EECS 470 Term Project

The term project is to build on the VeriSimple4 Alpha pipeline to create a more advanced pipeline with a few of the features we are studying in class. Projects will be done in groups of three to five students. All groups are required to implement certain "base" features and these base features vary by the number of group members. In addition, you will have the opportunity to add more advanced features to improve the performance of your pipeline. A significant portion of your grade will depend on the absolute performance of your pipeline (CPI and clock rate). The project is worth 25% of your course grade. Your project grade will be the weighted average of scores based on the following areas:

1. Implementation of base features: 20%. Did you implement all of the base features listed below?

2. Correctness and testing: 20%. Does your pipeline correctly implement the ISA, and did you convince us of that through your testing methodology?

3. Performance: 20%. How well does your pipeline perform on the test benchmarks provided (including, but not limited to, the ones used for Homework 3)? Performance will be calculated using the CPI derived from the Verilog simulator and the timing reported by the synthesis tool.

4. Additional features: 20%. Extra points for those who attempt ambitious designs. Ideally these points will be in addition to the performance points that these features provide. In the worst case, they are points for trying something bold that didn't quite work out.

5. Analysis: 10%. Did you uncover the impact of your features on performance? For example, on a superscalar machine how many instructions do you complete per cycle? What is the prediction accuracy of your branch predictor and/or BTB? How full is your ROB? If you add an interesting "Additional feature" it would be nice to learn how successful it is with respect to performance. Note that your grade won't suffer (or improve) from showing us that something is actually a **bad** idea. What we want to see is that you can measure how good or bad the idea/feature was. This data will show up in your report.

6. Documentation: 7%. Did your report describe your design, the motivation for your design decisions, your testing methodology, and your performance evaluation in a readable, concise manner? Although the documentation itself counts for only 7% of the project grade, I will be basing your scores for the other areas on the information you provide in your report. A poorly written report could bring down your scores in all areas.

7. Check-points: 3%. Did you meet the requirements of the first check-point? Was the module a reasonable one and did it work?

The proposal should include a list of the group members (with email addresses), optional features the group plans to implement (including motivation for choosing those features), an initial assignment of which group members are primarily responsible for which components, and a schedule with specific milestones to be achieved by each of the checkpoints (see below). This document isn't a contract, it is likely you will be changing your targets and goals as the semester goes on.

There will be two project checkpoints h

For both checkpoints you are to submit a brief (one-page) report indicating progress to date, progress relative to the original schedule, and any changes in the scope or direction of the project relative to the original proposal. In addition for the first check-point you are to turn in a working module for some substantial component of your project (ROB, BTB, rename, dependency checking for a superscalar, etc.). For this check-point you must also include a testbench for this module which checks if the module works correctly or not and the module must be able to synthesize.

By the second checkpoint, you should have all of the major components developed, synthesized and tested in isolation, and be well underway with the integration of these components into your final pipeline. All project groups must turn in a report detailing their findings on the last day of class (Wednesday April 21st). The report should be about 10 pages in length and include an introduction, plus details on the design, implementation, testing, and evaluation (analysis) of the new features. Your Verilog design will also be handed in and tested electronically.

Minimum requirements:

1. **I and D cache**. The base memory will have 100ns latency associated with it. You will be required to build an instruction and data cache to improve this. A modules for the main memory will be provided as will a basic I-cache.

2. **Multiple functional units with varying latencies.** You should split the Verisimple3 integer ALU into multiple units with different functions and potentially different latencies to improve your cycle time. Most integer operations (other than multiply) should take 1 cycle to execute. Branch target calculations and effective address calculations could also be split into separate units. Use the synthesis tool to guide your decisions. For your multiplier you are to use the one provided in programming assignment 2 although you may change the degree of pipelining as needed

3. **Either a superscalar implementation or an out-of-order implementation.** For the superscalar this means you have to have the *theoretical* ability to sustain completing two instructions per cycle. (There must be some code on which you can do this.) For the out-of-order implementation you need to be able to send instructions to the execution stage in an order other than program order. Your report must include a code segment that demonstrates this capability. Note that “better” and/or non-standard out-of-order implementations may count as advanced feature points.

Groups of 4 or 5 must implement some form of dynamic branch prediction complete with some means of predicting the address.

Groups of 5 must also implement **one** of the following.

1. The ability to process two load misses in parallel.
2. Out-of-order other than Tomasulo's I or II.
3. A next-line pre-fetch mechanism for the caches
4. A victim cache of at least 2 lines.

In addition to the varying requirements for different group sizes, Smaller groups will have a *slightly* easier time getting advanced feature and analysis points.

Optional features could include improvements to the above, doing more of the above then required, or something else novel. Some ideas include:

1. Deeper pipelining: can you reorganize the pipeline into more stages (or balance the stages better) to achieve a higher clock rate?
2. Dynamic scheduling using a scoreboard, reservation stations, etc.
3. Dynamic scheduling using Tomasulo's algorithm (T1, T2, or T3)
4. Maintain precise exceptions for an out-of-order implementation.
5. Fetch enhancements: return address stack, etc.
6. Memory hierarchy: write buffers; writeback vs. write-through data cache; non-blocking L1 data cache; dual-ported, banked L1 data cache (supports two accesses per clock if to independent banks);
7. Hardware and/or software prefetching for instructions and/or data.
8. Speculative execution allowed. The ability to complete execution but then be squashed before the machine state is changed. This should include the ability to speculate past branches.
9. Really hard things (at least to do right). Note that of these require speculative execution to be implemented before you can really do much.
   a) A value predictor
   b) Run-ahead execution

   c) A Cyclone-like scheduler

   d) Pipelining the dynamic execution logic

   e) A trace cache.