Project 4

Every hardware platform is designed to perform a basic set of operations. The operations are specified using agreed upon binary codes which, together, form the computer's *Instruction Set*. The instruction set comes in two flavors: binary, and symbolic. Writing programs directly in binary code is possible, yet tedious and unnecessary. Instead, we can write low-level programs using a symbolic language, called *assembly*, and have them translated into binary code by a program called *assembler*. In this project you will write some low-level assembly programs in the symbolic Hack machine language, and execute them on a program that emulates the operations of the Hack processor. This program, called the *CPU Emulator*, also features a built-in assembler.

Objectives

- Get a hands-on taste of writing and executing low-level programs in machine language
- Get acquainted with the Hack computer, before setting out to build it in the next project.

Tasks

Write and test the following two programs. When executed on the supplied CPU emulator, your programs should realize the behaviors described below.

Mult.asm (example of an arithmetic task): The inputs of this program are the values stored in R0 and R1 (RAM[0] and RAM[1]). The program computes the product R0 * R1, and stores the result in R2. Assume that R0 ≥ 0, R1 ≥ 0, and R0 * R1 < 32768 (your program need not test these conditions). The supplied Mult.tst and Mult.cmp scripts are designed to test your program on some data values.

Fill.asm (example of inputI/output task): This program runs an infinite loop that listens to the keyboard. When a key is pressed (any key), the program blackens the screen by writing "black" in every pixel. When no key is pressed, the program clears the screen by writing "white" in every pixel. You may choose to blacken and clear the screen in any spatial pattern, as long as pressing a key continuously for long enough will result in a fully blackened screen, and not pressing any key for long enough will result in a cleared screen. This program has a test script (Fill.tst) but no compare file – it should be checked by visibly inspecting the simulated screen in the CPU emulator.

The CPU Emulator

This program provides a visual simulation of the Hack computer. The program's GUI shows the current states of the computer's instruction memory (ROM), data memory (RAM), the two registers A and D, the program counter PC. It also displays the current state of the computer's screen, and allows entering inputs through the physical keyboard of your PC.

The typical way to use the CPU emulator is to load a machine language program into the ROM, execute the code, and observe its impact on the simulated hardware elements. Importantly, the CPU emulator enables loading binary .hack files as well as symbolic .asm files, written in the Hack assembly language. In the latter case, the emulator translates the assembly program into binary code on the fly, using its builtin assembler. Conveniently, the loaded code can be viewed in both its binary and symbolic representations.

Since the supplied CPU emulator features a builtin assembler, there is no need to use a stand-alone Hack assembler in this project.

Steps

We recommend proceeding as follows:

1. Open the tools/CPUEmulator in one window. Open a plain text editor (any editor) in another window.
2. Load the skeletal projects/04/mult/Mult.asm program into the editor.
3. Write / edit the code to implement your multiplication algorithm. Save the file.
4. Load Mult.asm into the CPU emulator. This can be done either interactively, or by loading and executing the supplied Mult.tst script.
5. Run the script. If you get any translation or run-time errors, go to step 3.

Repeat steps 1–5 for the second program, Fill.asm, using the projects/04/fill folder.

Tips

The Hack language is case sensitive. A common programming error occurs when one writes, say, @foo and @Foo in different parts of the program, thinking that both instructions refer to the same symbol. In fact, the assembler will generate and manage two different variables that have nothing in common.

Use upper-case letters for labels (like LOOP), and lower-case letters for variables (like sum).

Indent your programs, and write comments as needed. See the programs in the lecture or in the book and follow their examples.

As always, strive to write elegant, efficient, and self-explanatory programs.

References

CPU Emulator demo

CPU Emulator Tutorial (click *slideshow*)