EECS 470 Term Project Fall ‘07

The term project is to build on the VeriSimple4 Alpha pipeline to create a more advanced processor 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 30% of your course grade. Your project grade will be the weighted average of scores based on the following areas:

1. **Implementation of base features:** 25%. 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 programming assignment 3)? Performance will be calculated using the CPI derived from the Verilog simulator and the timing reported by the synthesis tool. *If you don't get synthesis working it will be very difficult to earn points here!*

4. **Additional features:** 15%. 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/Presentation Quality:** 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? Were you prepared for the second check-point?

A one-page project proposal is due **Friday 10/12**. The proposal should include a list of the group members (with email addresses), a group name (which we will use to name your directory in the source code control system, so group names may not contain spaces or special characters), base design details, optional features you are considering (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 (e.g., which component will you be implementing for checkpoint 1; see below). This document is not a contract; it is likely you will be changing your targets and goals as the semester goes on. However, the more detail you can give us up front on your plans, the better the feedback and advice you will receive from us will be.

The first project checkpoint is due on **Monday 10/29**. You must 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 you are to turn in a working module for some substantial component of your project (ROB, RS, BTB, etc.), along with a testbench for this module. The module must also be able to synthesize.*

The second project checkpoint is due on **Wednesday 11/14**. Another brief progress report is required, as for the first checkpoint. At this checkpoint you should plan to have your basic components integrated into a functional pipeline such that at the very least one instruction can be fetched, decoded, executed, and committed.

Your final project Verilog code (to be submitted electronically for testing) and written project report is due on **Monday 12/10** (the last day of class). The project report should be about 10 pages in length and include an introduction and details on the design, implementation, testing, and evaluation (analysis) of your pipeline, including specific discussion and analysis of any advanced features. You will also give an oral presentation on your project in class on that date. Oral presentations will be brief (10-15 minutes) and relatively informal.

**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. 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 Verisimple4 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. **An 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 nonstandard 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. Fast recovery from mispredicted branches.
3. Demonstrate recovery from an exception (e.g., null pointer load or division by zero).
4. A next-line pre-fetch mechanism for the caches.

**Advanced features:**
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. Issue memory accesses out-of-order (while still giving the correct results of course!). May be conservative or speculative here…
2. Superscalar dispatch, issue, and retirement.
3. Fetch enhancements: more sophisticated branch predictors, return address stack, etc.
4. 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); Hardware and/or software prefetching for instructions and/or data.
5. Data forwarding from loads to stores
6. Implement the Alpha conditional move (CMOVxx) instruction. This would include modifying at least one of the benchmarks to take advantage of this instruction.

Really hard things (at least to do right). Note that these require speculative execution to be implemented before you can really do much.
1. A value predictor (http://citeseer.nj.nec.com/lipasti96value.html)
2. A sophisticated hardware prefetcher
3. Run-ahead execution (must be on a umich machine to get to this paper.) (http://zizzer.eecs.umich.edu/papers/2003/hpca/mutlu_runahead.pdf)
4. Pipelining the dynamic execution logic (must be on umich machine to have permission to read these papers.) http://zizzer.eecs.umich.edu/papers/2000/micro/pipelining-scheduling.pdf
5. A trace cache.

Please feel free to come up with your own features. Just be sure to talk to us first….

**Other stuff**
- We will provide a starting point for a couple of modules. While you may or may not directly use these modules, they provide some helpful guidance for your project design. This information will be posted soon after P3 is turned in. You may reuse freely from P3 (the decoder is a good thing to not have to re-write).
- We recommend you stick with the directory structure of P3
- You **will** need to use a revision control tool.
  - We will provide subversion (svn) server for use by the class
  - The svn server will be put up shortly after you hand in your project proposals (once we know your group names)
  - You will hand in your project using the svn server (we will post hand-in instructions later in the semester)