ECE411 CP5
Final Report

By: Steven Patriarca
Jay Patel
Sean Kozak

Introduction

This report is meant to document the experience of the challenges and knowledge gained from completing the final MP in ECE 411: Computer Organization and Design. With this MP our team had to implement an advanced LC3b processor. Some of the advanced features include, five stage pipelined, a global two-level branch history table, two level cache hierarchy with a four-way, set associative L2 cache. This project certainly gave us a better understanding of the brain of a computer and some design tradeoffs to try and make that brain more efficient. Throughout the report we will discuss the challenges that came with completing this project and what we learned when overcoming those hardships.

Project Overview

This project was split into five checkpoints. The first three constructed the base of our processor, pipelined datapath, split L1 cache, L2 cache, and data forwarding. This base design could stand alone and fully process the programs given to it. However, this was not sufficient and the next two checkpoints were dedicated to implementing advanced designs making our processor and caches more efficient. The checkpoints were designed for groups of three students so the work was fairly distributed between the team with one member not doing much more work than the others.

Milestones

Checkpoint 1

In this checkpoint we need to design a very basic 5 stage pipeline with some of the lc3b instructions. The way we divided such that Jay did fetch and decode, Steven did the execute stage and sean did the memory stage and write back. The one challenges we meet in this part was typos and creating a naming convention.

Checkpoint 2

For this checkpoint we need to add the remaining lc3b instructions and added a cache arbiter to deal replace multi port memory we had in checkpoint 1. In this part Steven did the reaming memory accesses instructions, sean added the control flow instruction and jay added the cache arbiter. The big challenge we dealing with bug create because of the pipelined design such as getting the right data from the right stage at the right time.

Checkpoint 3

We implemented data forwarding and added an l2 cache to the design. For this part steven and sean worked on data forwarding and jay added the L2 cache to the data path. The challenges we faced that with data forwarding was how to detect data hazards. Problems we faced with the L2 cache was making it a multi cycle design.

Checkpoint 4

For this checkpoint we had to add branch prediction, eviction write buffer and performance counters. In this checkpoint jay added the performance counters, sean wrote the eviction write buffer, and steven designed a static branch predictor. Problem we faced in this design was how to flush bad instructions and making the write buffer invisible to the L2 cache.

Checkpoint 5

For this part we decide to try to make a Global 2-level Branch history table, 4 way set associative L2 cache and multi cycle L2 accesses. For this part the whole group worked on L2 cache. Steven worked on the Global 2-level Branch history table and jay worked on the multi cycle L2 accesses. Problem we faced in this design was implementing a true lru design. We also had trouble with the multi cycle delay and the controls signal the pipeline would receive.

Advanced Design Features

Global 2-level Branch History Table

The idea of this feature is to dynamically predict branch outcomes based on previous branch outcomes and program count. The Global 2-level Branch History table will attempt to speed up you program execution by taking the correct branch the first time and not having to flush your pipeline when the prediction is wrong. We mainly tested this feature on Competition Code 1 since this test code uses many branch instructions. When comparing the runtimes of this test code with an always taken branch predictor and our Branch History Table we saw a 23.8% speedup time while using our Branch History Table. While this speedup is great you must be cautious of the tradeoffs. There is not much of a performance hinderance when the Branch History Table does not predict correctly but it will still take up precious memory space and power consumption while offering no performance boost. This makes the feature not as helpful in code that does not branch often.

Four-way, Set Associative L2 Cache

This feature is implemented to try and reduce writebacks to main memory and reads from main memory since each way is 1 KB giving a total of 4 KB L2 cache. Our group decided to implement this from the beginning since we believed it would be a difficult part of our project if we switch L2 cache designs during implementation. Because of this, we do not have any performance analysis for this feature since we implemented it from the start. Theoretically, we believe a four-way, set associative would perform well when there is a code that reads and writes to the same or spatially close addresses. However, since each L2 cache way is 1 KB, this makes L2 reads slower and a hinder to performance.

Multi-cycle L2 Cache Hit

Unfortunately we were unable to fully implement this design feature. Our L2 cache took much of our time to fully implement, which took away from completing this feature. The premise of this feature was shortening the critical path at the cost of lengthening L2 cache hits, making

L2 cache reads take longer. Clearly, this implementation increases performance when there are not many L2 cache accesses.

Observations

As expected, our implementation of the four-way, set associative cache took much of our time. Continuous problems of incorrect next-state transitions, LRU rankings, and setting and resetting of data arrays arose. It took our group a considerable amount of time to fix all the bugs and complete a working L2 cache. For future designs, a more careful planning process will likely result in fewer bug creations and a better design overall.

With the problems that came with the L2 cache, we also struggled to implement the Eviction Write Buffer. Since the EWB works closely with the L2 cache, we believed some problems stemmed from the EWB but really having them come from the L2 cache implementation. Once the L2 cache was sorted out, the implementation of the Eviction Write Buffer was more trivial. Our team utilized the Competition test codes to test the correctness of the EWB. Since the L2 cache is so large, there were not many writebacks occurring in the first and second test codes. However, in the third test code we saw many writebacks occurring between L2 and main memory with the Eviction Write Buffer storing those writebacks. After completion of this code we decided to runit again without the buffer included in our design. To our surprise, the code ran significantly faster without the buffer. There was a 3.16x speedup without the buffer. We believe this came from the the code doing a significantly amount writes, reads, and writebacks making the buffer slow down our design. The Eviction Write Buffer is best utilized when there does not have to be many writebacks.

Conclusion

Our group certainly learned a significant amount throughout this project. It gave us a better understanding of the relationship between the processor, cache, and memory. We also learned the value of teamwork and communication to successfully meet deadlines and checkpoint requirements. We didn't get to implement everything we planned to but we did get to put advanced features like a Global 2-level Branch History Table, a four-way, set associative L2 cache, data forwarding, and a five stage pipelined design. Our team is proud of our accomplishments and we strive to learn more about computer design to become better engineers.