Altair 8800 Simulator

https://www.hackster.io/david-hansel/arduino-altair-8800-simulator-3594a6

Table of Contents

Acknowledgements ..... - 3 -
Highlights ..... - 3 -
Front Panel Elements ..... - 4 -
Auxiliary Switch Functions ..... - 5 -
Serial ports ..... - 8 -
Interacting with software via a terminal ..... - 10 -
Printer Emulation ..... - 11 -
Disk Drive Support ..... - 12 -
Hard Disk Support ..... - 13 -
Cromemco Dazzler Support ..... - 14 -
Configuration Menu ..... - 15 -
File System Manager ..... - 20 -
Debugging Capabilities ..... - 20 -
Example program: Kill-the-bit game ..... - 21 -
Loading 4k BASIC the old-school way ..... - 22 -
Altair Time Sharing BASIC ..... - 23 -
Music System ..... - 25 -
MITS Programming System II ..... - 27 -
8080 Instruction Set ..... - 31 -
Hardware setup for Arduino Mega 2560 ..... - 34 -
Pin Mapping for Arduino Mega 2560 ..... - 35 -
Hardware setup for Arduino Due ..... - 37 -
Pin Mapping for Arduino Due ..... - 38 -
Wiring an SD card to the Arduino Due ..... - 40 -

Acknowledgements

I must give many credits to Mike Douglas of altairclone.com who has spent countless hours collecting information about the Altair and made it all available on his web site for the community to use. Additionally, most of the software included in this simulator was collected, put into working condition and in many cases amended with additional functionality by Mike. Without his work, this project would have been significantly more complicated to put together, way less fun and would probably not happened at all.

Another big thanks to Martin Eberhard who has written many tools for the Altair community, for example the combo disk boot loader and hard disk loader which are used by the simulator. His ADEXER tool was invaluable for debugging the hard disk support. Thanks also for helping me sort through various issues with the hard disk emulation.

Highlights

- Runs at about the same speed as the original
- Emulated memory size (RAM) is 64KB
- ALTAIR extended BASIC ROM (16k) can be mapped to addresses 0xC000-0xFFFF
- Emulates one ALTAIR 88-SIO, 88-ACR and one or two 88-2SIO board(s). Each device can be mapped to the Arduino's serial port(s). Data sent to each device can be captured and replayed.
- The ACR device also supports using the CSAVE/CLOAD commands in extended BASIC. When invoking CSAVE, the program will automatically be saved to a file specified by the file name in the CSAVE command. For CLOAD, all programs saved with CSAVE will automatically be played back until the program specified by the file name is found (or not).
- By default, both the 88-SIO and 88-2SIO boards are mapped to the Arduino's main serial interface which is configured to 115200 baud 8n1. That interface be accessed via pins 0/1 or the Arduino's USB cable (on the Due make sure to connect the USB cable to the programming USB port, not the native port).
- Processes input/output on serial devices to deal with 7/8 bit output, upper-case input and backspace.
- Emulates 4 (can be configured to up to 16) Altair 88-DCDD disk drives
- Emulates an 88-HDSK hard disks drive with one unit (can be configured up to 4) and 4 platters.
- Emulates the 88-RTC-VI board which makes it possible to run ALTAIR Time Sharing BASIC.
- Many settings can be modified via a configuration editor (invoke by holding STOP and raising AUX1)
- Above specs apply when running on an Arduino Due. The simulator can also work when running on an Arduino Mega 2560. In that case, it runs at about 25% original speed and has 6k emulated RAM. Disk drive emulation is not supported on the Mega.

Front Panel Elements

Lights:

A15-A0	Shows the current address bus state (i.e. PC during normal operation)
D7-D0	Shows the current data bus state

Switches:

SW15-SW0	Address/Data entry switches
RUN	Execute instructions starting at current PC location
STOP	Stop program execution
SINGLE STEP	Execute single instruction at current PC location and increment PC
EXAMINE	Set PC to address from SW15-0 and show content of that address on D7-0
EXAMINE NEXT	Increment PC and show content of that address on D7-0
DEPOSIT	Store value of D7-0 at current PC address
DEPOSIT NEXT	Increase PC by one and store value of D7-0 at that address
CLR	Stop serial capture/replay activated by AUX2 (see below) Holding CLR down during power-up will initialize memory and CPU registers to 0
RESET	Reset processor (set PC to 0)
PROTECT	Mark current memory page as write-protected (cannot be written to)
UNPROTECT	Remove write-protect status of current page
AUX1/AUX2	See detailed information below

Auxiliary Switch Functions

AUX1 down:

Function depends on current setting of SW7-0:

0…0000: Print this list to serial interface

0…0001: Calculator (David Hansel, 2015)

- Flipping SW15 (on->off or off->on) executes operation selected by SW14-13 between operand currently showing on A15-8 lights and operand selected by SW12-8 switches. Result is shown on A15-8 lights.
- Operations (SW14-13): 00=Add, 01=Subtract, 10=Multiply, 11=Divide

0…0010: Kill-the-bit (Dean McDaniel, 1975)

- Kill the rotating bit. If you miss the lit bit, another bit turns on leaving two bits to destroy.
- Quickly toggle the switch, don't leave the switch in the up position.
- Before starting, make sure all the switches are in the down position.

0…0011: Pong game using front panel (Mike Douglas of altairclone.com)

- Left player quickly toggles SW15 to hit the "ball."
- Right player toggles SW8. Score is kept in memory locations 0x80 and 0x81 (left and right).
- Score is missed balls, so the lower number wins.

0…0100: Pong game using serial terminal (David Hansel, 2015)

- Needs terminal connected to serial interface. Terminal must understand escape sequences for cursor movement (“ESC-[“)
- Left player uses A/Z keys (paddle up/down), right player uses K/M keys (paddle up/down)

0…0101: 4k Basic (Bill Gates, Paul Allen, Monte Davidoff 1975)

- SW11 down: I/O to SIO device, SW11 up: I/O to 2SIO device (either setting will work in default simulator configuration)
- Answering N to questions about SIN/RND/SQR slightly increases available BASIC memory

0…0110: MITS 16k ROM Basic

- On Arduinoe Due, maps ROM extended BASIC to addresses 0xC000-0xFFFF until next RESET. (On Arduino Mega, ROM extended BASIC is always mapped to that range).
- Sets PC to 0xC000 and starts execution (i.e. starts BASIC)
- Before activating, set SW15-12 switches to
  0000: I/O goes to 2SIO device
  0010: I/O goes to SIO device
  (wither setting will work in default simulator configuration)
- When asked MEMORY SIZE, just press enter.
- When asked LINEPRINTER, enter capital O and press enter
- Available memory for BASIC programs is 48101 bytes (~3000 bytes on Arduino Mega)
- CSAVE and CLOAD commands can be used to save/load programs to/from internal storage

0…0111: MITS Programming System II (Due only)

- SW11 down: I/O to SIO device, SW11 up: I/O to 2SIO device (either setting will work in default simulator configuration)
- Editor, Assembler and Debugger
- See intro at the end of this document, google for full documentation

0…1000: Combo Disk Boot Loader ROM V2.05 (Martin Eberhard, Mike Douglas, 2016) (Due only)

- See Disk Drive Support section below

0…1001: ALTAIR Turnkey Monitor (Due only)

- Uses 2SIO device for I/O

0…1010: Music “Daisy, Daisy…” (Steve Dompier, 1975) (Due only)

- In configuration menu, set throttle delay to 5 before running (automatic throttle introduces noise due to the throttle-adjustment code that runs 40 times per second)
- An AM radio held on top of Simulator, close to D0-D7 lights will pick up the song
- See: http://www.digibarn.com/collections/weirdstuff/altair-sheetmusic/

0…1011: 8080 CPU Diagnostic (Microcosm Associates, 1980)

- Basic test, takes about a second to finish
- Outputs to serial interface (“CPU IS OPERATIONAL”)

0…1100: 8080 CPU Exerciser (Frank D. Cringle 1994, Ian Bartholomew 2009)

- Very thorough test that generates a CRC code for each group of tests.
- Outputs to serial interface.
- Full test takes about 4 hours. The "aluop <b,c,d,e,h,l,m,a>" section takes especially long

0…1101: Music System (Processor Technology, 1977) (Due only)

- Processor Technology’s Music System for the Altair
- Loads the music system (at 0x0) and ACUTER monitor (at 0xF000) into Altair RAM and starts ACUTER monitor.
- See “Music System” section below for more info

0…1110: Hard-Disk boot ROM V2.0 (Martin Eberhard, 2014) (Due only)

- See Hard Disk Support section below

0…1111: Enhanced Multi boot loader V3.0 (Martin Eberhard, Mike Douglas, 2016) (Due only)

- Set SW10-8 to select boot device:
  000=2SIO port 1, 010=SIO, 011=ACR, 110=2SIO port 2

0100000: Read Intel HEX data from serial input (primary host interface)
11xxxxxx: Save the 256-byte memory page currently selected on the SW15-8 switches to file #xxxxxx
10xxxxxx: Load the 256-byte memory page currently selected on the SW15-8 switches from file #xxxxxx

AUX1 up:
If STOP is held up while AUX1 is raised, then invoke the configuration editor (see Configuration menu below). Otherwise, run the program configured via the corresponding setting in the configuration menu.

AUX2 down:
If SW12 is up and SW13 is down, then mount disk in emulated disk drive (see *Disk Drive Support* section below).
If SW12 is up and SW13 is up, then mount image in emulated hard disk (see *Hard Disk Support* section below).

Otherwise, play back captured data or example programs. SW15-13 select the device to which data is played back and SW8-0 select what data is being played back:

- SW15 down: Use the serial device which is mapped to the host serial output (primary) and has last seen input/output activity. In most cases this will automatically select the intended device.
- SW15 up: Use serial device selected on SW14-13:
  00: 88-SIO (port 0x00/0x01)
  01: 88-ACR (port 0x06/0x07) (audio cassette interface)
  10: 88-2SIO, serial 1 (port 0x10/0x11)
  11: 88-2SIO, serial 2 (port 0x12/0x13)
- SW8 down: Play back basic/assembly example # selected on switches SW7-0 (see below)
- SW8 up: Play back captured data in file # selected on switches SW7-0 (see below)

Loading BASIC/assemly examples:

- SW7 up: BASIC example, SW7 down assembly example
- If SW6-0 are all 0 then a list of available examples will be transmitted
- Set SW6-0 to the example number in the list and push AUX2 DOWN to transmit example

- Playback can be stopped by pushing CLR or by pushing AUX2 DOWN again

Playing back captured data:

- Play back data previously captured via AUX2 up
- The file number to be played back must be selected via SW7-0
- Playback can be stopped by pushing CLR or by pushing AUX2 DOWN again

AUX2 up:

If SW12 is up and SW13 is down, then unmount disk from disk drive (see *Disk Drive Support* section below).
If SW12 is up and SW13 is up, then unmount image from hard disk (see *Hard Disk Support* section below).

Otherwise, capture data. SW15-13 select the device from which data is captured (same as SW15-13 settings in *AUX2 down* section above). SW7-0 specify the file number under which the captured data will be saved. Capturing continues until AUX2 is again pressed UP.

Capturing serial data can be used to save a BASIC program in BASIC (note that in extended BASIC the CSAVE/CLOAD commands provide an easier way of doing this):

1. Set SW7-0 to the desired storage file number
2. Set SW15-13 to 000 (to automatically select capture device)
3. type "list" (but **not** ENTER)
4. activate AUX2 up to start capture
5. press ENTER
6. wait until listing is finished
7. activate AUX2 up again to finish capture

To later restore a program:

1. Set SW7-0 to a storage file number under which serial data has been captured before
2. Set SW15-13 to 000 (to automatically select replay device)
3. type "new" to clear the current program
4. activate "AUX2 down" to start replaying the captured data
5. Ignore the "SYNTAX ERROR" at the end
   (reported because the final "ok" of the "list" function was also captured)

Capturing cassette data can be used to save BASIC variable contents in Extended BASIC or saving programs from MITS Programming System II

1. Set SW7-0 to the desired storage file number
2. Set SW15-13 to 101 (to capture from ACR device on I/O address 6/7)
3. activate AUX2 up to start capture
4. enter CSAVE*v [where v is the variable name that is supposed to be saved]
5. Repeat the previous step if more variables need to be saved
6. wait until listing is finished
7. activate AUX2 up again to finish capture

To later load the data:

1. Set SW7-0 to a storage file number under which cassette data has been captured before
2. Set SW15-13 to 101 (to replay to ACR device on I/O address 6/7)
3. Set SW8 to 1 (to select file replay, not BASIC example)
4. activate AUX2 down to start replaying the captured cassette data
5. enter CLOAD*v [where v is the variable name that is to be loaded]
6. Repeat the previous step if more variables need to be loaded
7. Activate AUX2 down to stop replay

Serial ports

The Emulator simulates an Altair 8800 with one 88-SIO, one 88-ACR and one or two 88-2SIO card(s) installed. These cards are visible to the software running within the emulator (for example ALTAIR BASIC).
All together there are up to seven emulated serial ports available to the software.

The Arduino, which runs the emulator, has its own serial ports that actual terminals can be connected to. The configuration editor allows the user to map emulated serial ports to real serial ports.

While the Arduino Mega has 4 serial ports, only one of them is available to use with the emulator because the pins that the others would use are connected to front panel elements such as LEDs or switches. The Mega's single serial port can be accessed either by plugging in the USB cable or by connecting a serial device to pins 0 (RX) and 1 (TX). Note that these pins use 5V levels, not the +12V/-12V used by real serial hardware. However, the levels can be boosted to proper serial levels using an adapter such as a MAX232.

When using the Arduino Due, there are up to five serial ports available for the emulator:

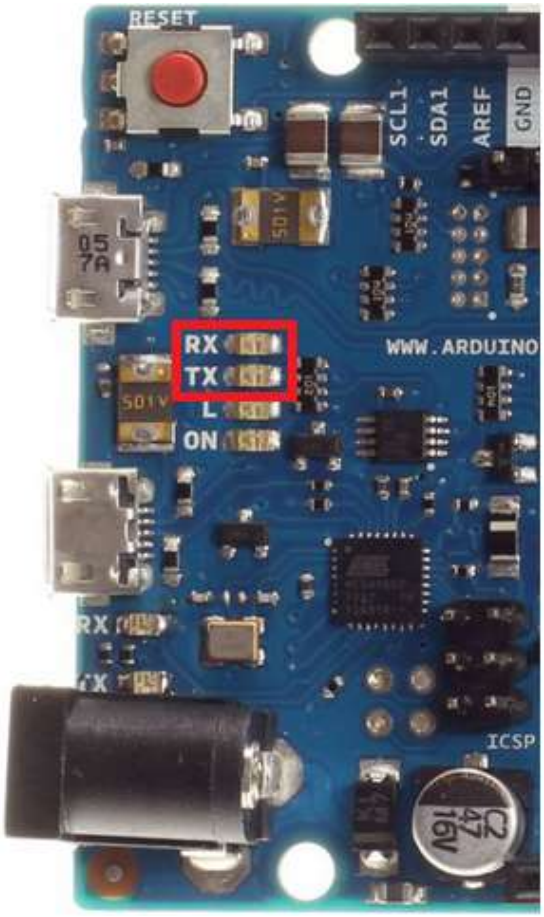
1. *Programming port USB connection.* After connecting a USB cable to the Programming Port of the Arduino it will show up on a PC as a serial interface. This is the default interface that the emulator uses (at 115200 baud, 8N1) when first activated or reset to its defaults.
   This connection can also be accessed on pins 0 (RX) and 1 (TX) of the Arduino Due **BUT** there appears to exist a problem with (some versions of) the Arduino Due where pin 0 (RX) does not work. For more information about that issue see here: https://forum.arduino.cc/index.php?topic=474483.0
   The pins are 3.3V and are **not** 5V tolerant. This port can be configured from 600 to 1050000 baud, except 1200 baud since opening a USB serial connection at 1200 baud causes the Due to erase its flash memory and go into programming mode. Baud rates of 300 and below are not available because the USB-to-serial chip on the Due does not support them: https://github.com/arduino/Arduino/issues/4714
2. *Native port USB connection*. The second USB port on the Arduino Due can also be used as a serial connection by plugging in a USB cable. This port does **not** have corresponding I/O pins and can only be accessed via USB. Since it is a pure USB connection the baud rate is irrelevant and is fixed at 115200.
3. *Serial port on pins 18 (TX) and 19 (RX)*. The pins are 3.3V and are **not** 5V tolerant. This port can be configured from 110 to 1050000 baud.
4. *Serial port on pins A6 (RX) and A7 (TX)*. This port is disabled in the default configuration since the same pins (A6/A7) are used for the PROTECT/UNPROTECT switch on the front panel. However, the Altair's PROTECT functionality is rarely (if ever) used and therefore the pins can be re-purposed to act as a serial port by changing two `#define` statements in the code (see below).

   > **IMPORTANT**: If pins A6/A7 of your Arduino are already wired to the PROTECT/UNPROTECT switch I highly recommend to physically disable the switch (for example by removing the connection of the middle pin ground) before enabling this port in the software. If the serial port is enabled in the software and the switch is connected, pressing either PROTECT or UNPROTECT can destroy pin A7 on the Arduino or possibly a connected serial device. This is because serial lines idle high (3.3V) and the switch, when pressed, connects the pin directly to ground. The best solution is to connect the middle pin of the PROTECT/UNPROTECT switch to ground via a 1k resistor (instead of directly). That way the switch can be used if enabled in software but does not cause damage when pressed while the serial port is enabled.

   To enable this port in the emulator, do the following before uploading the sketch to the Arduino:
   - In config.h, change
     "`#define USE_PROTECT 1`" to "`#define USE_PROTECT 0`"
   - In host_due.h, change
     "`#define USE_SERIAL_ON_A6A7 0`" to "`#define USE_SERIAL_ON_A6A7 1`"

   The pins are 3.3V and are **not** 5V tolerant. This port uses a software UART implementation which can be configured from 110 to 38400 baud.

5. *Serial port on pins RXL / TXL*. This port is disabled by default since it uses two I/O pins that are not connected to headers on the Arduino Due: the pins controlling the RX and TX LEDs next to the Native USB port (framed in red in the image on the right). These pins are accessible as digital pins 72 (RX) and 73 (TX) on the Arduino Due. Despite their location next to the Native USB port they have no connection to the port and can be freely controlled by software. To add a serial interface using these pins do the following:
    - (Optional but recommended): remove the RX/TX LEDs. I did try leaving the LEDs in place and just soldered onto the LEDs themselves and serial communication did work. However, the LEDs go to +3.3V through a 1k resistor which could possibly interfere with the serial signals. Your mileage may vary.
    - Solder wires to the pads on the left side of the LEDs (the side closer to the "RX" and "TX" labels). These will be the RX and TX wires for the serial connection. Take the required GND wire for the connection from any GND connection on the Arduino.
    - In file host_due.h, change
      `#define USE_SERIAL_ON_RXLTXL 0`
      to
      `#define USE_SERIAL_ON_RXLTXL 1`
      and upload the sketch to the Arduino.



The pins are 3.3V and are **not** 5V tolerant. This port uses a software UART implementation which can be configured from 110 to 38400 baud.

Interacting with software via a terminal

The main way to interact with programs on the Altair is via a terminal that is connected to a serial interface. The two most common serial interface cards used with the Altair were the 88-SIO and 88-2SIO. The 88-SIO offered one serial port which (although configurable via jumpers) was most often set to use I/O addresses 0 and 1. The 88-2SIO offered two serial ports (again configurable via jumper) that usually used addresses 16/17 (first port) and 18/19 (second port).

Most systems had at least one of these installed and most programs would expect to interact via either an 88-SIO at addresses 0/1 or the first port of an 88-2SIO at addresses 16/17. Some programs used the SW15-8 sense switches at startup to determine which one to use. For example:

4k BASIC and Programming System II: If SW11 is up then use an 88-2SIO at 16/17 otherwise use 88-SIO at 0/1.
16k ROM BASIC: If SW13 is up then use 88-SIO at 0/1 otherwise use 88-2SIO at 16/17.

The simulator simulates a system with both an 88-SIO and 88-2SIO installed at their default addresses. In the default configuration, **both** the 88-SIO and the first port of the 88-2SIO are mapped to the Arduino's main serial port, i.e. any output that goes to either one will show up on a terminal connected to the Arduino and inputs coming from the terminal will be sent to both cards. This was not possible in reality (it would mean to hook up one terminal to two serial ports) but in the simulated environment it works just fine. It allows users to not have to worry about properly setting sense switches before starting BASIC or other programs.

The one drawback is that when changing device settings in the configuration menu, you must first know which device is being used. The easiest way to determine that is to just un-map one of them in the configuration and see if the serial I/O still works.

7-bit vs 8-bit characters. Early terminals used only 7-bit characters and a number of Altair programs (such as 4k BASIC) use the 8th bit of a character to define end-of-string which would work fine since the terminal would ignore the 8th bit anyways. Modern terminals use the 8th bit and so display some strange graphics character for characters where the 8th bit is set. The simulator (in the serial device settings) offers a way to filter out the 8th bit by enabling the "Use 7 bits" setting.

Uppercase input. Early terminals only had upper case characters and so early Altair programs (4k BASIC again) can not handle lower-case characters. The simulator offers a serial device setting ("Serial input uppercase") that will automatically covert incoming lower-case characters to upper case.

Backspace handling. Some early Altair software does not use the backspace (0x08) character and instead has its own way of undoing inputs, e.g. in 4k BASIC receiving an underscore ("_") will print the underscore but internally delete the last character from the input buffer, i.e. have the functionality of a backspace. Other programs expect a "delete" (0x7f) character instead of "backspace" (0x08). The simulator offers some help by offering an option to translate between backspace and underscore or backspace and delete.

When converting back from underscore to backspace, the simulator actually sends a backspace-space-backspace sequence to delete the character left of the cursor even if the connected terminal does not do a destructive backspace.

Pasting text into the terminal. When pasting large amounts of text into the terminal, the terminal program usually sends that text at the given baud rate. So if the connection is using 9600 baud then a new character will arrive at the simulator about every millisecond. With many simulated programs (and especially when running the simulator on the Arduino Mega) this can cause characters to get lost because the simulated program can not keep up processing the characters at the rate that they arrive.

There are two ways to deal with this:

1. Lower the baud rate of the host serial interface and your terminal program, which forces the terminal to send characters at a lower speed. This works to some degree but (a) you may not be able to reduce the baud rate enough to guarantee that no characters are lost (especially when using the Mega) and (b) reducing the baud rate will also lower the transmission speed from the simulator to your terminal, which can become annoying when outputting large amounts of text.

2. The better option is to tell your terminal to add a delay between sending characters. TeraTerm allows this by changing the "Transmit delay" settings (In the Setup->Serial port menu). The terminal will still send at the given (fast) baud rate (and more importantly *receive* at the fast baud rate) but will wait for a specified amount of time before sending the next character. Moreover, TeraTerm allows to specify a delay after a carriage return/line feed. This is important because some programs (such as BASIC) need additional time after seeing a carriage return to process the input.
   Some testing has shown that for the Arduino Mega, setting the Transmit delay to 10msec/char and 200msec/line works well even with the host serial rate set at 115200 baud. On the Due, 3msec/char and 25msec/line should be sufficient. These are just some basic benchmarks. The optimum setting (not too much delay but also not too little) will also depend on the baud rate.
   Unfortunately it appears that Putty does not support a transmit delay.

Printer Emulation

One printer can be emulated by enabling the corresponding options in the configuration menu (see Configuratoin Menu section below). The printer emulation can be configured for an "Okidata" printer or "Centronics C700". Most of the included software (e.g. BASIC) support both (enter "O" or "C" at the LINEPRINTER prompt). Support for the Centronics printer is more complete since documentation for that printer exists on the Internet. The Okidata printer emulation was mostly put together by looking at other Altair system emulators.

Printer emulation does support interrupts. If interrupts are enabled by the software then the printer emulation automatically switches to real-time mode, i.e. it roughly simulates the time it would take the printer to print. This is necessary because otherwise (in an interrupt driven software system such as Time-Sharing BASIC) the printer would rapidly produce interrupts, disrupting access to other devices. Real-time operation can be forced even if interrupts are not enabled by enabling the "Force real-time mode" option in the configuration menu.

The printer output can be directed to either the host's primary or secondary serial port.

Disk Drive Support

Disk drive support is optional and requires a SD card attached to the Arduino Due's SPI header (the 2-row, 6-pin header marked "SPI"). See the *Wiring SD card to Arduino DUE* section at the end of this document about how to physically hook up the card.

Disk drive support is **not** available in the Arduino Mega build, mainly for two reasons: 1. The SPI pins on the Mega are directly connected to some general I/O pins which are already used for the front panel and 2. The Mega only provides 6k of emulated RAM. Most disk based programs require more than that.

The following files are expected to be found in the root directory of the (FAT format) card (the `disks` subdirectory in the source archive contains several disks including CP/M and Altair DOS):

- **DISKxx.DSK** (where xx is a 2-digit hexadecimal number): Disk images that the simulator can mount.
- **DISKDIR.TXT**: A text file whose contents will be sent to the serial connection (i.e. shown to the user) if front panel switches are set to 0001xxxx00000000 and the AUX2 switch is pressed down. This should contain information about each of the DISKxx.DSK files

To mount disks in the drive, use the AUX2 down switch:

- Set SW15-0 to: 0001nnnnDDDDDDDD where nnnn is a 4-bit number selecting the drive (i.e. drive 0-15) and DDDDDDDD is an 8-bit number selecting the disk number and press AUX2 down.
- The 8-bit disk number corresponds to the xx in the DISKxx.DSK files on the SD card.
- For example, setting SW15-0 to 0001 0010 0000101 and pressing AUX2 down will mount disk number 5 in drive 2.
- Selecting disk number 0 is a special case. If disk 0 is selected for mounting, it will not be mounted but instead the contents of the DISKDIR.TXT file will be sent to the serial output (i.e. shown to the user). Note that that means a file named DISK00.DSK can not be mounted.
- If a disk is already mounted in the drive the mounted disk will be unmounted before mounting the new disk
- If the disk file does not exist, it is like inserting an empty disk in the drive. If the operating system writes to the disk, the selected disk file will be created. So inserting a non-existent disk and then formatting that disk via the operating system (e.g. CP/M) will create a new empty disk.
- It is possible to mount the same disk in multiple drives. The simulator has no problem with that but the running operating system may get confused.

To unmount a disk from a drive, use the AUX2 up switch:

- Set SW15-0 to: 0001nnnnxxxxxxx where nnnn is a 4-bit number selecting the drive (i.e. drive 0-15) and press AUX2 up.
- It is not necessary to unmount disks before turning off the computer. Each write operation to a disk gets flushed to the SD card immediately so turning the computer off with disks mounted will not lose data.

To run a bootable disk image, first mount the disk and then start the Disk Boot ROM:

- Set SW0-7 to 00001000 (to select Disk Boot ROM)
- Press AUX1 down
- This will install the Disk Boot ROM at 0xFF00 and immediately start it. If a bootable disk has been mounted it should automatically start now

Hard Disk Support

Just like disk drive support, hard disk (88-HDSK) support requires a SD card and is not available on the Arduino Mega build. See the first two paragraphs of the "Disk Drive Support" section above for more details.

The 88-HDSK controller could handle up to 4 units, each holding up to 4 platters. In its default configuration, only one unit is enabled, which should be sufficient for most cases. Change the NUM_HDSK_UNITS setting in the config.h file to enable more (at most 4) units.

The following files are expected to be found in the root directory of the (FAT format) card (the `disks` subdirectory in the source archive contains some hard disk images including the Altair Accounting System):

- **HDSKxx.DSK** (where xx is a 2-digit hexadecimal number): Disk images that the simulator can mount.
- **HDSKDIR.TXT**: A text file whose contents will be sent to the serial connection (i.e. shown to the user) if front panel switches are set to 0011xxxx00000000 and the AUX2 switch is pressed down. This should contain information about each of the HDSKxx.DSK files

To mount hard disk images on a unit/platter, use the AUX2 down switch:

- Set SW15-0 to: 0011uuppDDDDDDDD where *uu* is the hard disk unit (2-bit, 0-3) and *pp* is the platter number within that unit (2-bit, 0-3). DDDDDDDD is an 8-bit number selecting the hard disk image number. Then press AUX2 down.
- The 8-bit disk image number corresponds to the xx in the HDSKxx.DSK files on the SD card.
- For example, setting SW15-0 to 0011 0001 0001100 and pressing AUX2 down will mount HDSK0C.DSK on platter 1 of unit 0.
- Selecting hard disk image number 0 is a special case. If disk image 0 is selected for mounting, it will not be mounted but instead the contents of the HDSKDIR.TXT file will be sent to the serial output (i.e. shown to the user).
  Note that that means a file named HDSK00.DSK can not be mounted.
- If a hard disk image is already mounted in the drive the mounted disk will be unmounted before mounting the new disk
- If the hard disk image file does not exist, it is like inserting a new platter in the hard disk. If the operating system writes to the hard disk, the selected image file will be created. Inserting a non-existent image and then formatting via the operating system (e.g. using ADEXER in CP/M) will create a new empty image.
- It is possible to mount the same image in multiple drives. The simulator has no problem with that but the running operating system may get confused.

To unmount an image from a unit/platter, use the AUX2 up switch:

- Set SW15-0 to: 0011uuppxxxxxxxx where *uu* is the hard disk unit (2-bit, 0-3) and *pp* is the platter number within that unit (2-bit, 0-3). Then press AUX2 up.
- It is not necessary to unmount images before turning off the computer. Each write operation to an image gets flushed to the SD card immediately so turning the computer off with disks mounted will not lose data.

To run a bootable hard disk image, first mount the image and then start the Disk Boot ROM:

- Set SW0-7 to 00001110 (to select Hard Disk Boot ROM)
- Press AUX1 down
- This will install the Hard Disk Boot ROM at 0xFC00 and immediately start it. If a bootable image has been mounted it should automatically start now. The Hard Disk Boot ROM always boots from unit 0, platter 0.

Cromemco Dazzler Support

In 1976 the Cromemco Dazzler graphics card for the Altair was released. The card accesses the Altair's memory via DMA to read pixel data and produces a picture of 32x32, 64x64 or 128x128 pixels, depending on memory configuration and color mode. While these resolutions seem almost laughable today, back in 1976 this was a major accomplishment.

The Simulator (when running on the Arduino Due) can emulate the Dazzler when paired with a client to display the picture. For more information on the Dazzler client, see my project at:
https://www.hackster.io/david-hansel/dazzler-display-for-altair-simulator-3febc6

To enable support for the Dazzler in the Simulator, do the following:

- Download the latest release of the Simulator firmware
- Change the USE_DAZZLER setting in config.h from 0 to 1
- Re-upload the modified firmware to the Arduino
- In the configuration menu, there should now be a "Configure Dazzler" setting.
  Change that setting to "USB Native Port".

After enabling support for the Dazzler, a number of new programs will be available for the "AUX1 down" function. All of these are (as far as I know) copyright Cromemco 1976:

- Kaleidoscope
- DazzleDoodle (*very* basic drawing program operated by joystick)
- DazzleMation (basic animation program, needs Cromemco serial card, see below)
- DazzleWriter (allows to type text, needs Cromemco serial card, see below)
- Life (Conway's game of life, expects input on first port of 2-SIO)
- Labyrinth (needs joystick)
- Chase (needs two joysticks)
- Space War! (needs two joysticks)
- Gotcha! (needs two joysticks)

For programs requiring a joystick, see the joystick instructions in the Dazzler project. Note that the Windows client recognizes joysticks plugged into the computer running the client.

For programs that need a Cromemco serial card, make sure the "SIO revision" setting in the SIO card configuration is set to "Cromemco".

Configuration Menu

The simulator configuration menu can be entered by holding STOP up and raising AUX1.
The following settings/actions are available:

- *Enable profiling*
  If enabled and the simulator is running (i.e. not in STOP mode) a message showing the current performance of the simulator is written to the (primary) serial output.
- *Set throttle delay* (Arduino Due only)
  Allows to throttle simulator performance. Throttling is done by busy-waiting, i.e. an empty loop counting down from the value selected here to zero. If set to "auto adjust", the simulator evaluates performance about 40x a second and automatically adjusts the throttle delay to get as close to 100% of the original (i.e. 2MHz clock frequency) as possible. If profiling is enabled too then the effect of changes in the throttle delay can easily be observed.
- *Enable serial panel*
  Shows a simple representation of the front panel lights and switches on the (primary) serial output. Mostly useful when STANDALONE mode is enabled.
- *Enable serial input*
  When stopped, the simulator processes the inputs described in the "Debugging capabilities" section below.
- *Enable serial debug*
  Print processor status and disassembled opcode during single stepping
- *Clear memory on powerup*
  If enabled, the simulated memory will be cleared (set to 0) when the simulator starts up.
  Otherwise, memory content is random (as with the original).
- *RAM size*
  Configures the size of the emulated RAM in steps of 1k. The emulated RAM starts at address 0 and is contiguous from there.
- *Aux1 shortcut program*
  Sets the program to be run when the AUX1 switch is raised. This can be any if the built-in programs available via AUX1 down or a disk. If a disk is selected, pushing up AUX1 will mount the disk and then install and run the disk boot loader.
- *Configure host serial*
  See "Host serial interface configuration" below.
- *Configure SIO/ACR/2SIO port 1/2SIO port 2*
  See "Serial device configuration" section below.
- *Configure Disk Drives* (only if disk drive emulation is enabled)
  See "Disk Drive configuration" section below.
- *Configure Hard Disks* (only if hard disk emulation is enabled)
  See "Hard Disk configuration" section below.
- *Configure Dazzler* (only if Dazzler emulation is enabled)
  See "Cromemco Dazzler support" section.
- *Configure Interrupts*
  See "Interrupts configuration" section below.
- *Manage Filesystem*
  Starts file system manager (see File System Manager section below).
- *File System manager for SD card*
  Starts a file system manager to modify the contents of the SD card (if attached). The file manager also allows transferring files from and to the card via XMODEM protocol.

- *Clear memory*
  Clear the memory of the simulator (set all to 0)
- *Save configuration*
  Saves the configuration. Up to 256 different configurations can be saved.
  Configuration #0 is automatically loaded when the simulator starts.
- *Load configuration*
  Loads a saved configuration
- *Reset to defaults*
  Resets all settings to their default values. This can also be done by holding RESET up during power-up of the Simulator.

Host serial interface configuration

- *Host Serial … settings*
  For serial ports that support configuration settings other than baud rate, this opens a sub-menu to configure baud rate, number of bits, parity and number of stop bits. Otherwise it toggles the baud rate used for the port (those ports always use 8N1 configuration). Note that different ports may have different ranges of possible baud rates. See "Serial ports" section above.
- *Primary host serial* (Arduino Due only)
  Selects which serial interface is used as the primary interface. All simulator related output (such as the configuration menu) is sent to the primary serial interface. Also, when auto-detecting the serial device for capturing/replaying data, only devices mapped to the primary serial interface are considered.
- *Apply host serial settings*
  When making changes to the host serial settings (baud rate, primary interface), those are not applied immediately. Select this option to apply the modified settings.

Interrupt configuration

The simulator can emulate a 88-RTC-VI board which provides a real-time clock and vector interrupt capability (both are necessary to run Altair Timesharing Basic). The following settings can be configured:

- *Real Time Clock*
  Can be either disabled or set to produce an interrupt at one of the following frequencies:
  0.06, 0.6, 6, 10, 60, 100, 1000 or 10000Hz
  Note that (despite the name) the frequency is based on simulated time, not real time.
- *Vector Interrupt Board*
  If set to "Interrupts connected directly to CPU" then the Vector Interrupt functionality is disabled.
  In that case, the device interrupt settings below can only be changed between "connected" or "not connected". Note that the simulator (unlike the original) allows to connect multiple devices to the CPU's interrupt line
  If this is set to "Use Vector Interrupt Board" then each device interrupt can be assigned a level/priority according to the 88-RTC-VI board's functionality. Consult the 88-RTC-VI board's user manual for more information about the interrupt levels.
- *[Device] interrupt*
  This configures the connection of the interrupt line for each of the listed devices. If the VI board is disabled, then an interrupt line can either be connected or not connected to the CPU. If the VI board is enabled, then the interrupt can be connected to a specific level on the VI board.

Disk drive configuration

If disk drive support is enabled, this menu allows to modify drive related settings:

- *Force real-time mode*
  If a running program that interacts with the disk drive does not enable interrupts for the drive then the drive emulation works in a rapid mode in which new data is presented to the program every time it checks if new data is available. This makes for very fast disk emulation. If interrupts are enabled for the drive then it operates in real-time mode, only producing interrupts when new data would be available on a real drive.
  Enabling the "Force real-time mode" option will always operate the drive in the slower real-time mode, making for a more realistic LED blinking pattern while interacting with the disk drive.
- *Drive n mounted disk image*
  Shows which disk image (DISKxx.DSK) is currently mounted in which drive and cycles through the available images.

Hard disk configuration

If hard disk support is enabled, this menu allows to modify drive related settings:

- *Force real-time mode*
  If a running program that interacts with the hard disk does not enable interrupts for the drive then the drive emulation works in a rapid mode in which new data is presented to the program every time it checks if new data is available. This makes for very fast hard disk emulation. If interrupts are enabled for the hard disk then it operates in real-time mode, only producing interrupts when new data would be available on a real drive.
  Enabling the "Force real-time mode" option will always operate the drive in the slower real-time mode, making for a more realistic LED blinking pattern while interacting with the hard disk.
- *Hard disk [unit n] platter m image*
  Shows which image (HDSKxx.DSK) is currently mounted in which unit/platter and cycles through the available images. The [unit n] is only shown if the simulator is set up to emulate more than one unit (NUM_HDSK_UNITS setting in config.h).
- *Reset hard disk controller*
  Resets the controller. Mainly this sets the CRDY flag. Do this if emulated software instructs you to turn the hard disk unit off and back on.

Printer configuration

To configure printer options, select "(5) Configure printer" in the main configuration menu. The following configuration options are available for printer emulation:

- *Printer type*
  Select which printer should be emulated or turn off printer emulation:
  Okidata – emulates an Okidata printer at I/O ports 2/3
  C700 – emulates a Centronics C700 printer at I/O ports 2/3
- *Map printer to interface*
  Selects the host's serial interface to which the emulated printer's output should be directed
- *Force real-time mode*
  If enabled, the printer emulation will enforce timing similar to a real printer (although not necessarily matching the actual emulated printer model). Provides a printer-like effect when watching the output.

Serial device configuration

The simulator emulates four serial devices connected to the Altair:

- 88-SIO card at port 0x00/0x01
- 88-ACR audio cassette interface at port 0x06/0x07
- 88-2SIO, card with serial 1 at port 0x10/0x11 and serial 2 at port 0x12/0x13
- A second emulated 88-2SIO card can be enabled by changing the following setting before uploading the sketch to the Arduino. In file `config.h` change:
  `"#define USE_SECOND_2SIO 0"` to `"#define USE_SECOND_2SIO 1"`
  The second 88-2SIO has serial 1 at port 0x14/0x15 and serial 2 at pot 0x16/0x17

For each of these devices the following settings can be configured in the configuration menu:

- *Map to host interface*
  Select to which host interface input/output of device gets directed.
  When using Arduino Mega, there is only one host interface (Serial). When using the Due, there is the primary and secondary interface. The physical interface (Serial/Serial1) that the primary interface maps to can be picked on the main setup screen. The other one becomes the secondary interface.
- *Simulated baud rate*
  If receive interrupts are disabled for a serial device and "Force baud rate" is off, the simulator just makes a new byte available for playback whenever the running program requests one. That way, playback runs as maximum speed without the program missing any characters. If receive interrupts are enabled however, the characters must be sent at a rate that gives the running program a chance to keep up. Similarly, if transmit interrupts are enabled, the program expects some time to pass between sending characters.
  This setting selects the baud rate at which input/output is processed. Note that the timing is based on simulated time, not real time. That means that on the Mega (since it is running at 25% original speed) a rate of 110 baud may actually look more like 25 baud in real time.
- *Force baud rate*
  If this option is on then the simulator will always use the given baud rate for receive and transmit, even if interrupts are disabled. This can be used to get more realistic timing for code that is not using interrupts.
- *Example playback NULs*
  When playing back data to the running program (e.g. the AM2 assembler), the program may need some extra processing time after a carriage return to process the previous line. This is done (as it would have been on the original) by sending a number of NUL (0) bytes after a carriage return. Note that this setting only affects the case when playing back examples stored in the simulator, not for captured data being played back. For captured data, make sure to set the program from which you are capturing to produce the proper number of NULs.
- *Use 7 bits*
  Some Altair programs (e.g. 4k BASIC) use the 8^{th} bit of a character to signal end-of-string, assuming that the output device only uses 7 bits. If this is the case, then the last character of any string will appear mangled in the output. If this option is enabled, the simulator will always clear the 8^{th} bit before sending it to the serial output.
  If the option is set to "autodetect", the s will detect (for some known programs) based on the memory location of the "OUT" instruction whether the bit needs to be cleared or not.
- *Serial input uppercase*
  Some Altair programs (e.g. 4k BASIC) only display uppercase characters and expect all input to be uppercase characters. If this option is enabled, the simulator will translate any incoming lower-case character to upper-case.

If the option is set to “autodetect”, the simulator will detect (for some known programs) based on the memory location of the “IN” instruction whether the translation is required.

- *Translate backspace to*
  Allows to map the backspace character to other characters expected by different ALTAIR programs:
  off – backspace is sent as backspace (ASCII 8)
  underscore – backspace is sent to the simulated program as an underscore ‘_’ (ASCII 95) and an underscore sent by the simulated program is interpreted as backspace. This is usefule for 4k BASIC and ROM BASIC.
  delete – backspace sent to the simulated program as a delete (ASCII 127) character. Time-sharing BASIC expects this
  autodetect – the simulator attempts to automatically determine which conversion is necessary by the memory location of IN/OUT opcode.
- *Enable CLOAD/CSAVE traps* (for ACR device only)
  If this option is enabled then the simulator will catch BASIC CLOAD/CSAVE calls and automatically save/load to the internal simulator storage, avoiding the need to manually start cassette capture/replay.
- *SIO board revision (for SIO device only)*
  Very early Altair SIO board had slightly different arrangement of bits in the status register. This design (revision 0) was changed very quickly. Very few programs exist that require the rev0 board, for example the original prototype Microsoft BASIC. This setting allows switching between the revisions.
  The setting also allows to switch the SIO card into “Cromemco” mode which emulates a Cromemco serial card used by some of the Dazzler programs. See “Cromemco Dazzler support” below.

File System Manager

The emulator includes a very simple mini file system to store the different types of data that can be saved/captured. The file system manager can be invoked by selecting (M) in the configuration menu.

The file system manager supports the following commands:

- **F**: Format file system (erases all files)
- **d**: Delete a file from the file system
- **r**: Read a file and show contents on screen

Debugging Capabilities

When simulation is stopped (i.e. the WAIT LED is on) and the "Serial debug" option is enabled in the configuration menu, the following keys have a function:

0-9,a-f	Toggle SW0-15 (only if stand-alone mode enabled in setup.h)
/	Prompt for value to set SW0-15 (only if stand-alone mode enabled in setup.h)
r	Run
o	Stop
t	Step
R	Reset
!	Hard reset (STOP+RESET)
X/x	Examine/examine next
P/p	Deposit/deposit next
U	AUX1 up
u	AUX1 down
s	AUX2 up
l	AUX2 down
Q	Protect
q	Unprotect
>	Run from address
B	Add breakpoint (only if breakpoints enabled in setup.h)
V	Delete last breakpoint
D	Disassemble (will prompt for start address, space bar continues, any other key exits)
M	Dump memory (will prompt for start address, space bar continues, any other key exits)
n	change number system (hexadecimal/octal/decimal)
C	Enter configuration menu
L	Load a program or data through serial input into simulated memory First value is start address, second value is length, followed by data bytes (all values separated by spaces). Easier to enter data this way than using the switches.
H	Load a program in Intel HEX format through serial input. https://en.wikipedia.org/wiki/Intel_HEX Useful to deposit programs and/or data directly into the simulated memory.

Example program: Kill-the-bit game

To enter the kill-the-bit game below into the ALTAIR:

1. Activate RESET
2. Set SW7-0 to first byte of program (041 octal, 00 100 001 binary)
3. Activate DEPOSIT
4. Set SW7-0 to next byte of program
5. Activate DEPOSIT NEXT
6. Repeat steps 4-5 until all bytes have been entered
7. If running the simulator on an Arduino MEGA, the game runs slow.
   To compensate, change the content of memory location 006 from 016 to 056 (all octal).

To run the game:

1. Activate RESET
2. Set SW15-8 switches to 0
3. Activate RUN

To save the game to Arduino storage:

1. Set SW15-8 to 0 (selects memory page 0)
2. Set SW7 to 1 (selects memory page operation)
3. Set SW6 to 1 (selects memory page save)
4. Set SW5-0 to 0 (selects file number 0)
5. Activate AUX1 down

To load the game from Arduino storage:

1. Set SW15-0 as above, except SW6 to 0 (selects memory page load)
2. Activate AUX1 down

```
                ; Kill the Bit game by Dean McDaniel, May 15, 1975
                ;
                ; Object: Kill the rotating bit. If you miss the lit bit, another
                ;         bit turns on leaving two bits to destroy. Quickly
                ;         toggle the switch, don't leave the switch in the up
                ;         position. Before starting, make sure all the switches
                ;         are in the down position.
                ;
0000                        org   0
0000 210000                 lxi   h,0     ; initialize counter
0003 1680                   mvi   d,080h  ; set up initial display bit
0005 010E00                 lxi   b,0eh   ; higher value = faster
0008 1A       beg:          ldax  d       ; display bit pattern on
0009 1A                     ldax  d       ; ...upper 8 address lights
000A 1A                     ldax  d
000B 1A                     ldax  d
000C 09                     dad   b       ; increment display counter
000D D20800                 jnc   beg
0010 DBFF                   in    0ffh    ; input data from sense switches
0012 AA                     xra   d       ; exclusive or with A
0013 0F                     rrc           ; rotate display right one bit
0014 57                     mov   d,a     ; move data to display reg
0015 C30800                 jmp   beg     ; repeat sequence
0018                        end

Here is the program in octal for easier entry into the Altair:
000: 041 000 000 026 200 001 016 000
010: 032 032 032 032 011 322 010 000
020: 333 377 252 017 127 303 010 000
```

Loading 4k BASIC the old-school way

The simulator provides a quick and easy way to load 4k BASIC by setting SW0-7 to 00000101 and pressing AUX1 down (see “Aux1 down” section above).

However, it is also possible to load 4k BASIC the original (slow) way via a boot loader.

Here’s how to load BASIC as if it came from a paper tape reader:

- Start the configuration editor and select (1) to configure the SIO device
- Make sure the device is mapped to the host serial port to which your terminal is connected
- Set the “Use 7 bits”, “Serial input uppercase” and “Translate Backspace to” settings to “autodetect”
- Set the simulated baud rate to 110 baud. Obvioulsy you can choose other baud rates here but 110 baud is the speed at which a typical TeleType tape reader operated. **Note**: the baud rate is based on simulated time (not real time), so when running on the Arduino Mega, 110 baud will be closer to 25 baud. All time estimates given below will be about 4x longer. I suggest using 600 baud in that case.
- Enable the “Force baud rate” option
- Exit the configuration editor
- Key in the SIO bootloader (same procedure as entering kill-the-bit above):
```
000: 041 256 017 061 022 000 333 000
010: 017 330 333 001 275 310 055 167
020: 300 351 003 000
```
- You may want to save the bootloader to a file so you can reuse it later (same procedure as for the kill-the-bit example above)
- Activate RESET (to reset program counter to 0, the start of the boot loader)
- Activate RUN. The boot loader is now running. You should see LEDs A0-A4 on, A5-A15 should be off.
- Set all switches to 0, then raise SW15, SW7 and SW6. Tthis selects example 0xC0 (SW7-0=0xC0) which is the 4k BASIC tape image to be played back to the SIO device (SW15-13=100)
- Activate AUX2 down. The HLDA LED will come on indicating that data replay has started.
- Immediately switch A15 down (otherwise the second stage boot loader will attempt to load from ACR instead of SIO).
- For the first 17 seconds (while the first-stage boot loader is running) the pattern on the A15-A0 LEDs should not change.
- After about 17 seconds the second stage (checksum) boot loader should start running.
  At that point, LEDs 15-12 should be off, A11-A7, A5, A2, A1, A0 should be on and A6, A4 and A3 should be very faintly flashing (about 10x per second, one flash per byte received).
- The second stage boot loader takes about 6 ½ minutes. Once it is done, the LED pattern should change to A9-A7 and A3-A0 on, all others off and your termial should show the “MEMORY SIZE?” prompt.
- After you answer the usual startup questions, BASIC should be up and running.
- To load a BASIC example program, set SW0-7 to the program number (e.g. 00011011 for a simple prime number computation) and activate AUX2 down.

Altair Time Sharing BASIC

In its default configuration the simulator allows to run Altair Time Sharing BASIC with two users, one on a terminal connected to the Arduino Due's main serial port (either use pins 0 and 1 or the USB connection), the other using the Due's Serial1 port at pins 18 and 19. See the next page for information on how to enable 4 users.

To set up the simulator, enter the Configuration Editor and make sure both the Serial and Serial1 baud rates are set properly for your two terminals. The primary host serial port can be either one, whichever is more convenient.

Next configure the emulated devices. Time Sharing BASIC only supports 2SIO cards, so set the SIO and ACR cards to "Not mapped". Configure the the two ports of the 2SIO card as follows:

- One mapped to primary, the other mapped to the secondary host interface
- Simulated baud rate: 2400 (anything higher can cause problems when trying to play back examples)
- Example playback NULs: 0
- Use 7 bits: on
- Serial input uppercase: on
- Translate backspace to: delete

Time Sharing BASIC relies on interrupts to give each user their proper time slice and uses the 88-RTC-VI (Real-Time Clock and Vector Interrupt board) to do so. Therefore, both the Real-Time Clock and Vector Interrupts must be set up properly in the simulator. Select "Configure Interrupts" sub-menu:

- Enable the Real-Time Clock and set it to 60Hz or 100Hz
- Enable the Vector Interrupt board (Use Vector Interrupt Board)
- Set the Disk drive interrupt to VI0
- Set the Real-Time Clock interrupt to VI1
- Set the 88-2SIO port 1 interrupt to VI2
- Set the 88-2SIO port 2 interrupt to VI2
- Set the 88-SIO and 88-ACR interrupts to "Not Connected"

For convenience, you may want to configure the "Aux1 shortcut program" setting to the Time Sharing Basic v1.1 disk. After everything is set, you may want to save the configuration so you can later just load it.

Ready to run Time-Sharing BASIC!

First, connect your two terminals to the Serial and Serial1 connection on the Arduino. Make sure the terminal settings match the **host** interface baud rates set above (NOT the simulated baud rates).
Next, either press AUX1 UP (if you have configured it) or mount the Time-Sharing BASIC disk and run the disk boot loader (see the Disk Drive Support section for more information). After a few seconds, the ALTAIR T/S DISK BASIC V1.1 prompt should appear. Answer the configuration questions as follows:

- RECONFIGURE? N (you may say L and verify that LEVEL1 is set to TIMER and LEVEL2 is set to 2 – 16, 18)
- MEMORY SIZE? [just press ENTER]
- NUMBER OF USERS? 2
- TERMINAL ADDRESS? 16
- REGION SIZE? 20000
- TERMINAL ADDRESS? 18
- REGION SIZE? 20000
- MOUNT PASSWORD? [pick a password and press ENTER]

After the last prompt, a BASIC startup message (ALTAIR T/S DISK BASIC V1.1) should show up on both terminals. Both terminals can be used independently to interact with BASIC and load and run programs. Refer to the ALTAIR Time Sharing BASIC manual (can be found online) on how mount disks and load programs.

To allow more than two users, the simulator software must emulate a second 88-2SIO card. To see if it is configured for two cards, start the configuration editor. If it shows options to change settings for “2SIO-2 port 1” and “2SIO-2 port 2” (in addition to “2SIO port 1” and “2SIO port 2”) then it is already set. Otherwise, you need to update the software:

1. Load `config.h` in an editor
2. Change “`#define USE_SECOND_2SIO 0`” to “`#define USE_SECOND_2SIO 1`”
3. Upload the simulator software to your Arduino

After that is done, start the configuration editor and map “2SIO-2 port 1” to the Arduino’s Native USB port. If you want an additional user, follow the instructions in the “Serial Ports” section to enable the serial port on the A6/A7 pins of the Arduino.

Configure the “2SIO-2 port 1” (and “2SIO-2 port 2”) ports following the instructions for the other ports above, mapping them to Native USB port (and Serial A6/A7). Likewise, configure the interrupts for the new ports:

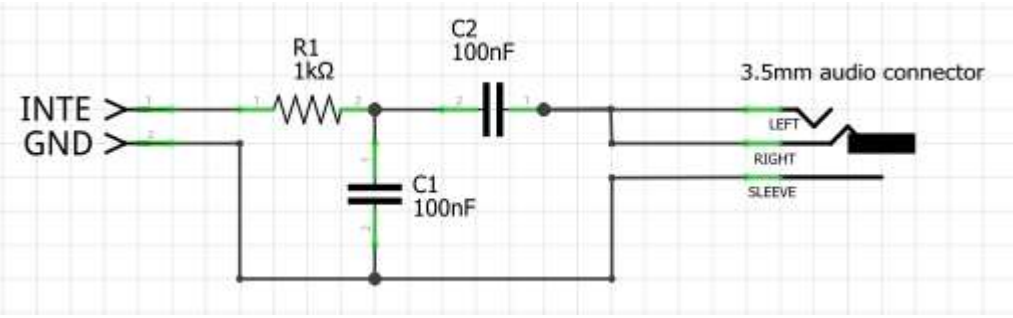
- Set the 88-2SIO port 1 interrupt to VI3
- Set the 88-2SIO port 2 interrupt to VI3

Now you can start Time-sharing basic either for three users (terminal addresses 16, 18, 20, region sizes of 14000) or four users (terminal addresses 16, 18, 20, 22, region sizes of 10000).

Music System

The Music System from Processor Technology was an affordable Music System for the Altair that required only a a minimal hardware addition. To learn more about the Music System and how it ran on the original Altair, head over to http://altairclone.com/music_system.htm where Mike Douglas has put together a great description.

You can run the music system on the simulator without any changes but in order to get any actual sound you will need to add just a minimal number of components: two capacitors and one resistor. Here's the schematic (taken from page 7 of the Music System documentation):



Connect the INTE input of the above schematic to the Arduino pin that drives the INTE light (pin D12 on the Due, pin D38 on the Mega) and GND to ground. The output is a line level mono signal that can be connected to the input of any amplified speaker system. It is not strong enough to drive headphones although but some mini earphones do produce a (not very loud) sound when plugged in.

The music system consists of two parts:

- The music system itself, which allows to enter, compile and play music (see User's Guide)
- The ACUTER monitor (a version of the CUTER monitor ported to the Altair and enhanced by Mike Douglas) which is necessary to load and save music as well as handling input/output (see User's Guide). The music system can **not** run on its own without the monitor.

To run the music system on the simulator, do the following:

- Enter the configuration monitor and set "throttle delay" to 12 (leaving it at automatic delay adds a weird vibrato effect when playing music)
- Make sure that 2SIO port 1 is mapped to your primary host interface
- Enable the "Serial input uppercase" option for the 2SIO port1
- Use a baud rate of no more than 9600 for your primary host interface (otherwise importing the HEX data for music examples will not work properly)
- Exit the configuration monitor
- Turn on switches SW0, SW2 and SW3, all others off
- Activate AUX1 down
- You should now see a ">" prompt in your terminal. This is the command prompt of the ACUTER monitor.
- Enter "EX 0" to initialize the music system itself.
- You should see "THE MUSIC SYSTEM (C) 1977 SOFTWARE TECHNOLOGY CORP"
- Enter "RET" to return to ACUTER (should see the ">" prompt)

Now we need to get some music into the system. Mike Douglas has collected a number of music examples for the system. Follow this link to get them (look in the subdirectories). To get an example into the system, we use the HGET command that Mike has added to ACUTER:

- Enter “HGET” in ACUTER
- You should see “Send/Rcv on port 0”
- Now click on one of the HEX files of music examples from the web site (e.g. SALLY.HEX) so it opens in your web browser. It should show many lines looking like
  :1008D3003430303130202F41495220574954482079
- Select all (likely CTRL-A) in your browser and copy it to the clipboard (CTRL-C)
- Now paste it into your terminal.
- The terminal should show many lines listing addresses and return to the “>” prompt when done.
- Now enter “EX 0” to go back to the music system (there will be **no** prompt after entering the system)
- Enter “FILE”, which causes the music system to scan memory for the imported music file
- Next, enter “SCORE”, which compiles the music
- Finally, enter “PLAY” to play the compiled music. You should hear the music in your speakers.

For a detailed description of the ACUTER and music system commands refer to their User manuals: ACUTER Music System. Here’s an example session (user input in ***bold italics***, system output normal):

```
>EX 0
THE MUSIC SYSTEM (C) 1977 SOFTWARE TECHNOLOGY CORP.   <77-05-19>
08D3 08D3
RET
>HGET
[paste contents of SALLY.HEX into terminal]
08D3
08E3
08F3
[…]
10E3
>EX 0
FILE
08D3 10F0
SCORE
10F1 17D1
PLAY
```

If desired, a piece of music that was imported via HGET can be saved within the simulator to a virtual tape by using the capture/replay functionality. To save an example to a file from within ACUTER:

```
>HGET
[paste contents of SALLY.HEX into terminal]
08D3
08E3
08F3
[…]
10E3
 [SW15, SW13, SW8, SW0 up, all others down => prepare to capture data from ACR to file #1]
[AUX2 up => start captur]
>SAVE SALLY 08D3 10F2
>
[AUX2 up => end capture]
```

To load an example from a file within ACUTER:

```
>GET
[SW15, SW13, SW8, SW0 up, all others down => prepare to play back file #1 to ACR]
[AUX2 down => start playback]
SALLY   08D3 0820
>
```

MITS Programming System II

[adapted from Mike Douglas' instructions at altairclone.com]

The MITS Programming System II allows development of 8080 assembly language applications on the Altair 8800 using just paper tape or cassette for mass storage. The package consists of an editor, assembler, debugger and a monitor that allows execution of these programs as well as the programs you may develop.
To use the programming package, the monitor program is loaded from paper tape or cassette in the same manner as loading BASIC. Once loaded, the monitor is then used to load the editor, assembler, debugger or user programs.
The monitor loads from paper tape or cassette using the same bootstrap loader as would be used for BASIC version 3.2. Sense switch settings are the same with the addition of A9 functionality:
A9 up – the monitor does NOT use serial input interrupts
A9 down – the monitor uses serial input interrupts

Loading the Monitor, Editor and Assembler

1) Set A15-A3 down and A2-A0 up.
2) Raise A9 if you want to disable interrupts. If interrupts are enabled, serial replay is slower (because it must run at the given baud rate). On the other hand, enabling interrupts enables the use of Ctrl-C.
3) Push AUX1 down to load the monitor.
4) The monitor prompt is two spaces and "?"
5) STOP! Do not type any commands to see "what happens." If what is typed is not a command, the monitor tries loading a program of that name from the cassette and hangs there until you provide that program. If the ABS device has been set to audio-cassette (AC), then Ctrl-C will return to the monitor prompt if interrupts were enabled during the boot process. Otherwise, follow the instructions below to restart the monitor from the front panel.
6) Assign the program load device to the cassette, type: "OPN ABS,AC<cr>"
7) Load the editor, type: "EDT<cr>". When loaded, the editor's prompt "*" is displayed. Type "E<cr>" to return the monitor.
8) Move the editor's buffer location into high memory so the assembler can reside in memory at the same time as the editor. For larger programs, more than the default 2K of buffer space will probably be needed as well. Here are settings for an 8K buffer:
   Type "DEP 5124<cr>" and enter "0<cr>" then "100<cr>" then ctrl-z.
   This specifies the 16-bit octal address 40000 (0x4000) for buffer start.
   Type "DEP 5530<cr>" and enter "0<cr>" then "140<cr>" then ctrl-z.
   This specifies the 16-bit octal address 60000 (0x6000) for buffer end.
9) Load the assembler, type: "AM2<cr>". When loaded, the assembler prompt is "*ASM*<cr>". Type "EOA<cr>" (end of assembly) to return to the monitor.
10) DBG, EDT and ASM share the same memory space (see Memory Allocation section below). If two of them are loaded sequentially, the system will just start the second one loaded, even if invoking the first. For example, if EDT is loaded first and then ASM, the monitor will invoke ASM if EDT is typed as a command. To force reloading of a component, use CLR to remove it from the program table, i.e. in the situation mentioned before, typing CLR EDT and then EDT will invoke the editor.

The computer is now ready to iteratively edit, assemble and run/test a program.

Using the Editor and Assembler

1. To start the editor, type: “EDT<cr>” This starts the editor and clears the edit buffer. To subsequently re-edit a program, type “EDT(R)<cr>” This leaves the existing source code in memory. If the “(R)” is left off, the program source will be erased.
2. At the editor prompt type “I” for insert. Ctrl-Z exits the insert mode. “P” prints (displays) the file with line numbers. “W” displays the file without line numbers. “D line[,line]” deletes line(s). “R line” replaces a line. “I line” inserts after the specified line. “I” by itself inserts before the first line. “E” exits the editor and returns control to the monitor.
3. End programs with:
   - BEG start label
   - END program name
   - EOA

   Where “start label” is the program entry point, “program name” is a three character program name. “EOA” means end of assembly.
4. Assign source file input to come from the edit buffer instead of a device and declare file type of ASCII (text file): Type, “OPN FIL,EB,A<cr>” This only has to be done once during a session (or if the FIL device was subsequently assigned to a different device during the session).
5. Run the assembler: Type “AM2<cr>”
6. Tell the assembler to take input from a file: Type “FILE<cr>” The assembler will run and show errors and undefined symbols. Note: The “Undefined Symbols” heading is displayed even when there are no undefined symbols. Control is returned to the monitor.
7. Run the program by jumping to the starting address: Type “JMP xxxxxx” where xxxxxx is the program starting address in octal.
8. You can enter the program into the monitor’s program table as follows: After a successful assembly, re-enter the assembler and preserve symbols: Type “AM2(P)<cr>”
9. Type “RUN name<cr>” where name is the three character name for the program. The program will run. From here on, the program can be run by typing the “name” specified at the monitor prompt. This step does not have to be repeated after subsequent assemblies if the entry address has not changed.

Memory Allocation

- The monitor is about 2.5K in length and uses RAM up through 0x0A3F
- The editor is about 2K in length and resides just above the monitor at 0xA40 – 0x11B1, followed by the default 2K edit buffer space from 0x11B2 – 0x19B1.
- Two versions of the assembler are available. Each are about 3K in length. ASM (assembler version 1) loads at the same address as the editor (0xA40). This is inconvenient for the iterative cycle of edit, assemble and test. As an alternative, AM2 (assembler version 2) loads just above the editor at 0x1350 – 0x1D78. The assembler’s symbol table grows up from 0x1D78. Note that the default location of the edit buffer conflicts with the load address of AM2 as the edit buffer grows. Before using the editor for longer programs, the location of the edit buffer should moved to a free area in memory by using the monitor DEP command to patch the buffer start address into octal locations 5124-5125 and the buffer end address+1 into octal locations 5530-5531.
- When using the AM2 assembler, a program loaded at 0x2000 leaves about 650 bytes of symbol table space (0x1D78 – 0x1FFF). This is enough for small demo programs of 100 lines or less. Otherwise, a higher starting address should be used for the target program
- The debugger is is about 2K in length and overlays the editor at 0xA40 – 0x133F. The debugger and AM2 assembler can both reside in memory at the same time. This makes it easy to patch programs with the assembler while debugging the program.

Loading and Saving Files

1. Assign the FIL device to the audio cassette and specify ASCII files:
   Type “OPN FIL,AC,A<cr>”
2. In the editor, type “L<cr>” to load a source file. Type “S<cr>” to save a source file.
3. After loading a source file into the editor, you can re-assign FIL to the edit buffer for use as the assembler input: Type “OPN FIL,EB,A<cr>”
4. When you want to save an edited file after the edit and assemble process, be sure to set the FIL device back to AC before using the “S” command from within the editor.
5. Optionally, you can take assembler source file input from the audio cassette. After assigning FIL to the AC (see step 2), in the assembler, type “FILE name<cr>” where “name” is the three character source file name for the assembler to read. The source file is then read directly from the audio cassette instead of the edit buffer.

Additional Notes

- When using the edit buffer as the source for the assembler, the source file must be “rewound” by editing the source file in between successive assemblies.
- To restart the monitor, stop the machine, set all switches off except A6 (i.e., set address of octal 100). Raise examine, set front panel switches back the way they were at load time, then depress run.
- When sending large amounts of text (e.g. pasting source code into the serial terminal), some characters may get lost. Refer the section about pasting text into the terminal in the “Interacting with running programs” about possible solutions.

Example usage #1: assemble directly from input

1) Make sure simulator is configured to send SIO output and SIO2 port 1 output to your terminal
2) *[STOP]+[RESET]* reset Altair
3) *[A15-A3 down, A2, A1, A0 up]* select program #7 (PS2 monitor) using SIO card with interrupts
4) *[AUX1 down]* load and run PS2 monitor and mount PS2 cassette tape
5) OPN ABS,AC assign ACR to ABS device (to load editor/assembler)
6) AM2 start assembler (to load it into memory)
7) *[CLR]* un-mount the PS2 tape
8) *[A7 up, A2 down, A1 up]* select ASM example #2 (“dump” example from PS2 documentation)
9) *[AUX2 down]* start ASM source code example replay
10) RUN DUMP runs DMP example directly from assembler

Example usage #2: insert into edit buffer, then assemble

1) Make sure simulator is configured to send SIO output and SIO2 port 1 output to your terminal
2) *[STOP]+[RESET]* reset Altair
3) *[A15-A3 down, A2, A1, A0 up]* select program #7 (PS2 monitor) using SIO card with interrupts
4) *[AUX1 down]* load and run PS2 monitor and mount PS2 cassette tape
5) OPN ABS,AC assign ACR to ABS device (to load editor/assembler)
6) EDT start editor (to load it into memory)
7) E end editor
8) AM2 start assembler (to load it into memory)
9) EOA exit assembler
10) DEP 5124 move edit buffer to 0x4000-0x6000 = 8K length
11) 0 start address 100-000 = 40000 octal = 0x4000
12) 100
13) *[CTRL-Z]* end of input
14) DEP 5530
15) 0 end address 140-000 = 60000 octal = 0x6000
16) 140

17) *[CTRL-Z]* end of input
18) EDT start editor again (with new buffer)
19) I enter "insert" mode
20) *[CLR]* un-mount the PS2 tape
21) *[A7 up, A2 down, A1 down, A0 up]* select ASM example #1 (PONG)
or [A7 up, A2 down, A1 up] select ASM example #2 (DUMP)
22) *[AUX2 down]* start ASM source code example replay
23) *[CTRL-Z]* exit "insert" mode after example is loaded
24) E exit editor
25) OPN FIL,EB,A select edit buffer as input device
26) AM2(S) go into assembler (S parameter to print symbol table at end)
27) FILE load input file (from edit buffer)
28) EOA return to monitor
29) JMP 20000 run program

Example usage #3: insert into edit buffer, assemble, write to file, run file

1) Make sure simulator is configured to send SIO output and SIO2 port 1 output to your terminal
2) *[STOP]+[RESET]* reset Altair
3) *[A15-A3 down, A2, A1, A0 up]* select program #7 (PS2 monitor) using SIO card with interrupts
4) *[AUX1 down]* load and run PS2 monitor and mount PS2 cassette tape
5) OPN ABS,AC assign ACR to ABS device (to load editor/assembler)
6) EDT start editor (to load it into memory)
7) E end editor
8) AM2 start assembler (to load it into memory)
9) EOA exit assembler
10) DEP 5124 move edit buffer to 0x4000-0x6000 = 8K length
11) 0 start address 100-000 = 40000 octal = 0x4000
12) 100
13) *[CTRL-Z]* end of input
14) DEP 5530
15) 0 end address 140-000 = 60000 octal = 0x6000
16) 140
17) *[CTRL-Z]* end of input
18) EDT start editor again (with new buffer)
19) I enter "insert" mode
20) *[CLR]* un-mount the PS2 tape from ACR
21) *[A7 up, A2 down, A1 down, A0 up]* select ASM example #1 (PONG)
22) *[AUX2 down]* start ASM source code example replay
23) *[CTRL-Z]* exit "insert" mode after example is loaded
24) E exit editor
25) OPN FIL,EB,A select edit buffer as input device
26) AM2(S,A) go into assembler (S parameter to print symbol table at end,
A to dump output binary to file)
27) *[A15, A13, A8 up, A7 down]* Prepare to capture file #1 from ACR device
28) FILE load input file (from edit buffer)
[wait for "SENSE SW 15 FOR DUMP" message]
29) *[AUX2 up]* start capturing
30) *[A15 down, A15 up]* tells AM2 to start writing
[wait for "?" prompt from PS2 monitor]
31) *[AUX2 up]* finish capturing
32) *[AUX2 down]* start replay
33) PONG load and run PONG

8080 Instruction Set

```
Conventions in instruction source:
    D   = Destination register (8 bit)
    S   = Source register (8 bit)
    RP  = Register pair (16 bit)
    #   = 8 or 16 bit immediate operand
    a   = 16 bit Memory address
    p   = 8 bit port address
    ccc = Conditional

Conventions in instruction encoding:
    db  = Data byte (8 bit)
    lb  = Low byte of 16 bit value
    hb  = High byte of 16 bit value
    pa  = Port address (8 bit)

Dest and Source reg fields:
    111=A   (Accumulator)
    000=B
    001=C
    010=D
    011=E
    100=H
    101=L
    110=M   (Memory reference through address in H:L)

Register pair 'RP' fields:
    00=BC   (B:C as 16 bit register)
    01=DE   (D:E as 16 bit register)
    10=HL   (H:L as 16 bit register)
    11=SP   (Stack pointer, refers to PSW (FLAGS:A) for PUSH/POP)

Condition code 'CCC' fields: (FLAGS: S Z x A x P x C)
    000=NZ  (Zero flag not set)
    001=Z   (Zero flag set)
    010=NC  (Carry flag not set)
    011=C   (Carry flag set)
    100=PO  (Parity flag not set - ODD)
    101=PE  (Parity flag set - EVEN)
    110=P   (Sign flag not set - POSITIVE)
    111=M   (Sign flag set - MINUS)
```

Inst	Encoding	Flags	Description
ACI #	11001110 db	ZSCPA	Add immediate to A with carry*
ADC S	10001SSS	ZSCPA	Add register to A with carry
ADD S	10000SSS	ZSPCA	Add register to A
ADI #	11000110 db	ZSCPA	Add immediate to A
ANA S	10100SSS	ZSCPA	AND register with A
ANI #	11100110 db	ZSPCA	AND immediate with A
CALL a	11001101 lb hb	-	Unconditional subroutine call
Cccc a	11CCC100 lb hb	-	Conditional subroutine call
CMA	00101111	-	Complement A
CMC	00111111	C	Complement Carry flag
CMP S	10111SSS	ZSPCA	Compare register with A
CPI #	11111110	ZSPCA	Compare immediate with A
DAA	00100111	ZSPCA	Decimal Adjust accumulator
DAD RP	00RP1001	C	Add register pair to HL (16 bit add)*
DCR D	00DDD101	ZSPA	Decrement register*
DCX RP	00RP1011	-	Decrement register pair
DI	11110011	-	Disable interrupts
EI	11111011	-	Enable interrupts
HLT	01110110	-	Halt processor
IN p	11011011 pa	-	Read input port into A
INR D	00DDD100	ZSPA	Increment register
INX RP	00RP0011	-	Increment register pair
Jccc a	11CCC010 lb hb	-	Conditional jump*
JMP a	11000011 lb hb	-	Unconditional jump*
LDA a	00111010 lb hb	-	Load A from memory
LDAX RP	00RP1010 *1	-	Load indirect through BC or DE
LHLD a	00101010 lb hb	-	Load H:L from memory*
LXI RP,#	00RP0001 lb hb	-	Load register pair immediate*
MOV D,S	01DDDSSS	-	Move register to register*
MVI D,#	00DDD110 db	-	Move immediate to register*
NOP	00000000	-	No operation
ORA S	10110SSS	ZSPCA	OR register with A
ORI #	11110110	ZSPCA	OR immediate with A
OUT p	11010011 pa	-	Write A to output port
PCHL	11101001	-	Jump to address in H:L
POP RP	11RP0001 *2	*2	Pop register pair from the stack
PUSH RP	11RP0101 *2	-	Push register pair on the stack
RAL	00010111	C	Rotate A left through carry*
RAR	00011111	C	Rotate A right through carry
Rccc	11CCC000	-	Conditional return from subroutine
RET	11001001	-	Unconditional return from subroutine
RLC	00000111	C	Rotate A left
RRC	00001111	C	Rotate A right
RST n	11NNN111	-	Restart (Call n*8)
SBB S	10011SSS	ZSCPA	Subtract register from A with borrow
SBI #	11011110 db	ZSCPA	Subtract immediate from A with borrow
SHLD a	00100010 lb hb	-	Store H:L to memory*
SPHL	11111001	-	Set SP to content of H:L
STA a	00110010 lb hb	-	Store A to memory
STAX RP	00RP0010 *1	-	Store indirect through BC or DE
STC	00110111	C	Set Carry flag
SUB S	10010SSS	ZSCPA	Subtract register from A
SUI #	11010110 db	ZSCPA	Subtract immediate from A
XCHG	11101011	-	Exchange DE and HL content
XRA S	10101SSS	ZSPCA	Exclusive OR register with A
XRI #	11101110 db	ZSPCA	Exclusive OR immediate with A
XTHL	11100011	-	Swap H:L with top word on stack

*1 = Only RP=00(BC) and 01(DE) are allowed for LDAX/STAX
*2 = RP=11 refers to PSW for PUSH/POP (cannot push/pop SP).
When PSW is POP'd, ALL flags are affected.

Encoding	Inst	Flags	Description
00000000	NOP	-	No operation
00000111	RLC	C	Rotate A left
00001111	RRC	C	Rotate A right
00010111	RAL	C	Rotate A left through carry*
00011111	RAR	C	Rotate A right through carry
00100010 lb hb	SHLD a	-	Store H:L to memory*
00100111	DAA	ZSPCA	Decimal Adjust accumulator
00101010 lb hb	LHLD a	-	Load H:L from memory*
00101111	CMA	-	Complement A
00110010 lb hb	STA a	-	Store A to memory
00110111	STC	C	Set Carry flag
00111010 lb hb	LDA a	-	Load A from memory
00111111	CMC	C	Complement Carry flag
00DDD100	INR D	ZSPA	Increment register
00DDD101	DCR D	ZSPA	Decrement register*
00DDD110 db	MVI D,#	-	Move immediate to register*
00RP0001 lb hb	LXI RP,#	-	Load register pair immediate*
00RP0010 *1	STAX RP	-	Store indirect through BC or DE
00RP0011	INX RP	-	Increment register pair
00RP1001	DAD RP	C	Add register pair to HL (16 bit add)*
00RP1010 *1	LDAX RP	-	Load indirect through BC or DE
00RP1011	DCX RP	-	Decrement register pair
01110110	HLT	-	Halt processor
01DDDSSS	MOV D,S	-	Move register to register*
10000SSS	ADD S	ZSPCA	Add register to A
10001SSS	ADC S	ZSCPA	Add register to A with carry
10010SSS	SUB S	ZSCPA	Subtract register from A
10011SSS	SBB S	ZSCPA	Subtract register from A with borrow
10100SSS	ANA S	ZSCPA	AND register with A
10101SSS	XRA S	ZSPCA	Exclusive OR register with A
10110SSS	ORA S	ZSPCA	OR register with A
10111SSS	CMP S	ZSPCA	Compare register with A
11000011 lb hb	JMP a	-	Unconditional jump*
11000110 db	ADI #	ZSCPA	Add immediate to A
11001001	RET	-	Unconditional return from subroutine
11001101 lb hb	CALL a	-	Unconditional subroutine call
11001110 db	ACI #	ZSCPA	Add immediate to A with carry*
11010011 pa	OUT p	-	Write A to output port
11010110 db	SUI #	ZSCPA	Subtract immediate from A
11011011 pa	IN p	-	Read input port into A
11011110 db	SBI #	ZSCPA	Subtract immediate from A with borrow
11100011	XTHL	-	Swap H:L with top word on stack
11100110 db	ANI #	ZSPCA	AND immediate with A
11101001	PCHL	-	Jump to address in H:L
11101011	XCHG	-	Exchange DE and HL content
11101110 db	XRI #	ZSPCA	Exclusive OR immediate with A
11110011	DI	-	Disable interrupts
11110110	ORI #	ZSPCA	OR immediate with A
11111001	SPHL	-	Set SP to content of H:L
11111011	EI	-	Enable interrupts
11111110	CPI #	ZSPCA	Compare immediate with A
11CCC000	Rccc	-	Conditional return from subroutine
11CCC010 lb hb	Jccc a	-	Conditional jump*
11CCC100 lb hb	Cccc a	-	Conditional subroutine call
11NNN111	RST n	-	Restart (Call n*8)
11RP0001 *2	POP RP	*2	Pop register pair from the stack
11RP0101 *2	PUSH RP	-	Push register pair on the stack

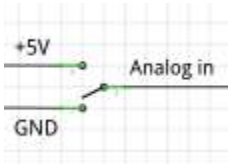
*1 = Only RP=00(BC) and 01(DE) are allowed for LDAX/STAX
*2 = RP=11 refers to PSW for PUSH/POP (cannot push/pop SP).
When PSW is POP'd, ALL flags are affected.

Hardware setup for Arduino Mega 2560

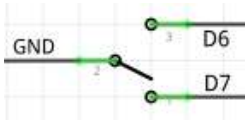
What makes the Arduino Mega 2560 perfect for this project is that it has exactly the right number of I/O pins to connect all the Altair's input/output elements:

Altair:	12	Status LEDs (INT,WO,STACK,HLTA,OUT,M1,INP,MEMR,PROT,INTE,WAIT,HLDA)
	8	Data bus LEDs (D0-D7)
	16	Address bus LEDs (A0-A15)
	16	Input switches (SW0-15)
	16	Function switches (RUN,STOP,EXAMINE,EXAMINE NEXT,DEPOSIT,DEPOSIT NEXT,RESET,CLR,PROTECT,UNPROTECT,AUX1 UP/DOWN,AUX2 UP/DOWN)
	2	Connections for serial RX/TX
	70	**(digital) inputs/outputs required**
Arduino Mega 2560:	54	Digital I/O pins
	16	Analog input pins (can be used for digital input)
	70	**I/O lines available**

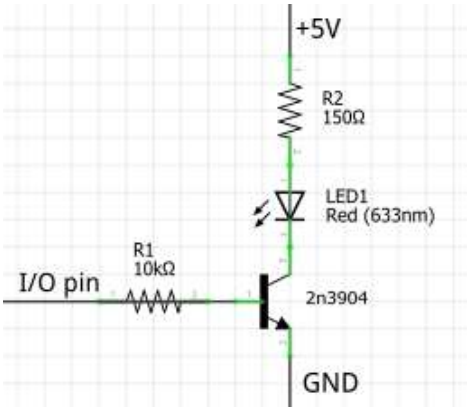
For simplicity, we connect the Altair's 16 input switches (SW0-15) to the Arduino's 16 analog inputs. These are SPDT On-On switches that alternate between two settings:



For the function switches we use the Arduino's internal pull-up resistors and just switch GND to the appropriate digital input. These are momentary SPDT (On)-Off-(On) switches. For example, the EXAMINE/EXAMINE NEXT switch is connected like this:



To drive the output LEDs we just use a simple transistor switch circuit for each LED (to prevent overload on the Arduino if too many of the LEDs are on at the same time):



See the following page for a mapping of exactly which front panel element maps to which Arduino I/O pin.

Pin Mapping for Arduino Mega 2560

Mapping Arduino pin to Altair element

Arduino Pin	Altair Element	Direction	Arduino Pin	Altair Element	Direction
D0	Serial RX	in	**D35**	A10	out
D1	Serial TX	out	**D36**	A9	out
D2	AUX2 UP	in	**D37**	A8	out
D3	AUX2 DOWN	in	**D38**	INTE	out
D4	STEP	in	**D39**	PROT	out
D5	SLOW	in	**D40**	WAIT	out
D6	EXAMINE	in	**D41**	HLDA	out
D7	EXAMINE NEXT	in	**D42**	D7	out
D8	DEPOSIT	in	**D43**	D6	out
D9	DEPOSIT NEXT	in	**D44**	D5	out
D10	OUT	out	**D45**	D4	out
D11	M1	out	**D46**	D3	out
D12	INP	out	**D47**	D2	out
D13	MEMR	out	**D48**	D1	out
D14	AUX1 UP	in	**D49**	D0	out
D15	AUX1 DOWN	in	**D50**	HLTA	out
D16	PROTECT	in	**D51**	STACK	out
D17	UNPROTECT	in	**D52**	WO	out
D18	RESET	in	**D53**	INT	out
D19	CLR	in	**A0**	SW0	in
D20	RUN	in	**A1**	SW1	in
D21	STOP	in	**A2**	SW2	in
D22	A0	out	**A3**	SW3	in
D23	A1	out	**A4**	SW4	in
D24	A2	out	**A5**	SW5	in
D25	A3	out	**A6**	SW6	in
D26	A4	out	**A7**	SW7	In
D27	A5	out	**A8**	SW8	In
D28	A6	out	**A9**	SW9	In
D29	A7	out	**A10**	SW10	In
D30	A15	out	**A11**	SW11	In
D31	A14	out	**A12**	SW12	In
D32	A13	out	**A13**	SW13	In
D33	A12	out	**A14**	SW14	In
D34	A11	out	**A15**	SW15	In

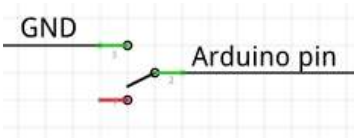
Mapping Altair element to Arduino pin

Altair Element	Arduino Pin	Direction	Altair Element	Arduino Pin	Direction
SW0	A0	in	**A0**	D22	out
SW1	A1	in	**A1**	D23	out
SW2	A2	in	**A2**	D24	out
SW3	A3	in	**A3**	D25	out
SW4	A4	in	**A4**	D26	out
SW5	A5	in	**A5**	D27	out
SW6	A6	in	**A6**	D28	out
SW7	A7	In	**A7**	D29	out
SW8	A8	in	**A8**	D37	out
SW9	A9	in	**A9**	D36	out
SW10	A10	in	**A10**	D35	out
SW11	A11	in	**A11**	D34	out
SW12	A12	in	**A12**	D33	out
SW13	A13	in	**A13**	D32	out
SW14	A14	in	**A14**	D31	out
SW15	A15	in	**A15**	D30	out
STOP	D21	in	**D0**	D49	out
RUN	D20	in	**D1**	D48	out
SINGLE STEP	D4	in	**D2**	D47	out
SLOW	D5	in	**D3**	D46	out
EXAMINE	D6	in	**D4**	D45	out
EXAMINE NEXT	D7	in	**D5**	D44	out
DEPOSIT	D8	in	**D6**	D43	out
DEPOSIT NEXT	D9	in	**D7**	D42	out
RESET	D18	in	**INT**	D53	out
CLR	D19	in	**WO**	D52	out
PROTECT	D16	in	**STACK**	D51	out
UNPROTECT	D17	in	**HLTA**	D50	out
AUX1 UP	D14	in	**OUT**	D10	out
AUX1 DOWN	D15	in	**M1**	D11	out
AUX2 UP	D2	in	**INP**	D12	out
AUX2 DOWN	D3	in	**MEMR**	D13	out
Serial RX	D0	in	**PROT**	D39	out
Serial TX	D1	out	**INTE**	D38	out
			WAIT	D40	out
			HLDA	D41	out

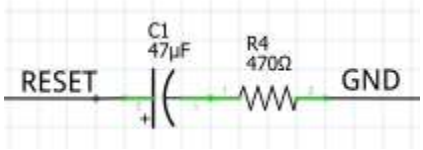
Hardware setup for Arduino Due

The setup for the Arduino Due is similar to the Arduino Mega (see above), just the pins are different (because the simulator accesses some of the pins directly by their processor register to increase efficiency). See the next page for the Arduino Due connections.

- LED drivers: Same as for the Arduino Mega. Note that the LED driver still uses +5V even though the Due runs at 3.3V (because the voltage is only used for lighting the LEDs).
- Function switches: Same as for the Arduino Mega (only different pins)
- Address switches: On the Due we can access the analog input pins just like digital inputs, including the ability to switch on an internal pullup resistor. That slightly simplifies the setup for the SW0-SW15 switches as we do not have to connect Vcc to the switches, only GND:



- The reset circuit on (some) Due boards is not very reliable at power-up (this seems to be a common problem), leaving the Due sometimes in a blocked state after power-up. If you experience this, a simple workaround is to add a capacitor and resistor to keep the RESET line low for a bit longer at power-up. This has worked fine for me and not caused any side effects.



- In the mapping on the next pages you will see some Arduino pins shown with two labels, for example A0 (D54). In this case, A0 is the label on the board and D54 is the digital pin number that relates to that connection (i.e. Analog input 0 (A0) can be accessed as Digital input 54 (D54))

Pin Mapping for Arduino Due

Mapping Arduino pin to Altair element

Arduino Pin	Altair Element	Direction	Arduino Pin	Altair Element	Direction
D0	Unused		**D36**	A2	out
D1	Unused		**D37**	A3	out
D2	INT	out	**D38**	A4	out
D3	WO	out	**D39**	A5	out
D4	STACK	out	**D40**	A6	out
D5	HLTA	out	**D41**	A7	out
D6	OUT	out	**D42**	SW14	in
D7	M1	out	**D43**	SW15	in
D8	INP	out	**D44**	A15	out
D9	MEMR	out	**D45**	A14	out
D10	WAIT	out	**D46**	A13	out
D11	D7	out	**D47**	A12	out
D12	INTE	out	**D48**	A11	out
D13	PROT	out	**D49**	A10	out
D14	D4	out	**D50**	A9	out
D15	D5	out	**D51**	A8	out
D16	SW9	in	**D52**	RESET	in
D17	SW8	in	**D53**	CLR	in
D18	Serial TX	out	**A0 (D54)**	STEP	in
D19	Serial RX	in	**A1 (D55)**	SLOW	in
D20	RUN	in	**A2 (D56)**	EXAMINE	in
D21	STOP	in	**A3 (D57)**	EXAMINE NEXT	in
D22	HLDA	out	**A4 (D58)**	DEPOSIT	in
D23	SW10	in	**A5 (D59)**	DEPOSIT NEXT	in
D24	SW11	in	**A6 (D60)**	PROTECT	in
D25	D0	out	**A7 (D61)**	UNPROTECT	in
D26	D1	out	**A8 (D62)**	SW0	in
D27	D2	out	**A9 (D63)**	SW1	in
D28	D3	out	**A10 (D64)**	SW2	in
D29	D6	out	**A11 (D65)**	SW3	in
D30	AUX1 UP	in	**DAC0 (D66)**	SW4	in
D31	AUX1 DOWN	in	**DAC1 (D67)**	SW5	in
D32	AUX2 UP	in	**CANRX (D68)**	SW6	in
D33	AUX2 DOWN	in	**CANTX (D69)**	SW7	in
D34	A0	out	**SDA1 (D70)**	SW12	in
D35	A1	out	**SCL1 (D71)**	SW13	in

Mapping Altair element to Arduino pin

Altair Element	Arduino Pin	Direction	Altair Element	Arduino Pin	Direction
SW0	A8 (D62)	in	**A0**	D34	out
SW1	A9 (D63)	in	**A1**	D35	out
SW2	A10 (D64)	in	**A2**	D36	out
SW3	A11 (D65)	in	**A3**	D37	out
SW4	DAC0 (D66)	in	**A4**	D38	out
SW5	DAC1 (D67)	in	**A5**	D39	out
SW6	CANRX (D68)	in	**A6**	D40	out
SW7	CANTX (D69)	In	**A7**	D41	out
SW8	D17	in	**A8**	D51	out
SW9	D16	in	**A9**	D50	out
SW10	D23	in	**A10**	D49	out
SW11	D24	in	**A11**	D48	out
SW12	SDA1 (D70)	in	**A12**	D47	out
SW13	SCL1 (D71)	in	**A13**	D46	out
SW14	D42	in	**A14**	D45	out
SW15	D43	in	**A15**	D44	out
STOP	D21	in	**D0**	D25	out
RUN	D20	in	**D1**	D26	out
SINGLE STEP	A0 (D54)	in	**D2**	D27	out
SLOW	A1 (D55)	in	**D3**	D28	out
EXAMINE	A2 (D56)	in	**D4**	D14	out
EXAMINE NEXT	A3 (D57)	in	**D5**	D15	out
DEPOSIT	A4 (D58)	in	**D6**	D29	out
DEPOSIT NEXT	A5 (D59)	in	**D7**	D11	out
RESET	D52	in	**INT**	D2	out
CLR	D53	in	**WO**	D3	out
PROTECT	A6 (D60)	in	**STACK**	D4	out
UNPROTECT	A7 (D61)	in	**HLTA**	D5	out
AUX1 UP	D30	in	**OUT**	D6	out
AUX1 DOWN	D31	in	**M1**	D7	out
AUX2 UP	D32	in	**INP**	D8	out
AUX2 DOWN	D33	in	**MEMR**	D9	out
Serial RX	D19	in	**PROT**	D13	out
Serial TX	D18	out	**INTE**	D12	out
			WAIT	D10	out
			HLDA	D22	out

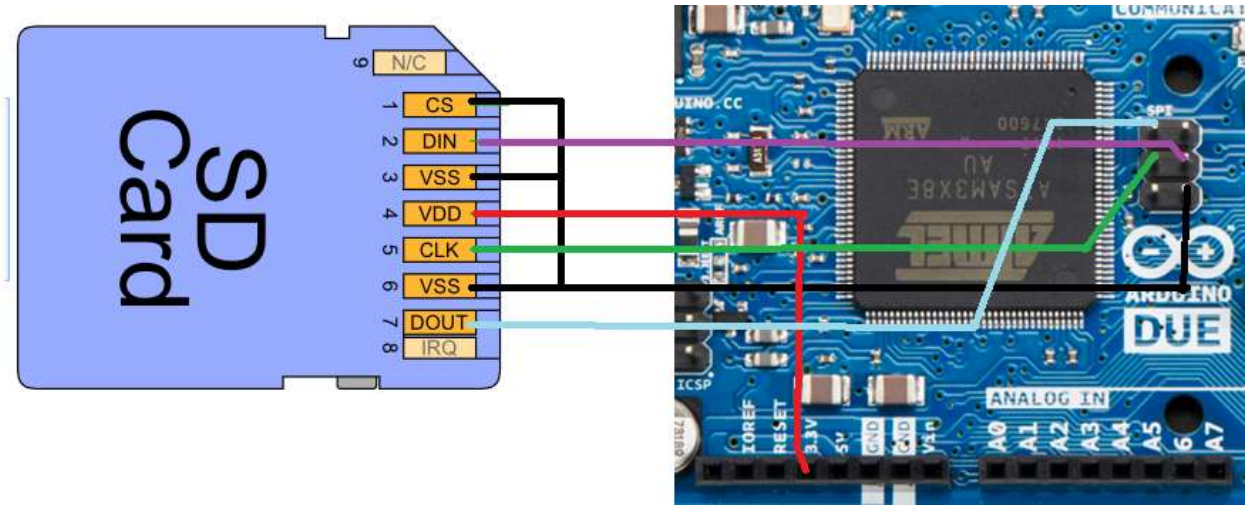
Wiring an SD card to the Arduino Due

Standard Arduino SD card shields will not work with the Arduino Due because the SPI pins are not connected to the D13-D11 pins as in other Arduino board. That is a good thing because we're already using the D13-D11 pins for front panel elements.

On the DUE, the SPI pins are **only** available on the separate 2-row, 6-pin SPI header (labeled "SPI" on the board).

There are commercial products such as the SparkFun Level Shifting microSD Breakout board that provide an SD card slot. That board is certainly works but is overkill since the Arduino DUE (like the SD cards) operates on 3.3V so no level shifting is required.

In fact, an SD card can be wired directly (without any other required electronic elements) to the SPI header on the Due:



Unfortunately, there is no 3.3V pin on the SPI header on the Arduino Due so that wire must go to the (separate) 3.3V pin. Do not wire the 5V output from the SPI header to the card. Doing so could damage the SD card and/or the Due.

The CS (chip select) pin is wired directly to GND, so the SD card is always selected. That saves us from having to find another I/O pin on the Arduino to use for chip select. The SD library on the Arduino requires to specify a Chip Select output pin but the simulator software sets that to the HLDA light output pin which as a side effect gives a "sd card active" visual indicator.

I recommend getting a microSD card with a microSD-to-SD adapter. Take the microSD card out of the adapter and create a cable by soldering wires directly to the adapter's pins and connecting them to the SPI header using the wiring given above. The adapter now serves as the socket for the microSD card, which can be plugged in and taken out easily.