CSEE 4824 Final Project - Spring 2024

The final project is to build on the VeriSimpleV RISC-V pipeline from project 3 to create an out-of-order processor based on the designs we've gone over in class. Projects will be done in **groups of five or six students**. All groups are required to implement an out-of-order processor with the base features listed below. Each group will also implement multiple advanced features based on the group's interests to improve performance. A significant portion of your grade will depend on your processor's absolute performance (CPI and clock period).

1 Grading

This project is worth 35% of your course grade and will be graded by the weighted average of scores in the following areas:

1. **Implementation of base features** (23%): Did you implement all of the listed base features?
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.
4. **Advanced features** (17%): Points for ambitious designs and interesting implementations. 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. These are calculated out of 17 points based on what advanced features you implement, see below for details.
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 advanced 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, your scores for other areas will be based on the information you provide in your report. A poorly written report could bring down your scores in all areas.
7. **Milestones** (3%): Did you meet the requirements of the first milestone? Was the module a reasonable one and did it work? Were you prepared for the second and third milestones?

2 Project Proposal

A one to two-page project proposal is due **Thursday 3/21 at 11:59 pm** on CourseWorks. The proposal should include:

- Your team number (and a name, if you like)
- A list of group members (with UNI and email addresses)
- Base design details (are you planning on implementing a P6-style design? R10K? Something else?)
- Advanced features you are considering
- A schedule for which specific milestones will be achieved by each of the milestones (e.g., which component will you be implementing for milestone 1)

- A "group charter", outlining your expectations for one-another, including:
  - How many hours per week are expected per person
  - When during the week each person is typically free and when they are not
  - When the team plans on doing most of its work
  - When and on what the team members will work as a group, and when they will work individually
  - Who will largely be responsible for which tasks. This should include technical things (e.g. designing the branch predictor, writing tests for the LSQ, etc.) as well as non-technical things (scheduling meetings, taking minutes, etc.)
  - Known conflicts that will take each person away from their work for a prolonged period (off-site interviews, family gatherings, holidays, etc.) or other major class deadlines that will take a person away from the project for more than a day
  - When and where team meetings will be held. In addition to a regular "work" meeting schedule, we recommend scheduling at least one non-work meeting a week to touch base and to interact without formally discussing the project. Holding such meetings over lunch is often a good idea. It can help with communication and morale. Each person should sign this document (digitally is fine) indicating that they have read and agree with it.

Your proposal is not a contract; it is likely you will be changing your targets and goals as the semester goes on, but changes should be made with group consensus. 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 to discuss your proposal.

Group members will also periodically fill out peer-evaluations on whether their partners are participating in the project and acting respectfully towards one-another. 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.

3 Milestones and Submission

There will be multiple milestones due for this project where you will submit progress reports and meet with the instructors about your status on the project. The second and third milestones are not hard requirements, but recommended targets for your group to hit.

The first milestone is due **Thursday 3/28**. For this milestone, you are to turn in a brief progress report and a working module for some substantial component of your project (we recommend the Reservation Station, although Reorder Buffer is another good option). Your brief (one-page) report should indicate your 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. Your module must have a corresponding testbench, and be able to generate correct output in simulation (turn it in by creating a specific branch of your git repository with the working module).

The second milestone is due **Thursday 4/4**. Another brief progress report is required, and your basic components (including Instruction cache) should be integrated into a functional pipeline such that most non-memory operations can be correctly fetched, decoded, executed, and committed.

The third milestone is due **Thursday 4/11**. Another brief progress report is required though there will be no required group meeting. At this point you should have begun working on memory operations, and should have the project working in simulation for around 75% of the test programs.

Your final project Verilog code (to be submitted electronically for testing) is due **Thursday 4/18 at 11:59pm**. The written project report is due **Thursday 4/25 by 11:59pm**. 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.

There will also be brief, relatively informal oral presentations at the end of the semester. More details about these will be given later.

4 Processor Requirements

4.1 Base Features (required)

1. **A base out-of-order processor.** You will be implementing one of the out-of-order designs covered in class, either P6 or R10K style, although nonstandard designs may be approved. Specifically, your design will need to be able to send instructions to the execution stage in an order other than program order and your report must include a code segment that demonstrates this capability. Your project must support in-order commit or otherwise provide a mechanism for exceptions to be handled correctly (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.**

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. 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. **Instruction and Data cache.** The base memory will have a 100ns latency associated with it. You will be required to build separate instruction and data caches to improve this. The main memory module will be provided, as will a basic I-cache module. **Your instruction and data cache may not be larger than 256 bytes each (so 512 bytes in total).**

4. **Branch prediction with address prediction.** You must implement, at a minimum, a branch target buffer and a bimodal branch predictor. More sophisticated predictors may count as advanced feature points.

4.2 Advanced Features (17 points, at least one difficult)

Advanced features are graded out of 17 points (with any over 17 counting as $\frac{1}{3}$). Each group must implement at least one difficult advanced feature and some other advanced features. We suggest that you target earning 16-18 points total. In all cases, *the quality of your feature implementations will play a major role in how we score it.*

Some "difficult" and "simple" advanced features are listed below (though you should feel free to suggest others!). Those with more stars (*) are treated as increasing in difficulty. Difficult features are generally worth 6-8 points plus 2 per star (*8-10, **10-12, ***12-15). The simpler features are generally only worth 0.5-3 points. However, these numbers are not absolute and will depend on implementations. Going for multiple high difficulty features has been done in the past, but please talk to us about the risks associated with that. Groups with fewer than 5 people (dropping the class, life issues, etc.) will have a reduced requirement for advanced feature points. Also, better and/or nonstandard base feature implementations may count as advanced feature points - talk with staff if you have any ideas here. Lecture numbers (e.g. L[N]) and textbook chapters (e.g. H&P [N]) for some features are notated as a reference. You are also encouraged to do your own research or propose ideas not listed here.

Difficult Advanced Features (estimated: 6-8 points, +2 per *):

- Superscalar execution (2-way, 3-way*, N-way**) (L5, L7, H&P 3.8)
  - 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)
- Simultaneous Multithreading (SMT) with 2-way superscalar execution*** (H&P 3.12)
- Early branch resolution (before the branch hits the head of the RoB) (L9)
- Multi-path execution on low-confidence branches (this may not help performance much...)

- Early tag broadcast (L7)
  - Optionally, with speculative scheduling of instructions dependent on load hits
- Speculating on load dependencies and forwarding optimally in nearly all cases (L10, H&P 3.6)

Simpler Advanced Features (estimated: 0.5-3 points):

Some simpler features like prefetching are fairly trivial, while others can be more complex based on your implementation. Talk to staff to get a good sense of difficulty. The ones with a † are generally simpler but tend to give a solid return-on-investment in either performance or debugging.

- Fetch enhancements (H&P 3.9)
  - More sophisticated branch predictors †
  - Return address stack
- Memory hierarchy improvements (H&P 2)
  - Instruction and/or data prefetching †
  - Dual-ported, banked L1 data cache (supports two accesses per clock if to independent banks)
  - Associative caches †
  - Non-blocking L1 data cache
  - Victim cache
- Issue memory accesses out-of-order (while still giving the correct results of course!) using a Load-Store Queue (H&P 3.6)
- Data forwarding from loads to stores (H&P 3.6)
- Having a really good GUI debugger †
  - This entails something beyond just dumping the state of your machine to text files. You should be able to pause, step through cycle-by-cycle, and graphically view the state of various components of your machine. Ncurses is a popular choice.

5 Final Suggestions

Here are some final suggestions for getting started:

- **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).

- **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).

- **Read the updated README and use the Makefile.** The README for project 4 has been updated with instructions for using the new module testbench system in the Makefile. This will help with managing all the different modules and testbenches you will be writing.