This assignment is worth 6% of your grade in the class and is graded out of 100 points.

VeriSimple4 is a simple pipelined implementation of a subset of the Alpha instruction set, written in synthesizable, behavioral Verilog. The structure of VeriSimple4 is very similar to the MIPS pipeline covered in the text. We have provided you with a base version of VeriSimple4 that has absolutely no hazard detection logic. This version of Verisimple4 inserts 4 invalid instructions after every instruction to remove any possibility of any hazard. You will need to change that so that more than one valid instruction can be found in the pipeline at a time. This change will be in if_stage.v and is discussed in the pipeline slides.

You are to extend the VeriSimple4 pipeline to support full hazard detection and bypassing. Your solution is subject to the following restrictions:

- Branches should resolve in the same stage they are currently resolved in.
- All forwarding must be to the EX stage, even if the data isn’t needed until a later stage.
- Any stalling due to data hazards must occur in the decode stage. (That is, if stalling is required the dependent instruction should stall in the decode stage.) Obviously, instructions following the stalling instruction in the IF stage will have to stay in the IF stage. Put another way, if you need to insert an invalid instruction, it should be inserted in the EX stage (as was done in the slides in the first few lectures this semester.)
- If you wish to insert anoop you must invalidate the instruction. Otherwise your CPI numbers will be wrong.
- If there is a structural hazard in the memory, you should let the load/store go and have the fetch stage wait on getting memory.

Add control logic to detect all possible structural, control and data hazards. Add bypass paths (forwarding) to eliminate data hazards whenever possible (within the rules stated above), and stall instructions if (and only if) there is a data hazard that cannot be resolved by bypassing. You should predict branches are not taken and squash if incorrect. Verify that your (faster) pipeline produces the same results (in memory) as the original version does.

Your synthesized version of the code (.vg files) as well as your behavior model (.v file) will be tested, so it does need to synthesize. You will also submit your Verilog code electronically using the same process as in previous programming assignments.

The grading will involve a number of test programs being run and us checking to be sure you’ve gotten the correct results, correct CPI, correct pipeline output, and are actually following the directions specified above. Additional files, including the VeriSimple4 files can be found on the website.

Other notes/hints:

- Be careful with forwarding and register 31!
- Synthesized runs of the pipeline can take as long as 4 minutes on the class machines, probably longer on other boxes. Be patient.
- There is a LOT of Verilog code. Take the time to try to understand how things work. The discussion slides are helpful as a starting point.
- Start early!