Program #2

Due Friday, Nov. 16, 2018 at 11:59pm

Collaboration

You must work on this assignment by yourself. You may not look up solutions for any part of this project from any resources (e.g. previous years' solutions, the Internet, other textbooks, etc.) including code you have written for similar assignments in other courses. You are allowed to look up C++ standard library information, and you are allowed to look online for explanations of compiler errors. You are allowed to discuss this project with the instructor, and peer tutors. You are not allowed to discuss the program with any other students or non-students until after the submission due date and time.

Description

For this assignment, you will determine the execution time of different sequences of MIPS assembly instructions when different pipelined datapaths are modeled.

Your main program should accept the name of a file to parse from as a command line argument. Each line of the file should have a correctly formatted MIPS assembly instruction (the instructions that must be supported are the same instructions as in lab #5) **or** a correctly formatted binary encoding of a MIPS instruction (the instructions that must be supported are the same instructions as in lab #5). However, you should make sure to check for correctness. If there are any errors in the input file, you should only print out an error message to stderr and exit.

If there are no errors with the input file, you need to determine the execution time for the following different pipelines:

1. Ideal – Ignore all data dependencies when determining total execution time
2. Stall – Model the impact of data hazards on the pipeline, assuming no data forwarding. Instructions should simply stall until values become available in the register file (assuming writing in first half of cycle and reading in second half of cycle).
3. Forwarding – Model the impact of data hazards on the pipeline, assuming data forwarding.

For all of these pipelines, you can ignore the impact control instructions would have on the next program counter and just assume the next instruction in the assembly file is the next instruction that will be executed.

Deliverables and grading

- Your grade will be based 50% on correctness, 15% on design, 10% on testing, 15% on style (including commenting), and 10% on your design document.
- You must use a Makefile that compiles your code with –Wall flag (you will lose points for any warnings) and your final executable should be named PIPESIM. In the directory ~kshaw/share/cs301/prog2 I've placed an example input assembly file and output file. **Your output must match the format in my output file or you will lose points for correctness.** (You will lose points if you print out anything other than what is indicated in my output file, so make sure you remove any debugging print statements.)
- You are required to reuse code (with modification) from lab #5, particularly the ASMParser, Instruction, OpcodeTable, and DependencyChecker classes. Your are welcome to use any other code I have given you or that you have written (for example, for program #1) to complete this assignment. And, you are welcome to use whatever STL classes you like.

- Your design document (named DESIGN) should explain at a high level your design. Meaning, it should describe what each class does (including any significant break down of functionality into new methods) and how the classes fit together. Please see the sample design document provided for lab #4. In addition to discussing each class individually, make sure you add significant methods added to these classes.
- **You are required to use inheritance in order to earn at least half of the design points. Meaning, you should have a base class named something like Pipeline which has two derived classes named something like StallPipeline and ForwardPipeline. The two derived classes only implement functionality that differs from the functionality in the Pipeline class; you should make sure to divide up your code into methods so that the smallest amount of code possible needs to be written in these derived classes. To make sure you have fully implemented inheritance correctly, make sure dynamic binding results in the correct version of an overridden method being called. (ex. `Pipeline *ptr = new StallPipeline(); ptr→overriddenFcn();` results in the version of overriddenFcn() in the StallPipeline class being invoked.)**
- Remember to follow all the code guidelines indicated in the class handout and follow good object oriented design practices. I am happy to discuss your design with you to give you suggestions for improvement.
- You need to submit at least one input file other than the one I am handing out. It should demonstrate that different sequences of instructions with dependences result in different execution times for the different pipeline implementations. Remember that different stall times occur depending on the instructions involved in the data dependence. For example, you'll stall longer if there is a load instruction producing a register value than if an add instruction produces the register value when data forwarding is used. The relative distance between the two dependent instructions will also impact the number of cycles stalled. Your test file should make sure to handle all of the possible different combinations that result in different stall durations. You must document your testing in your README file. (Note: since the number of instructions you're testing is small, your testing should be exhaustive.)

Note: I have not taught you how to implement inheritance in C++. You are far enough along in your academic career to be able to find language specific information on your own; you already understand the general concept of inheritance from CS221. I suggest you enter something like "C++ inheritance" or "C++ override method" into your favorite search engine and/or look at one of the library's e-books on C++ to determine how to implement inheritance. Once you have looked into various resources about inheritance in C++, you are welcome to come talk to me about this issue. However, you will have to explain to me what you learned, how you learned it, and what you don't understand.

Submitting Your Work

You must submit your work using a script called turnin. Remember to comment your code, to put your name in each file, and make sure you deallocate any dynamically allocated memory.

While you're in your solution directory, type the following command:

```
turnin -v -c cs301 -p prog2 *
```