Week 7 assignment: The Hack CPU

1 Tasks

1. Create a computer capable of running Hack in Logisim!
2. Use any extra time to get more assembly practice in. In particular, **you should get to at least the end of section 5 of week 5's assignment before starting this one** — Hack assembly is going to be the foundation of the unit going forward, while this week's assignment is mostly only relevant to this week's material.

2 Required software

In addition to Logisim, you will need the assembler from the nand2tetris software suite. The demonstrations in the video lectures should give you a good idea of how to use this software, and the official documentation is available from the unit page. All of it runs on Windows, Linux and Mac OS.

In order to run this software (and anything else from the nand2tetris suite) on your home computer, you will need to install the Java Runtime Environment, which you can download here. (It's already installed on the lab computers.) If you are getting an error about javaw.exe being missing, the most likely reason is that you don't have the Java Runtime Environment installed.

3 Implementing Hack

At last it's time to actually build a computer! You've built most of the components already — the Logisim skeleton file provided on the unit page includes copies of the ALU and program counter from weeks 2 and 4, and you'll be using the pre-made Logisim versions of RAM, ROM and registers. You also know from this week's videos how Hack assembly translates to machine code and roughly how the Hack CPU fits together. The outline in video 2 isn't a complete solution, and it's not the only way to do it, but it's enough that you should be able to fill in the gaps using the skills you picked up in the first half of the unit.

Since this is an architecture unit rather than a hardware design unit, we're not simulating the screen and keyboard directly. Instead, while a "true" Hack implementation would use 32KB RAM with a 14-bit address space, you should use 64KB RAM with a 15-bit address space — so the extra RAM can hold the memory-mapped addresses 0x4000–0x6000 for screen and keyboard.

As you proceed, you may run into situations where it's not clear how the CPU should behave. As a rule of thumb, if the situation isn't specified as part of the Hack ISA, then it doesn't matter what your CPU does in response. For example, if A contains 0xFFFF, then it doesn't matter what the result of executing `D=M` is, since M is only defined when A is a valid memory address.

Logisim tip: You will almost certainly find it useful to have multiple clock inputs in your circuit. You can manually cycle all these clock inputs at once using the "Simulate → Tick once" option (Ctrl+T).

4 Testing your CPU

After testing a few individual instructions, the easiest way to stress-test your CPU is to load in a reasonably complex program and see what goes wrong, comparing side-by-side with the CPU simulator if necessary. The ROM in the Logisim skeleton provided is pre-loaded with a solution to the Collatz conjecture question from last week's assignment. If you'd rather load your own program, there's a reasonably easy way to do this:

1. Using the nand2tetris Hack assembler, compile your program to machine code, resulting in a .hack file.
2. Download and run the bin2hex program linked on the unit page from the command line. The first argument should be your .hack file, and the second argument should be a new output file. This will convert each binary word of machine code into a hex word, separated by spaces. (If you are not running Windows, you will need to compile bin2hex from source — we wrote it in C, so you should be able to do this.)

3. Open the output file in your favourite text editor, select it all, and copy it to the clipboard.
4. Right click on your ROM in Logisim and choose “Edit contents...”.
5. Select address 0 and hit Ctrl+V (or Command+V on a Mac). The file contents should now appear in the window.
6. Select ”Close Window”. Your program is now loaded into ROM and ready to test.

You’ll find the CPU simulator runs much slower than the tick frequency in Logisim. To give you an idea, you should expect the pre-loaded Collatz conjecture program to take around 10 seconds to run with RAM[0] set to 0x002C, or around 3–4 seconds with RAM[0] set to 0x0005. If it’s taking much longer than that, something has gone wrong.

If you want to simulate keyboard inputs then the easiest way is to pause simulation on Logisim, edit RAM[0x6000] to match the input you want to simulate, then resume simulation.