TCSS371 – Machine Organization
Homework 4 – Simulator
25 Points

Follow the coding and commenting conventions that you have been using in the course. Your program when run as is must display numbers 9 to 1 and final display of the simulator. We do not want to see any other output.

Submit a zip file of all your .java files (NO OTHER files) on Canvas. Name it **uwnetidSimulator.zip** or **uwnetid1uwnetid2Simulator.zip,** using your uwnetids. Your names must be in the header of each java file that you modified. No late work or email submissions will be accepted.

Simulator Instructions

Download a copy of the incomplete LC3 simulator starter code from Canvas. It includes the following files:

- BitString.java
- Computer.java
- Simulator.java

BitString: The simulation is based on the class BitString. The existing code provides many methods that can manipulate BitString objects for you and you don't need to write additional methods.

A BitString contains an array of char, but the values in each char are '0' or '1'. There is an associated length member that keeps track of how full the array is (anywhere from 0 to 16).

Computer: The Computer class uses BitStrings to represent the contents of all registers and memory locations. The memory and register file are treated as arrays of BitString. To execute a single instruction, you will generally need to use a couple of BitStrings out of the Computer object, convert them to integers, do some calculations on the integer and vice versa. By frequently displaying the Computer object during development, you can verify that individual instructions are behaving properly.

Simulator: The class Simulator includes a short LC3 program in two different forms, assembly language and binary. This program prints the numbers 9 down to 1 and halts.

Finish the simulator so that this program is loaded and executed. The required instructions are ADD, LD, BR, OUT, HALT. To finish the simulation, you will need to at least do the following:

- Add methods to execute the unimplemented instructions (like ADD, LD, BR, OUT, HALT).
- Modify main method so that the program is loaded into the computer. Follow the instructions in main for this. **DO NOT** call execute method in a loop here. The execute method must be only called once.
- Modify the execute method in Computer class, so that it executes instructions until it runs into a HALT instruction.
- Write Junit test methods for each instruction in ComputerTest class and you will be graded on these test methods.

You can model ADD, LD and BR after the NOT instruction provided. You need to examine the bits in IR and modify memory, condition codes, registers, and the PC as the current instruction would if executed by the LC3.

For OUT and HALT, which would normally trigger trap routines, the simulator should instead perform the required action. In the case of HALT, the execute method should quit (by exiting its loop) and for OUT, a single character should appear on the IDE's console based on the ASCII code currently in R0.

Your program does not need to execute any other LC3 instructions, but your code should be general enough to work with similar assembly programs. You will note that the amount of memory available in the simulator is quite small - roughly 50 words. This is enough for running simple programs.

If you want to try other programs, the LC3 Edit program will create a .bin file containing 0/1 strings for any assembly program that you compile. The trap vector and interrupt vector tables are not included in this simulation. We are also omitting input/output related hardware such as KBDR/KBSR and DDR/DSR.

As a general strategy, you might execute single-instruction programs to test the quality of your simulator before testing a multi-line program.

Extra Credit: Implement AND, LDI, STI, LDR, STR in addition to the above instructions for extra credit – 2 points for each instruction implemented. You must create test methods in ComputerTest class to test these new methods. You will be graded on the test methods.