CS2110 Fall 2011
Homework 6

This assignment is due by:

Day: Wednesday, October 5th

Time: 11:54:59pm

Rules and Regulations

Academic Misconduct

Academic misconduct is taken very seriously in this class. Homework assignments are collaborative. However, each of these assignments should be coded by you and only you. This means you may not copy code from your peers, someone who has already taken this course, or from the Internet. You may work with others **who are enrolled in the course,** but each student should be turning in their own version of the assignment. Be very careful when supplying your work to a classmate that promises just to look at it. If he turns it in as his own you will both be charged. We will be using automated code analysis and comparison tools to enforce these rules. **If you are caught you will receive a zero and will be reported to Dean of Students.**

Submission Guidelines

1. You are responsible for turning in assignments on time. This includes allowing for unforeseen circumstances. If you have an emergency let us know ***IN ADVANCE*** of the due time supplying relevant documentation (i.e. note from the dean, doctor's note, etc). Extensions will only be granted to those who contact us in advance of the deadline and no extensions will be made after the due date.

2. You are also responsible for ensuring that what you turned in is what you meant to turn in. No excuses, what you turn in is what we grade. In addition your assignment must be turned in on T-Square. When you submit the assignment you should get an email from T-Square telling you that you submitted the assignment. If you do not get this email that means that you did not complete the submission process correctly. Under no circumstances whatsoever we will accept any email submission of an assignment. Note: if you were granted an extension you will still turn in the assignment over T-Square.

3. There is a random grace period added to all assignments and the TA who posts the assignment determines it. The grace period will last at least one hour and may be up to 6 hours and can end on a 5 minute interval; therefore, you are guaranteed to be able to submit your assignment before 12:55AM and you may have up to 5:55AM. As stated it can end on a 5 minute interval so valid ending times are 1AM, 1:05AM, 1:10AM, etc. **Do not ask us what the grace period is we will not tell you**. So what you should take from this is not to start assignments on the last day and depend on this grace period past 12:55AM. There is also no late penalty for submitting

within the grace period. If you can not submit your assignment on T-Square due to the grace period ending then you will receive a zero, no exceptions.
4. Although you may ask TAs for clarification but you are ultimately responsible for what you submit.

Submission Conventions

1. All files you submit for assignments in this course should have your name at the top of the file as a comment for any source code file, and somewhere in the file, near the top, for other files.
2. In addition any code you write must be clearly commented and the comments must be meaningful. You should comment your code in terms of the algorithm you are implementing we all know what the line of code does.
3. When preparing your submission you may either submit the files individually to T-Square (preferred) or you may submit an archive (zip or tar.gz only please) of the files.
4. If you choose to submit an archive please don't zip up a folder with the files, only submit an archive of the files we want.
5. Do not submit compiled files that is .class files for Java code and .o files for C code.

Overview

If you haven't already, read the “Getting Started” pdf for instructions on getting the LC-3 simulator setup on your machine

For this homework, you will write a program in LC-3 assembly that disassembles an LC-3 assembly program. What is disassembling? It's exactly what it sounds like. It is the opposite of assembling. You take the assembled binary instruction and output the assembly instruction that corresponds to it

For example,
You're given the number: **0x1000**
The disassembled instruction is: **ADD R0, R0, R0**

The format of each instruction is listed in PattPatelAppA.pdf, and you should follow this formatting for your program.

This program consists of two parts;

1. Write a program in a high-level language such as Java that disassembles a list of instructions.
2. From your high-level implementation, convert to assembly.

Part 1: High-level Program

When writing assembly programs, it is easier to first implement the program in a higher level language and then convert from that to assembly, because you can picture how each high-level instruction could map down to assembly. As such, you should only use statements that map easily to assembly. For instance, avoid multiply and divide, because the LC-3 has nothing comparable, and they're computationally intensive on the LC-3 when implemented. This means do not use objects, do not use complex string functions do not use complex library functions.

As a note, you will be given an operation that does a right-shift and another that prints out a number stored in a register to the LC-3 console, so you won't have worry about how to do those things. More on this in the assembly section.

The program should execute as follows;

1. Fetch an instruction from a list of instructions
2. Determine the type of instruction (ADD, AND, etc)
3. Print out the instruction based on type.
4. Repeat until the instruction 0xFFFF is reached in the list.

0x0E00 should disassemble to **BR 0** or **BRNZP 0** (your choice both are the same).

As a special case,
0x0000 should disassemble to **NOP**, which means NO OPERATION. These instructions effectively do nothing. There are more special cases of this format that could be disassembled into a NOP so be sure to catch them all!

For TRAPs you only have to know to disassemble **0xF025** as **HALT**. For any invalid instructions, opcode or other traps such as GETC, RTI, or the reserved opcode 1101, you can just simply display them as **ERROR**.

You must include all of your test cases in your high-level implementation. We will be running your high-level implementation to test if you have indeed implemented it. You are free to use Java, Python, Ruby, C, C++ to implement this.

Part 2: Assembly

Once you have a working implementation in high-level code, you are to convert it to assembly.
Take it step-by-step and think about how you could map each instruction one at a time.
Write your assembly in the provided "disassemble.asm" file

The list of instructions to disassemble will be at x5000. The sentinel value is 0xFFFF.
An example test case we could use is

Address	Value
x5000	x0000
x5001	x01FF
x5002	xF025
x5003	xFFFF

There are 3 instructions for you to disassemble in the above test case.

Along with the .fill pseudo-op which fills a memory location with a number, there is another pseudo-op that will come in handy called .stringz. This pseudo-op stores the following string at that memory location (i.e. **STRN_HELLO .stringz "Hello World"** would store "Hello World" at the label STRN_HELLO) Note that this string has a trailing 0 (read as ASCII character 0 NUL) which is the convention for strings.
Operations that use strings depend on that 0 to know when the string ends.

There's a TRAP called PUTS that will print a string starting at the address in R0 to the console.

Right-Shift
To use the right-shift (x>>y), put x into R1 and y into R0 then call **JSR RSHIFT**.
The shifted number will be stored in to R0.

Print Number
To print a number to the console, place the number you want to print into R0 and call **JSR PRINT_NUM**. The number will output on the console in hex format.

NOTE: Do not worry what the JSR instruction does. Treat it as a blackbox call that does magic for now. As long as you meet the preconditions (i.e R1 contains X R0 contains Y when it completes R0 will contain X >> Y). You will learn this instructions use in the next homework. Do not use this instruction for anything other than Calling Right-shift and Print Number.

A Few Requirements

1. Your code must assemble and run with NO WARNINGS
2. Comment your code! This is especially important in assembly, because it's much harder to interpret what is happening later, and you'll be glad left notes to let you know sections of code or certain instructions are contributing to the code. Comment things like what registers are being used for and what not so intuitive lines of code are actually doing. To comment code in LC-3 assembly just type a semi-colon (;), and the rest of that line will be a comment.
   Avoid stating the obvious in your comments, it doesn't help in understanding what the code is doing.

   Good Comment
   ```
   ADD R3, R3, -1    ;counter--
   BRp LOOP          ;if counter == 0 don't loop again
   ```

   Bad Comment
   ```
   ADD R3, R3, 1     ;Decrement R3
   BRp LOOP          ;Branch to LOOP if positive
   ```

3. Print your output to LC-3 console.
4. DO NOT assume that ANYTHING in the LC-3 is already zero. Treat the machine as if your program was loaded into a machine with random values stored in the memory and register file. To randomize memory in simpl and Complx State > Randomize.

5. ABSOLUTELY DO NOT use R6 and R7. These registers are used internally by the code we gave you. If you touch them buildings will crumble, butterflies will spontaneously combust!
6. Test your assembly. Don't just assume it works and turn it in. At the bottom of disassemble.asm is the space you're program reads the instructions from. Add instructions or invalid operations here to test the output.

Deliverables

1. Your working high-level code (.py, .rb, .java, .c, .cpp)
2. disassemble.asm