CPSC3300 Project 4

Due date: 11:59PM Dec. 3rd

In this project you are asked to simulate a CPU (as seen in Zybooks 4.4). The instructions supported by this simulator are the ones in the Project 2. Follow the given rules:-

1. You are required to follow MVC (Model View Controller) design pattern for your software architecture. You can check what MVC is at <https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller>
2. For the Model, you should use a good abstraction of the CPU, e.g., separate logic blocks as objects.
3. "Controller" will allow different granularities of runtime. You are required to have a version of the controller that runs the entire program, however you may wish to implement a "single-step" controller that does one at a time with press of keyboard button.
4. "View" - at least one "text" view that will display a "scoreboard" that shows the contents of PC, registers, and memory; also of logic block statistics - they should use the observer pattern to implement this. Note: The view is expected to update every cycle. For the text view, that means it should print the scoreboard on every cycle.
5. You should track # cycles for a given program. Also, you should track ALU arithmetic operations (how many add, sub, etc ops). Note: Some instructions besides "add" use add. example is beq; this counts as an ALU arithmetic op, incrementing the PC does not.
6. You should track # of memory reads/writes too. Note that for simplicity, you don’t need to implement separate instruction and data memories illustrated in zybooks 4.4. You just need one memory that contains instructions and data together.
7. The control should track the # of each individual instruction.

The program should accept two command line arguments. The first will be a binary instruction file generated by your Project 2. The second will be a memory file that should be read and loaded into the "memory" section. All addressing starts at 0x0.

# **Things to avoid**

You should not just have one long "main". You should use structures appropriately. You are allowed to use C++ only if you have had Brian Malloy's 2-D game design experience.

# **General Requirements**

a. You can work individually or in teams of 2. No discussion with other teams is allowed.

b. You should get started early.

c. You should include a readme with YOUR NAMES and documentation about their project; e.g. design decisions and specifics.

d. You should use object-oriented design principles.

# **Notes about Output**

You are free to use language of your choice, however, you should create a makefile with "all" and "clean" targets. You can add additional arguments for your simulator (for single stepping etc), but ./smolmips myprog.bin mydata.bin should give the expected behavior, where myprog.bin is the output of the assembler from Project 2 and mydata.bin is the associated memory. make and that command should be enough to run the program - no other script setups.

# **Language and Skeleton Code**

You are encouraged to use Java because it is easy to express object oriented designs. A skeleton for the project has been provided in Java—you are NOT required to use it but may find the outline helpful. You can use C if you know how to use object-oriented principles in C. If you don't you will lose points. You can also use C++ if you have experience with Brian Malloy's 2-D game design. You are not supposed to use libraries except the default builtin.

# **Sample Input**

A sample input program and memory have been provided. They have been translated for you but the raw assembly is also provided if you would like to edit it and re-translate using your previous code. The provided files are not intended to provide full test coverage and should be used as a small valid input that can aid your development. You do not need to create or submit any input files but may find it helpful to validate your approach.

# **Bonus and Extra Points**

a. If anyone is super eager, you can earn up to 10 extra points if you create a GUI view.

b. Also up to 20 points if you create a pipelined version (this time you need to separate instruction and data memories), but the pipelined version scoredboard must track each stage.