The Retro Computer Manual

Retro Computer Instruction Manual

A guide to using retro-inspired 8080/Z80 based computers, including the Altair-Arduino and IMSAI 8080esp. Or, to put it another way, "OK you've built your kit - now what? What are you going to do with it?"

Table of contents

Introduction to system architecture

Conceptually, the Altair and IMSAI computers were a very straightforward design. Inside the box was a front panel with LEDs and switches, a power supply, and a "back plane" circuit board with almost much nothing but a set of interconnected 100-pin slots - known as the S-100 Bus.

Figure 1: An IMSAI opened up and wantonly displaying the S-100 Bus

Into those slots would be placed the various cards that comprised the computer system: first the CPU card (with an Intel 8080 or a Zilog Z80 CPU), and then a RAM card (initally composed of static RAM chips).

In theory that's all what was needed to make a working computer. Code could be entered using the front panel switches, and the LEDs provided a form of output.

In practice, this wasn't enough and users also installed floppy and hard drive controller cards,

ROM cards, more RAM, serial cards (for driving terminals and printers), parallel cards (for more printers), and even graphic display cards and camera capture cards.

Memory Map

The 8080 and Z80 CPUs could address up to 64Kb of memory(*) and this would start at address 0000h and (funds permitting) go all the way to FFFFh. Memory was used to store program code and any data, and when the user set all the front panel address switches to off and toggled RESET and EXAMINE, the program counter was set to 0000h and the computer would display the contents of memory address 0000h. (Static memory chips would often contain random data, and so the data LEDs would display a random pattern.)

Description	Address
Start of memory	0000h
Maximum supported memory	FFFFh

Initially the Altair shipped with 256 bytes of memory, and the IMSAI with 1Kb of memory. Why? Because memory was very expensive in 1975!

However, it soon became apparent that a lot more would be needed in order to run useful software: Microsoft's smallest version of BASIC alone required 4Kb, and so expansion cards were soon on the wish list of every computer owner.

ROMs were expensive, and so sometimes it was necessary to "boot strap" the computer manually to get a disk drive working instead on relying on code in a ROM. This would entain entering a set of assembly language instructions directly using the front panel and executing them - hopefully triggering the disk drive to load more, and execute those: thus booting up CP/M or some other application. We'll cover how you might do that in another section.

Memory Map using CP/M

If CP/M was loaded, memory was arranged in a specific way. Again, the memory started at 0000h, but this time CP/M included some of its code there: a set of jumps to useful routines, and of course, code at 0000h that would jump to the code that brought up the famous A: prompt to allow the user to enter commands.

The CP/M memory map looks like this:

Description	Address
Start of memory	0000h
'Low storage' area..	0000h
..used by CP/M	00ffh
TPA - free memory for programs aka "Transient Program Area"	0100h
CCP - console command processor	xxxxh
BDOS - Basic Disk Operating System	xxxxh
BIOS - Basic Input Output System	xxxxh
Maximum possible memory space	FFFFh

Description	Address

The exact address of the CCP, BDOS and BIOS components of CP/M would vary from system to system. Depending on much memory you had installed, you would literally build your own custom version of CP/M to make full use of it. Your bespoke CP/M would also include the necessary drivers for the floppy and hard drives you added to your system.

CP/M might not be what we think of an Operating System today. It didn't provide a graphical user interface, or allow multuple programs to run at once. It didn't provide networking, or automatic updating of software or anything that we take for granted today.

CP/M was a lot more basic, and in essence wasn't much more than an implemetation of an agreed set of basic functions, tailored for a specific set of hardware, coupled with a command line to let the user launch utilities and programs, and the promise that any application would be loaded in at address 0100h and executed. And that was it.

Here is an example of some code. Any application can call the CP/M "Print" function by calling the BDOS start address (00005h) and passing in the special value of 9, and the code in the BDOS will do the actual printing. It's like a primative SDK.

```
; 8080 ASM code to display "hello world"

PRINT EQU 9                     ; The Print function
BDOS EQU 5                      ; The address of the BDOS code vector

org 0100h                       ; All user code starts at 0100h

LXI D, STRING                   ; Get the address of the string to print
MVI C, PRINT                    ; Load register C with the magic code for printing
CALL BDOS                       ; Call the BDOS
RET                              ; End program

; The BDOS Print needs a string ending in $
STRING: DB 'Hello World!', 10,13,'$'
```

(BTW, if you like, you can enter this code in ED, and them assemble it with ASM, link it with LOAD and then run it under CP/M. We'll cover that later.)

This was basic, but it was also enough to create a standard platform that software developers could start to take advantage of. The weird and wonderful hardware of the day was abstracted away behind the BDOS and BIOS, allowing developers to write code and create powerful and useful apps. And write they did - CP/M was the most popular software platform in the world at the time.

ROMs

ROMs could be installed into the system, and they too would need to appear in the same memory map. For example, a floppy disk drive might come with a ROM of drivers, and that ROM would expect to be at a specific address - say F000h. The user could then start the code in the ROM running by selecting F000h on the address switches, toggling EXAMINE and RUN.

Some expansion cards could also combine ROM and RAM, swapping them in and out as required (they couldn't exist in the memory map at the same location at the same time).

(*) The 8080 could in theory address up to 128Kb, using the second 64Kb for stack memory.

Front panel

A distinguishing feature of the Altair and IMSAI computers is their primary user interface - the front panel covered in LEDs and switches.

Although initally overwhelming, there is a logal pattern to the designs which you'll soon come to understand and appreciate. They provide a unique look into the working of the computer in a way that no modern system can offer.

Altair 8800

Figure 2: An Altair 8800

There are four sets of LEDs on the Altair front panel:

LED	Description
Status	Located in the top left, these ten LEDS describe the current state of the processor
WAIT/HLDA	Located just under the Status LEDs, these LEDs let you know if the computer is running (WAIT will be off), and if a HOLD has been acknowledged .
Data	Located in the right right, these eight LEDs can signify the byte stored at a specific address.
Address	Located in the middle of the panel, these sixteen LEDs represent an address in memory, from 0000h to FFFFh

Status LEDs

LED	Description
INTE	An interrupt has been enabled.
PROT	The memory currently referenced by the program counter is read only. Rarely used by an Altair-Duino.

LED	Description
MEMR	The address bus will be used to specify the memory to be read.
INP	The address refers to an input device.
MI	The CPU is processing the first part of an instruction.
OUT	The address refers to an output device.
HLTA	The assembly instruction HALT has been executed, and acknowledged.
STACK	The address bus hols the Stack Pointer's push-down stack address.
WO	The operation being executed is a Write or Output operation.
INT	An interrupt request has been acknowledged.

WAIT/HLDA LEDs

LED	Description
WAIT	The computer is not currently executing code.
HLDA	A HOLD has been acknowledged.

Data LEDs

When the computer is not running, the WAIT light is on, and EXAMINE or EXAMINE NEXT has been toggled, these LEDs represent the data stored at the current address.

Address LEDs

When the computer is not running, the WAIT light is on, and EXAMINE or EXAMINE NEXT, or DEPOSIT or DEPOSIT NEXT, or RESET has been toggled, these LEDs represent the current address.

Altair control switches

Button	Action
STOP/RUN	Start or stop the computer from executing code at the current address.
SINGLE STEP	If the computer is not currently running code, the WAIT light will be on, and SINGLE STEP will execute the next instruction in memory. Some systems have a SLOW option when toggled down, which repeatedly steps through code.
EXAMINE	Set the program counter to the address set by the 16 switches, and display the byte at this address.
EXAMINE NEXT	Increment the program counter by 1, and display the byte at this updated address.
DEPOSIT	Write the byte represented by the right-most switches into the memory at the current address.
DEPOSIT NEXT	Increment the program counter by 1, and write the byte represented by the switches into this updated address.
RESET	Set the program counter to zero. If the computer is still running code, it will continue at address 0.

Button	Action
CLR	Send a clear command to external equipment - unsupported by default on Altair-Duino/IMSAI8080esp systems.
PROTECT/UNPROTECT	Mark a section of memory as read only. Unsupported by default on most Altair-Duino systems.
AUX1	Action depends on specific hardware installed. On Altair-Duino used to select various options.
AUX2	Action depends on specific hardware installed. On Altair-Duino used to select various options.

Altair Address, Data and Sense switches

These switches serve multiple purposes.

Specify an address

If the computer is not currently running code (the WAIT LED is on) then you can use these sixteen switches to select the address of the program counter. When all are down, the address is 0000f. When all are up, it's FFFFh.

Typically you would set the switches to specify an address, and then toggle EXAMINE. The address LEDs will change to reflect the same address, amd the data LEDs will display the byte at that address.

Specify data

With an address set, you might want to write a new value at that location. At this point you can use the eight right-most switches to specify the value, and then toggle DEPOSIT or DEPOSIT NEXT. The position of the switches will define the byte that is now written to memory.

Sense

If a program is running, the eight left-most switches can be used to provide into into the computer. The computer can use IN(0) to read the current state as an 8-bit number.

Examples

This can be rather confusing, so let's look at some very specific examples.

Example 1 - Start at memory address and count upwards

- Toggle STOP in case the computer is doing something.
- Set all the address switches to off i.e. down.
- Toggle RESET. This will reset the program counter to zero. All the address LEDs will be off.
- Toggle EXAMINE. As the switches are all down, you're still using the program counter at zero, so nothing will appear to happen.

Figure 3: Altair status leds

Figure 4: Altair status leds

Figure 5: Altair data leds

Figure 6: Altair address leds

Figure 7: Altair control switches

Figure 8: Altair control switches

- Toggle EXAMINE NEXT. The program counter will be incremented. The A0 LED will turn on. This is memory address 0001f
- Continue to toggle EXAMINE NEXT. The program counter will continue to increment, and the address LED's will count upwards in binary.
- When you get bored (hopefully in less than 65,384 times) toggle RESET and the counter will start again.

Example 2 - Read the contents of memory

- Toggle STOP, and RESET. You're back to program counter zero.
- Toggle EXAMINE. This time look at the data LEDs - the value shown is the byte at memory address 0000f.
- Toggle EXAMINE NEXT. Now the data LEDs show the value at memory address 0001f.
- Continue toggling EXAMINE NEXT for a few times. As the address LEDs show the current address, the data LEDs show the contents of memory.

Example 3 - Writing to memory and confirming the data is correct.

Altair floppy disk operation (MITS Disk Controller)

The Altair can access multiple floppy disks in multiple drives. With a real Altair, you would place the floppy disks into the drives. With the Altair-Duino, the name of the disks, and drives to mount them in, are specified with the switches and the use of the AUX1 and AUX2 toggles. So this is NOT going to work on a real Altair - it's a trick to simulate using real floppy disks. It's not possible to connect a real, physical floppy disk drive to an Altair-Duino. It's for your own good, really.

This section refers for the **MITS Disk Controller**. To use the **Tarbell Disk Contoller** see the section **Altair floppy disk operation (Tarbell Disk Controller)**.

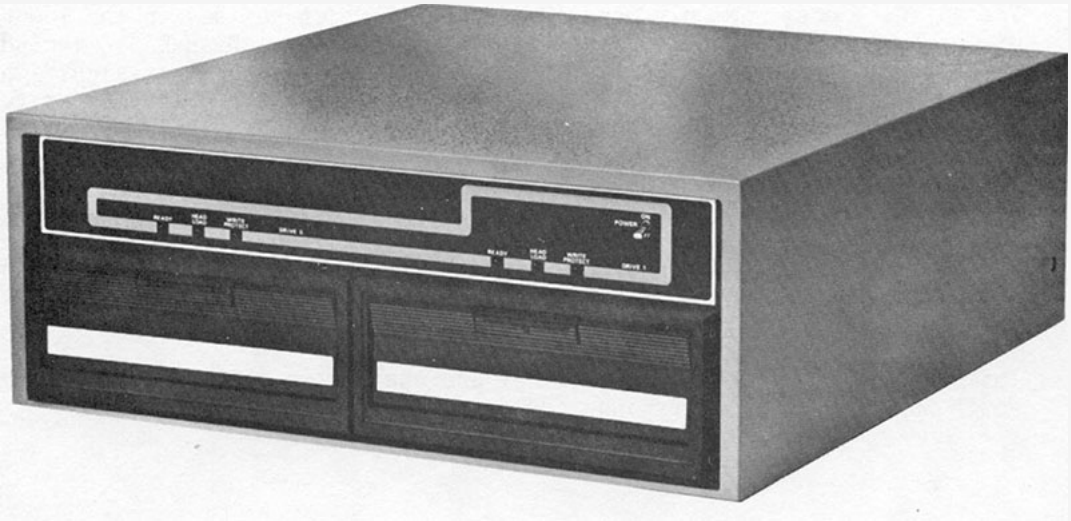

Figure 9: An Altair 8800 floppy disk drive

Display a list of available floppy disks

To see a list of available disk images that can be loaded, enter the following pattern on the address toggle switches (1 means the switch is up):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0

And toggle **AUX2** to **DOWN**.

Note: If you have added your own disks to the local file system (see The Altair Local File System), you will need to manually add the descriptions to the DISKDIR.TXT file. Be sure to terminate each description with a CR and LF character.

Mount a specific floppy disk in the default drive

The disks on the Altair-Duino's file system are named "DISK01.DSK", "DISK02.DSK" and so on. To select a specific disk, set switch **12** to **UP** and then enter the binary representation of the disk name on switches **4** to **0**:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	0	0	0	0	0	0	x	x	x	x	x

For example, to mount disk DISK12.DSK (by default, **Colossal Cave Adventure**), set the following switches:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	0	0	0	0	0	0	1	0	0	1	0

And toggle **AUX2** to **DOWN**.

Note: This will mount the disk, as if you had physically placed a disk in the drive, but it will not run anything. If you need to boot from it, see "Boot from a floppy disk" below.

Selecting different drives

The Altair can support up to 16 floppy drives. So far we've used the default, drive 0, which appears as drive **A:** under CP/M.

To select a different drive, you use switches **11** to **8**:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	x	x	x	x	0	0	0	0	0	0	0	0

And toggle **AUX2** to **DOWN**.

Here are how the first four drives are named under CP/M:

	11	10	9	8
A:	0	0	0	0
B:	0	0	0	1
C:	0	0	1	0
D:	0	0	1	1

Unmounting a floppy disk

To remove a floppy disk, enter the drive number as above, but toggle **AUX2** to **UP**.

Boot from a floppy disk

The disk in drive 0 can be used to boot the Altair, and this is typically used to start CP/M. The code required to load and boot the disk is stored on a Disk Boot ROM. Here is how you activate the Disk Boot ROM:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
?	?	?	?	?	?	?	?	0	0	0	0	1	0	0	0

Then toggle **AUX1** to **DOWN**.

This installs the Disk Boot ROM to address 0xFF00 in the Altair's memory, and starts it running. If a bootable disk is in drive 0, it will boot from it. If you are booting CP/M, you will see the **A:** prompt.

Example

Here's how to boot from DISK13.DSK (by default, CP/M) and mount DISK10.DSK (by default, **The Hitchhikers Guide to the Galaxy**) in drive **B:**.

1. Initialize the Altair

Hold up **Stop** and **Reset**

2. Install and boot CP/M

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	0	0	0	0	0	0	1	0	0	1	1

AUX2 to **DOWN**.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0

AUX1 to **DOWN**.

CP/M will boot.

3. Mount Hitchhikers Guide

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	0

Toggle **AUX2** to **DOWN**.

If you now enter **DIR B:** you'll see the Hitchhikers disk ready to play.

Altair hard disk operation (MITS 88-HDSK Controller)

The Altair-Duino system can access four hard disks, specified with the switches and the use of the AUX1 and AUX2 toggles. In other words, this is not going to work on a real Altair - it's a trick to simulate using hard disks on the Altair-Duino. There is no way to connect genuine MITS hard drives to the Altair-Duino (just in case you happened to find some).

Note: This section requires that a SD card be present on your Altair-Duino.

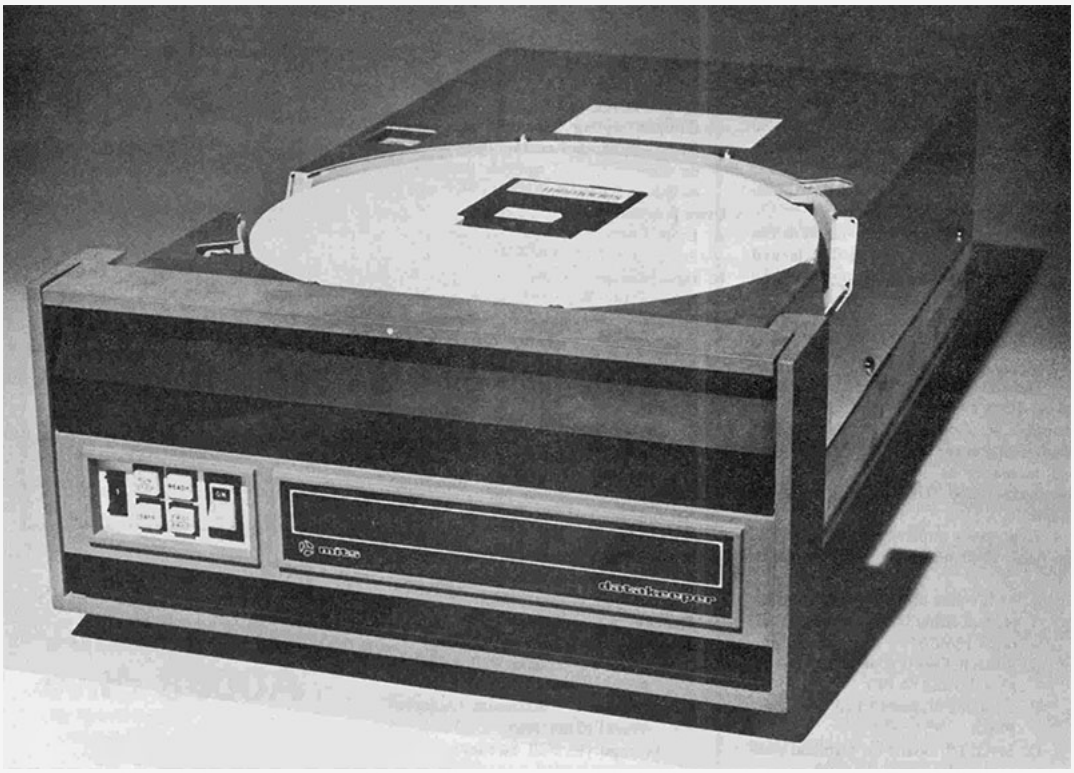

Figure 10: An Altair hard drive system from MITS

Display a list of available hard disk images

To see a list of available disk images that can be loaded, enter the following pattern on the address toggle switches (1 means the switch is up):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0

And toggle **AUX2** to **DOWN**.

Note: If you have added your own disks to the local file system (see **The Altair Local File**

System), you will need to manually add the descriptions to the HDSKDIR.TXT file. Be sure to terminate each description with a CR and LF character.

Mount a specific hard disk in the default drive

The hard disks on the Altair-Duino's file system are named "HDSK01.DSK", "HDSK02.DSK" and so on. To select a specific disk, set switches **13** and **12** to **UP** and then enter the binary representation of the disk name on switches **7** to **0**:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	0	0	0	x	x	x	x	x	x	x	x

For example, to mount disk HDSK03.DSK (by default, **Mike Douglas' CP/M**), set the following switches:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	1

Note: This will mount the disk, but it will not run anything. If you need to boot it, see "Boot from a hard disk" below.

Selecting different drives and platters

Note: the exact number of hard disk drives available depends on the build of the Altair-Duino your computer has been programmed to support. By default only 1 drive is available, but you can re-build the Altair-Duino image if you need more. See "Building your Altair-Duino image".

The Altair can support up to 4 hard drives connected at once. So far we've used the default, unit 1, which appears as drive **A:** under CP/M.

To select a different drive unit, you use switches **11** and **10**:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	x	x	0	0	0	0	0	0	0	0	0	0

If your system supports 1 drive, as is the default, then you will see an error if you try to mount any drive other than Unit 1.

Hard drives also have **platters** which refer to the physically disks of recording material inside the drive mechanism. It's possible to select the platter number too, using switches **9** to **8**:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	0	0	x	x	0	0	0	0	0	0	0	0

In practice this doesn't do much and you should keep switches **9** and **8** set to off.

Boot from a hard disk

The disk in drive 0 can be used to boot the Altair, which is typically used to start CP/M. The code required to load and boot the disk is stored on a Disk Boot ROM. Here is how you activate the Disk Boot ROM:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
?	?	?	?	?	?	?	?	0	0	0	0	1	1	1	0

Then toggle **AUX1** to **DOWN**.

This installs the Disk Boot ROM to address 0xFF00 in the Altair's memory, and starts it running. If a bootable disk is in drive 0, platter 0, it will boot from it. If you are booting CP/M, you will see the **A:** prompt.

Example

Here's how to boot from HDSK03.DSK (by default, CP/M) and then mount floppy DISK10.DSK (by default, **The Hitchhikers Guide to the Galaxy**) in drive **B:**.

1. Initialize the Altair

Hold up **Stop** and **Reset**

2. Install and boot CP/M

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	1

AUX2 to **DOWN**.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	0	0	0	0	0	0	0	1	1	1	0

AUX1 to **DOWN**.

CP/M will boot.

3. Mount Hitchhikers Guide

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	0	1	0	0	0	0	1	0	0	0	0

Toggle **AUX2** to **DOWN**.

If you now enter **DIR B:** you'll see the Hitchhikers disk ready to play.

Note: Mounting hard drives and floppy drives side-by-side has some quirks. I would have expected that setting switch **8** and not **9** would have mounted the Hitchhikers disk at drive B:, but it didn't. Also the above pattern will mount it at drive C: at the same time.