CS 3339
Computer Architecture
Fall 2014

Project 4

Due: *Thursday, October 30 at 3:00 PM*

All project files have to be submitted using TRACS. Please follow the instructions at http://tracsfacts.its.txstate.edu/trainingvideos/submitassignment/submitassignment.htm. Note that files are only submitted if TRACS indicates a successful submission. See the end of this handout for what files need to be submitted. This project can be done *individually or in pairs*. You have to be able to explain all code that you submit.

4 MIPS Execution Time Simulation

Augment your interpreter from Project 2 such that it accurately *calculates* the running time of an 8-stage pipelined MIPS implementation with ID-stage branch resolution. No new code is provided. The pipeline has the following stages: IF1, IF2, ID, EXE1, EXE2, MEM1, MEM2, and WB.

4.1 Execution Time [100 points]

Have your code count the precise number of cycles it would take to execute the programs. Note that it initially takes seven cycles to start up the pipeline. Assume that no structural hazards exist and that there are no half-cycle register reads or writes, i.e., everything happens at the same clock edge. Further assume full data forwarding where possible. If forwarding is not possible, insert the minimum number of bubble cycles (stalls) necessary between the instructions in question until forwarding works. All instruction inputs are needed at the beginning of the respective stage. Most instructions need their inputs in the EXE1 stage, except *jr*, *beq*, and *bne*, which need them already in the ID stage, and the second *sw* input, which is only needed in the MEM1 stage (the base register is needed in EXE1). Instruction results become available at the end of the EXE2 stage, except *mult* and *div*, which become available in WB (i.e., cannot be forwarded), *lw*, whose result becomes available in MEM2, and *jal*, whose result becomes available in ID. For simplicity, let's assume that *trap* instructions follow the same timing as add instructions. See the previous project for which registers the different *trap* instructions read and write. The *jr*, *j*, and *jal* instructions are always followed by two flush cycles (stalls). The *beq* and *bne* instructions are followed by two flush cycles only if they are taken.

Print the *exact* number of cycles it would take to execute the programs on a CPU with the above parameters (i.e., until the last instruction exits the pipeline) as well as the number of flush cycles due to control transfers and the number of bubble cycles inserted due to data dependencies (see below for an example).

I recommend the following approach. Create two arrays whose elements represent the pipeline stages. One array records which register, if any, is the destination of each instruction in the pipe-line. The other array records, for each instruction in the pipeline, in which stage it generates its result (treat *$hi* and *$lo* as one register since they are always written together). A flush overwrites an existing pipeline entry with a NOP that does not write any register. Similarly, a bubble inserts a NOP into the pipeline. Whenever an instruction needs to read an input register, determine, based on the content of the two arrays, whether it will be able to do so in time (either from the register file or from the forwarding network) or whether it has to wait, in which case the appropriate number of bubbles needs to be inserted. Note, the *$zero* register is always available. Do everything from the point of view of the control unit, i.e., in the ID stage. I suggest calling a function for each source to which you pass the source register number as well as the latest stage where this input is needed. Similarly, call a function for each instruction to which you pass the destination register number, if any, as well as the earliest stage in which the result will become available. Check your results using the following equation: instrs = cycles - 7 - bubbles - flushes.

The following is the expected result for *sssp.mips*. Your output must match this format verbatim.

```
CS3339 -- MIPS Runtime Simulator
running sssp.mips

 7 1

program finished at pc = 0x400440  (449513 instructions executed)
cycles = 993588
bubbles = 492078
flushes = 51990
```

Code Requirements

- Make sure your code compiles with gcc and runs on zeus.cs.txstate.edu.
- Make sure your code is well commented.
- Make sure your code does not produce unwanted output such as debugging messages.
- Make sure your code's runtime is not excessive.
- Make sure your code is correctly indented and uses a consistent coding style.
- Make sure your code does not exceed array bounds.
- Make sure your code does not include unused variables, unreachable code, etc.

File Submission

- Delete all files that you do not need anymore such as core and *.o files.
- Make sure your code complies with the above code requirements before you submit it.
- Any special instructions or comments to the grader should be included in a "README" file.
- Upload your final **`cycles.c`** file (all code you write has to go into this file) and the optional **`README.txt`** file onto TRACS. Both files have to be text files, not PDF etc.
- For group projects, include a comment at the beginning of the source code that lists both group members. The two group members have to submit identical files.
- Upload each file separately and do not compress them.
- Do not submit any unnecessary files (e.g., provided or generated files).

You can submit your file(s) as many times as you want before the deadline. Only the last submission will be graded. Be sure to submit at least once before the deadline.

October 23, 2014