ECE411

Fall 2021

MP4

Always_f(@F)

OOO RISC-V Microprocessor

Siddhant Singh, Hemang Nehra, Suraj Sharma

ss59, hnehra2, suraj2

Mentor TA: Eric Dong (ericd3)

Table of Contents

Introduction

For MP4, our group decided to implement a 32bit Dynamically Scheduled RISC-V microprocessor inspired by the Tomasulo algorithm. Our motivation to design and verify a high performance high ILP RISC-V microprocessor was predicated on implementing computer architectural techniques found in commercial microprocessors. We really enjoyed the challenging design and verification process of our microprocessor from scratch. We perused research papers and various resources to design and optimize our microprocessor.

Our functional microprocessor supports the RV32I ISA and includes optimizations such as dynamic branch predictor, hardware prefetching with stream buffers, and a 2kB 4-Way SA L1 I-Cache and D-Cache memory subsystem.

Project Overview

The design process of the Out-of-Order microprocessor constituted of many stages. Our initial team meeting consisted of going through the deliverables for each checkpoint to form a project roadmap as well as formalize the datapath of our design. After consulting with our mentor TA on our proposed design process, we sat together to design and verify a parameterized circular queue, which serves as our backbone for the Reorder Buffer, Load/Store Queue, Instruction Queue and Store after Branch hazard detection mechanism.

After the verification of our circular queue, we split up the tasks amongst ourselves to streamline the process. The inter-relation of individual modules to one another meant a close knit design process with constant communication with team mates to discuss design specifications and areas of performance bottlenecks. This process allowed us to make swift decisions impacting functionality or performance. The involvement of the entire group also allowed us to debug faster and help each other debug the design.

Our verification process was made easier by a modular design and verification approach. However, this still meant that edge-cases, previously unimagined, would hamper the functionality of our design and required further investigation.

Design Description

Overview

Our design and datapath have transformed completely since the inception of this project, owing to the high complexity of the design and many moving parts in the design. Moreover, our design also had to adapt to various advanced features that are planned/implemented in our processor.

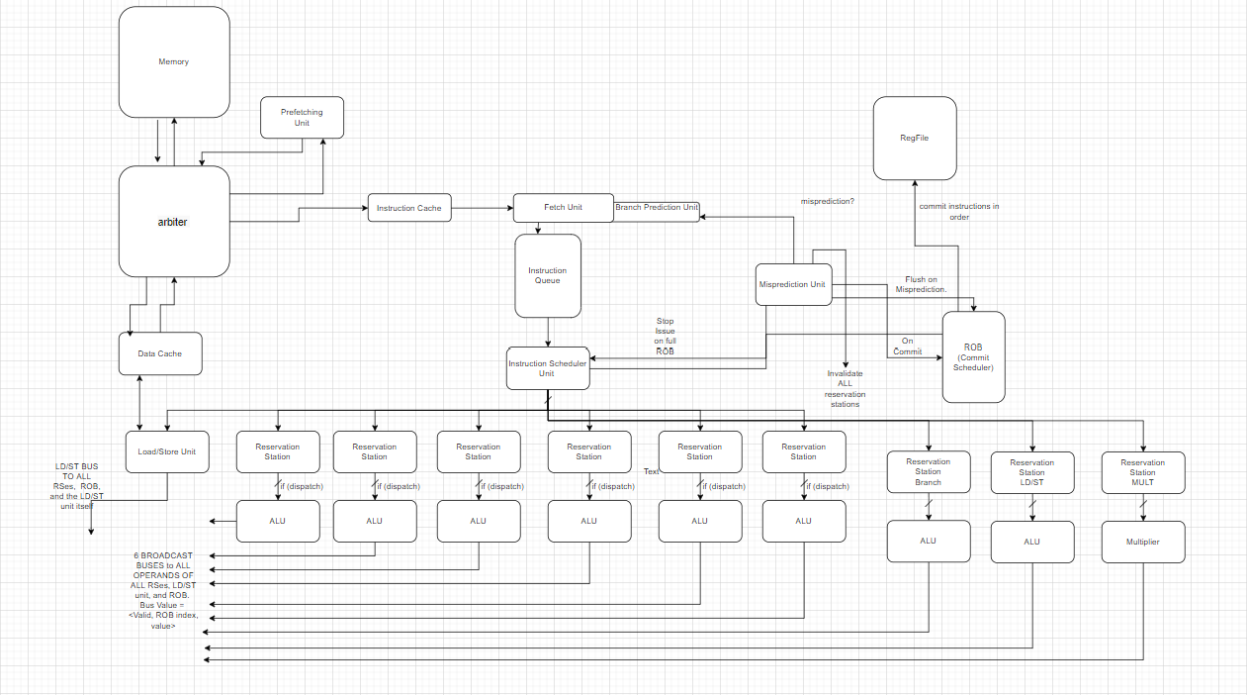


Fig 1: Top level overview of our OoO microprocessor

Milestones

1. Checkpoint 1

While checkpoint 1 did not have any points associated with it for team's implementing an OoO microprocessor, our objective was to have formalize our design by this checkpoint and have a robustly-verified circular queue ready to be used as a backbone in our design.

After we unit tested the circular queue with all imaginable edge-cases, we were ready to start implementing the Reorder Buffer, Instruction Queue and the Load/Store Queue.

2. Checkpoint 2

Checkpoint 2 required us to have a semi-functional microprocessor which did not have branch prediction or cache implemented. This meant, the test program tested during this checkpoint, “mp4-cp1.s”, had a lot of stall instructions (nops) to allow the processor to resolve load/stores and branches. The functionality of this test program was determined by the end-values in our register file for R0-31.

For this checkpoint, we implemented the Instruction Queue, Instruction Scheduler, Load/Store Queue, Reorder Buffer and Reservation Stations for resolving branches, load/store target addresses and executing other instructions.

Before we began verifying our design with the “mp4-cp1.s”, we wrote our own test program consisting of multiple instructions to unit test the design and the edge cases. After this, we began verifying our design with “mp4-cp1.s” and encountered bugs with load/store operations. The bugs in ths design were due to unaccounted for edge cases which proved to bottle down the execution of our microprocessor.

3. Checkpoint 3

Checkpoint 3 required us to have a fully functional microprocessor with a branch predictor capable of running “mp4-cp2.s” and “mp4-cp3.s”. This period also coincided with the our implementation of optimizations such as hardware prefetcher and 4-Way SA cache.

The high-use of branches in the “mp4-cp2.s” program meant that our branch predictor was tested to its limit, including flushing mechanisms in the vent of misprediction. Branch predictor proved tricky to integrate as it introduced many new bugs into our design which needed resolving as well as highlighting control hazards that were previously ignored during the design process. The short execution time of the checkpoint 2 code meant it was relatively easy to debug using the Waveform. We were able to get the “mp4-cp2.s” running in a short time.

However, our biggest hurdle proved to be the “mp4-cp3.s” testcode. The long execution time (368x the execution time of “mp4-cp2.s”) meant endless hours were spent debugging the test program. During the verification of “mp4-cp3.s”, we realized the need for creating a software model for our design to hep identify the faults in our design. However, with a short turnaround required, the team remained steadfast on verifying our design.

Our biggest bugs remained the branch predictor and the interaction between memory during flushes on misprediction. Since we had not accounted for pending memory requests when issuing the flush signal, we were corrupting the data in our cacheline, causing incorrect execution of instructions. Once we identified this

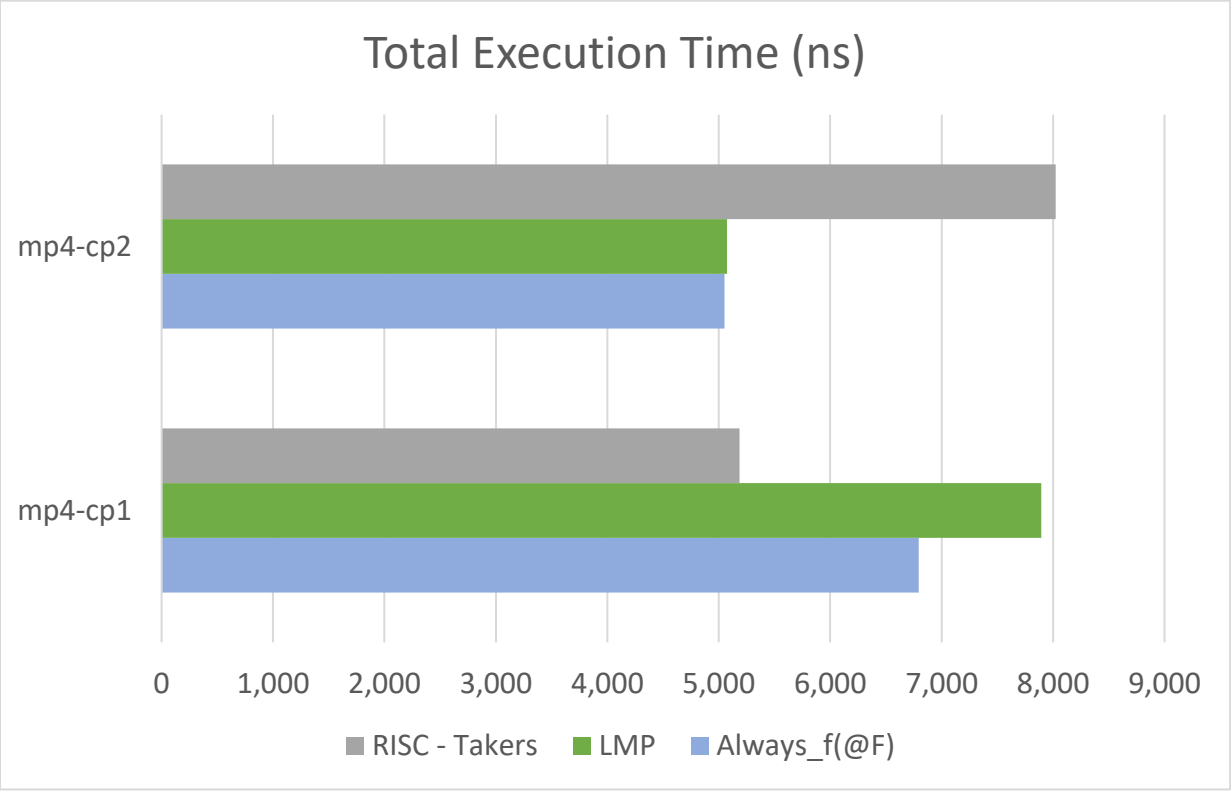
issue, we focused on creating a robust mechanism to resolve this hazard which proved to fix the functionality of our design with the “mp4-cp3.s” test program.

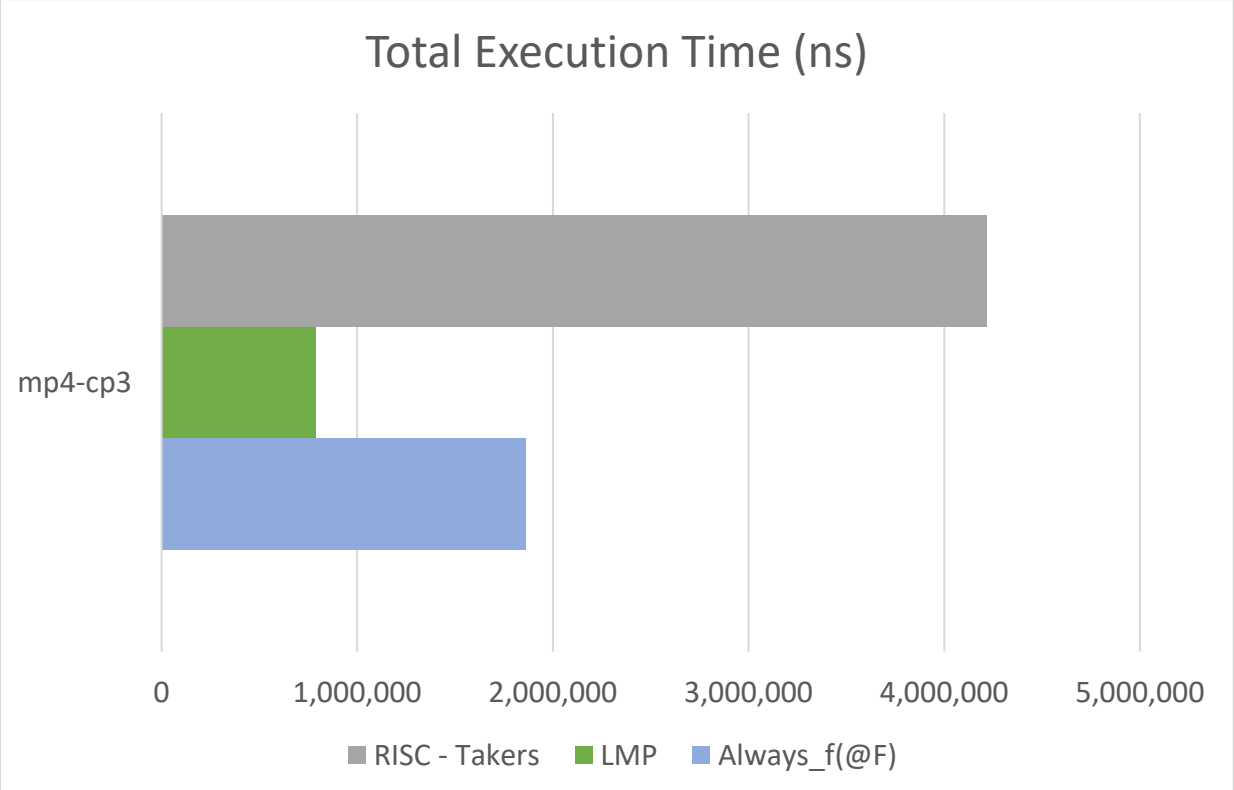
Advanced Design Features

Out-of-Order Execution

1. **Design:** Our design was inspired by the Tomasulo Algorithm to design a dynamically scheduled RISC-V microprocessor. While the lecture notes provided a solid background on the architectural technique, the lack of any formal documentation for the project meant we had to resort to research papers and additional resources on YouTube to design our processor. Our design finds its performance improvements compared to a statically scheduled processor by executing the instructions out-of-order but “retiring” them in order to avoid any hazards. This is possible due to the integration of the Reorder Buffer, initially omitted in the Tomasulo Algorithm.
   The design process involved painstakingly long conversations about possible bottlenecks and resolving these bottlenecks. One avoidable bottleneck was the size of the ROB, which we decided to set to 32 along with the Instruction Queue and the Load/Store Queue. This allowed our processor to avoiding stalling instruction scheduling due to the ROB being full and enabled higher performance.
2. **Testing:** The complex nature of the design meant that the testing of our OoO microprocessor was painstaking. While we debugged using Waveforms on Modelsim, we realized while debugging the “mp4-cp3.s” test program that a software model is imperative for a smooth debugging process. The time constraint meant that we were unable to implement a software model which hampered our debugging for the competition codes. The emergence of unimagined edge-cases further compounded our design process as changes to our design need to be introduced to resolve those hazards. There remains a big scope of improvement in our verification efforts on our microprocessor, something we aim to change for the future.
3. **Performance Analysis:** Our **Fmax** was **98.81MHz**. Our microprocessor offered promising results compared to the conventional design option of a 5-stage pipelined CPU. From the figures illustrated, we can see that our design shows 50% improvement in execution time for cp3 code and 60% for cp2 code while taking 30% longer for cp1 testcode compared to the design of team RISC – Takers. When our design is compared to team LMP, we can see that our design twice as long for cp3 code and 16% faster for cp1 testcode. The execution time for cp2 codes were near identical for both the designs. We believe the massive performance difference

comes from their implementation of pipelined cache coupled with our aggressive instruction prefetching which increases D-cache miss penalty.





Hardware Prefetcher

1. **Design:** The motivation to integrate a sequential hardware prefetcher with stream-buffers originated from our analysis of the I-cache hit-rate and miss penalty and the desire to reduce the miss penalty. We integrated a hardware mechanism which predicts the next set of instructions that need to be fetched from memory. Through the arbiter interface, we are able to serve I-cache misses when the memory is servicing another request if the prefetcher stream buffer has already fetched the set of instructions the I-cache is requesting for. This helped us reduce our cache hit time to 1 cycle in the best case scenario.
2. **Testing:** The testing of the prefetcher included observing memory requests by the I-cache, D-cache and the prefetcher. The prefetcher is designed to take the lowest priority in terms of memory requests to memory, after I-Cache misses and D-Cache misses. Monitoring the states in the arbiter and serving I-cache misses through the stream buffers was the focus on verifying the hardware prefetcher.
3. **Performance Analysis:** The prefetcher was the source of the biggest performance increase for our microprocessor, providing a ~30% improvement in execution time on average for all test codes. This is due to aggressive prefetching and a 512 instruction window for the stream-buffers, enabling 1 cycle I-cache miss penalty and thus our processor hardly stalls due to "lack of instructions" in the instruction queue. We prioritized keeping our instruction queue full at any given time to aid the superscalar integration into our design.

Local Branch Predictor

1. **Design:** The importance of branch prediction to the speedup of the microprocessor meant we placed a lot of emphasis on the branch predictor. We designed a private history dynamic branch predictor that utilizes a 2bit up/down counter. We also integrated a Branch Target Buffer of size 128 and the Branch History Table has a size of 1024, owing to 10 bits acting the indexing bits to avoid conflicts faced in the "mp4-cp3.s" test program. We also had to integrate the flushing mechanism in the event of a misprediction which was issued from the ROB, on the commit of a mispredicted branch, requiring changes to the ROB and commit logic.
2. **Testing:** The testing of the branch predictor included a lot of hazards that needed to be accounted for, including store operations after a mispredicted branch. In order to handle this hazard, we included a stalling mechanism in the Load/Store queue which waits for the branch before it to be resolved before it is executed.
3. **Performance Analysis:** The branch predictor offered us the speculative performance increase that we were looking for. It allowed our processor to not

stall on branch operations and handle mispredictions with flushing of IQ, ROB, LD/ST unit, and RS's. The low frequency of branches in "mp4-cp1.s" and "mp4-cp2.s" meant that the branch predictor did not have the desired accuracy, which was expected. However, we achieved an accuracy of ~89.39% on the execution of "mp4-cp3.s" test code.

Cache

1. **Design:** While we were provided with a direct-mapped cache, we decided against using this cache as we wanted to avoid the conflicts in our cache. Instead, we decided to implement a 2kB 4-Way Set-Associative Cache with Pseudo-LRU replacement policy for our L1 I-Cache and D-Cache. When increasing the size of our cache, we were mindful not to do so by just increasing the sets as that would increase our conflicts.
2. **Testing:** The testing of our cache was, surprisingly, the biggest bottleneck in the verification of "mp4-cp3.s" test code. This was due to the fact we had neglected the interaction between memory and flushing. This meant we kept corrupting the data in the cache, sometimes causing our processor to stall out or execute the instructions incorrectly. When we recognized this issue, we created a robust mechanism to avoid corrupting the data on flushes. This fixed the implementation of our design on the test codes provided.
3. **Performance Analysis:** The cache proved to serve significant performance gains in our design, allowing us to implement superscalar functionality into our design as well. The biggest bottleneck is the response from burst memory requiring ~25 cycles as opposed to single cycle response set up with magic memory. However, we have seen massive performance gains in D-cache hit for load/store operations. We had a I-Cache hit rate of 99.8% and D-Cache hit rate of 60.4% for "mp4-cp3.s".

Advanced Features in the Works

Following is the list of Advanced Features that are currently in the pipeline for our design. This includes features that were developed parallel to our debugging efforts for checkpoint and competition codes. Our efforts to continue debugging competition code meant that we had to put a pause on our efforts of verifying our design with the optimizations. Stay tuned for further integration of these features into our design to increase the performance of our microprocessor.

Superscalar

We plan on integrating superscalar functionality into our design with 2-Way Issue. We have already designed the mechanism for multiple issues but are yet to verify this design. We also plan on eliminating the single cycle single commit from ROB by introducing multiple commits (5 commits in a single cycle) to eliminate another performance bottleneck in our design. We already have the provision for this in our design but needs to be verified with our design.

Wallace Tree Multiplier

We plan on implementing a RV32M extension for our microprocessor by integrating a Wallace Tree multiplier and divider into our design. Stay tuned for this extension.

Gshare Branch Predictor

We plan on implementing a Gshare Branch Predictor with Tournament. Stay Tuned.

Additional Observations

Our design performed very well compared to a similarly optimized pipelined design which coincides with the architectural advantages of dynamically scheduling instructions to execute them out of order. While our Fmax was 98.81MHz, we actually expected a worse Fmax considering the complexity of our design and extremely long combinational paths, forming very long critical paths. We believe we can further increase our Fmax by optimizing our combinational logic and pipelining certain modules of our design to further optimize our design.

Conclusion

Our design is a culmination of 8 weeks of painstaking effort made by every member of this team. We were able to accomplish our goal of designing and verifying a dynamically scheduled RISC-V microprocessor inspired by the Tomasulo algorithm. We saw significant performance increase compared to statically scheduled designs which confirms the advantages of OoO design. We were able to further increase our performance by implementing optimizations such as hardware prefetcher, branch prediction and cache. While we plan on integrating further optimizations, we saw significant improvement with these optimizations alone.

This design process proved to a very valuable learning experience for all of us and has increased our affinity with computer architecture. The design process was challenging but very rewarding as we conceptualized a design from scratch and acquired many new skills during the process. This project has given us insight into how complicated modern processors are and the design process involved. We are excited to continue our learning in the field through further investigation and hope to work on state of the art processors in the industry.