Project 5: Computer Architecture

CS 220

Background

In previous projects we've built the computer's basic processing and storage devices (ALU and RAM, respectively). In this project we will put everything together, yielding the complete Hack Hardware Platform. The result will be a general-purpose computer that can run any program that you fancy.

Objective

Complete the construction of the Hack CPU and computer platform, leading up to the top-most Computer chip.

Chips

Chip (HDL)	Description	Testing
Memory.hdl	Entire RAM address space	Test this chip using Memory.tst and Memory.cmp
CPU.hdl	The Hack CPU	Recommended test files: CPU.tst and CPU.cmp. Alternative test files (less thorough but do not require using the built-in DRegister): CPU-external.tst and CPU-external.cmp.
Computer.hdl	The platform's top-most chip	Test by running some Hack programs on the constructed chip. See more instructions below.

Contract

The computer platform that you build should be capable of executing programs written in the Hack machine language, specified in Chapter 4. Demonstrate this capability by having your Computer chip run the three test programs given below.

Testing

Testing the Memory and CPU chips: It's important to unit-test these chips before proceeding to build the overall Computer chip. Use the the test scripts and compare files listed above.

Testing the Computer chip: A natural way to test the overall Computer chip implementation is to have it execute some sample programs written in the Hack machine language. In order to perform such a test, one can write a test script that (i) loads the `Computer.hdl` chip description into the supplied Hardware Simulator, (ii) loads a machine-level program from an external `.hack` file into the ROM chip-part of the loaded `Computer.hdl` chip, and then (iii) runs the clock enough cycles to execute the loaded instructions. We supply all the files necessary to run three such tests, as follows:

Program	Comments
Add.hack	Adds up the two constants 2 and 3 and writes the result in RAM[0]. Recommended test: ComputerAdd.tst and ComputerAdd.cmp. Alternative test (less thorough but only requires usage of the built-in RAM16K): ComputerAdd-external.tst and ComputerAdd-external.cmp.
Max.hack	Computes the maximum of RAM[0] and RAM[1] and writes the result in RAM[2]. Recommended test: ComputerMax.tst and ComputerMax.cmp. Alternative test (less thorough but only requires usage of the built-in RAM16K): ComputerMax-external.tst and ComputerMax-external.cmp.
Rect.hack	Draws a rectangle of width 16 pixels and length RAM[0] at the top left of the screen. Recommended test ComputerRect.tst and ComputerRect.cmp. Alternative test (less thorough but does not require usage of any built-in chips): ComputerRect-external.tst and ComputerRect-external.cmp.

Before testing your Computer chip on any one of the above programs, read the relevant `.tst` file and be sure to understand the instructions given to the simulator. Appendix B of the book may be a useful reference here.

Resources

The relevant reading for this project are Chapter 5, Appendix A, and Appendix B (as a reference). Specifically, all the chips described in Chapter 5 should be implemented in the Hardware Description Language (HDL) specified in Appendix A.

The resources that you need for this project are the supplied Hardware Simulator and the files listed above. The project files are available in a ZIP archive file available on the course web site.

Implementation Tips

Complete the computer's construction in the following order:

1. **Memory:** This chip includes three chip-parts: RAM16K, Screen, and Keyboard. The Screen and the Keyboard are available as built-in chips, and thus there is no need to implement them. Although the RAM16K chip was built in Project 3, we recommend using its built-in version, as it provides a debugging-friendly GUI.

2. **CPU:** This chip can be constructed according to the proposed CPU implementation given in Figure 5.9 of Chapter 5, using the ALU and register chips built in Projects 2 and 3, respectively. We recommend though using built-in chip-parts instead, in particular ARegister and DRegister. The built-in versions of these two chips have exactly the same interface and functionality as those of the Register chip specified in Chapter 3; however, they feature GUI side-effects that come handy for testing purposes.

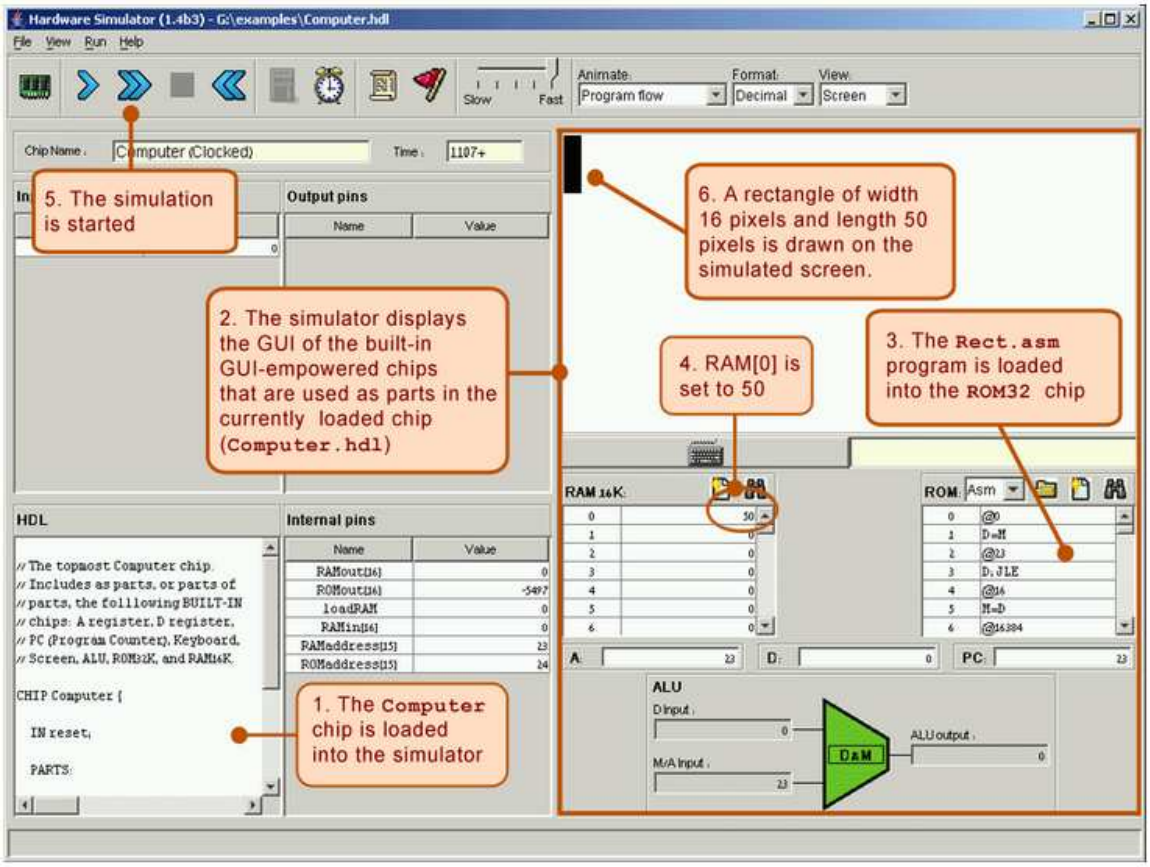
   In principle, your CPU implementation may include internal chips of your own specification, i.e. chips not mentioned in Figure 5.9 of Chapter 5. However, this is not recommended, and will most likely yield a less efficient CPU design. If you choose to create new chips not mentioned in the book, be sure to document and unit-test them carefully before you plug them into the architecture.

3. **Instruction memory:** Use the built-in ROM32K chip.

4. **Computer:** The top-most Computer chip can be constructed according to the proposed implementation shown in Figure 5.10 of Chapter 5.

Tools

All the chips mentioned in this project, including the topmost Computer chip, can be implemented and tested using the supplied Hardware Simulator. Here is a screen shot of testing the `Rect.hack` program on a Computer chip implementation.



The Rect program illustrated above draws a rectangle of width 16 pixels and length `RAM[0]` at the top-left of the screen. Now here is an interesting observation: normally, when you run a program on some computer, and you don't get the desired result, you conclude that the program is buggy. In our case though, the supplied Rect program is bug-free. Thus, if running this program yields unexpected results, it means that the computer platform on which it runs (`Computer.hdl` and/or some of its lower-level chip parts) is buggy. If that is the case, you have to debug your chips.

Submission and Assessment

If you can't finish the project on time, submit what you've managed to do, and relax. All the projects in this course are highly modular, with incremental test files. Each hardware project consists of many chip modules (`*.hdl` programs), and each software project consists of many software modules (classes and methods). It is best to treat each project as a modular problem set, and try to work out as many problems as you can. You will get partial credit for partial work.

What if your chip or program is not working? It's not the end of the world. Hand in whatever you did, and explain what works and what doesn't in a README file. If you want, you can also supply test files that you developed, to demonstrate working and non-working parts of your project. Instead of trying to hide the problem, be explicit and clear about it. You will get partial credit for your work.

See the next page for the assessment rubric. Submit the following as a single ZIP archive in Canvas:

1. A README file containing the names of all group members. This file may also contain other information, as described above.
2. All your HDL files.
3. Nothing else.

Project 5: Computer Architecture

Student name(s): _____

Grading method: The implementation of some chips was described in the book, and some chips are simpler than others. The different weights assigned to the chips below reflect this variance. If the chip passes *all* the tests specified in the supplied test script, it receives two thirds of its allotted points. The remaining third reflects our evaluation of the way the chip is built.

Generally speaking, we prefer implementations that *use as few chip parts as possible,* even if it implies a less efficient chip design.

Chip	***Working?***	***Well built?***	***Comments***
Memory	/ 17	/ 8	
CPU	/ 35	/ 18	
Computer	/ 15	/ 7	
Total	/ 67	/ 33	

Total grade: _____