Assembly Language Programming on the IMSAI 8080.

by Bruce E. Hall, W8BH

Not long ago I caught the retrocomputing bug. I wanted to relive some of the excitement I had as a kid with the early 8080 and Z80 microcomputers. So, I visited Tindie.com and clicked the 'Purchase Now' button for a RC2014 Zed. It was great!

Then, while browsing the Internet, it happened again. I found a kit for the IMSAI 8080, that amazing computer I could only *dream* about as a kid. The kit uses modern components to emulate the original IMSAI 8080. Its beautiful front panel was impossible to resist.

The kit instructions are thorough. A series of YouTube videos compliment the written guide.

Once built, however, you are on your own. Referring to the original IMSAI 8080 user manuals is a daunting task. The path forward is unclear if you are new to retrocomputing and want to do assembly language programming.

Read on if you want to compile assembly language programs for the IMSAI 8080. I will explain what you need and how to start writing your first assembly language programs. I assume that you have some knowledge of programming and understand your way around bits, bytes, and hexadecimal numbers.

The first half of this writeup is "Getting to know the IMSAI". It is meant for those who are meeting this retro hardware for the first time. Never rush into programming without knowing your programming environment. If you are already a seasoned user, skip to part 2.

Table of Contents

Part 1: Getting to know the IMSAI.

Throughout this document, when I refer to the IMSAI 8080 or just IMSAI, I am referring to the IMSAI 8080 replica (also known as IMSAI 8080esp) by The High Nibble.

What you need for Part 1:

- Assembled and working IMSAI 8080
- PC running Microsoft Windows
- Tera Term software for Windows

Connecting to a terminal

If you've just finished assembling your kit and are wondering what to do next, here you go:

First, set up a terminal program on your computer. I am a long time PuTTY user. But I have to say, Tera Term is hard to beat. Give it a try!

Tera Term setup

Get a copy of the .exe installer. Install and run the software. Tera Term (github.com)

Serial port: Set speed to 115200, 8N1. No flow control. No transmit delays.

Font: change font to Terminal/Regular/12 point

Windows: change text color to Green (R0/G255/B0)

Save setup to the default configuration file.

Next, physically connect your PC to the IMSAI 8080 with a USB cable, as you did in the Testing I phase of the assembly. Confirm, via the Device Manager, that there is a communications port open between the PC and the IMSAI 8080. It should appear as a line item under ports, such as "Silicon Labs CP210x USB to UART Bridge

(COM3)". The port number shown must correspond to the Serial Port chosen in your Tera Term setup.

When the IMSAI 8080 is connected to the PC, you should see bootup information displayed on your terminal emulator. If not, check your serial port configuration.

Flip the power switch on your IMSAI 8080 to ON. You will see more bootup information displayed on the terminal, as the simulation software on the ESP32 runs through its startup sequence. You may have to screen back to see it.

What happens now depends on your startup configuration. If the front panel WAIT LED is lit, you'll need to press <Run> on the front panel to run any pre-loaded software.

Putting the IMSAI 8080 on your local WiFi network

After you have a working serial connection, put the IMSAI 8080 on your local network. There are several ways to do it.

The following method uses the desktop UI:

1. Get to desktop UI on your PC by selecting IMSAI8080 as the WiFi access point then browsing to 192.168.4.1
2. Click on SYS:
3. Click on Gear icon in top-right corner. It will bring up CFG: screen
4. Click on boot.conf.
5. Edit the "SSID=" and "Password=" entries to make your local WiFi network
6. Click the Save icon in top-right corner. You should see a confirmation message.
7. Click Close icon in top-right corner.
8. Using the method below, change your current startup configuration to include bit3, which is WiFI-STA bit. So, for example, the default configuration would change from 0000h to 0008h.
9. Power cycle the IMSAI 8080.

If you prefer editing files on the SD card instead, do this:

1. Set the startup configuration to include bit3=1, which is WiFI-STA bit. For example, the default configuration would change from 0000h to 0008h.
2. Power off the front panel and remove power from the IMSAI 8080.
3. Remove the SD card. A few small tools will help extract the card through the case ventilation slot. I suggest using a popsicle stick for activating the SD push-to-release mechanism, then using tweezers to gently remove the card.
4. Load the SD card in your PC using an appropriate adapter.
5. Navigate to the imsai/conf folder.
6. Right-click on the boot.conf file and open it with Notepad.
7. Change "SSID=", adding your local network name. Mine is "SSID=Cheetos"
8. Change "Password=", adding your WiFi network password.
9. While you are at it, consider changing the time zone string "TZ=". Here in the Eastern US , the appropriate string is "TZ=EST5EDT,M3.2.0/2,M11.1.0/2"
10. Save the file.
11. Return the SD card to the IMSAI 8080 using that popsicle stick, with the SD label facing away from you and the gold connection pads facing towards you.
12. Reconnect and power up.

SD card information

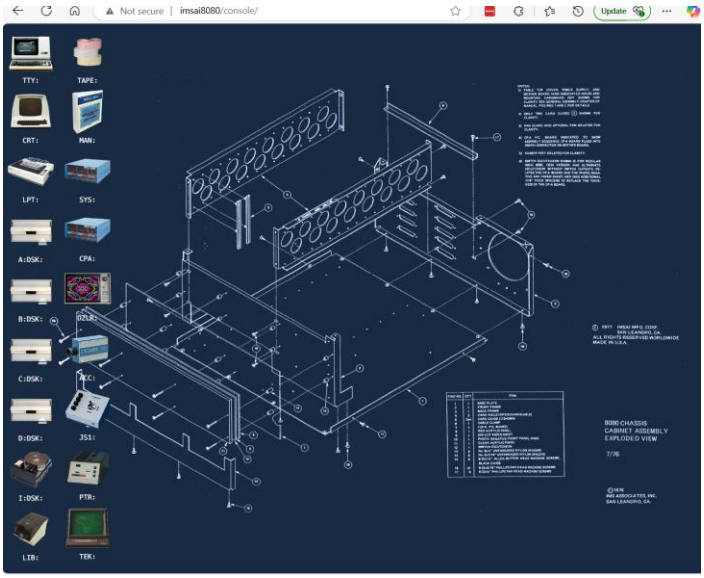
You can read the SD card in your PC! Name is "Z80PACK"
It has 3 folders:
- *IMSAI – for IMSAI 8080 software*
- *Z80 – for Cromenco Z80 software*
- *WWW – files for web browser UI*

Inside the IMSAI folder you will find these subfolders:
- *Conf – for the configuration files*
- *Disks – for the disk library*
- *Manual – for the included manuals*
- *Roms – for the available roms, in Intel .hex format*
- *Tapes – for the available tapes, in Intel.hex format*

After configuring the machine for local WiFi access, type "imsai8080" into your browser, and you should be able to view the web interface.

You have several ways of interacting with the IMSAI 8080: the front panel switches, the USB serial port, two RS-232 serial ports, and the web interface. Keep in mind that the TTY: web interface and the USB serial port are mutually exclusive. When the web TTY is active, the serial/USB port is not, and vice versa. If you want to switch from the web TTY back to the serial port, click on the plug icon (green plug in top-right corner of the TTY window), turning it to red. To switch back to web TTY, click on the plug icon again, turning it from red to green.

The Browser/Desktop User Interface



The desktop interface is an interactive web page as shown on the right. Each icon corresponds to a device on the emulated system: floppy drives, a hard drive, serial ports, a printer, etc. Click on an icon to interact with it.

TTY is the teletype device, which is a text-based console. One click will open it, and the "underscore" icon will close it. Use <alt><enter> to toggle full screen mode. The plug icon indicates The TTY's connection status to your system. A green icon means this TTY window is connected to one of the machine's serial ports; a red icon indicates this window is disconnected. In the default configuration (0000h), TTY won't do much. Once a monitor program or CP/M is loaded, however, TTY comes alive.

CRT is an emulated high-speed console. You will need a special configuration setting (see below) to enable its use. Try 8258h to enable it. You can use CP/M commands to transfer control between the CRT and TTY:

- To transfer control from CRT to TTY: type "STAT CON:=TTY:" in the CRT window.
- To transfer control from TTY to CRT: type "STAT CON:=CRT:" in the TTY window.

A:DSK, B:DSK, C:DISK, and D:DISK are the emulated floppy drives. To use the floppy drives:

- Hover over the icon to see the name of the mounted floppy disk, if any. A red-circle sub-icon in the lower-right corner means that the drive is empty.
- Click on the small up-arrow icon to eject the mounted disk. After confirming, the icon will change to a red "no disk" icon.
- To mount a different floppy, open LIB:, select a floppy*.dsk file, and carefully drag-drop it to the center of an empty drive. You can also copy a floppy *.dsk file into LIB: from your PC desktop file explorer.
- Clicking (or double-clicking) on a drive isn't going to show the disk contents. You will have to use CP/M and your console, a la 1976, to discover what is on the disk.
- Each disk emulates a single-sided 8" IBM disk, which contains about 250K of information.

I:DSK is your 4M hard drive. Access its contents through CP/M.

MAN: contains pdf manuals for the IMSAI, CP/M, and programming. There is a LOT of not-easy-to-digest information there! Click on a manual to open it. Access is slow.

PTR: is a very cool paper tape reader. Open TAPE: and drag-drop a tape onto the reader. Click the plug icon to enable the connection, then click 'Read' to read the tape. You will learn how to use it later.

SYS: is a detailed summary of the current system configuration. The gear icon is useful for changing some of the system settings. Don't use it unless you know what you're doing.

LPT: is the emulated line printer. From here you can send program output to your Windows printer. To copy output from TTY to LPT, press <ctrl>P. Make sure the LPT window is open first! Type <ctrl>P again to stop. To copy a CP/M file to LPT, type "PIP LPT:=<filename>".

Startup Configuration

There is a lot to discover before starting down the Assembly Language rabbit hole.

There is a Github repository for the IMSAI 8080. The repository contains the files and instructions for updating firmware. Incidentally, this firmware determines which machine you are emulating. You can change your IMSAI 8080 machine to a Cromenco Z-1 machine, and vice versa, by selecting the alternate firmware file.

The IMSAI has several different boot configurations, including the following:

- Which CPU should be emulated, the 8080 or the Z80?
- What CPU speed does it run at?
- How should it connect to WiFi (access point or local net station)?
- How should the 64K address space be allocated (RAM vs ROM)?
- What software should be loaded (CP/M? Basic? Monitor? Nothing?)
- Should the software start automatically, or wait to be started?
- Should undocumented CPU instructions be allowed?

These settings are kept on the ESP32 in non-volatile storage "NVS" as a 32-bit value. Only the lower 16 bits are used.

To see your current setting, observe the boot sequence from your terminal emulator when you reconnect power via the USB cable. Look for the line "nvs: settings = 0000xxxx", where xxxx are the hexadecimal value of the boot configuration. The SYS: device on the web interface also displays startup settings.

Changing the startup configuration

The procedure is outlined on the High Nibble website (Configuration) and on this video. Briefly,

1. Turn front panel power switch off: <Pwr Off>
2. Hold <Examine> switch in up position
3. While holding switch up, press a Reset Button on ESP32. There are two reset buttons available: one on the ESP module itself, on the rear of the device, and another through a case ventilation hole on the right side of the device. This second switch is best accessible with a popsicle stick applied through the top ventilation hole, angling the reset-button end of the stick towards you.
4. Wait a second, then release the <Examine> switch.

If you do everything correctly, you will see running lights on the Int/Run/Wait/Hold LEDs.

5. Key in the new configuration settings, using the 16 front panel address switches. See below for more information on these switch settings.
6. Raise the <Deposit> switch. The value keyed in will be echoed to the address LEDs.
7. Press the reset button again.
8. To start the emulation, Turn on the front panel switch: <Pwr On>

Startup Configuration Details

The 16 address switches on the front panel are used to key in the 16 startup configuration bits, with switch up=1 (option enabled) and switch down=0 (option disabled). The default configuration is 0000h, where all bits are 0:

Default Startup Configuration: **0000h** (Memory Map 0, 8080 at 2 MHz, WiFi AP mode):

b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
Auto-Run	Reserved	Reserved	Reserved	Banked ROM	Memory Map	Memory Map	Memory Map	Unlim speed	4/2 Mhz	No undoc codes	Z80 CPU	WiF-STA	Log level	Log level	POST
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

A full description is beyond the scope of this introduction. Here are a few useful combinations. These combinations all emulate a 4 MHz Z80 with WiFi in local network (STA) mode.

Code	Description/Use
0058	Front-Panel Switches – no bootloader
0158	Boot to CP/M or IMSAI monitor
0258	Boot to CP/M with CRT (VIO) support
0358	Boot to the Memon/80 monitor
0458	Boot to XYBasic

The code is a 16-bit hexadecimal value. Each digit in the code corresponds to four bits. As shown above, bits15-12 correspond to the left-most digit; bits11-8 to the 2nd digit, and so on. For example, Code 0158 would be entered as follows, flipping each address switch up for 1 and down for 0:

0				**1**				**5**				**8**			
0	0	0	0	0	0	0	1	0	1	0	1	1	0	0	0

To enable auto-run (no need to press <run> on boot), change the first digit from 0 to 8. For example, the startup configuration 8158h will boot directly to CP/M, assuming a CP/M system disk is in drive A. If not, the system will boot to the IMSAI monitor program.

CP/M Basics

If you are just getting into (or back into) retrocomputing, CP/M is a good place to start. There are plenty of documentation and online tutorials for CP/M 2.2. Go check them out!

To auto-boot CP/M on your machine, set your NWM configuration settings to 8158h or 8958h using the above instructions. And mount the disk named “cpm22” (or similar) on drive A.

CP/M looks and feels like MSDOS, but there’s enough difference to trip you up.

- DIR A: Display list of files on drive A: Use wildcards for file matching.
- ERA <file> Erase the file. Wildcards are OK.
- PIP <newf>=<oldf>[optns] Copy command. To copy to another drive: PIP H:=A:FILE.TXT
- REN <newfile>=<oldfile> Rename the file.
- USER <n>, n=0-15 Switch to user area of disk drive, with its own directory.
- TYPE <filename> Echo contents of a file to console. Use <ctrl>S to pause.
- HELP <command> Get help with CP/M commands.

Note: PIP and STAT have *lots* of options. Here are a few useful ones:

- STAT DSK: to see the disk drive capacity and characteristics

- STAT <filename> to see the file size, in records and Kbytes.
- PIP <dest>=<source> copy the file named <source> to <dest>

CP/M Text Editors

I don't fancy text editing on the IMSAI 8080. Years ago, I used "vi" and loved it. But that is ancient history. I find it much easier to create and edit files on my PC, transferring them to the IMSAI as needed.

The standard CP/M text editor is a line editor called ED. It is terrible (my opinion only, mind you). Unless you are going for that authentic 1976 experience, you have better choices.

One step up from ED is Word-Master. It's on the CP/M disk. It uses WordStar-like <Ctrl> commands for cursor control. So, if you like those, you'll feel right at home.

If you are not a WordStar fan, try TE by Miguel Garcia. The ANSI version works great.

Amazingly, the visual text editor of choice for some is… Turbo Pascal! You can think of it as a fast and nimble text editor – with a Pascal compiler thrown in for free. A copy of the disk image is on my GitHub page. To use it, run Turbo.com, press 'E' for Edit, and enter the name of the file you wish to edit. The arrow keys will move you within the file. <Ctrl>V will toggle insert/overwrite mode. Press <Ctrl>K-D to exit the editor. Press 'Q' to quit.

CP/M Disks

If you are like me, you watched CP/M 2.2 boot, then entered DIR to see a directory listing. And you noticed drives A: - D: and I: Did you look at what was on those other drives?

Drives A through D emulate a standard 8" IBM single density disk, which is organized in 77 tracks, numbered 0-76. Each track has 28 sectors, and each sector contains 128 bytes, giving a total disk size of 77*26*128/1024 = 250K bytes. The first two tracks are reserved for system use, leaving about 243K of usable space. Drive I: is organized as a 4MB hard disk with 4080K of usable space.

Type "STAT" at the CP/M prompt to see how much free space is available on each drive/slice currently in use. On my system:

```
C>stat
A: R/W, Space: 11k
B: R/W, Space: 23k
C: R/W, Space: 149k
D: R/W, Space: 72k
I: R/W, Space: 3662k
```

The web interface allows you to eject and mount disks. Empty drives have a small red sub-icon in the drive's lower right corner; mounted drives have an "eject disk" sub-icon displayed. You can even add disk images to LIB: by drag-dropping them from Windows. To copy a disk image from the IMSAI to Windows, drag the disk image to the download icon in the top right corner of the LIB: window.

CP/M subdirectories

Sorry, there aren't any. Each drive is limited to a single directory. You cannot create subdirectories in CP/M, such as "A:\GAMES\ZORK.COM". For organizational purposes, however, each drive can be divided into 16 different user areas. From the CP/M 2.2 manual,

> The USER command allows maintenance of separate files in the same directory. In the syntax line, n is an integer value in the range 0 to 15. On cold start, the operator is automatically logged into user area number 0, which is compatible with standard CP/M 1 directories. You can issue the USER command at any time to move to another logical area within the same directory. Drives that are logged-in while addressing one user number are automatically active when the operator moves to another. A user number is simply a prefix that accesses particular directory entries on the active disks. The active user number is maintained until changed by a subsequent USER command, or until a cold start when user 0 is again assumed.

So, type 'User 1' and you are presented with a new, empty directory! (Technically, you are seeing the same directory, but viewing only files associated with User 1.) You now have access only to files in the User 1 area. Switch back to 'User 0' and the directory reappears. Hint: If at some point your directory seems wrong or suspiciously empty, try switching to User 0.

Typing DIR at the prompt shows you only User 0 files. Don't assume that you are seeing all the files on that drive.

Moving files between Windows and CP/M

Can we drag/drop individual Windows files to a CP/M disk? Nope, that's not possible in the current web interface. Let's do a file transfer the old-fashioned way, either by modem (XMODEM) or paper tape.

<u>Transferring a file to CP/M via XMODEM:</u>

1. Close the web TTY: socket by clicking the icon, turning it from green to red.
2. Using your terminal emulator, establish a serial connection with the IMSAI8080 at 115200 baud, 8/N/1, and software flow control.
3. Load the comms disk from the LIB: into drive B: and go to drive B:

4. From the terminal, type "**XMODEM <filename > /R /X0**" The /R option puts XMODEM into receive mode, and /X0 specifies transfer over the console port.
5. From the terminal emulator, send the windows file via XMODEM. On Tera Term, this is done from the file menu: File > Transfer > XMODEM > Send.
6. If successful, CP/M's XMODEM will report the number of blocks received.

Transferring a text (non-binary) file to CP/M via Tape:

1. Open the paper tape reader device, PTR:
2. PTR opens with its socket closed (red). Click the socket icon to open (green).
3. Drag/drop the Windows file, panel.obj, onto the paper tape. Acknowledge the transfer.
4. At the CP/M prompt, enter "**PIP <filename>=PTR:".** Cursor control on TTY will be suspended.
5. On the PTR, press the "read" button. Wait until the reader is finished.
6. On the PTR, press the "EOF" button. Cursor control on TTY should resume.

Using XY Basic under CP/M

XYBasic can be found on a disk in the LIB: Mount it on an unused drive and type XYCPMFP to run it. You can find the manual at nesssoftware.com. Here are a few commands to get you started.

To:	Type:
Start BASIC	XYCPMFP <enter>
Exit BASIC and return to CP/M	<ctrl>B
Renumber a program, starting at 100	RENUM 100
Load a program from disk	LOAD "<filename>"
Save the current program to disk	SAVE "<filename>" Do not use a file extension; the extension ".XYB" will be added automatically
Save program as an ASCII file	SAVE "<filename",A Do not use a file extension, the extension ".BAS" will be added automatically.
View files (from within BASIC):	DIR
Erase current program & start anew	NEW
List the current program	LIST
Edit a program line	EDIT <linenumber> While in edit, press <space> to advance forward through line, press <esc> to leave, press <i> to insert a character, press <d> to delete a character, press <return> to quit editing.
Terminate a running program	<ctrl>C
Continue a running program	CONT

Here is a quick demo program to get started:

```
10 FOR I = 1 TO 10
20 PRINT TAB(I) "Hello"
30 NEXT
```

Enter the program via keyboard, or copy the listing:

1. Select the program text & copy to Windows clipboard in the usual manner (<ctrl>C).
2. Paste the text into Tera Term using <alt>V. For longer listings, use Tera Term File > Send File and choose a file to send. The entire file will be copied into XYBasic. If you get errors when transmitting files, add a transmit delay of a few mS/char.
3. The listing can be saved to disk from within Basic, as above.

Type 'RUN' to run the program. Type <ctrl>B to exit BASIC and return to CP/M.

A BASIC program: fun with ANSI Control Codes

I've avoided BASIC for decades. But now it's time to play and add a little color to our monochromatic ANSI terminal. The terminal's foreground and background colors can be controlled by sending ANSI escape codes. In general, these codes consist of the escape character + left bracket "[" + the remaining code. Some examples:

Code	Description
<esc>[3#m	Set foreground color to # (0..7). "<esc>[33m" is yellow text.
<esc>[9#m	Set bright foreground color to # (0..7)
<esc>[4#m	Set background color to # (0..7). "<esc>[41m" is red background
<esc>[10#m	Set bright background color to # (0..7)
<esc>[38;5;#m	Set foreground color to # (0..255)
<esc>[48;5;#m	Set background color to # (0..255)
<esc>[38;2;r;g;bm	Set foreground color to RGB
<esc>[48;2;r;g;bm	Set background color to RGB
<esc>[2J	Clear the screen with background color
<esc>[H	Put cursor at 0,0
<esc>[r;cH	Put cursor at row, column

The basic colors are 0=black, 1=red, 2=green, 3=yellow, 4=blue, 5=purple, 6=cyan, and 7=white. In BASIC, the escape character is sent by "PRINT CHR$(27);" The semicolon is required to stop a carriage return from being sent. Start BASIC, then enter a few PRINT statements using these codes. For example, PRINT CHR$(27);"[44m" will change the background color to blue. What does PRINT CHR$(27);"[2J" do?

FP.BAS is a small BASIC program that reads the "programmed input" front panel switches and displays their value using ANSI control codes. Try it!

To print a basic program (or any CP/M text file, for that matter), use PIP to copy the file to the line printer LPT:

PIP LPT:=<filename>

Then, go to the web interface and open LPT. View the listing. Click on the printer icon to print via Windows.

Devices and I/O ports

The IMSAI 8080 communicates with external devices (keyboard, display, printer, etc) over its Input/Output ports. The 8080/Z80 CPU supports up to 256 ports. The CPU executes OUT and IN instructions over those ports to transmit & receive data. Hardware interfaces often require for than one I/O port for proper operation. For example, a UART on a Serial I/O board requires two ports: one for data and another for control. Bear in mind that the IMSAI 8080esp only emulates the original hardware.

The system emulates two SIO cards for four (4) serial ports total. Each serial port is controlled with two Z80 I/O ports. In addition, there is an emulated Line Printer which uses one Z80 I/O port:

I/O	Emulation	*CP/M Devices*	*Default* Input use	*Default* Output use
02h	SIO1.port A	TTY:	Console data	Console Display
03h	"		Keyboard status	Port Configuration
04h	SIO1.port B	CRT: UR2: UP2:	VIO keyboard	CRT display
05h	"		VIO keyboard status	Port Configuration
22h	SIO2.port A	UL1: PTP: PTR:	Web Tape Reader	Web Tape Punch
23h	"		Tape Reader Status	Port Configuration
24h	SIO2.port B	UC1: UP1: UR1:	Virtual Hayes Modem	Virtual Hayes Modem
25h	"		Modem status	Port Configuration
F6h	PIO	LPT:	Line printer status	Line Printer

In the default configuration, if the CPU issues an OUT instruction to port 02h, data is sent to the TTY display. If we sent the same data to CPU port 22h, it would go to the paper tape punch.

Device	Description
UART0	1st physical RS-232 port (or USB port)
UART1	2nd physical RS-232 port
WEBTTY	TTY window on Desktop UI
VIOKBD	CRT window keyboard on Desktop UI
MODEM	WiFi-based Hayes Modem
WEBPTR	PTR: device on Desktop UI
S132TTY	VT100 TTY for S132 board
S132VIO	IMSAI CRT for S132 board

But not always! We choose the devices connected to the four ports. There are currently 8 available devices, as shown in the table. The four serial ports are assigned to *one or more* of the 8 devices by 4 lines in the systems boot.conf file. You can edit this file directly from the SD card, or via SYS: on the desktop UI (click the gear icon and select boot.conf). The default assignments are:

```
# SIO-2 default port mappings (HAL)
SIO1.portA.device=WEBTTY,UART0
SIO1.portB.device=VIOKBD
SIO2.portA.device=WEBPTR,UART1
SIO2.portB.device=MODEM
```

Notice that SIO1.portA is first mapped to the TTY window on the desktop interface, as discussed above. If the device is not available, however, it will be connected to UART0. Similarly, SIO2.portA (CPU ports 22h/23h) is first mapped to the PTR device on the desktop interface. If PTR is not available, however, it will be connected to UART1, which is the 2nd RS-232 port on the back of the IMSAI.

The virtual devices, including the desktop TTY, CRT, and PTR, are connected/disconnected by clicking on their "plug" icons. Green is connected and red is disconnected. SIO1.port A (I/O ports 2&3) is first mapped to the desktop TTY and, if that is disconnected, is mapped to the first RS-232 port (or USB port). Clicking the plug icon will toggle control between the two.

Can we configure a port to be connected to both devices at the same time? Yes, add a plus sign like this: `SIO2.portA.device=WEBPTR+,UART1`

Using the RS-232 serial ports

There are two physical serial ports on the back of the IMSAI 8080, labelled RS232-1 and RS232-2. They are driven by a single MAXIM232 chip, and electrically connected to the ESP32 by means via two headers labelled "patch" and "comms".

The physical RS-232 ports are not connected until you add the appropriate jumpers. The required jumpers are described here: RS-232 configuration.

RS232-1 and the USB port on the ESP32 share the same data pins and are mutually exclusive. If you are using the USB cable for communication you cannot use RS232-1, and vice versa. I connect to my PC with the USB cable, so communicating over USB is easiest. It does not require any additional cabling. As noted above, use the virtual TTY plug icon to toggle the TTY between the desktop UI and Tera Term/Putty/etc.

To establish a connection from RS232-2 to your PC, do the following:

1. Install 4 jumpers on the Patch and Comms headers to connect UART1 to RS232-2.
2. Connect a **null-modem cable** from the RS232-2 port to an RS232-USB adapter.
3. Connect the USB side of the adapter to an open USB port on your PC.
4. Open a second terminal session in Tera Term: File > New Connection.
5. Go to Setup > Serial Port and set it to115200 baud, 8N1. No flow control.
6. Consider changing the font color to distinguish the sessions: Setup > Window.

This new terminal session is connected to SIO2.portA (ports 22h,23h), which is the paper tape device under CP/M. Try sending a text file to it like this: PIP PTP:=BIOS.ASM.

You can also receive files. Try PIP TEST.TXT=PTR:, then enter text from the second terminal. For best results, turn on local echo and set newlines to CRLF. Type a <ctrl>Z to end the file. Now, enter TYPE TEST.TXT from the first terminal to see the file.

Notes on using the Paper Tape Device:

- It is disconnected at first (red). Click the plug icon to connect it (green).
- The tape reader offers a way to transfer files to/from Windows. To transfer a Windows (ASCII-only) file to a CP/M disk:
    - Drag/drop Windows file into the tape library (TAPE:)
    - Drag/drop file from the tape library into the paper tape reader
    - "PIP <filename>=PTR:". Cursor control on TTY will be suspended.
    - On the PTR, press the "read" button. Wait until the reader is finished.
    - On the PTR, press the "EOF" button. Cursor control on TTY should resume.
    - <filename> should now appear in the disk directory
- To transfer a CP/M (ASCII) file to Windows:
    - On the PTR, press "clear" to remove any existing tape
    - Enter "PIP PTP:=<filename>. Wait until the punch is finished.
    - On the PTR, press "Download". The file "tape.bin" will appear in the Windows downloads folder
    - Rename the file back to its original name in CP/M
- To punch a tape from a CP/M file:
    - On the PTR, press "clear" to remove any existing tape
    - enter "PIP PTP:=<filename>. Wait until the punch is finished.
- To read a paper tape into a CP/M file:
    - Load the file onto the paper tape.
    - "PIP <filename>=PTR:". Cursor control on TTY will be suspended.
    - On the PTR, press the "read" button. Wait until the reader is finished.
    - On the PTR, press the "EOF" button. Cursor control on TTY should resume.
- Using the tape reader under the Memon/80 monitor:
    - Use Command TE to see the contents of a tape:
        - Load the tape from TAPE: to PTR:
        - On the console, type TE<enter>
        - On the PTR, press "Read".
        - The tape contents will display on the TTY console

        - Press <ctrl>C on the console to exit
    - Command HD <addr><count> will transfer memory contents AS AN INTEL HEX FILE
    - Command HL will transfer an INTEL HEX FILE on tape to memory
        - Load the tape onto the paper tape device
        - On the console, enter the command HL<enter>. The Intel HEX file should already contain the address where it should be loaded.
        - On the paper tape, press "Read"
        - A "." Will appear on the console for each record read. When the tape is finished, the console will show the number of records read.

Using the modem & TELNET:

You can't beat Telnet for that authentic 1980's computer experience. Telnet was originally developed in 1969 for ARPANET and is still in use today. The IMSAI 8080 includes an emulated Hayes Modem, which can connect to real telnet servers over an internet IP connection. There is no desktop UI for the modem (yet), so we use a terminal emulation program called Kermit for communicating with it. Here is the procedure:

1. Load the comms disk into B: drive and start Kermit: B:Kermit
2. At the Kermit-80 prompt, type "SET PORT UC1". This is the Hayes modem port.
3. Type: "CONNECT" to start communication on the modem port. From this point forward, make sure that the <caps lock> key is on. The modem only accepts upper-case commands.
4. Type: AT<return>. If you get an "OK" response, the modem is connected.
5. Type: ATDTELEHACK.COM There is no space between the AT command for Dial and the telnet server, which is telehack.com.
6. Enjoy! Telehack is a wonderful place to get lost. Read about it here: Telehack manual. Try commands like FINGER, JOKE, WEATHER, or TIME. I recommend avoiding graphic-intensive commands like starwars and aquarium: they run very slowly over the simulated dialup modem.
7. Type :+++ to return the modem back to command mode
8. Type: ATH to hang up (disconnect from the telnet site)
9. Type: <ctrl>\C to return to Kermit control
10. Type: EXIT to leave Kermit and return to the CP/M prompt.

PART 2: Assembly Language Programming

Let's start with programming a bare-bones IMSAI 8080, using only the front panel switches and LEDs. In other words, there is no keyboard, no video display, and no system firmware. It is just us and the front panel. The workflow is

1. Create the assembly language program on a PC.
2. Enter the program on the IMSAI 8080 using the front panel switches.
3. Execute the program.

Let's be honest: entering byte after byte via toggle switches is tedious and error prone. And it requires mentally converting each hexadecimal value into binary.

The sample program I chose is short and sweet, only 8 bytes in length. It reads the front panel input "programmed input" switches and copies their values to the "programmed output" LEDs. You may download the following "panel.asm" from my GitHub page.

```
.ORG 100H
START:  IN      A,(0FFH)     ; read front panel switches
        CPL                  ; invert bits in register A
        OUT     (0FFH),A     ; Put it on the output LEDs
        JP      START
.END
```

The first line ".ORG 100H" tells the assembler where to put the program. A natural place to put it is at the start of memory, 0000h. The first 256 byte, or "page" of RAM is used by many systems as a place to store pointers to special routines, so we will avoid that and start at 0100h. This is also the default location to load CP/M programs.

The next line, using the IN instruction, loads data from port FF into Register A. If we output this value to the front panel LEDs, they will appear inverted. This was done on the original system to save a few pennies on hardware. So, we must use an instruction to invert the value, which is CPL. The following line "OUT (0FFh),A" copies the inverted value in A to output port FF, which is the location of our LED display. Finally, JP (jump) loops back to the start of the program.

Assemblers

There are many assembler choices. I use Telemark Assembler, or TASM. If you want to develop on the IMSAI 8080, there is an 8080-assembler bundled with CP/M. I decided against using it since I prefer coding and compiling Z80 code on my PC.

TASM installation

1. Copy the assembler files to a convenient spot: "c:\tasm".
2. Edit the environment variables to include this path.
3. Add the variable TASMTABS and set its value to c:\tasm
4. Open a command prompt and type tasm. Make sure it works. Check out the compiler options.
5. *RTFM*. It is online.

TASM notes for those who don't read the manual:

- Invoke the assembler with "tasm -80 <options> <filename>".
- The object file format is Intel Hex by default (-g0).
- To produce a binary object file, use option -g3 (same as -b), which also forces option -c (contiguous block output). This option is useful for code destined for ROM: all locations will have a known value. Often used with -f, below.
- Fill option (-fFF) fills unused memory with 0FFH. Similarly, -f00 fills with zeroes.
- The Intel Hex file is 24 bytes per line by default. Change it to 16 bytes with -o10 (or 32 bytes with -o20).
- Numbers are decimal by default.
- Enter hex numbers like this: $E3F4, 0E3F4H (number cannot begin with alpha digit).
- Enter binary numbers like this: 01101B, %01101
- String constants may be used with DB, BYTE directives & can include \r, \n,\t, \\, \"
- Examples of Byte and Word directives. (.BYTE, .byte, and .db are equivalent)

```
.db $0F
.dw 0FFHE0
.db 'a'
.db "Hello", 13, 10, "World",0
.byte "Hello World\r\n"
```

- Labels are case sensitive, up to 32 characters in length.
- Symbols can be defined at compile time with -d option. For example, -dINTMODE1
- IFDEF/ELSE example:

```
#IFDEF INTMODE1
<code here>
#ELSE
<code here>
#ENDIF
```

- Code usually starts with the .ORG directive and ends with the .END directive:

```
.ORG $0100      ; start program at $0100
<code here>
.END             ; follows the last line of code/data
```

- You can add additional .ORG statements in the program listing. For example, ".ORG $+8" will reserve 8 bytes of space without assigning any values.

Your first assembler program: panel.asm

Let's create a super-simple, "Hello, World" style assembler program which reads eight front panel switches and copies their status to corresponding front panel LEDs. Below is the code for panel.asm. You may type or copy/paste it into your preferred Windows code editor (I use Notepad++), or download it from my bhall66 GitHub page:

```
        .ORG 100H
START:  IN      A,(0FFH)      ; read front panel switches
        CPL                   ; invert bits in register A
        OUT     (0FFH),A      ; Put it on the output LEDs
        JP      START
        .END
```

The first line locates the program's ORiGin (starting point) at 0100h. This is a very common place to start user applications, since the first 256 bytes of memory from 0000 to 00FFh are reserved in CP/M.

The next four lines make up the program. "IN A,(0FFH)" takes the contents I/O port FF and puts it into the A register. Reading from FF reads the values of the eight "programmed input" switches, one switch per bit. Writing to port FF done by the OUT instruction, places the value of A back onto the "programmed output" front panel LEDs. The LEDs are wired in such a way that a logic 0 state, rather than a logic 1, causes an LED to light. The CPL instruction is not necessary, but it inverts all 8 bits so that the LEDs light when the switch is flipped up, rather than down.

Finally, the END directive informs the compiler that there is no more code or data to compile.

Assemble the program using the TASM assembler using the following DOS command "TASM -80 panel.asm". The "-80" option directs the assembler to use Z80 mnemonics and generate Z80 code.

As an aside, I prefer Z80 mnemonics, and will use them throughout my examples, but 8080 syntax works just as well. The following table shows the panel.asm program using Z80 and 8080 syntax.

panel.asm (Z80 mnemonics)	panelT.asm (8080 mnemonics)

`START: IN A,(0FFH)`
`CPL`
`OUT (0FFH),A`
`JP START`
	`START: IN 0FFH`
	`CMA`
	`OUT 0FFH`
	`JMP START`

If you want to use 8080 mnemonics, use the assembler's -85 option: "TASM -85 panelT.asm". I've also created a file, panelD.asm, formatted for the DRI assembler that comes with CP/M 2.2. The generated code for all three is the same:

```
DB FF 2F D3 FF C3 00 01
```

If TASM reports any errors, review your code for syntax errors. Make sure you have used the correct assembly options.

The assembler will generate two output files, a list file with the .lst extension, and a .obj object file (in Intel Hex format, by default).

Examine the file panel.lst. You should see the following listing:

```
0001   0100             .ORG 100H
0002   0100 DB FF       START:  IN   A,(0FFH)    ; Get a value
0003   0102 2F                   CPL              ; invert bits
0004   0103 D3 FF               OUT (0FFH),A     ; Send it to the output LEDs
0005   0105 C3 00 01            JP   START       ; looping forever
0006   0108             END
tasm: Number of errors = 0
```

This program is 8 bytes in length. The orange box highlights the eight individual bytes that make up the program and are the values that you will need to key into the front panel. The yellow box to the left indicates the address where each byte is located, starting at location 0100h and ending at 0107h. Armed with this knowledge, we are ready to use the program.

What do you think about Z80 assembly language programming? Are you already familiar with the concepts, or do you want to learn more? The classic book for learning more is "Programming the Z80" by Rodnay Zaks. It is a useful reference for the remaining programs in this tutorial. You can find copies of it on the Internet in all the usual places.

Before creating other programs, it is important to know how to transfer programs to the IMSAI. The next section discusses 9 different ways to load software into your IMSAI 8080.

Nine ways to get an assembly language program into the IMSAI 8080

There are many ways to enter programs on the IMSAI 8080. I will demonstrate 9 ways: using the front panel switches, using the IMSAI monitor ROM, using the Memon/80 monitor program, and 6 different ways from within CP/M itself.

Method #1: How to enter a program using only the front panel switches

1. The program is 8 bytes long: DB FF 2F D3 FF C3 00 01.
2. Make sure the startup configuration is using Memory 0 (64K RAM, no ROM). For this example, use configuration 0058h or something similar.
3. Toggle the <reset> switch to begin at address 0000h.
4. Set the address to 0100h by flipping down all address switches except for bit 8, which should be raised, and pressing <examine>.
5. Key in the first byte of the program, which is DB, onto the switches marked "Address-Data" (hexadecimal DB = binary 111011011)
6. Toggle up <Deposit> to save this byte to address 0100h. You should see the LEDs on the Data bus change to the value DB. If not, check your switches and press <Deposit> again.
7. Enter in the second byte, which is FF (all switches up). This time, press <Deposit Next> to deposit it into the next address location, which is 0101h. You should see the address change to 0101h and the data change to FF.
8. Continue entering the data bytes: 2F<deposit next>D3<deposit next> and so on until the last two data bytes: 00<deposit next>01<deposit next>.
9. Now check your work: return to address 0100h (all address switches should be down except bit 8) and press <examine>. You should see D3 on the data bus. Press <Examine-next> seven times, checking each program byte between each press.
10. When you are convinced that the program is correct, return to address 0100h and press <run>.

If successful, the "Programmed output" row of LEDs will mimic the "Programmed input" switches below them. Toggle a few switches. Do the LEDs change accordingly? Now press <stop>. Do the LEDs still change?

If you had any difficulty getting this to work, please check out a video by "ShadowTronBlog" which demonstrates keying a different program.

Method #2: Using the IMSAI IEEE Monitor

The IMSAI monitor is a step up from the front panel. It is instructive to try. But, unless you have a compelling reason to use the original monitor, Memon/80 (described below) is a more forgiving and convenient choice.

A superset of the IMSAI IEEE monitor commands is found in the VIO manual, page III-43. Commands G, A, X, L, and Q in the VIO manual do not work. All commands must be in upper case. Addresses must be given as 4-digit values.

Command	**Description**
Call subroutine	C <addr> Call a subroutine at <addr>, returning control to the monitor on a return instruction
Display Memory	D <start addr> <end addr> Displays contents of memory block from <start> to <end>
Examine Memory	E <addr> Displays contents at address & allows entry of new data: Press <space> to go to next location Press <-> to display previous location Press <enter> to quit Enter two hex digits to modify memory
Move Memory	M <start addr> <end addr> <dest addr> Copy memory block start..end to the destination address
Fill Memory	F <start addr> <end addr> <fill byte> Fill memory block start..end with the fill byte
Search Memory	S <start> <end> <2 byte pattern> Search Memory from start to end for a 2-byte pattern. For example, “S 0000 0200 02DB will search 0000-0200h for the two bytes 02 DB.
Boot from A: drive	B

Read from Disk	R <tt> <ss> <addr> <unit> Example: “R 00 01 0100 2” will read disk #2, track 0, sector 1 to memory address 0100h
Write to Disk	W <tt> <ss> <addr> <unit> Example: “W 00 01 0100 2” will write to disk #2, track 0, sector 1 from memory address 0100h. Dangerous command!!
Read paper tape	H Reads characters “from port 2” in Intel HEX format. Does not seem to work with paper tape device.
Test Memory	T <start addr> <end addr> Tests memory block start..end by writing and reading all 255 bit patterns. Memory contents are NOT destroyed. Return to monitor prompt = no errors found. If T is entered without start or end address, it will search all memory until a bad location (or ROM) is encountered. The first bad memory (or ROM) encountered will be displayed as: <addr> <byte found> <byte expected>
Execute/Jump	J <addr>

To enter the IMSAI IEEE Monitor:

1. Ensure that Drive A: contains no disk. Eject the disk if one is present.
2. Enter startup configuration 0158h.
3. From the front panel, when you press <Pwr On>, the address bus should display D800h. If it does not, enter D800 on the address switches<examine>
4. Press <run> to start the monitor.
5. Go to TTY: (make sure socket icon is green) and press <space> a few times.
6. You should get the IMSAI MONITOR startup message and a “?” prompt

To enter a program, type “E 0100” at the monitor prompt. The contents of address 0100h will be displayed and it will wait for your edit. Enter the same program as before (DB FF 2F D3 FF C3 00 01), using a space after each byte value. When you are finished, press <enter>. Entering data with the monitor is much faster than using the front panel switches!

To confirm the data entry, enter “D 0100 0107<enter>”. This will show you the 8 bytes starting at 0100h. You should see the following: 0100 DB FF 2F D3 FF C3 00 01

When everything looks correct, run the program by entering “J 0100”. The program will run. You can <stop>, <start>, and <single step> through it as before.

To return to the monitor, <stop>, set the address to D800 <examine> and <run>. Press <space> on the console and the MONITOR prompt should reappear.

Method #3: Using the Memon/80 Monitor

The monitor manual is located on your SD card. It can be downloaded directly from the web portal of your IMSAI 8080. *RTFM*. Memon/80 is better documented and easier to use than the original IMSAI IEEE monitor.

To use it, set the startup configuration of your machine to **0358h** and press <run>. You will be greeted by the “Memon/80 2.6 by M. Eberhard” prompt. There is no need to repeat the manual, so here is an abbreviated list of comands:

Command	Description
DU	DU <addr> <count>. Dumps (displays) memory contents. ^C to stop.
EN	EN <addr> Enter hex data into memory at address. <return> to quit.
FI	FI <addr> <count> <value>. Fill memory with a given value.
SE	SE <addr> <byte>’Hall’ <byte>. Seach memory for a byte/string pattern.
BO	BO <drive>. Boot from floppy drive
EX	EX <addr>. Call subroutine at <addr> & return to monitor with <ret> instr
HD	HD <addr> <count>. Dumps Intel HEX file from memory to Tape Punch.
HL	HL. Loads Intel HEX file from Tape Reader into memory.
TE	Terminal Exchange. Keyboard goes to Tape Punch, Tape Reader to Console.
IN	IN <port>. Input a byte from port & show it on console
OT	OUT <port> <data>. Outputs <data> to <port>
?	Help, an abbreviated list of available commands.

To enter the program, enter “EN 100” <enter>. The address 0100h will be displayed and wait for your input. Enter the program data, with a space between each byte: DB FF 2F D3 FF C3 00 01 <enter>.

This data does not have to be entered manually. You could load it from a Windows text file instead. To do this, establish a console connection with a terminal emulator. Then, at the 0100: prompt, send an ASCII text file that contains two-digit hexadecimal values separated by spaces. In Tera Term, this is done from the file menu: File > Send File...

Alternatively, you could drag the program file in HEX format (panel.obj) from Windows to your paper tape device, enter HL<enter> from the monitor, and <read> the tape.

To confirm the data entry, enter "DU 100 8<enter>". This will show you the 8 bytes starting at address 0100h. You should see the following: 0100: DB FF 2F D3 FF C3 00 01.

When everything looks correct, run the program by entering "EX 100" <enter>. The program will run. You can <stop>, <start>, and <single step> through it as before.

To return to the monitor, <stop>, set the address switches to F800h <examine> and <run>. The MONITOR prompt should reappear.

Booting CP/M from this monitor is easy. Mount the CP/M disk in Drive A:, then enter "BO" to boot to CP/M. You can return the monitor by entering the following on the front panel: <stop>, set address switches to F800h, <examine>, <run>

Method #4 (CP/M): Using DDT

Boot CP/M in your usual manner to see the startup message "54K CP/M VERS 2.2 Bxx" and drive A prompt. Here are four different ways to enter your assembly language program from within CP/M:

1. Enter manually via DDT
2. Enter manually via SID
3. Transfer from Windows via the virtual Paper Tape device
4. Transfer from Windows via XMODEM.

I will describe each of these methods below. Try them to decide which method you prefer.

To use DDT, type DDT at the A> prompt. The prompt will change to "?". Here are a few DDT commands:

Command	Example usage
D (display)	D100: Displays 192 bytes of memory, starting at 0100h
F (fill)	F100,140,01: Fills memory block <0100..0140h> with byte 01h
G (go)	G100: execute program starting at 0100h.
L (list)	L100: disassemble and list program starting at 100h
M (move)	M100,110,A000,10: moves <0100-0110h> to A000h
S (set)	S100: examine 0100h and alter its contents. Type '.' to stop

All numerical input is in hexadecimal. Once in DDT, type "S100" to see/modify memory at 0100h. Enter the first byte of the program "DB"<enter>. Continue for the next 7 bytes: FF

2F D3 FF C3 00 01, hitting the return key between each byte. Lower case values are converted to upper case. Press '.' to stop data entry.

Check your work by displaying memory contents with "D100". Try L100 to see a listing of your program, noting that it is displayed using 8080 mnemonics.

You could enter G100 to start the program, but the program is erased when you return to CP/M. There is a better way! Enter <ctrl>C to leave DDT, then issue the following CP/M command:

save 1 panel.com

This will save 1 page (256 bytes) of memory starting at 0100h to a CP/M disk file named PANEL.COM. Perfect! Once the program starts, you can return to CP/M from the front panel by pressing <reset>.

Method #5 (CP/M): Using SID

The successor to DDT was SID, the symbolic instruction debugger. Type SID at the CP/M prompt to start SID. A "#" prompt will appear. Again, here are a few of its commands:

Command	Example usage
C (call)	C100: Call a routine at 0100h, returning to SID on a Return instruction.
D (display)	D100: Displays 192 bytes of memory, starting at 0100h
F (fill)	F100,140,01: Fills memory block <0100..0140h> with byte 01h
G (go)	G100: execute program starting at 0100h.
L (list)	L100: disassemble and list program starting at 100h
M (move)	M100,110,A000,10: moves <0100-0110h> to A000h
S (set)	S100: examine 0100h and alter its contents. Type '.' to stop
W (write)	W<filename>,100,140: write <100h-110h> to file <filename>

The DDT and SID commands shown above are nearly the same. Notice the W command allows us to create a file from within SID, something that DDT cannot do. So, to create the executable panel.com file, you have two choices:

1. Leave SID and enter the CP/M command "SAVE 1 PANEL.COM", like above; or
2. Within SID, enter "WPANEL.COM,100,110".

Method #6 (CP/M): Using the virtual paper tape device

There is no need to manually enter your program byte by byte. Here's how to load your assembly language object file (INTEL HEX format) to a CP/M using the virtual paper tape device:

1. Open the paper tape reader device, PTR:
2. PTR opens with its socket closed (red). Click the socket icon to open (green).
3. Drag/drop the Windows file, panel.obj, onto the paper tape. Acknowledge the transfer.
4. At the CP/M prompt, enter "PIP PANEL.HEX=PTR:". Cursor control on TTY will be suspended. If you want to see data echoed to the console as it is read, add the E option like this: "PIP PANEL.HEX=PTR:[E]"
5. On the PTR, press the "read" button. Wait until the reader is finished.
6. On the PTR, press the "EOF" button. Cursor control on TTY should resume.
7. Type "LOAD PANEL.HEX" to create an executable PANEL.COM file

Method #7 (CP/M): Using XMODEM or other file transfer program

We can eschew the web interface and transfer files over a serial connection with XMODEM. Use a Windows terminal emulator that supports XMODEM, such as Tera Term.

1. Close the web TTY: socket by clicking the icon, turning it to red.
2. Using your terminal emulator, establish a serial connection with the IMSAI8080 at 115200 baud, 8/N/1, and software flow control.
3. Load the comms disk from the LIB: into drive B: and go to drive B:
4. From the terminal, type "**XMODEM <filename.HEX> /R /X0**" The /R option puts XMODEM into receive mode, and /X0 specifies transfer over the console port. Use the filename extension HEX so that it can easily be converted into a COM file.
5. From the terminal emulator, send the windows file via XMODEM. On Tera Term, this is done from the file menu: File > Transfer > XMODEM > Send.
6. If successful, CP/M's XMODEM will report the number of blocks received.
7. Type the CP/M command "LOAD <filename>.HEX" This creates a .COM file of the same name that is ready to run.

XMODEM is not the only file transfer program available to you. On the comms disk you will find two others, QTerm and Kermit.

Method #8 (CP/M): send file to console

Here is a great method for transferring small files from Tera Term to CP/M.

1. Set the console to your Tera Term window, if it isn't already.
2. In CP/M, type "PIP <filename>.HEX=CON:
3. Now, from Tera Term, choose File> Send File… and choose the file to send.
4. Press <crtl>Z to end the file transfer.
5. In CP/M, type "Load <filename> to create an executable .COM file

This works best for small files. For larger files, you may need to slow the transfer by adding 1 or 2 mS delay to each character (Tera Term > Setup > Serial Port). You may also need to temporarily change your transmit newline setting from CR to LF (Tera Term > Setup > Terminal > New Line Transmit = LF).

Method #9: forget the PC! Create, edit, and assemble in CP/M.

Why bother transferring programs from the PC when you can do everything on the IMSAI? It's a more authentic experience. It can be quite enjoyable, provided you use a good assembler and editor. Here is the workflow for developing an 8080 assembly language program on the IMSAI.

1. Create and edit the assembly program using your favorite text editor. For small files I use TE: "TE TEST.ASM"
2. Run the 8080 assembler: Type "ASM TEST". Do not enter the .ASM extension.
3. The assembler will create two files:
    a. TEST.HEX, which is the object file in Intel Hex format
    b. TEST.PRN, which is the file listing.
4. Optionally, view the files "Type TEST.PRN" to see the output.
5. Optionally, load the files into the DDT or SIM debugger: "SID TEST"
6. Create the executable. "LOAD TEST" will load TEST.HEX and create an executable file named TEST.COM
7. At the CP/M prompt, type "Test" to run TEST.COM.
8. Toggle the front panel <reset> switch to return control to CP/M.

If you prefer Z80 mnemonics, you'll need to install a Z80 assembler on your CP/M system. I like Z80ASM by SLR Systems. This Z80 assembler creates .COM files directly.

Your 2nd assembler program: UART.ASM

In the examples below, I assume that you compile the program on your PC using TASM or a similar assembler, and you use one of the options above for transferring the program to the IMSAI.

Bare-metal means interfacing directly with machine hardware (that is, if we were using an actual IMSAI 8080 and not an emulation). There is nothing between your program and the I/O ports. Your code is smaller and runs faster.

The first program, panel.asm, was a bare-metal program. Let's do another bare-metal program which reads and writes to the console. Serial I/O on an IMSAI 8080 is handled by an Intel 8251 USART chip. This chip supports two serial channels, A and B. Each channel requires two I/O ports, one for data and the other for control. The SIO hardware is described in Chapter 9 of the IMSAI 8080 manual.

To initialize serial communication, the 8251 expects two bytes on the channel's control port. The first byte sets the mode and the second sets the command.

UART Mode byte: CA							
C				A			
bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
Stop bits		Parity		Data bits		Baud	
1	1	0	0	1	0	1	0

UART Command Byte: 27							
2				7			
Sync Hunt	Reset	RTS=-0	Error Reset	Send break	Rx Enable	DTR=0	Tx Enable
0	0	1	0	0	1	1	1

From page 9-36 of the IMSAI manual, the mode byte (CAh) sets the data format to 2 stop bits, no parity, and 7 data bit:

- b7,b6: 11= 2 stop bits
- b5,b4: 00 = no parity
- b3,b2: 10 = 7 data bits
- b1,b0: 10 = clock/16 baud rate

The command byte 27h configures hardware control and enables data transmission. We want to set RTS/CTS both low and Transmit/Receive to be enabled.

We can check serial communication status by reading from the same control port. The status byte informs us of any communication errors,

UART Status Byte	Bit
DSR input status	7
Break detect	6
Framing Error	5
Overrun Error	4
Parity Error	3
Transmitter Empty	2
Receiver ready	1
Transmitter ready	0

the status of the hardware DSR input line, if a data byte has been received, and if the UART is ready to send. See the Intel 8251 manual for more information.

Channel A is connected to the console/terminal. When read, it looks for keyboard input. Writing to channel A send the data to the teletype TTY. I/O ports for the SIO board are mapped as follows:

I/O port address	UART Function	IN (read port)	OUT (write port)
02h	Chan A, data	Read Kbd data	Send data to TTY
03h	Chan A, control	Read Kbd status	Set mode, cmd
22h	Chan B, data	Read paper tape	Write paper tape
23h	Chan B, control	Read PTR status	Set mode, cmd

Let's use this SIO hardware information to create a program that looks for keyboard input and echoes it to the console/TTY. Refer to file uart.asm on my GitHub page.

```
.ORG    100H
START:  LD   A,0CAh      ;7 BITS, NO PARITY, 2 STOP
        OUT  (3),A       ; UART A - SET MODE
        LD   A,027h      ; ENABLE TX & RX
        OUT  (3),A       ; UART A - SET CONTROL
U1:     IN   A,(3)       ; CHECK STATUS
        AND  2           ; LOOK @ RX READY BIT ONLY
        JP   Z,U1        ; WAIT FOR A CHARACTER
        IN   A,(2)       ; UART A - READ CHAR INTO A REG
        OUT  (2),A       ; UART A - WRITE CHAR TO CONSOLE
        JP   U1          ; WAIT FOR NEXT CHARACTER
.END
```

The first four lines, two LD instructions and two OUT instructions, send the mode byte CAh and 27h to the UART's channel A control register on port 03.

The next 3 lines, highlighted in yellow, wait for a key on the keyboard to be pressed. First , an IN instruction at line U1, reads control port 03 to get the UART status. This value is then logically ANDed with value 00000010 to ignore all status information except for the "receiver ready" status on bit 2. If this bit is negative, the result is 0, causing the following Jump-on-zero to loop back to U1.

When a key is pressed, data is received by the UART, the Rx receive bit is 1, and the looping stops. IN A,2 reads the data into register A and OUT 2,A sends that data to the screen.

Compile it. Try different methods of transferring the object file to CP/M, converting the file to a .COM file, and running it. Toggle the <Reset> front panel switch to return to CP/M. For fun, try modifying the code to double echo each keypress. Can you modify the program to send data to the paper tape device?

As seen above, bare-metal programming requires detailed knowledge of the hardware. It also limits your code to computers which use the same hardware. A computer that uses a Z80 SIO chip for serial communication, for example, would not work with code meant for the Intel 8251.

Your 3rd assembly program: ECHO.ASM

CP/M provides standardized I/O functions, letting us write code that will run on computers with vastly different hardware. Its BIOS/BDOS provides a hardware abstraction layer that shields the programmer from underlying hardware details, both simplifying program development *and* improving portability.

Let's rewrite the last program using CP/M BDOS calls. By using CP/M, this program should run on all CP/M machines, no matter which UART device is used for serial communication. Here is an abbreviated list of CP/M functions related to Basic I/O:

#	Function
0	System Reset
1	Console Input, returns with character in A
2	Console output, call with character in E
3	Auxiliary (RDR) input, returns with character in A
4	Auxiliary (PCH) output, call with character in E
5	Printer output, call with character in E
6	Direct console I/0, Call with E=0FFH to get status. Returns char in A, 0 otherwise.
7	Get IOBYTE.(same value as IOBYTE at addr 0003h)
8	Set IOBYTE, call with byte in E
9	Output "$"-terminated string, pointed to by DE.
10	Buffered console input, buffer addr at DE. First byte of buffer is size.
11	Console status, returns A=0 if no characters are waiting.
12	Return B=H=system type, A=L=CP/M version (v2.2=22h)

The standard way to call CP/M routines is to access them through the BDOS. The function number is loaded into the C register, and a call is placed to 0005h. Consider this code:

```
.ORG    100H
START:  LD   C,6          ; CP/M CONSOLE INPUT
        LD   E,0FFh       ; FF= GET INPUT STATUS
        CALL 5            ; RETURNS STATUS IN A
        OR   A            ; SET FLAGS
        JP   Z,START      ; LOOP UNTIL CHAR READY
        CP   '~'          ; IS CHAR A TILDE "~"?
        RET  Z            ; YES, EXIT TO CP/M
        LD   E,A          ; NO, GET RDY TO ECHO
        LD   C,2          ; CP/M CONSOLE OUTPUT
```

```
        CALL 5              ; ECHO CHAR TO SCREEN
        JP   START          ; AND GET NEXT CHAR
.END
```

The highlighted code in yellow represents the character input loop. BDOS function 6 returns the character in A, if one is available, and 0 otherwise. The instruction "OR A", is a quick, one-byte method to set the zero flag if A=0. The next two lines, starting with the compare CP, look for the termination character "~". The RETurn statement gracefully returns control back to CP/M so that we don't have to reset the system. Finally, BDOS function 2 is used to echo the character on the screen.

Try it! Can you change the termination character to something else?

Writing strings: rocks.asm

Here is a small program that writes a string to the console:

```
START:  LD   C,9            ; BDOS Fn9 = String Output
        LD   DE,STR1        ; POINT TO THE STRING
        CALL 5              ; AND PRINT IT
        RET
STR1:   .DB "This IMSAI 8080 rocks!",13,10,"$"
```

I've omitted the ORG and END directives, but know they are still required. BDOS function 9 is used to output a string. This function requires the DE register pair to contain the string's address.

The last line contains the string itself. ".db" stands for "define byte", meaning that byte data follows. It can be a single byte or a comma-separated list of bytes such as a decimal number (0-255), a character in single quotation marks ('g'), or string of characters in double-quotation marks ("hello"). The last data item, $, marks the end of a CP/M string. Without it, your code will print characters beyond the confines of your program until it encounters a $ - somewhere. Don't forget the $.

Reading strings: read.asm

Here is a program to read the console and store the input in memory for future use.

```
START:  LD   A,80       ;buffer size 80 char
        LD   DE,BUF     ;point to buffer
        LD   (DE),A     ;set maximum length
        LD   C,10       ;bdos 10 = console read
        CALL 5          ;get input from user
        RET             ;return to CP/M

BUF:                    ;buffer structure ----
BUFLEN: .DB 0           ;first byte = max length
STRLEN: .DB 0           ;second byte = actual length
```

```
BUFDAT: .FILL 80,0      ;80 bytes of char data
```

The call to BDOS function #10 gets user input, echoing it to the screen, until the return key is pressed. It requires a buffer, or block of memory, to store the entered character data. The address of this buffer is put into the DE register pair. The highlighted block shows how to declare this buffer. Notice how the first two bytes of the buffer hold special meaning. The first byte must contain the size of the buffer. Our program must set this value before the function is called, and does so by the two instructions "LD A,80" and "LD (DE),A". The second byte is the actual length of the data; that is, how many characters the user typed in.

Compile and create an executable for it in the usual manner. Then run it and enter a string of characters followed by the <return> key. The program ends. What happened to the string you entered? Answer: it went into the memory buffer.

We'll need a debugger to see our string in memory. I recommend SID for this purpose. Type "SID READ.COM". At the SID prompt, type "L100" to list the instructions starting at memory location 0100h. You should see something like the following:

```
0100  MVI  A,50
0102  LXI  D,010C
0105  STAX D
0106  MVI  C,0A
0108  CALL 0005
010B  RET
```

That's the program in memory, albeit with 8080 mnemonics. Type "C100" to execute it. By using SID's call function, control will return to SID when the program ends. Enter a string of characters as before. When you see the "#" SID prompt, type D10C to display memory starting at 010Ch. You should see the text you entered. The first byte should be hex 50 (decimal 80), the buffer size. The second byte should be the length of the string you entered. From SID, type <ctrl>C to return to CP/M.

Putting it together: talk.asm

The last two examples were meant to show how the CP/M functions work. Let's put them together in a longer program that reads and writes strings:

```
START:  LD   DE,STR1
        CALL OutStr       ;"Favorite food?"
        CALL InStr        ;get user response
        CALL CRLF         ;new line

        LD   HL,STRLEN    ;point to str length
        LD   A,(HL)       ;get string length
        OR   A            ;is it zero?
        RET  Z            ;if so, done
```

```
LD   B,0
LD   C,A          ;get str Length in BC
INC  HL           ;HL points to chars
ADD  HL,BC        ;point beyond last char
LD   (HL),'$'     ;Add $ to end of string

LD   DE,CHARS
CALL OutStr       ;display user response
LD   DE,STR2
CALL OutStr       ;" is really tasty"
CALL CRLF         ;skip a line
JP   START        ;loop
```

This program prompts for the user's favorite food and responds that the food is tasty. The two previous programs are incorporated as the routines OutStr and InStr, respectively.

A few instructions must be added to make it all work. Code highlighted in orange loads the string length byte into register A. The OR A instruction sets the zero flag if the length is zero, so that the program can gracefully exit when there is no response. Code in yellow prepares the user input for subsequent display, terminating it with the required dollar symbol. The HL register is used to point to the location where the $ will be placed. The location is calculated as the start of the string (HL + 1) plus the string length (which is in register pair BC). The 16-bit add instruction ADD HL,BC does this addition.

Simple math: dogyears.asm

In the last program, code for string I/O was packaged into routines. It is handy to keep a collection of routines for future use. I put my routines in a file called bhUtils.asm. Now, check out the silly and irreverent program "dogyears.asm", which estimates a dog's biological age as calendar-age x 7.

There is no built-in Z80 instruction for multiply. We can write our own multiply routine (or copy one off the 'net), but for simple multiplication it's easier to use addition and subtraction. The following code takes an age in the HL register and multiplies it by 7:

```
LD    B, H          ; multiply age x 7 as follows:
LD    C, L          ; copy HL to BC
ADD   HL, HL        ; HL x 2
ADD   HL, HL        ; HL x 4
ADD   HL, HL        ; HL x 8
SBC   HL, BC        ; HL x 7 (subtract one, get it?)
CALL  printNum16
```

The first two lines copy HL to BC. Next, use the ADD HL, HL instruction to add HL to itself, thereby multiplying it by 2. Three of these ADD instructions have the effect of multiplying HL by eight (2 x 2 x 2 = 8). Finally, the SBC instruction subtracts one multiple, making the

result HL x 7. You should probably clear the carry before SBC, but we will assume no carry here.

More examples

For a few more programming examples, please check out dump.asm, which is a memory dump program, and mapper.asm, which displays the IMSAI's RAM/ROM in a colorful way. Now have fun writing your own assembly language programs.

Bruce.

Last updated: 24 April 2025