ZIP-8: Portable CHIP-8 Emulator

Ben Grant

Abstract

I developed an interpreter for the CHIP-8 virtual machine/retro fantasy console in the modern systems language Zig. I chose an architecture which, alongside Zig's cross-compilation support, makes the emulator easily portable to a wide variety of environments, and I've been able to run it in a browser with WebAssembly and on multiple microcontroller platforms.

Introduction

CHIP-8 is a system for making games and other applications developed in the 1970s. It may be variously classified as a programming language or a virtual machine. It uses an assembly language, but it also has certain high-level instructions (like drawing a sprite on the display) which make it much easier to make graphical software. Originally ran on the COSMAC VIP microcomputer, it has 4,096 bytes of memory, sixteen 8-bit registers and a single 12-bit one for accessing memory, a 64-by-32 monochrome display, a hexadecimal keypad, and a buzzer speaker. It has 35 instructions, and has some similarities to RISC architectures in that you can only operate directly on registers, not memory, but it also has a few very complicated instructions that make that definition fuzzier.

Related work

Despite its age, CHIP-8 has retained a strong hobbyist community. Innumerable people have developed emulators to run CHIP-8 code on modern computers, as well as new software written for CHIP-8. That said, some of the most useful implementations and resources I've encountered are:

- Matt Mikolay's CHIP-8 documentation
- Octo, a high-level CHIP-8 assembler and development environment on the web. Octo also has its own extension to CHIP-8, called XO-CHIP, which adds more advanced graphics and sound support. I used Octo to develop new CHIP-8 ROMs to run on ZIP-8.

Preliminary results

ZIP-8 is currently working on the three platforms mentioned in Abstract:

- It compiles with WebAssembly to a module only 12 kB in size, which I run in a web application wrapper that displays the screen contents with HTML5 `<canvas>`.
- It runs on the Raspberry Pi RP2040, a low-cost but powerful 32-bit ARM microcontroller. The RP2040 is fast enough to drive DVI displays, and HDMI is backwards-compatible with DVI, so this version can output to almost any HDMI monitor.
- It runs on the Microchip ATmega4809, an 8-bit AVR microcontroller with only 6,144 bytes of RAM. This version has been the most challenging by far. I thought that the limited RAM capacity would make it technically possible but difficult to run CHIP-8 (there are 2,048 bytes of headroom available on top of the memory CHIP-8 needs), but so far I have not yet gotten the overhead low enough to run full CHIP-8. Instead, I limit the CHIP-8 system's memory to 1,024 bytes, which is still enough to run many games. This version connects to a 128-by-64 pixel monochrome OLED display, over SPI, and it can run the emulator and interface with the display fast enough to run most games at the full 60 Hz.

Both of the microcontroller platforms support requesting input from a keyboard over I^{2}C, which could be implemented by any other microcontroller connected to a grid of keys.

It’s been possible to run ZIP-8 on so many platforms because Zig has very good cross-compilation support, and because I made all functionality available with a simple C interface. This means it can be called by JavaScript code that imports a WebAssembly module, or by C++ code in the Arduino IDE targeting my microcontroller platforms. While my internal functions use more advanced types that can’t be expressed in the C ABI, it’s easy to create wrapper functions using only simple types (e.g. the main data structure for the CPU is expressed in C as a `void *`) that call the Zig functions internally.

Next steps

There are a lot of directions I could take this work. First, ZIP-8′s portable API is technically incomplete currently as it cannot report whether sound is playing to the host environment. This is a very simple API to add, though: CHIP-8′s sound interface is only a buzzer that can be on or off, and the CPU already tracks enough information to tell whether it should be playing.

There is a somewhat popular extension to CHIP-8 called SUPER-CHIP. It adds higher resolution display support, scrolling (which is helpful since changing the entire display at once would use a lot of code), and miscellaneous other features. I could add support for those instructions. I could also consider adding support for Octo’s XO-CHIP extension, which adds color support to the display, quadruples the RAM to 16,384 bytes, and adds much better sound support. The most difficult part of that, I think, would be sound because that requires audio synthesis with very precise timing.

Another useful feature would be compatibility flags. Since CHIP-8 has had so many different implementations, they have not all agreed on certain details of how certain instructions should be executed. Software that assumes these instructions work one way will not currently work on ZIP-8. I could add flags which toggle between different modes for all these instructions, to allow more code to work.

I’ve had several ideas about adding multiplayer support in some form. CHIP-8 games can implement multiplayer by assigning different areas of the keypad to different players’ controls. Since the state of the emulator is so self-contained and it can execute so quickly on modern processors, it would likely be possible to have a peer-to-peer system where each device, when it receives inputs from other devices, rolls back the emulator state to what it was when those inputs occurred and replays it up to the current time with corrected inputs. This is a similar system to what’s used by many fighting games. There’s also a possible implementation where the server handles all processing. Clients might even act just as dumb controllers in this case (if, say, players are seated at the same monitor which is controlled by the server, so they don’t need to see it on their own devices). Or, the server could try to efficiently stream the display contents to connected clients.