK: ani     0       ;(Port Mask)

FDE2 CADEFD             jz      GETBYT  ;(may become jnz) wait for data

                ; Fall into GETNOW

                ;---Subroutine-----------------------------------------
                ; Get a byte from the transfer port
                ; This code gets modified once the input port is known
                ; On Entry:
                ;   b=checksum so far
                ; On Exit:
                ;   a = input character
                ;   Z set if received byte matched previous checksum
                ;   b := b+a
                ;-----------------------------------------------------
                GETNOW:                 ;call to flush port
FDE5 DB00       GBDP:   in      0       ;(Data Port place)get data byte


FDE7 B8                 cmp     b       ;set Z if this byte matched cksum
FDE8 F5                 push    psw     ;temp save
FDE9 80                 add     b       ;update checksum in B
FDEA 47                 mov     b,a
FDEB F1                 pop     psw     ;recover data byte
```

```
FDEC C9                 ret             ;A=byte, B=checksum

                ;---Table---------------------------------------------------------
                ;Port Parameters: One 2-byte entry for each input port:
                ; Byte 1 = Data port address * 2 + JNZ flag
                ; (The status port is assumed to immediately preceed the data
                ; port.)
                ; Byte 2 = ready mask for data input
                ;-----------------------------------------------------------------
FDED 2201       PTABLE: db      S2RXDA*2,S2RDF          ;0:2SIO A (2 stop bits)
FDEF 2201               db      S2RXDA*2,S2RDF          ;1:2SIO A (2 stop bits)
FDF1 0301               db      SIORXD*2+1,SIOIDR       ;2:SIO
FDF3 0F01               db      ACRRXD*2+1,ACRIDR       ;3:ACR
FDF5 4280               db      P4DA0*2,P4RDF           ;4:4PIO Port 0
FDF7 0A02               db      PIORXD*2,PIORDF         ;5:PIO
FDF9 2601               db      S2RXDB*2,S2RDF          ;6:2SIO B (2 stop bits)

                ; The last table entry is just a copy to fill it out.

FDFB 2601               db      S2RXDB*2,S2RDF          ;7:2SIO B (2 stop bits)

FDFD                    end
```