CS270 Programming Assignment 6
"LC-3 Simulator & Assembler Project"

Programs due Friday, May 4 (via Checkin by 5:00pm)
Homework and programming assignments are to be done individually.

NO LATE SUBMISSION FOR THIS ASSIGNMENT

Version 0.9 (will be finalized by 5:00 pm, Wednesday April 25)

Goals

1. Improve your understanding of the LC-3 machine architecture by building a simulator.
2. Further enhance your C skills by extending the simulator to include a shift instruction.

The Assignment

In this assignment, you will develop a version of the LC-3 simulator using C. Due to the scale of this project, we have provided you with a large portion of the program code and a working version of the simulator executable (thanks to Fritz). The simulator is actually a stripped down version of the code in the lc3sim tool, and the gui interface that you are familiar with also works. However, large parts of the simulator are missing, and it is your job to complete it.

PART A

1. Download the executable and skeleton code from http://www.cs.colostate.edu/~cs270/.Spring12/Assignments/PA6/PA6.tar. It contains all the files needed to develop the simulator as well as a simple file **testST.asm**, which is your first test case. When you extract the file it will create a directory named **PA6**.

Study the code in `lc3sim.c` (we will walk you though this in Recitation). This is also your first opportunity to read a significant piece of code (about 1000 lines) that has been written by someone else and is not fully documented. Your job is to complete the implementation of the functions in `mysim.c` (this is the single file that you will submit for this part). In addition to the functions `sign_extend` and `write_memory`, the most significant function that you will need to write and complete is `execute_one_instruction`. This function is called by lc3sim.c, and you need to understand exactly how this happens.

Your simulator should be built up incrementally. Currently, `write_memory` does nothing. This is why, when you start the simulator, nothing is initialized (even the operating system is not loaded). As soon as you get this working, you will be able to start your simulator and see that the operating system is loaded. After this, you testing should consist of writing very small (incomplete), short program fragments of one or two instructions that you test in a very incremental manner.

PART B

Extend the simulator to include a new instruction for bit-shift (using the unused opcode, 1101). Like the ADD/AND instructions there are two ways to use this (immediate or register). The instruction specifies a destination register DR, a source register SR1, (in fields, `IR[11:9]` and `IR[8:6]`, respectively) and optionally an additional source register SR2, (specified in IR[2:0]). The bit IR[5] is used to encode this. If that bit is 1, the amount of the shift is encoded as an immediate value. If the bit is 0, the shift is stored in SR2, just like in ADD/AND instructions. The amount by which to shift is a 5-bit 2's complement value, i.e., in the range, 16 and +15 that come wither from the immediate field of the instruction or from SR2[4:0]). A positive value of this operand causes a left-shift, adding zeroes on the right. A negative value shifts the bits right, adding zeroes on the left.

To be able to test the SHIFT instruction, your PA6.tar contains an executable newlc3as. This is an assembler that recognizes the operation SHIFT.

Submission Instructions

Submit a tar.gz file called PA6.tar.gz with your implementation (subject to revision). If you were unable to complete Part-B (i.e., tokenizing and parsing the symbol table file) and end up using the provided solution, then also submit a README file describing your progress with implementing Part-B to be considered for partial credit. Do the same if you were unable to complete Part-A. Failure to submit the README file will result in no partial credit.

Late Policy: Late submissions are not allowed for this assignment.

Grading Criteria

To grade the assignment, we will examine and run the program on a number of .asm and .obj files (20 points for a provided file and 30 points for our own test files for part A, and 10+20 for part B), making sure that a correct .hex file is generated in both cases. 10 points will be given for following the instructions in the assignment. The remainder of the grade will be given for style considerations: commenting, indentation, naming (20 points). For the first time this semester, points will be deducted for excessive code size (-5 points), compiler warnings (-2 points per, up to -10 points) and compiler errors (-10 points per, up to -30 points). Remember, you will not receive any partial credit if you fail to submit the README file.