Team Fortville:

Chip-8 Emulator

Matthew Ehrler - mehrler@uvic.ca

Justin Underhay - underhay@uvic.ca

Christopher Hettrick - chettrick@uvic.ca

Josh Williams - joshwill@uvic.ca

USER GUIDE:

To run the program, simply perform a ‘gradle run’ from the command line in the project root directory. To add a new rom to run, include the rom file in the project’s ‘src/main/resources’ directory and update the ‘args’ input variable in the ‘gradle.build’ file with the rom’s filename. Arguments may also be specified as ‘gradle run --args=<*romname.ch8*>’ on the command line.

Available roms are:

breakout.ch8, hidden.ch8, input.ch8, invaders.ch8, pong.ch8, tank.ch8, tetris.ch8

The original Chip-8 key mapping:

1	2	3	C
4	5	6	D
7	8	9	E
A	0	B	F

Key mapping on a modern computer keyboard:

1	2	3	4
Q	W	E	R
A	S	D	F
Z	X	C	V

For example, in the game ‘Pong,’ keys ‘1’ and ‘Q’ are used to move to paddle up and down, respectively.

PROJECT OVERVIEW:

Our project is a software simulation of the 8-bit Chip-8 instruction set. Chip-8 is an instruction set designed in the mid 1970's by Joseph Weisbecker to allow for easier programming of games on the COSMAC VIP and Telmac 1800 8-bit Microcomputer. Our implementation uses multithreading and a 2-stage pipeline. The first pipeline stage is for fetching and decoding instructions, and the second is for executing instructions and writing back results to memory or registers. Each stage runs in their own thread. The opcode to be executed and relevant data is passed between threads using an intermediary buffer. When the buffer is empty the fetch/decode stage will fill it with the next instruction and data. The execute/writeback stage will execute the instruction when the buffer is full. The original Chip-8 architecture's 4KB of memory and various registers are simulated in their respective classes using arrays. The Chip-8's 64x32 screen and 16 button keyboard is simulated using Java Swing and a keyboard listener. The buzzer is simulated with the Java sound library. The overall architecture is depicted in Figure 1.

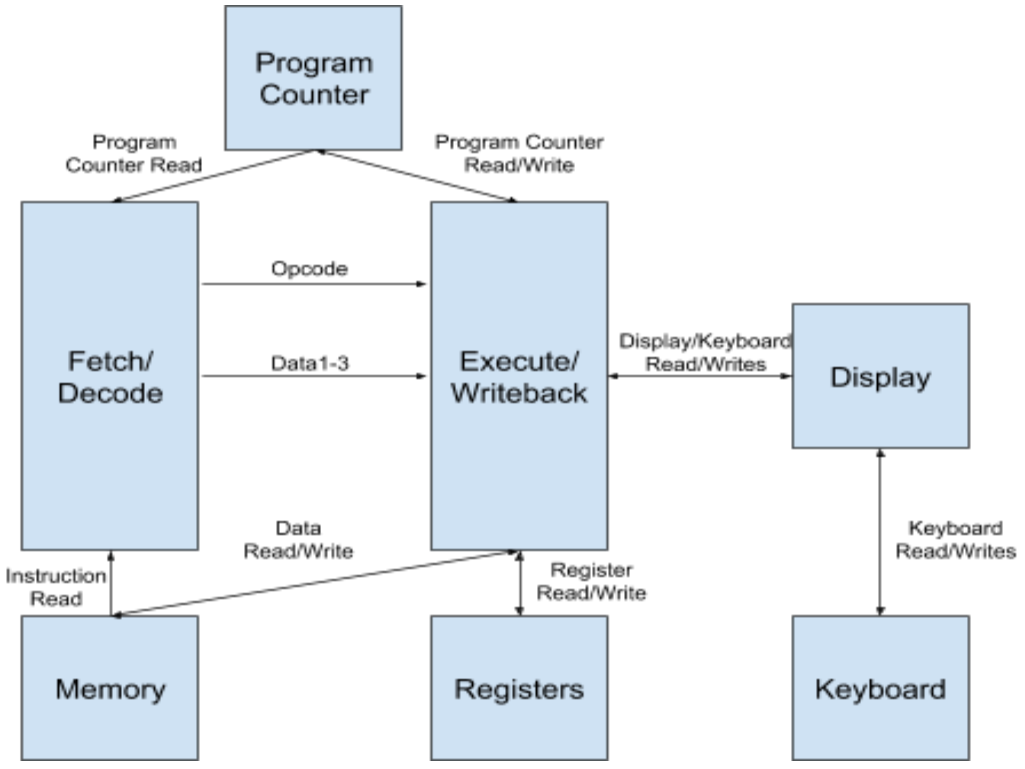

Figure 1: Chip-8 simulator architecture

CLASS OVERVIEWS:

CHIP8.java: Takes a filename as an argument from the command line (uses pong.ch8 if none specified), loads the supplied rom/instructions into memory, and creates the Register, Memory, Display, FetchBuffer, FetchDecode, and ExecuteWriteback classes. It then creates and runs two threads, one for the FetchDecode class, and the other for the ExecuteWriteback class.

FetchDecode.java: If the instruction buffer is empty, it will run a clock cycle and decrement timers. If the instruction buffer is full, it will wait. In each clock cycle the decoded instruction information specified by the program counter is loaded into the buffer, the buffer is set to full, and the program counter is incremented.

ExecuteWriteback.java: If the instruction buffer is full, it will run a clock cycle. If the instruction buffer is empty, it will wait. In each clock cycle it will gather the decoded instruction information from the buffer, set it to empty, and then execute that instruction.

Buffer.java: Holds and passes instruction data between the two threads.

Display.java: Creates a 1290 pixel by 680 pixel display, and uses 20 pixel by 20 pixel blocks to imitate the 64 by 32 Chip-8 display. This class has setPixel(), drawSprite(), and clear() methods.

Keyboard.java: Uses 16 keys (1-4, Q-R, A-F, Z-V). This class has getKey(), waitForKey(), keyPressed(), and keyReleased() methods.

Sound.java: This class has methods to set the tone, length, and volume of a beep.

Registers.java: This class sets up 16 programmer visible registers, 16 stack locations for subroutine calls, a stack pointer, an index register, a program counter, a sound timer, and a delay timer. This class has methods to access and change these registers.

Memory.java: This class sets up a 4096-byte memory space and loads into this memory the supplied Chip-8 rom file. This class also sets up sprites in memory for a basic font of hexadecimal digits. This class has load() and store() methods.

35 Opcodes: Perform various operations such as adding two registers together, jump to a memory address, or setting the delay timer.

REFERENCES:

[1] http://devernay.free.fr/hacks/chip8/C8TECH10.HTM#0.0

[2] https://web.archive.org/web/20130903140414/http://chip8.com/?page=73