A Z80 CP/M emulator for the SAMD51

Created by Dave Astels

https://learn.adafruit.com/z80-cpm-emulator-for-the-samd51-grand-central

Last updated on 2021-11-15 07:30:34 PM EST
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview</td>
<td>3</td>
</tr>
<tr>
<td><strong>Z80</strong></td>
<td>5</td>
</tr>
<tr>
<td><strong>CP/M</strong></td>
<td>7</td>
</tr>
<tr>
<td>• Builtin Commands</td>
<td>8</td>
</tr>
<tr>
<td>• Standard Programs</td>
<td>9</td>
</tr>
<tr>
<td><strong>RunCPM</strong></td>
<td>10</td>
</tr>
<tr>
<td>• Modifying RunCPM</td>
<td>11</td>
</tr>
<tr>
<td>• Changes to Support the Grand Central</td>
<td>11</td>
</tr>
<tr>
<td>• abstraction_arduino.h</td>
<td>12</td>
</tr>
<tr>
<td>• RunCPM.ino</td>
<td>12</td>
</tr>
<tr>
<td><strong>I/O in Z80 Assembly</strong></td>
<td>14</td>
</tr>
<tr>
<td>• Turning on a LED</td>
<td>15</td>
</tr>
<tr>
<td><strong>Wrap Up</strong></td>
<td>17</td>
</tr>
</tbody>
</table>
Overview

Retro computing is very popular. It's interesting to experience what computing was like in the 80s when 8-bit processors were the standard and 64K of memory was generous.

While you can get a taste of this by using an ATMega328 based board such at the venerable Arduino UNO or a Metro 328, but it's not quite the same as using a 6502 or a Z80.

You can go all the way to the hardware level and get something like the RC2014 Z80 computer kit (https://adafru.it/DJX) for the full experience, or you can stick with an emulator on your desktop machine.

This guide takes you somewhere in-between. It looks at running a Z80 emulator on state of the art microcontroller hardware. Specifically, a Grand Central M4 Express. This board is built around the SAMD51 MCU which has plenty of memory and power to run an emulator in real time. Just as importantly, it has an onboard SD card slot which is needed to house CP/M's virtual disks.

While this guide is written for the Grand Central M4, any SAMD51 board has plenty of power and memory to handle the job. You'll have to add an SD card shield/wing and make corresponding changes to the code for the board definition and SD interface.

We'll start with a brief discussion of the Z80 CPU, what it was and why it was so important in the history of personal computing. Then we'll talk briefly about CP/M, one of the more popular operating systems for the Z80.
The emulator we'll use is RunCPM. It can be found on github (https://adafru.it/DJY). There are a few changes required to make it work on the GrandCentral, mostly just to tell it about the board. We'll take care of those and get it running.

With it running, we'll take a brief look at operating CP/M running on the Z80 that is emulated by the Grand Central.

Finally, we'll come full circle in a way and explore how to use the Grand Central's I/O pins from Z80 assembly language.
The Z80 is an 8-bit CPU* that was extremely popular and common in the late 70s to mid 80s, being significantly more powerful than most of its competitors. It was used in many early hobbyist computers as well as several commercially successful machines (the Sinclair ZX80, 81, and Spectrum; Tandy's TRS-80 line of computers, and others). It was used in various scientific and graphing calculators, many coin-op arcade games (e.g. PacMan, Frogger, Galaxian, Galaga, Dig Dug), and several analog synthesizers (including models from Moog, Oberheim, and Roland). Nintendo's Game Boy, Game Boy Color, Game Boy Advance/SP/Micro used a variation of the Z80. In all, it was one of the key CPUs of the day.

While Intel invented the microprocessor, it was a startup, Zilog, that designed and developed the Z80 as an improvement on Intel's 8080 CPU. Zilog was started (and the Z80 designed by) Federico Faggin who (before leaving Intel to start Zilog) had led development of the 4004 and 8080 CPUs.

The Z80 essentially took over from the 8080 to become one of the three main processor families at the time. The other two were the 6502 family from MOS Technologies (notable for it's use in pre-Mac Apple computers, Commodore's VIC20
and C64, and many others) and the 6800 family from Motorola. The Z80 had several advantages over the older 8080 design, including:

- an enhanced instruction set
- more registers
- additional addressing modes
- simplified interfacing (power, clock, memory refresh, simpler buses) that required fewer external chips in a design (i.e. simpler and cheaper to use)

The focus here is on getting RunCPM working on the Grand Central, so this is as much of an intro to the Z80 we'll do here. For more, check out the wealth of material at z80.info. The author wrote a more detailed 2-part introduction to Z80 hardware and assembly language in issues 7 & 8 of Hackspace Magazine. A couple books worth reading, if you want a deeper dive into the Z80, are Build your own Z80 computer. Design guidelines and application notes by Steve Ciarcia and Programming the Z80 by Rodnay Zaks.

*CPU stands for Central Processing Unit, which is the part of a computer that executes instructions (the programs). MCUs (micro controller unit) like the ATmega328 and SAMD51 contain a CPU as well as memory and I/O circuits. The SAMD51's ARM Cortex-M4 core is the CPU. The ATmega328 has an AVR core as its CPU.
CP/M was a popular operating system for Z80 based systems. It was minimal compared to operating systems like Unix or OSX (or even Windows). It was more comparable to early MS-DOS. In fact MS-DOS was initially a port of CP/M to the 8086 CPU used in the original IBM PC. That story is interesting, and depicted nicely in Pirates of Silicon Valley.

Like MS-DOS, or a command (Windows) or terminal (OSX and Linux) window, CP/M is a purely textual command-line interface commonly accessed via a serial terminal.

It is simple and small, but gives the basic functionality required to manage disks, write, assemble/compile, and run programs. It also provides system level functionality to programs. Things like reading and writing files, serial input and output, and so on.

From the CP/M manual:

CP/M is a monitor control program for microcomputer system development that uses floppy disks or Winchester hard disks for backup storage. Using a computer system based on the Int8080 microcomputer, CP/M provides an environment for program construction, storage, and editing, along with assembly and program checkout facilities. CP/M can be
easily altered to execute with any computer configuration that uses a Zilog Z80 or an Intel 8080 Central Processing Unit (CPU) and has at least 20K bytes of main memory with up to 16 disk drives.

File names in CP/M are up to 8 characters for the name, followed by a period, followed by up to 3 characters for the type:

```
filename.typ
```

This is where MS-DOS got it's 8.3 filename format.

CP/M had the wildcards that we still use today when specifying filenames: ? matches a single character and * matches part or all of a file name or type.

Many commands will accept a file reference with wildcards, referred to as an ambiguous file reference (or afn) while some only allow file references without wildcards, referred to as an unambiguous file reference (or ufn). These abbreviations are used in the manual and will be used here.

**Builtin Commands**

ERA afn

Erase a file or files.

DIR afn (or simply DIR)

List the names of matching files.

REN ufn1=ufn2

Rename a file

TYPE ufn

Display the contents of as ASCII file. There was no unicode back then.
Standard Programs

There are some programs that are a standard part of CP/M. So standard that they are called Transient Commands, i.e. commands that aren't part of the CP/M core program, but are loaded, executed, and disposed of as required. Some of the more useful ones in this context are briefly described below. See the CP/M manual for more information.

STAT
Displays information on the current disk, particular files, and device assignments.

ASM
Assembles the specified program and creates an executable version.

LOAD
Load an Intel HEX format code file and creates an executable version.

DDT
The CP/M debugger.

ED
The CP/M text editor.

SUBMIT
Submit a file of commands for batch execution.

DUMP
Display the contents of a file in hex.

This of course is a bare minimum introduction to CP/M. There is a wealth of information on, and code for, CP/M at the Tim Olmstead Memorial Digital Research CP/M Library (https://adafruit.it/DK5).
RunCPM

RunCPM is a Z80 emulator bundled with various versions of CP/M which run on the emulated Z80. It includes all the tools you need to explore Z80 assembly language programming (with access to the Grand Central's IO capabilities). Microsoft Basic is included and other languages are available online (https://adafru.it/DK6), including Modula-2 (an interesting language that was Niklaus Wirth's follow up to Pascal).

There are several different ways to build RunCPM: which one you use is determined by the platform you want to build it for. We'll use the Arduino approach to build it for the Grand Central so open the RunCPM.ino in the Arduino IDE. Before that you'll need the IDE set up to build for the Grand Central. See the Grand Central tutorial (https://adafru.it/DK7) for details on doing that.

If you are new to the Arduino IDE, you may want to read the following tutorials:

- Getting Started with Arduino (https://adafru.it/pcg)
- All About Arduino Libraries (https://adafru.it/dit)
Modifying RunCPM

To use RunCPM you'll need an MicroSD card. Follow the instructions in Getting Started section of the RunCPM Readme (https://adafru.it/DK8) to set up the card. The contents of your card should be as shown below (with the card mounted on a linux workstation). The B directory isn't strictly required, but provides another virtual disk to play around with:

```
ls -R
.
A
CCP-CCPZ.60K CCP-DR.60K CCP-Z80.60K CCP-ZCP2.60K CCP-ZCP3.60K
B
CCP-CCPZ.64K CCP-DR.64K CCP-Z80.64K CCP-ZCP2.64K CCP-ZCP3.64K

./A:

./A/0:
1STREAD.ME CONSOLE8.Z80 INFO.Z80 RSTAT.COM Z80ASM.COM
ASM.COM DDT.COM LOAD.COM RSTAT.SUB Z80ASM.PDF
BDOS.ASM DISKDEF.LIB LUA.COM RSTAT.Z80 Z80CCP.ASM
BDSEOQ.LIB DISPLAY.LIB LUAINFO.LUA STAT.COM Z80CCP.SUB
BDOS.LUA DUMP.ASM LUA.SUB SUBMIT.COM ZCPR2.ASM
BDOS.SUB DUMP.COM LUA.Z80 SUBMIT.0 SUBMITD.COM ZCPR2.SUB
CAL.COM ED.COM LU.COM SYSGEN.COM ZCPR3.ASM
CCP.ASM EXIT.COM MAC.COM TE.COM ZCPR3.SUB
CCP.SUB EXIT.SUB MAKEFCB.LIB UNARC.COM ZEYALL.COM
CCP-ZCP3.BIN EXIT.Z80 MBASIC.COM UNCR.COM ZEXDOC.COM
CCPZ.SUB FORMAT.COM MLOAD.ASM USQ.COM ZSID.COM
CCPZ.Z80 FORMAT.SUB MLOAD.COM XMODEM.COM ZTRAN.COM
CLEAN.SUB FORMAT.Z80 MLOAD.DOC XSUB.COM
CONSOLE7.COM HELLO.LUA MOVCPM.COM Z2HDR.LIB
CONSOLE7.Z80 INFO.COM OPCODES.DOC Z3BASE.LIB
CONSOLE8.COM INFO.SUB PIP.COM Z3HDR.LIB

./B:

./B/0:

>:
```
RunCPM is written to be very portable. Only a couple changes had to be made to get it running on the Grand Central.

The official RunCPM repo (https://adafruit.it/DJY) contains the GrandCentral related changes.

Changes to Support the Grand Central

Since the official repo now contains the Grand Central changes, this is for information /education and as a guide if you want to get it running on other M4 boards.

**abstraction_arduino.h**

This was just a matter of setting HostOs for the Grand Central:

Line 8 changes from:

```c
#ifdef ARDUINO_SAM_DUE
```

to:

```c
#if defined ARDUINO_SAM_DUE || defined ADAFRUIT_GRAND_CENTRAL_M4
```

**RunCPM.ino**

Similar to the above change, we need to add Grand Central support here as well. In particular, the specifics of the SD card and LED interfaces, as well as the board name. At about line 28 we add a branch to the board configuration #if structure:

```c
#elif defined ADAFRUIT_GRAND_CENTRAL_M4
    #define USE_SDIO 0
    SdFat SD;
    #define SDINIT SDCARD_SS_PIN
    #define LED 13
    #define LEDinv 0
    #define BOARD "ADAFRUIT GRAND CENTRAL M4"
```
For reasons that aren’t clear (possibly due to the separate SPI bus for the SD card) the SD card and file system has to be initialized a bit differently, so we have at about (after adding the above) line 108:

```c
#if defined ADAFRUIT_GRAND_CENTRAL_M4
  if (SD.cardBegin(SDINIT, SD_SCK_MHZ(50))) {
    if (!SD.fsBegin()) {
      _puts("\nFile System initialization failed.\n");
      return;
    }
  #else
    if (SD.begin(SDINIT)) {
  #endif
```

The original line that the above replaces is:

```c
if (SD.begin(SDINIT)) {
```

That’s all it took to get RunCPM up and running. The terminal interface is via the USB connection which made it very easy. Not all terminal emulators will work perfectly, though. CP/M expects a VT100 type terminal, so not everything will work nicely (the author had some problems with MBASIC).

With the microSD card loaded/in-place and the Grand Central connected with board & port set appropriately, build and install onto the Grand Central.

Now you can pretend to be a kid in the basement learning this strange new world of computing in the early 80s. Welcome to the author’s childhood.
RunCPM provides access to Arduino I/O capabilities through CP/M's BDOS (Basic Disk Operating System) interface. By loading the C register with a function number and a call to address 5, additional functionality that has been added to the system can be accessed.

For these functions, the number of the pin being used is placed in the D register and the value to write (when appropriate) is placed in E. For read functions, the result is returned as noted.

PinMode

```
LD C, 220
LD D, pin_number
LD E, mode ;(0 = INPUT, 1 = OUTPUT, 2 = INPUT_PULLUP)
CALL 5
```

DigitalRead

```
LD C, 221
LD D, pin_number
CALL 5  Returns result in A (0 = LOW, 1 = HIGH).
```

DigitalWrite
LD C, 222
LD D, pin_number
LD E, value ;(0 = LOW, 1 = HIGH)
CALL 5

AnalogRead

LD C, 223
LD D, pin_number
CALL 5

Returns result in HL (0 - 1023).

AnalogWrite (i.e. PWM)

LD C, 224
LD D, pin_number
LD E, value (0-255)
CALL 5

Turning on a LED

Using the above PinMode and DigitalWrite calls, it's easy to write some code to turn on a LED connected to, for example, pin D8. Use ED (the editor) to enter the following into the file LED.ASM. You could do this on your workstation directly on the SD card since ED is a beast from another time and, quite possibly, another dimension.

; Turn on a LED wired to pin 8
org 100h ;start address
mvi c, 220 ;pinmode
mvi d, 8 ;digital pin number
mvi e, 1 ;value (0 = low, 1 = high)
push d ;save arguments
call 5 ;call BDOS
pop d ;restore arguments
mvi c, 222 ;digital write
call 5 ;call BDOS
ret ;exit to CP/M

Then use the ASM command to assemble it:
A>asm led
CP/M ASSEMBLER - VER 2.0
0111
000H USE FACTOR
END OF ASSEMBLY

RunCPM Version 3.7 (CP/M 2.2 60K)

This produces several files. LED.PRN is a text file containing your assembly language
program along with the machine code it assembles to. Each line has 3 columns:
address, machine code, and assembly language.

A>type led.prn

0100    org 100h
0100 0EDC   mvi c,220
0102 1608   mvi d,8
0104 1E01   mvi e, 1
0106 D5    push d
0107 CD0500 call 5
010A D1    pop d
010B 0EDE   mvi c, 222
010D CD0500 call 5
0110 C9    ret

There is also now a LED.HEX file. We can use the LOAD command/program to convert
it into LED.COM which can be executed.

A> load led

FIRST ADDRESS 0100
LAST ADDRESS 0110
BYTES READ 0011
RECORDS WRITTEN 01

Now it can executed:

A>led

which will turn on the LED connected to pin D8.
So now we can read and write digital and analog I/O from Z80 assembly language code that's running on a Z80 emulated on the Grand Central. That seems pretty round-about.

While that's true, the point is to be able to play around with Z80 assembly language (and CP/M in this case) without having to find or build an actual Z80 system (although that can be its own kind of fun).

Wrap Up

The SAMD51 has plenty of horsepower and memory to run emulators of various retro hardware, probably faster than the original systems. The Metro Grand Central M4 Express has an onboard microSD slot that's the final piece needed for RunCPM, avoiding the need for an SD shield.

The focus here is on getting RunCPM running on the Grand Central, so there's been a very brief introduction to the Z80. For more, check out the wealth of material at [Z80.info](https://adafruit.it/DJZ). The author wrote a more detailed 2-part introduction to Z80 hardware and assembly language in issues 7 & 8 of [Hackspace Magazine](https://adafruit.it/DJ-). A couple books worth reading, if you want a deeper dive into the Z80, are *Build your own Z80 computer. Design guidelines and application notes* (https://adafruit.it/DK0) by Steve Ciarcia and *Programming the Z80* (https://adafruit.it/DK1) by Rodnay Zaks. Both are now freely available at the provided links.

While it may not be directly applicable to most of what you find yourself doing, it can be instructive to delve into the low level workings of processors. It will give you a deeper understanding of how computers actually work. Using older 8-bit CPUs for this simplifies the exercise compared to delving into the working of a modern CPU or MCU.