EECS 470 Term Project Fall 2022

The term project is to build on the VeriSimpleV RISC-V 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 five or six students. All groups are required to implement certain "base features" and these base features vary by the number of group members and your group's interests. 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 35% of your course grade. Your project grade will be the weighted average of scores based on the following areas:

1. Implementation of base features: 23%. 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: 17%. 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? Were you prepared for the second and third check-points?

A one-page project proposal is due on Monday 10/10 turned in as a group assignment on Gradescope. The proposal should include a list of the group members (with email addresses), 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). It should also include how the group is planning on communicating (slack, e-mail, etc.). 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 10/24. You will submit this via the grader (as project 4) and via git. 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 self-checking testbench for this module. The module **must** be able to synthesize. You should have at least a few modules done at this point and you should have a detailed diagram of your design. Your team must be using git and checking things in on a regular basis.

The second project checkpoint is due on Thursday 11/10. 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 most non-memory operations can be correctly fetched, decoded, executed, and committed.

The third checkpoint is due on Friday 12/2. Another brief progress report is required though there will be no required group meeting. At this point you should have the entire project working in simulation for more than 75% of the testbenches and each component synthesizing correctly.

Your final project Verilog code (to be submitted electronically for testing) is due on ~~Thursday 12/8 at 8pm~~. The written project report is due on Friday 12/9 by 8pm. The project report should be about 10-20 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. Oral presentations will also be held on or around that same date (depending on scheduling, but expect 12/9). Oral presentations will be brief 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. Your project must support in-order commit or otherwise provide a mechanism for exceptions to be correctly handled.
4. **Branch prediction with address prediction**

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. In order to earn full advanced feature points, each group should implement one difficult advanced feature and a few other advanced features. Some "difficult" advanced features are listed below, <u>though you should feel free to suggest others</u>. Those with *s are treated as especially difficult.

✓

- Superscalar execution (3-way*, arbitrary **)
- Exception handling*
- Simultaneous Multi-threading (SMT) with 2-way superscalar execution ***
- ✓ Early branch resolution (before the branch hits the head of the RoB)
- Multi-path execution on low-confidence branches (this may not help performance much…)
- Speculating on load dependencies and forwarding optimally in nearly all cases.

Some other, simpler, additional features are listed below. Some like pre-fetching are fairly trivial, while others can be more complex depending on the implementation. Talk to the class instructor to get a good sense of difficulty, though the ones with + here are generally *simpler* and tend to give a solid return-on-investment.

- Fetch enhancements: more sophisticated branch predictors+, return address stack, etc.
- Memory hierarchy: non-blocking L1 data cache; associative caches+; dual-ported, banked L1 data cache (supports two accesses per clock if to independent banks); Hardware prefetching for instructions and/or data+.
- Add new instructions.
- Having a really good GUI debugger+.

Grading of advanced features

Out of the 17 points for advanced features, the ones without a * are generally worth 6-8 points, the ones with a * are generally worth 8-10, the ones with ** are generally worth 10-12 and the ones with *** are generally 12-15. The simpler features are worth 0.5 to 3 points. If you should earn more than 17 points in this category, any points earned over 17 will be divided by 3. So if you end up with 23 points in advanced features, we'll give you 19. We suggest groups target one difficult feature and 2-3 simpler ones while having a few more ones ready to go in the event you have time to add them. Going for 2 difficult features has been done in the past, but please talk to us about the risks associated with that. *In all cases, the quality of your implementation of each feature will play a major role in how we score it.* We suggest you target earning 16-18 points in this category. Groups of 5 get 2 extra advanced feature points and 1 "extra" point (not capped). This levels the playing field a bit, but groups of 6 still have an advantage.

Other stuff

We have a whole document of suggestions but here are some highlights:

- **We will provide a starting point for a couple of modules**. While you may choose to 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 **<u>will</u>** need to use git and make your repo available to the 470 staff.
- **Code your first module as a group**. It will help immensely with getting everyone on the same page and working out coding standards as a team. For larger groups this can feel like a waste of time—it is worth it in the end. Classrooms have been found to be great places to gather for initial meeting. They have projectors and many have whiteboards.