**ECEN 4593 Fall 2019 – Class Project Final Report**

1. **Project Summary**

In this project, we simulated a RISC-V CPU that was both instruction-accurate and timing-accurate. We started with learning the basics of RISC-V assembly, both in terms of structure and how the language is processed, and then began building up a diagram and the related code. The class and accompanying project phases gradually introduced new challenges and issues – data hazards, simulation of physical bit flipping, cache systems – and then helped us understand how to solve those problems. We also were exposed to how RISC-V is changing the industry and will be showing up in our professional lives for years to come, including its advantages over other architectures and why it is a good fit for up-and-coming technologies.

1. **Challenges**

The first challenge that this project presented was learning how to handle the CODAL language and translating our hardware diagrams into code. This required us to intimately understand how RISC-V works, from instruction binary translation to pipelining. This is something that was learned with time and with lots of support from both phase instructions and the professor, but it requires a mental switch and newfound understanding of the software. Next and related to the first challenge was debugging. I frequently had one line of code or one bite missing, and the structure of the simulation changes as a result. The “sleep on it” method is great for debugging, allowing you to return to the code with fresh eyes later. The final challenge came in the form of time – the pacing of the phases is admittedly relentless, but it encourages students to keep up with the class and instruction material. I tended to reserve one set time – Sunday evenings – to work on the project. This helped me remain focused on the tasks but also was close to the semi-regular Sunday 10 PM deadlines and largely prevented me from following my own advice to “sleep on it”.

1. **Takeaways**
2. As the class pounds into our heads, DEBUG, DEBUG, DEBUG. Having a methodical process to check for errors – for example, retracing steps using lab guidelines or going through the entire pipeline to ensure all registers/signals are processed properly – allows one to double-check themselves and avoids frustration when things that *should* work do not.
3. I really appreciated the project structure that deadlines are not firm but HIGHLY recommended – it felt more similar to what I’ve heard from industry and helped me feel more comfortable asking for help when I was struggling, even after the deadline had passed.
4. This one is kind of sappy, but: it is amazing how simple fundamentals build up to create a functioning computer. With what boils down to a few MUXes and some form of data storage, a basic system can start doing operations that would have taken humans thousands of seconds as compared to less than one.
5. **Suggested Improvements**

The only major item that would improve this class would be a solid debug process document with common errors. For example, Phase 2 would start the debug document with suggestions about ensuring that all the bits are present in the instruction pieces and that everything is concatenated properly. Then, as the pipeline structure is introduced, items like “Ensure that all widths are defined and all signals have proper widths” or “Ensure that registers are pipelined using the {pipeline} command” can be added. These simple errors clearly make a difference and save both the student and grader time ensuring that all requirements are met!