ECE 411

Spring 2020

Team letUsGraduate

MP3 Final Report

Kevin Lee (changml2), Taras Ambrozyak (tarasa2), Shounak Ray (sray10)

Introduction

The goal of this project was to design and implement all the components needed for a RISC-V, 5-stage pipelined microprocessor. Along with having five pipelined stages, the processor was to have two levels of cache, hazard detection, data forwarding, and branch prediction. Each of these components were added in order to improve the efficiency of our processor. Being able to design the processor and its components on our own allowed us to get a deep understanding of, not only how each individual component works and why they are essential, but how each of the components must interact together in order to have a working, pipelined processor.

Project Overview

Initially, this project started out as a blank canvas of sorts. Our group was required to have a number of set functional components in our design, but we were free to choose any additional features that we desired to implement. The goal was simple in the beginning; we wanted to have a baseline design done with some advanced features implemented by the deadline. Therefore, our plan going forward was to finish all of the baseline components and rendevouz to discuss the advanced features that we wished to implement. Our actual execution of this project did not follow this plan. The workload was initially evenly assigned to each person, with certain people having specialties that rounded out to be about the same amount of work overall. Taras was in charge of most research and design components, while Kevin and Shounak were in charge of implementation and debugging. This quickly fell apart by the end of Checkpoint 1. At the end of Checkpoint 1, Shounak went missing until around the last week of the project. This meant that the workload now had to be readjusted to be done by two people. Taras and Kevin were now the primary contributors of the project, meaning that Taras and Kevin now had to inherit the many responsibilities originally assigned to Shounak. This led to several complications in our design, the most prominent complication being a lack of time and too much work. As such, our group started falling behind starting at Checkpoint 2, and we never really recovered. The lack of time bled into our advanced design features as well, meaning that with each passing day, we had less and less time to work on any advanced features. By the last week of the project, it became very clear to us that there was no longer an option to design any advanced features, and therefore our goals changed. We had to sacrifice the advanced features in order to get the baseline design working, and that's just what we did. Our group managed to get the baseline design working by the final deadline in exchange for not having implemented a single advanced design feature.

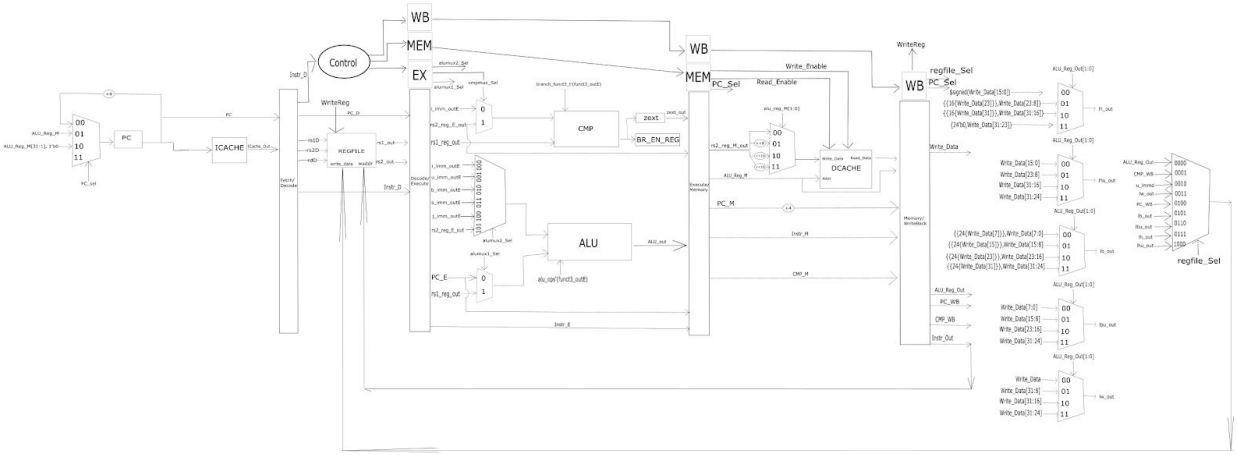
Milestones

During this project, we had four major checkpoints.

Checkpoint 1

In the first checkpoint, we designed our basic, pipelined processor to execute basic RV32I ISA instructions. The five stages we implemented for our processor were the fetch stage, decode stage, execute stage, memory stage, and writeback stage. Between each stage, we had sets of registers so our processor would be able to carry over an instruction’s specific values, such as

the PC and the ALU output. Once we believed that our design was correct, we used the provided test code and modelsim to test if our processor would execute the instructions correctly. Initially, we had a problem with our branch instructions as our PC value would branch one instruction too far. We realized that this was caused by the fact that we passed in the incorrect stage's PC value and were able to fix this quickly. Once the branch instructions were working, we were able to complete the provided test code, and from there, we tested further functionality by changing the instructions in the provided test code. These tests showed us that we were able to execute the basic instructions of the RV32I ISA in our basic, pipelined processor. We then began working on our arbiter and split L1 cache designs. Our L1 caches were designed to be split, one for storing instructions and one for storing data. For this design, we planned on using our cache design from MP2, one for the instructions and one for data. In order to allow the caches to communicate with main memory, we needed an arbiter. We designed our arbiter to take the outputs from our two caches and pass them to our cacheline adapter, which would communicate with the main memory. Having the arbiter allowed our processor to ensure that only one of our two caches would try to access the main memory at a time. Once we finished these designs, we began working on Checkpoint 2.

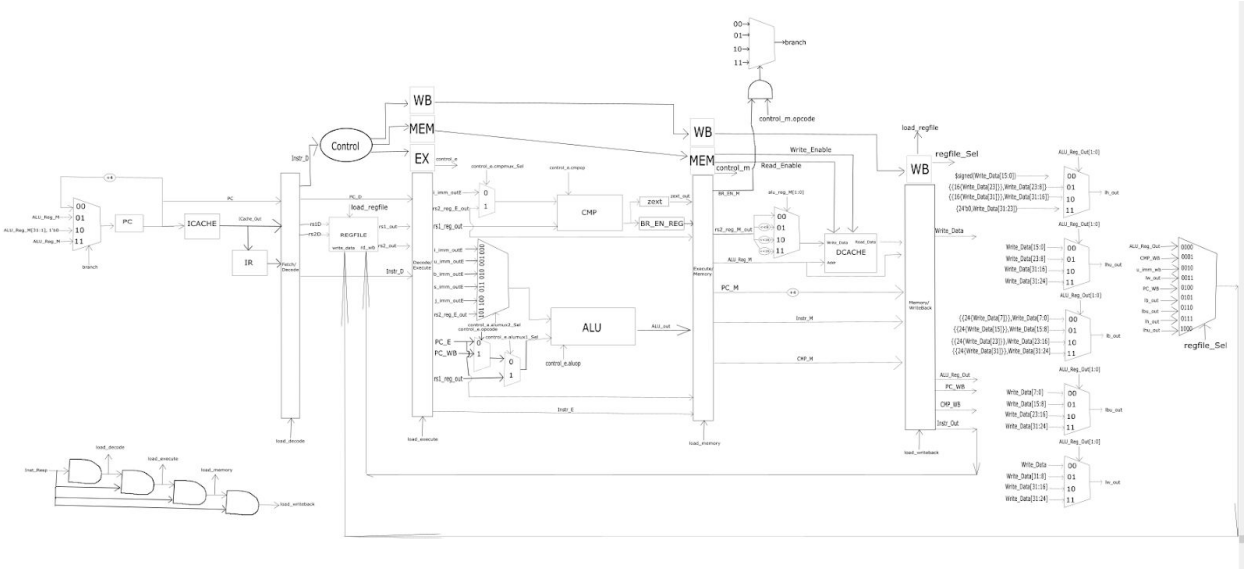


Checkpoint 1 Pipelined Design

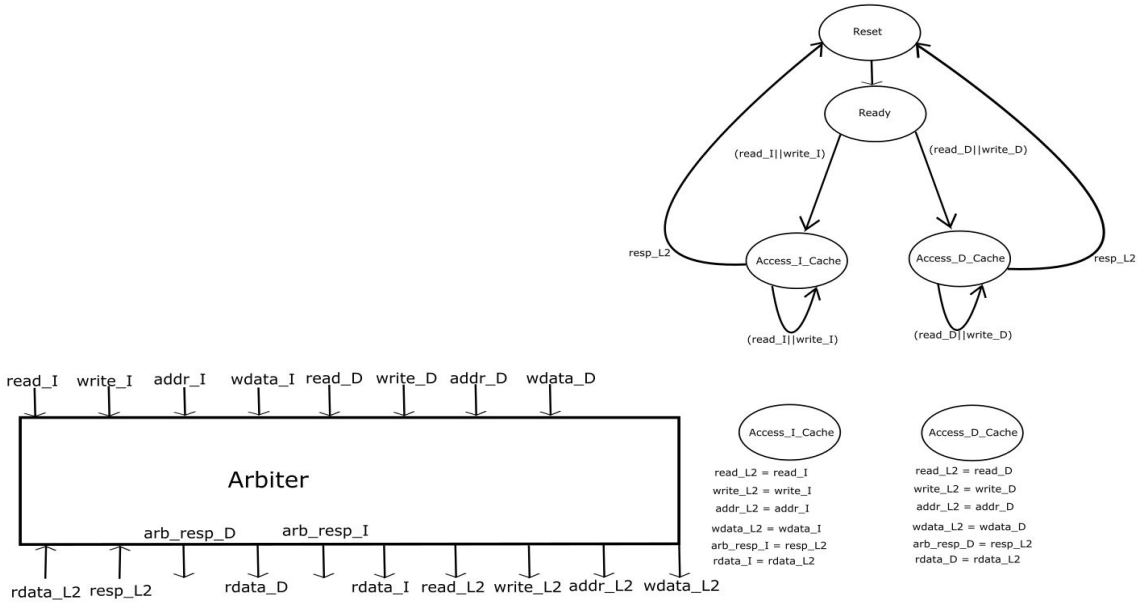
Checkpoint 2

For Checkpoint 2, we needed to implement the designs that we had made in Checkpoint 1. This was troublesome as none of our group members had a working MP2 cache. We all had reads working but not writes. After inspecting one of our MP2 designs, we believed that the datapath for the cache was correct, but the control for it was not. We decided to redesign the cache control and added this version in place of our split L1 caches. We then implemented our arbiter design. Once we had everything in place, we ran the provided Checkpoint 2 test code, and using modelsim, we saw that our cache would sometimes output empty data when there was actually a hit. However, when going through the signals in modelsim, we saw that our arbiter was indeed working as it should, so we realised our issue was purely a cache issue. Upon further inspection, we found our empty data issue was due to the LRU being selected incorrectly. This caused the wrong data to be read from our cache, and once we fixed this bug, our Checkpoint 2 test code worked correctly. We also viewed our modelsim and saw that

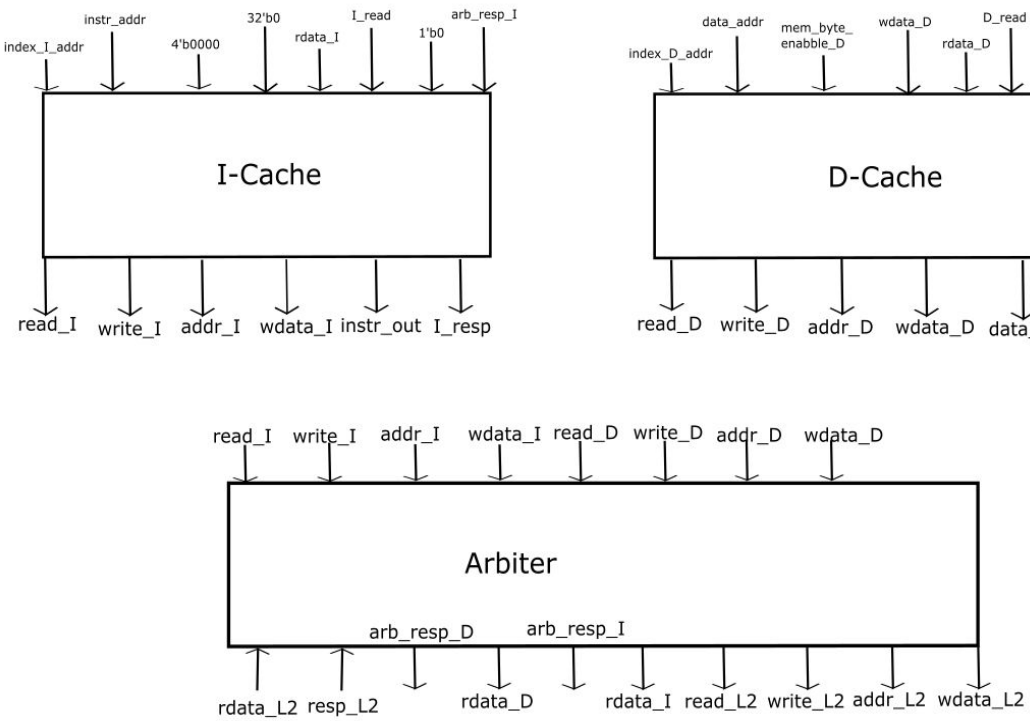
including the L1 caches allowed our processor to be much more efficient since our caches would only take one cycle to respond in cases of a hit, whereas, main memory took many cycles to respond as seen by the waveform on a cache miss. Once the L1 cache was working correctly in our design, we instantiated shadow memory. When we did this and ran the test code, we realized we had some shadow memory errors. These shadow memory errors were the result of changing the requested data address or instruction address too quickly. Therefore, in order to fix this issue, we took advantage of the fact that we could use registers to delay the addresses by the correct amount of cycles. From there, we started to design our hazard detection unit, forwarding unit, branch prediction, and L2 cache.



Updated Checkpoint 2 Pipelined Design



Checkpoint 2 Arbiter Design



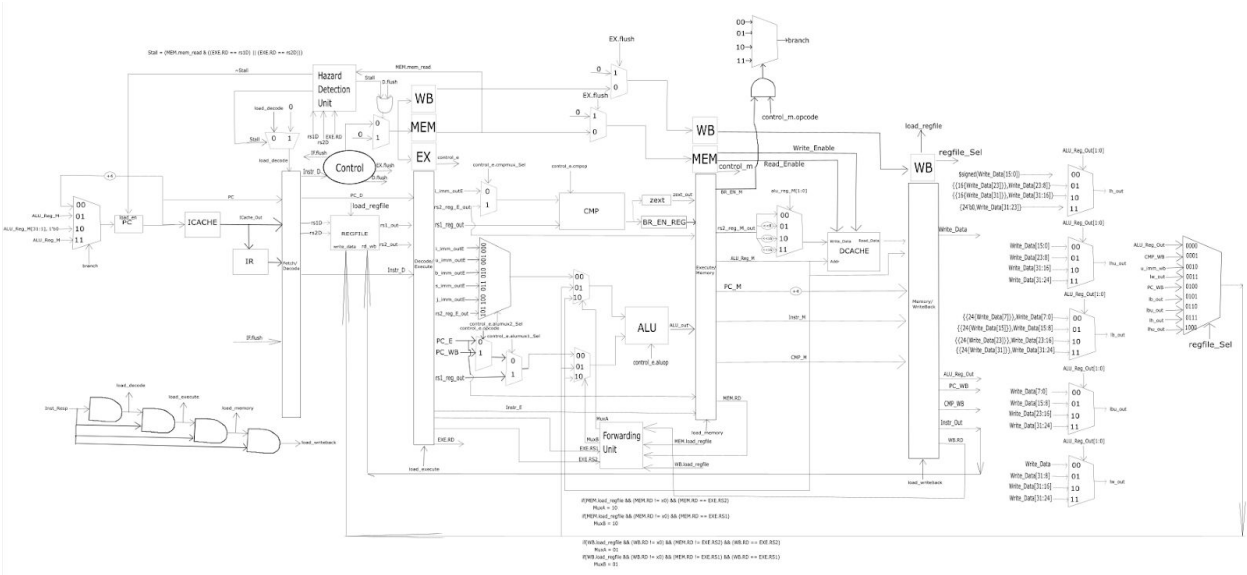
L1 Cache and Arbiter Connections

Checkpoint 3

The hazard detection unit allowed for instructions with data hazards to execute correctly by stalling the pipeline for one cycle to allow the data hazard to be resolved. The hazard detection unit was also responsible for flushing the pipeline for control hazards caused by branching. Since we implemented static-not-taken branch prediction, our processor would continue to load instructions if it detected a branch instruction. So, our hazard detection unit flushed the pipeline of these instructions if the branch ended up being taken, i.e. our prediction was incorrect. The forwarding unit allowed our processor to forward data between instructions. This was done by checking if one of the instruction's source registers were the same as the destination register of either of the previous two instructions. If so, we would forward the data that was to be written to the regfile directly to the stage in which it was needed, most commonly the execute stage. This allowed for quicker execution at times since the processor would not need to wait for the regfile to be updated before executing an instruction, which relied on updated register values. In order to test our forwarding unit, hazard detection unit, and branch prediction, we ran the provided test code and viewed the waveforms in modelsim. Using this method, we were able to see that our forwarding unit was indeed passing values from stage-to-stage as necessary. We also saw that our hazard detection unit was correctly flushing the pipeline after a taken branch so none of the instructions loaded after the branch actually got executed; this showed that control hazards were being handled. We also saw that our hazard detection unit inserted stalls when a data hazard was found, so this gave the first instruction enough time to finish its memory stage before the next instruction would ask for that data. Seeing as we were able to pass the test code

and we saw that our waveforms looked correct, we were confident that our newly added units were acting as expected. After we had those units working, we added in our L2 cache, which was similar to our MP2 cache design. We tested this cache by comparing the amount of time an L1 cache miss would take with the L2 cache implemented against how long the miss would take without the L2 cache. We used modelsim to view the waveforms and saw that having the L2 cache gave a performance increase in the case of an L1 cache miss and L2 cache hit, so we were confident that our L2 cache was working as well.

In addition to our practical implementations, in Checkpoint 3, we also had to instantiate our rvfi monitor. Our rvfi monitor followed a similar example that was given by Nathan on the ECE 411 piazza, where we would make packets that contained all the information that the rvfi required. This packet was created in the fetch stage. From there, it would be assigned the signals that the rvfi required that were available in the fetch stage like instruction address. The other signals that were not yet ready from the current stage (fetch) were left blank. As our commands would move forward in our pipeline, the rvfi packet would move along with it. At each stage, more signals were being assigned, until finally, all of the signals in the rvfi packet were assigned a pertinent value by the time the packet reached the writeback stage. The packet in its current form in the writeback stage is what was sent and committed to our monitor. This packet held all the expected information that the rvfi monitor was expecting, and by using this packet, we managed to pass the rvfi tests.



Checkpoint 3 Pipelined Design with Forwarding and Hazard Detection

Checkpoint 4

This is where our project reached its conclusion. Due to time constraints we were unable to do any advanced features. However, we were able to complete the baseline design and had every baseline feature successfully implemented.

Conclusion

This project demanded the utmost effort and ingenuity from our group. Of the design objectives that were laid out to us, our group managed to achieve the baseline design. This included a fully functioning 5 stage pipelined RV32I ISA, split L1 caches for both instruction and data, a unified L2 cache, an arbiter to handle the caches, hazard detection, data forwarding, static branch prediction, shadow memory for both caches, and an rvfi monitor. With these components of the project implemented, we fulfilled our goal of having a working baseline design. Despite this, we were unable to fulfill our additional goal of having any advanced features implemented due to a lack of time. Throughout this project, we came across a very interesting issue where, similar to a hydra growing two heads when one is severed, our code would produce multiple bugs when one bug was solved. From this, we came to the conclusion that at every point of a design, no matter what project you are working on, it is very important to document any and all changes in order to have a record of said changes in case one needs to abort and revert to a stable build. We also learned the importance of having a detailed and well thought-out paper design, as having these as references made implementing and debugging our features go much more smoothly.

Reference:

Due to the excessive size of some of our images, they may appear to be small and unreadable in the context of this paper. The full size images are available on our group's github page : https://github-dev.cs.illinois.edu/ece411-sp20/letUsGraduate