CPSC3300 Project 3

In this project you are asked to simulate a CPU (as seen in textbook 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 textbook 4.4. You just need one memory that contains instructions and data together.
7. The control should track the # of each individual instruction.

Argument for input memory: A file generated by your Project 2. File should be read and loaded into the "memory" section. Addressing starts at 0x0.

**Things to avoid**

You should not just have one long "main". You should use structures/classes appropriately.

**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 are required to design memory well because we are going to plug the cache in later.

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

e. You should use object-oriented design principles.

**Language**

You are encouraged to use Java because it is easy to express object-oriented designs. You can also use C++. You should create a Makefile with "all" and "clean" targets. Simulator should take one argument - the program to simulate. You can add additional arguments or whatever (for single stepping etc), but ./smolmips myprog.bin should give the expected behavior, where myprog.bin is the output of the assembler from Project 2. make and that command should be enough to run the program - no other script setups.

**Bonus and Extra Points**

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