EECS 470 Term Project Winter '21

The term project is to build on the VeriSimple4 RISC-V pipeline from project 3 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 students (other sizes must be approved by staff). 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). 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. Note that for every test program that your design does not complete correctly, we factor it into your relative performance as *10% slower than the worst-performing correct design.* So correctness is far more important that performance – you can't get performance points until your design works. Furthermore, we need an accurate cycle time for your design, so you can't get *any* performance points if your design does not work correctly after synthesis.

4. Optional 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. There are guidelines of typical optional features we've seen in the past (and their approximate point values). However, you are encouraged to propose something new.

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 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 to the course staff via Gradescope by 11:59 pm on **Friday 2/26**. The proposal should include:

- A list of group members (with email addresses)
- Base design details (are you planning on implementing a P6-style design? R10K? Something else?)
- Optional design features you are considering
- A plan for how often you will meet, using what medium (e.g. Zoom, VS-Code live, etc)
- An initial assignment of how the workload will be distributed and who is responsible for what
- A schedule for which specific milestones will 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. We will schedule meetings with groups the following week to discuss your proposal.

The first project checkpoint is due on **Thursday 3/11**. You will submit this by making a specified commit to Bitbucket. 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 (we recommend the Reservation Stations), along with a testbench for this module. The module must also be able to synthesize.

The second project checkpoint is due on **Thursday 3/25.** 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, with an Instruction Cache.

The third checkpoint is due on **Tues 4/6**. 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 **Friday 4/16 at 11:59pm**. The written project report is due on **Monday 4/19 by 11:59pm**. The project report should be about 8 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. Details on oral presentations will be given towards the end of the semester.

In addition, group members will periodically fill out peer-evaluations on the relative contributions of the group members. This will not be used to "micro-manage" points at the end of the project (all members will usually receive the same grade except in unusual circumstances), but we would like to identify group dynamic issues early on. Please come to the staff early if you have concerns about keeping all group members productive.

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. **Both your instruction and data cache may not be larger than the current size of 256 bytes, each (so 512 bytes in total).**

2. **Multiple functional units with varying latencies.** You should split the supplied 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 (although you only need to demonstrate branch misprediction recovery). **The number of CDBs in your design is limited to the superscalar-ness of your design. E.g. if you are designing a 3-way superscalar design, you may not have more than 3 CDBs.**

4. **Dynamic branch prediction.** You must implement, at a minimum, a branch target buffer and a bimodal branch predictor. More sophisticated predictors garner extra feature points.

5. **Advanced features.** You must attempt one of the ~~starred~~ "difficult" features below.

6. **Submit a Test Case to the Autograder**. Each group must submit a meaningful test case to the autograder. More details are at the end of the document

Advanced features:
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, though you should feel free to suggest others. Those with *s are treated as especially difficult.

- Superscalar execution (3-way*, N-way **)
  - I.e. issue, execute, retire width >1. Scalar width is minimum width of entire pipeline
  - Note that you may not include more CDBs than the narrowest part of your design (see above)
- 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…)
- Early tag broadcast
  - Speculative scheduling of instructions dependent on loads hits*
- Loads issue speculatively past pending stores

Some other, simpler, additional features are listed below. Some like prefetching 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.

- Issue memory accesses out-of-order (while still giving the correct results of course!) using an LSQ.
- Data forwarding from stores to loads.
- Fetch enhancements: more sophisticated branch predictors, return address stack, etc.
- Memory hierarchy: non-blocking L1 data cache; associative caches+; victim cache; dual-ported, banked L1 data cache (supports two accesses per clock if to independent banks); Hardware prefetching for instructions and/or data+.
- Having a really good GUI debugger+.
- Automated regression testing infrastructure

Grading of advanced features
Out of the 15 points for advanced features, the ones without a * are generally worth 6-8 points, the ones with a * are generally worth 8-10 and the ones with ** are generally worth 10-12. The simpler features are worth 0.5 to 3 points. If you should earn more than 15 points in this category, any points earned over 15 will be divided by 3. So if you end up with 21 points in advanced features, we'll give you 17. 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 14-16 points in this category. Approved groups of 4 get 2 extra advanced feature points.

Additional details for RISC-V:

You will support RV32IM, without fences, division, CSR operations and system calls.

Each group must submit a test case, written in assembly or C. The requirements for this test case will be:

- 2500+ instructions executed
- Maps to meaningful data structure, algorithm or workload
- Must include only supported instructions (no divides, fences, etc)
- No self-modifying code or stack overflow

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 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 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. Plan on a strategy for meeting and working remotely (see Piazza for a list of suggested tools)