Altair Software Development using CP/M

Martin Eberhard 20 January 2011
Updated 7 June 2013

This paper describes a simple environment for developing machine-code software for the Altair Computer. Part of the goal is to develop software in the spirit of the period when the Altair was commercially available, particularly when Altair software was loaded from paper tape.

Working with Altair Paper Tapes

Altair paper tapes were designed to be loaded via a Bootstrap Loader that may be toggled into the Altair from the front panel, or ROM-resident in a tiny optional ROM. Because front panel toggling is tedious and prone to errors, MITS went to extremes to minimize the length of the Bootstrap Loaders – a typical Bootstrap Loader is only 22 bytes long.

The Altair Paper Tape Format

An Altair paper tape comprises eight sections:

1. **Null Leader**. This is where the paper label gets pasted, and is never read by the computer. It should be long enough to wrap around the roll of paper tape and protect it.
2. **Leader**. This is typically 64 to 256 bytes of a specific character that is ignored by the bootstrap loader. Note that the specific character differs between versions of MITS software.
3. **Checksum Loader**. This program gets loaded and executed by the Bootstrap Loader. It loads the Program File, verifies correct load, and executes the program. It is written on the paper tape in reverse-order.
4. **Checksum Loader Gap**. This is a string of nulls after the Checksum Loader and before the Program File. On earlier versions MITS software, this gap was 0 bytes long. On newer versions, it is about 12 bytes long. This gap gives the Checksum Loader time to initialize.
5. **Program File**. This is the program that will be executed, for example, Altair Basic. It is written as a sequence of checksummed records.
6. **Go Record Gap**. This is a string of nulls after the Program File and before the Go Record. On earlier versions of MITS software, this gap was 0 bytes long. On newer versions, it is about 12 bytes long. (This gap seems unnecessary, as it takes no more time for the Checksum Loader to get ready for a Go Record than it takes to get ready for another Program File record.)
7. **Go Record**. This is a 3-byte sequence that tells the Checksum Loader where to begin execution. Most (all?) MITS programs execute at 0000h.
8. **Null Trailer**. This is a length of blank tape at the end that is not read by the computer. It should be long enough to protect the end of the Go Record from abuse caused by paper tape winders, etc. I suggest at least 18” of Null Trailer.

Bootstrap Loader

The Bootstrap Loader loads the Checksum Loader from the beginning of the paper tape into memory (last byte first), and then executes it. The paper tape must start someplace in the Leader, and the

Bootstrap Loader will ignore the Leader characters to find the beginning of the Checksum Loader. The Checksum Loader's load address is hard-coded into the Bootstrap Loader.

MITS published several different Bootstrap Loaders – one for each of the several I/O ports that they supported. They also changed these Bootstrap Loaders with different versions of their Checksum Loader (which changed a couple of times as MITS published newer versions of Altair Basic).

For each IO port's Bootstrap Loader, only one byte changed as they revised the Checksum Loader. This byte both defines the ignored Leader character and also is the length of the Checksum Loader + 1.

The various versions of the Altair Bootstrap Loader are printed in the Altair manuals. Here is the Bootstrap Loader for the 88-2SIO board. Note that the byte at Address 011 changes with Basic version.

Octal Address	Octal Data	Mnemonic	Comment
000	076	MVI A,ARESET	RESET 2SIO PORT 0 ACIA
001	003		
002	323	OUT A0CTRL	
003	020		
004	076	MVI A,A0INIT	INITIALIZE 2SIO PORT 0
005	**021**		2 STOP BITS (025 FOR 1 STOP BIT)
006	323		
007	020		
010	041	LXI H,LADDR	HL=LAST ADDRESS OF CHECKSUM LOADER
011	**302**		256 FOR BASIC 3.2, 302 FOR BASIC 4.X
012	**0X7**		017 FOR 4K, 037 FOR 8K, 077 FOR EXT.
013	061	LXI SP,STACK	SET UP STACK FOR EFFICIENT RETURNS
014	032		STACK IS RIGHT AT THE END OF THIS CODE
015	000		
016	333	IN A0STAT	GET READER STATUS
017	020		
020	017	RRC	TEST RX DATA REG FULL
021	320	RNC	RETURN IF NO DATA YET
022	333	IN A0DATA	GET READER DATA
023	021		
024	275	CMP L	IS THIS A LEADER CHARACTER?
025	310	RZ	IGNORE IF SO
026	055	DCR L	NEXT ADDRESS, SET Z FLAG IF DONE
027	167	MOV M,A	PUT CHECKSUM LOADER BYTE IN MEMORY
030	300	RNZ	RETURN IF L<>0 - NOT DONE YET
031	351	PCHL	JUMP TO CHECKSUM LOADER
032	013		RETURN ADDRESS ON THE STACK
033	000		

Checksum Loaders

MITS used a couple of different Checksum Loaders with different versions of their software. The primary difference was how they interpreted the front panel switches, used to specify the port used for loading, and also to specify the port used as a console by the loaded program. Because the switch settings are used by both the Checksum Loader and by the loaded program (e.g. Altair Basic), you must use the

correct checksum loader for a given program. (For example, using the Checksum Loader for Altair Basic version 4.1 with the program file for Altair Basic version 3.1 will yield undesirable results.)

Here is how the Checksum Loader and Altair Basic version 4.X interpret the front panel switches. Switches A8-A11 define the Load Port (used by the Checksum Loader), and switches A12-A15 determine the Console Port for Altair Basic. The switches are interpreted as follows:

Switch					
A15	A14	A13	A12		
A11	A10	A9	A8	PORT	CHANNELS (OCTAL ADDRESSES)
0	0	0	0	2SIO, 2 STOP BITS	20, 21
0	0	0	1	2SIO, 1 STOP BIT	20, 21
0	0	1	0	SIO	0, 1
0	0	1	1	ACR (Cassette)	6, 7
0	1	0	0	4PIO	40, 41, 42, 43
0	1	0	1	PIO	4, 5
0	1	1	0	HSR (High-Speed Reader)	46, 47

Note that neither the ACR nor the HSR should be assigned to be the Console Port.

The version of the Checksum Loader that was meant for Altair Basic 4.X uses a table (beginning at address XXACh) to specify the behavior of the supported I/O ports. Each table has 3 entries: the Control Port address, either a “JZ” instruction or a “JNZ” instruction, and a “Receiver Data Ready” bit mask. The Data Port address is assumed to be one greater than the Control Port address for all supported ports. The JZ/JNZ byte specifies the polarity of the Receiver Data Ready bit in the Control Port: JZ means active-high ready bit; JNZ means active-low. The bit mask simply masks off all bits read from the Control Port except the Receiver Data Ready bit.

Because of its table-driven nature, changing or adding a load port to this Checksum Loader is relatively simple: replace one of the table entries with the required values for your new port. Note that changing the Checksum Loader’s table will not make your new port work with Altair Basic, which has its own table that must be found and modified.

If you add a new load port to this table, it will take switch setting 0111, and you must also change the maximum allowed switch value (address XX21h) in the Checksum Loader. Additionally, since adding another table entry will change the length of the Checksum Loader, you must change one byte in the Bootstrap Loader, and change the Leader Character, since these are both equal to the length of the Checksum Loader + 1. For this reason, it is a lot easier to replace a table entry (for example the HSR entry) with the parameters for your new Load Port.

The source code for the Altair Checksum Loaders that came with Altair Basic version 3.2 and 4.1 are published together with this document.

The Program File

The Program File comprises a number of Program Load Records. These are generally written in ascending address-order, with the exception of the first Program Load Record, the one that loads at address 0000h. This Program Load Record is generally loaded last, so that the bootstrap loader will still be in memory should a checksum error occur during loading.

The Program Load Record begins with a 3Ch, followed by a 1-byte Program Byte Count, and a 2-byte Memory Load Address where this block should be stored, low address byte first. Next is a variable

number of bytes of program data, as specified by the Program Byte Count. The block ends with a 1-byte Checksum, which is simply the sum, truncated to one byte, of all the bytes of data in the Record, including the low and high Memory Load Address bytes, but **not** including the Sync Byte and the Program Byte Count:

Byte	Hex Value	Function
0	3Ch	sync byte
1	NN	Program Byte Count
2	ll	Low byte of Memory Load Address
3	hh	High byte of Memory Load Address
4	<NN bytes>	NNN bytes of program data
NN+4	cc	Checksum byte

The Go Record

The paper tape data ends with a Go Record comprising one byte of 78h followed by the 2-byte Go Address of the program. When the Checksum Loader encounters a Go Record, it jumps to the specified address.

Byte	Hex Value	Function
0	78h	Sync byte
1	ll	Low byte of Go Address
2	hh	High byte of Go Address

The Development Environment

This is a CP/M 2.2-based development environment, using Digital Research's assemblers, ASM or MAC. You can run CP/M 2.2 native on a CP/M machine, or you can run CP/M in emulation on a PC running DOS.

CP/M 2.2 under DOS

To run CP/M 2.2 under Microsoft DOS, you need a program called 22NICE.COM, which is an excellent CP/M emulator, written by Sydex. You can download a "trial" version of this program from http://www.cpm80.com/ or from http://www.gaby.de/edownl.htm. Though Sydex no longer offers 22NICE.COM for sale, if you contact them at www.sydex.com, they will probably sell you a license as they did for me. (While you are at it, purchase a copy of 22DISK.COM, which enables a PC to read and write CP/M-compatible disks.)

22NICE.COM works great in a DOS window under Windows XP. (I have not tried it under Vista or Windows 7.) 22NICE.COM even works great in a DOS window under Windows XP that in turn is running under VMware Fusion on a MacBook Pro, which I find greatly amusing.

The Assembler and Loader

22NICE.COM requires you to run any .COM file through their GENCOM.COM utility to make it executable in their environment. For Digital Research's ASM, you would type "GENCOM ASM.COM". This will rename ASM.COM to ASM.CPM and create a tiny run file called ASM.COM. Now ASM will work as it is supposed to – at the DOS prompt, you can type "ASM MYFILE" to assemble MYFILE.ASM. You will need to do the same thing with LOAD.COM: "GENCOM LOAD.COM". If you want to use MAC (Digital Research's Macro Assembler), then you will need to run MAC.COM through GENCOM as well.

You will likely want to create a little DOS batch file for assembling programs, called MAKE.BAT:

```
ASM %1
LOAD %1
GENCOM %1
```

This will make a 22NICE-executable file from MYFILE.ASM when you type MAKE MYFILE.

You can download Digital Research's ASM and LOAD at http://www.retroarchive.org/cpm/os/os.htm (download the file called STDCPM22.ZIP.) If you want to use the macro-assembler, you can download MAC at http://www.cpm.z80.de/binary.html (Look for the link called MAC BINARY.)

Debugger

Digital Research's DDT works great in this environment. I generally have a DEBUG definition in my code that turns off hardware accesses so that I can debug basic functionality in the 22DISK environment. Again, you will need to run DDT through GENCOM.

Editor

For that true retro feel, you can use Digital Research's ED (available the same place as ASM and LOAD). Or you can use your favorite CP/M screen editor such as WordStar. I recommend using the DOS NOTEPAD editor for simple, reliable text editing when using the 22NICE.COM environment. You can also use WORDPAD, though its tabs are inconsistent with NOTEPAD.

Note that if you use an Apple-based text editor such as the default TextEdit program, you will have problems because Mac's use only a Carriage Return to indicate end-of-line, whereas both CP/M and DOS use a Carriage Return-Line Feed for end-of-line. There are a few Mac-based text editors, such as Text Wrangler that will allow you to change the end-of-line character string to be CP/M compatible.

Working Directory

I strongly recommend running 22NICE.COM in a directory close to the root because of DOS's file-name limitations. I suggest creating a directory called "C:\CPM" for this purpose. In this directory, you should have the following files:

- 22NICE.COM
- CSAVE.COM (one of 22NICE's files)
- GENCOM.COM (another 22NICE file)
- MAKE.BAT (Your MAKE batch file)
- ASM.COM (created by GENCOM)
- ASM.CPM (renamed by GENCOM)
- LOAD.COM (created by GENCOM)
- LOAD.CPM (renamed by GENCOM)
- DDT.COM (created by GENCOM)
- DDT.CPM (renamed by GENCOM)
- MYFILE.ASM (whatever program you are creating)
- MYFILE.HEX (created by ASM)
- MYFILE.PRN (created by ASM)
- MYFILE.CPM (created by LOAD and renamed to have .CPM extension by GENCOM)
- MYFILE.COM (created by GENCOM)

Some Useful Tools

I've found or written a few tools to be very useful for working with paper tape files. Here are the ones that I have found:

Hex File Editor

It is sometimes very useful to edit binary file directly in hex. For example, some of the tape files for Altair Basic that I have found on the web have junk at the end, because the original file was not an even multiple of 128 bytes, and DOS filled in the end with junk (usually 55H). With the right editor, you can trim off this junk or replace it with nulls. Also, when creating tape files, it is useful to look at the results.

For this purpose, I use the program called XVI32.COM, which runs under Windows. You can get it for free here: www.chmaas.handshake.de/delphi/freeware/xvi32/xvi32.htm, though it would be classy to donate at this website.

Terminal Program

You will need a program that can send binary files to an RS-232 serial port, perhaps via a USB-to-RS232C dongle. Unfortunately, many of the available terminal programs do a bad job with binary files, for example, eating (and failing to transmit) EOF characters. The one I found that works is RealTERM version 2.0.0.57. (Note that the preceding version of this program had the "eats EOF characters" problem!)

You can get RealTERM here: http://realterm.sourceforge.net/

Also, not all USB-to-RS-232C dongles do the right thing with binary files – some tend to eat XON and XOFF characters, or eat nulls.

Utilities Written by Me

I have written a few useful CP/M utilities for creating Altair paper tapes. They are briefly described here, and their source code and executable code are published with this document.

PUNCH.COM, PUNVER.COM, and TREAD.COM all are useful if your reader/punch is a Teletype or similar device that communicates via 20 mA current loop, since few PCs support 20 mA current loop. In essence, the Altair serves as a level translator and buffer.

MAKEALT.COM

MAKEALT.COM creates an Altair paper tape file from components. You supply the Program File (in binary, hex, or Altair paper tape format), the Checksum Loader (in hex format), the sizes of the various gaps and leaders on the tape, and an optional human-readable banner that will be punched in the Null Leader. MAKEALT.COM will produce a .TAP file that can be sent directly to a paper tape punch to create a bootable tape.

If you want to replace the Checksum Loader on an existing paper tape fie (for example, to support a new Load Port), you can use MAKEALT.COM to do this: "MAKEALT OUTPUT.TAP=INPUT.TAP CHKSUM.HEX".

PUNCH.COM

PUNCH.COM is a brief Altair program that reads a binary file from the 2SIO port 1, and writes it to 2SIO port 0. The two ports can be at different baud rates. Flow control is via RTS/CTS hardware handshaking. The program incorporates a 256-byte buffer, so that the program will work correctly even with a transmitter that does not respond immediately to handshaking.

The idea of PUNCH.COM is to connect port 1 to a PC and port 0 to a paper tape punch (such as a Teletype). The PC can send a tape file to its RS232C port, and the Altair will convert to the right baud rate and communications protocol to create a paper tape.

PUNCH.COM gets loaded and runs at address 100h. (This way, a bootstrap loader can remain at address 0.)

PUNVER.COM

PUNVER.COM is another brief (front panel loadable) program. This program reads a file from port 1, and another from port 0. It compares the two programs until a difference is encountered. When a difference is encountered, the program stops the reader on port 0 (if the reader has a run/stop control), stores information about the difference in memory, and executes a HALT.

Note that PUNVER.COM is not useful for debugging a PC serial port/terminal program that eats certain characters (for example an EOF character), since the same characters that got stripped out when the tape was punched will get stripped out during verify.

PUNVER.COM gets loaded and runs at address 300h. (This way, you can have both PUNCH.COM and PUNVER.COM in memory at the same time.)

TREAD.COM

TREAD.COM is the reverse of PUNCH.COM, reading from paper tape on port 0, and sending the file to port 1. This program allows you to read a paper tape into a PC. It ignores leading nulls on the tape.

TREAD.COM loads and runs at address 0.

An Example

I have a Ghielmetti FER204A high-speed paper tape reader, perfect for loading Basic really quickly. This device communicates via RS232C, and uses hardware handshaking for flow control. I want to load paper tapes with this device using Port 1 of my 88-2SIO, and with a Teletype ASR33 as the console on Port 0. I also want to use Port 1 for sending files from the PC to the Teletype, to be punched on tape, so Port 1 needs to be wired as a standard RS232C port with hardware flow control.

Wiring the Altair 88-2SIO

My Teletype has a Hewlett Packard modification that allows remote control of the punch. This modification requires +12V and -12V as well as RTS to enable this feature. Details of this modification and how to use it are documented elsewhere.

The Ghielmetti FER204A's DB25 connector is pinned out like this:

Pin	RS232C Sig	Direction	Polarity	Note
2	TxD	FER204A→	RS232C Standard	Paper tape data
3	RxD	FER204A←		Not Used
4	RTS	FER204A→	Active Low	Same as pin 20 (DTR) (1)
5	CTS	FER204A←	Active Low	Drive pins 5 & 6 together
6	DSR	FER204A←	Active Low	Drive pins 5 & 6 together
7	GND			Signal Ground
20	DTR	FER204A→	Active Low	Same as pin 4 (RTS) (1)

(1) The FER204A was slightly modified to make this signal active-low.

Port 1

I wired the Port 1 of the Altair 88-2SIO to connect to the high-speed paper tape reader this way:

Port 1 Onboard Setup (9600 Baud, RS232C, hardware handshaking)

Signal	Connect	Connect
RxD	E3 – J1	J2 – S2.7
TxD	E5 – N6	N5 – S2.8
CTS	E2 – I2	I1 – S2.1
DCD	E1 – J3	J4 – S2.2
RTS	E4 – N8	N7 – S2.3
GND	S2.4 – S2.10	
Baud Rate	CLK1 – 9600	

Port 1 Wiring Harness (Standard RS232C pinout, compatible with a 25-pin to 9-pin PC-type serial cable)

P1 Pin	Wire Color	DB25P Pin	8250 Signal	RS232C Signal	Direction
1	Brown	4	CTS	RTS	Port→2SIO
2	Red	20	DTR	DCD	Port→2SIO
3	Orange	5, 6, 8	RTS	CTS, DSR, DCD	2SIO→Port
4	Yellow	7	GND		
5	Green				
6	Blue				
7	Violet	2	RxD	TxD	Port→2SIO
8	Gray	3	TxD	RxD	2SIO→Port
9	No Wire				
10	Black	1	Protective GND		

Port 0

I wired Port 0 of the Altair 88-2SIO to connect to my Teletype this way:

Port 0 Onboard Setup (110 baud, 20 mA current loop, RTS output used to enable TTY punch)

Signal	Connect	Connect	Connect	Connect
RxD	D3 – K12	K11 – Y7	Y8 – S1.6	Y9 – S1.7
TxD	D5 – L1	Z2 – S1.5	S1.4 – S1.10	
PTR Enable	D4 – L2	IC L.4 – M3 (1)	M4 – S1.3	
-12V	IC N.1 – S1.8 (1)			
+12V	IC N.14 – S1.9 (1)			
Baud Rate	CLK0 – 110 Baud			

(1)These connections are made on the solder-side of the 2SIO

Port 0 Wiring Harness (non-standard 20 mA current loop pinout)

P1 Pin	Wire Color	DB25P Pin	Signal Name	Direction
1	Brown	N/C		
2	Red	N/C		
3	Orange	19	–PTR Enable	2SIO→TTY
4	Yellow	3	TxD Return (GND)	2SIO→TTY
5	Green	13	TxD	2SIO→TTY
6	Blue	11	RxD	TTY→2SIO
7	Violet	2	RxD Return	TTY→2SIO
8	Gray	17	-12V	2SIO→TTY
9	White	18	+12V	2SIO→TTY
10	Black	1	GND	

Modifying the Checksum Loader

I decided to replace the Altair Basic 4.X Checksum Loader's HSR option with an option that allows loading from Port 1 of the 2SIO. This required replacing the last 3-byte load port table entry with one that supports the 2SIO. This means that front panel switches A11…A8=0110 will select 2SIO Port 1 for

loading. I used my own program, MAKEALT.COM, to replace the Checksum Loader on the original MITS .TAP files. I used my own PUNCH.COM program to punch the tape using the Teletype.

Modified Table:

```
3FAC 11CA01 LPTABL: DB   SIO2R0,JZ,SIO2RDF    ;0=2SIO PORT0, 2 STOP BITS
3FAF 11CA01         DB   SIO2R0,JZ,SIO2RDF    ;1=2SIO PORT0, 1 STOP BIT
3FB2 01C201         DB   SIORXD,JNZ,SIOIDR    ;2=SIO PORT
3FB5 07C201         DB   ACRRXD,JNZ,ACRIDR    ;3=ACR PORT (CASSETTE)
3FB8 21CA80         DB   PIO4DA0,JZ,PIO4RDF   ;4=4PIO PORT
3FBB 05CA02         DB   PIORXD,JZ,PIORDF     ;5=PIO PORT
                ;   DB   HSRRXD,JZ,HSRRDF     ;6=HSR PORT (HI-SPEED RDR)
3FBE 13CA01         DB   SIO2R1,JZ,SIO2RDF    ;6=2SIO PORT1
```

Modifying the Boot Loader

The boot loader also requires a few minor modifications to load from the 88-2SIO Port 1. (Changes are shown in **bold**.):

Octal Address	Octal Data	Comment
000	076	RESET 2SIO PORT 0
001	003	
002	323	
003	**022**	**CONTROL PORT**
004	076	INITIALIZE 2SIO PORT 1
005	**025**	2 STOP BITS (025 FOR 1 STOP BIT)
006	323	
007	**022**	**CONTROL PORT**
010	041	HL=LAST ADDRESS OF CHECKSUM LOADER
011	302	256 FOR BASIC 3.2, 302 FOR BASIC 4.X
012	**0X7**	017 FOR 4K, 037 FOR 8K, 077 FOR EXT. BASIC
013	061	SET UP STACK FOR EFFICIENT RETURNS
014	032	STACK IS RIGHT AT THE END OF THIS CODE
015	000	
016	333	GET READER STATUS
017	**022**	**STATUS PORT**
020	017	RRC: TEST RX DATA REG FULL
021	320	RETURN IF NO DATA YET
022	333	GET READER DATA
023	**023**	**DATA PORT**
024	275	CMP L: IS THIS A LEADER CHARACTER?
025	310	RETURN IF SO
026	055	DCR L: CHECKSUM LOADER LOADS LAST FIRST
027	167	PUT CHECKSUM LOADER BYTE IN MEMORY
030	300	RETURN IF L<>0 - NOT DONE YET
031	351	PCHL: JUMP TO CHECKSUM LOADER
032	013	RETURN ADDRESS ON THE STACK
033	000	

Using This Configuration

To load Basic 4.1 from the FER204A high-speed paper tape reader:

(1) Load the above bootstrap loader into the Altair at address 0.
(2) Examine address 0
(3) Set the Altair's front panel switches A15…A8 = 00000110 (only A9 and A10 up). This selects the Teletype on Port 0 as Basic's console, and the FER204A on Port 1 as the load device.
(4) Set the paper tape into the FER204A, so that the reader head is somewhere in the Leader.
(5) Toggle the Altair's RUN switch.
(6) The FER204A will begin reading, and in a few moments, the Basic sign-on dialog will be printed on the Teletype.