ECE 411

Spring 2021

Pipelined RISC-V Processor
Final Report

Tony Joo, Albert Yoo, and Andrew Zhang
TA: Shreyas Mohan

Introduction

For our final project, we developed a pipelined RV32I Processor in SystemVerilog. The processor implements all of the RV32I instruction set architecture (ISA) except FENCE*, ECALL, EBREAK, and CSRR instructions. Pipelining is very important in computer architecture because it improves the CPU performance by increasing the number of instructions executed simultaneously. Our report starts off with an overview of our project. We then describe all of our checkpoint designs in detail with diagrams. Afterwards, we talk about our advanced features that we added on top of our pipelined processor in detail.

Our pipelined processor has 5 main components: pipeline registers, control memory, hazard detection, data forwarding, and cache. The 5-stage pipelining has 5 states: Instruction Fetch (IF), Instruction Decode (ID), Execute (EX), Memory (MEM), and Write-back (WB). To be able to execute the instructions simultaneously in each of these stages, we needed 4 pipeline registers named IF/ID, ID/EX, EX/MEM, and MEM/WB to denote the in-between stages respectively. These registers store useful information such as register values, program counter, memory read data, opcodes, control signals, and so on. For the second component, control memory (control ROM), is used to generate control signals for the processor. Instead of running through the states in a state machine, we carry the control signals through each pipeline register because each pipeline stage will have different control signals concurrently. To handle data hazards, the processor needs to notice if the concurrent instruction needs an updated register value before executing the instruction. Also, the processor needs to handle control hazards by detecting a branch enable instruction and stall the pipeline until the branch has been taken. Finally, we connect our cache to our design by separating it into data cache and instruction cache. Because our design has only one bus, we also needed to implement an arbiter to decide the priority to serve each cache.

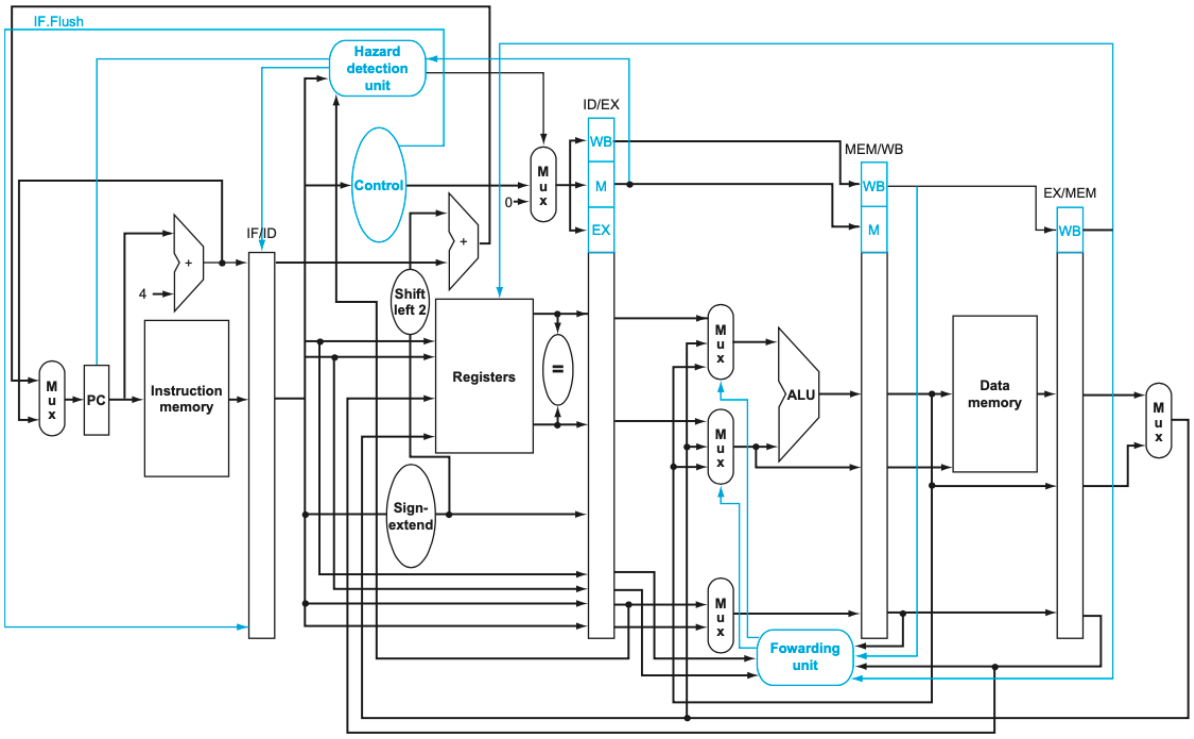

Figure 1. RISC-V pipelined processor

Project overview

The goal of our project was to design and implement a pipelined RV32I processor and improve the functional pipelined RV32I processor with additional advanced features. We met anywhere from 3-5 days each week to work on the project. The first meeting of each week used to get organized and discuss how we wanted to design the checkpoints, usually by drawing the design schematic. Then we would implement the design together by utilizing Visual Studio's Liveshare feature which would let us work on the code simultaneously. We realized the great importance of having the design implemented correctly because our design was the only reference that we look up on when coding.

Design description

Milestones

i. Checkpoint 1: RV32I ISA and basic pipelining

For this checkpoint, all group members implemented the pipeline registers and control ROM of the pipelined processor design by simultaneously editing the code. The members of the group implemented additional features to pipeline our RISC-V processor. We added pipeline registers which carried over instruction data such as the register numbers, funct3, and opcode, the program counter, and the control word containing control signals. We also added a control ROM which initialized the control signals of the control word based on the opcode, funct3, and funct7 inputs. Our testing strategy was to run the assembly program mp4-cp1.s and verify if the waveform values matched the expected values as we traced through the test code.

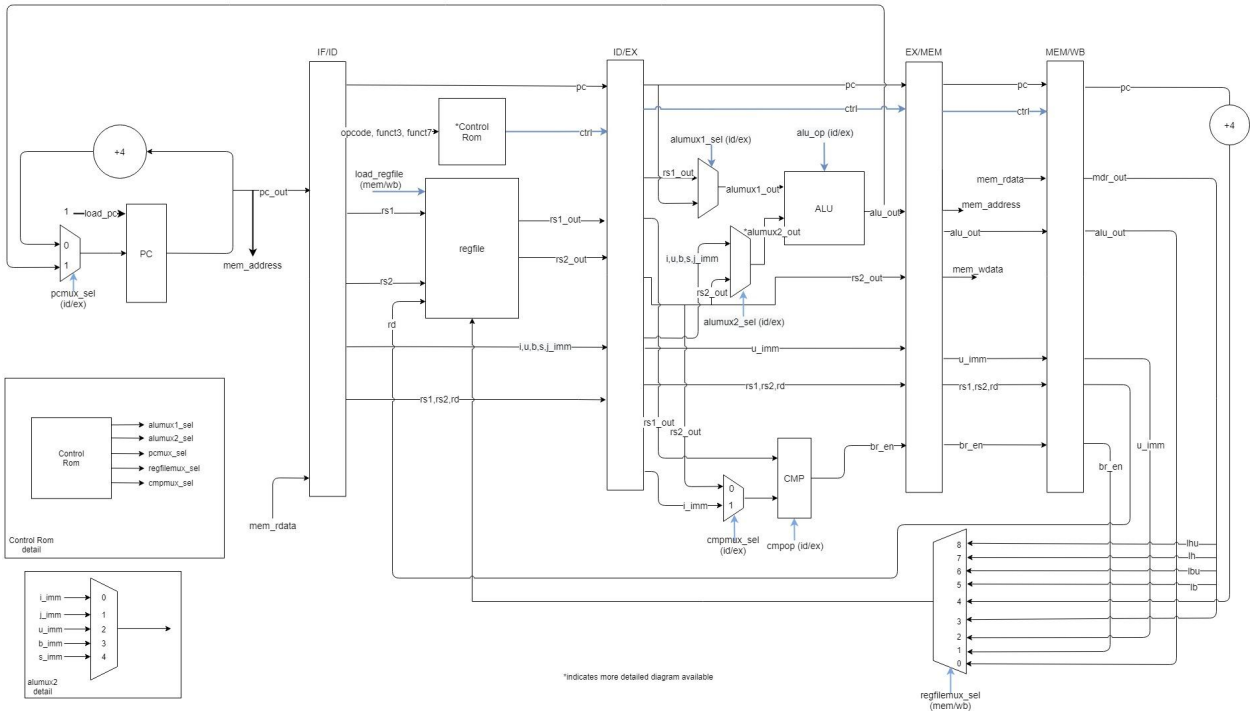

Figure 2. CP1 datapath design

ii. Checkpoint 2: L1 cache + hazards and static branch prediction

For this checkpoint, all group members implemented the hazard detection and forwarding, including static-not-taken branch prediction for all control hazards. We also implemented an arbiter and connected it to the instruction and data caches. Our testing strategy was to run the assembly program mp4-cp2.s and verify if the waveform values matched the expected values as we traced through the test code. We first tested the code with only the branch prediction without caches plugged in and then tested after implementing the arbiter with caching.

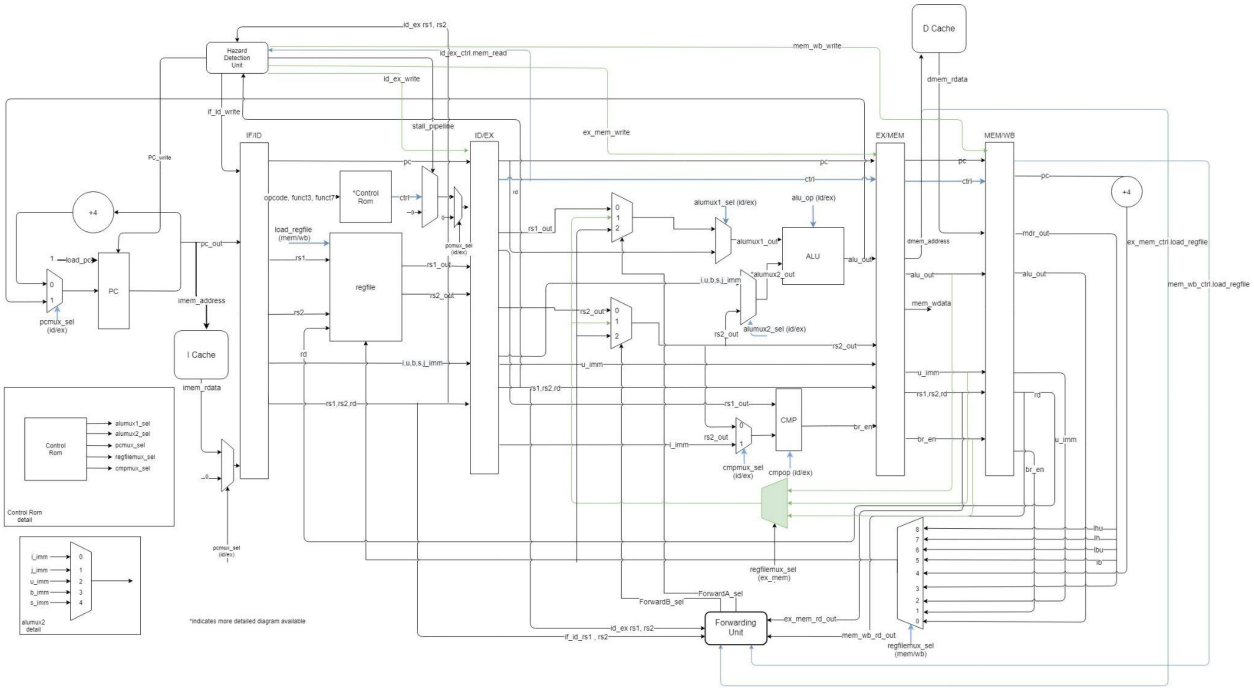

Figure 3. CP2 datapath design

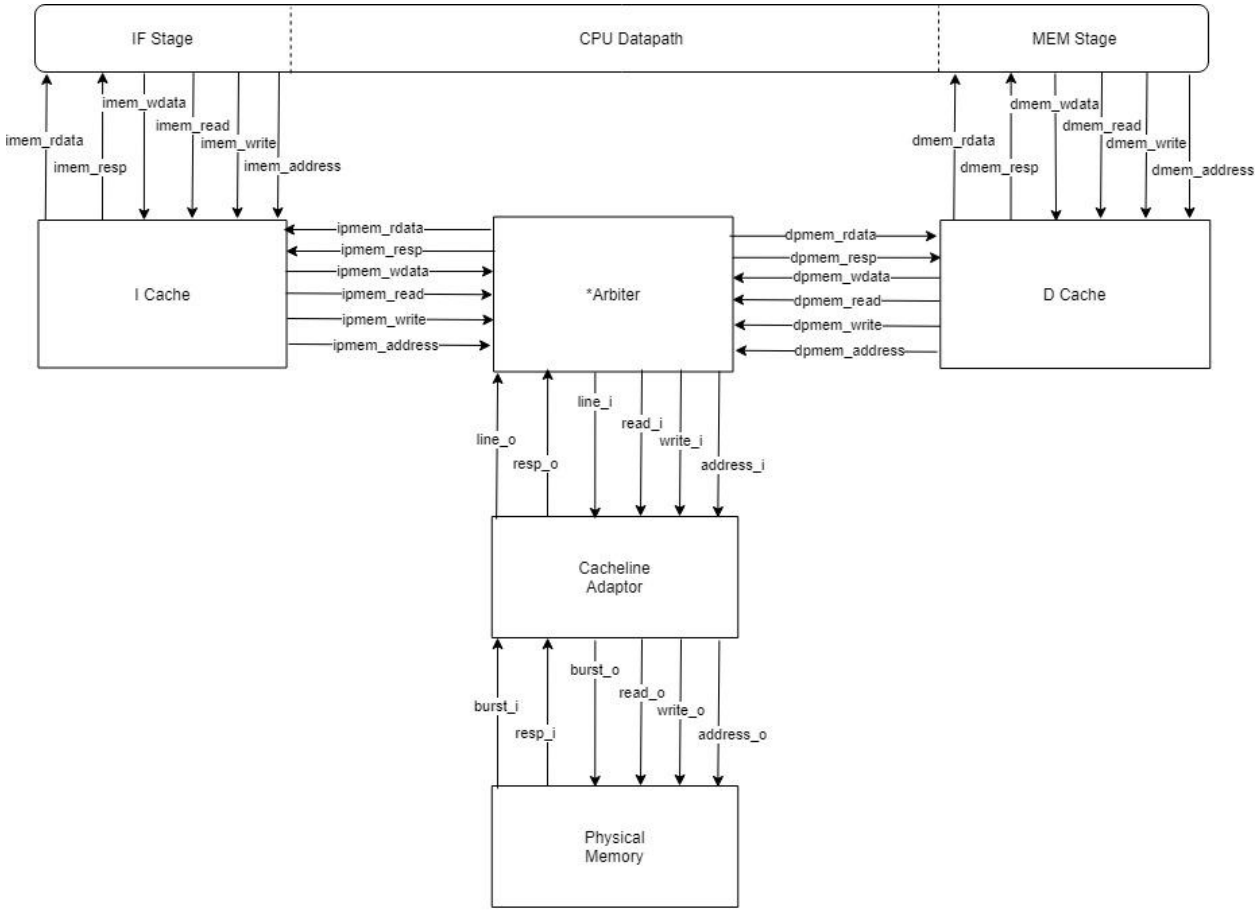

Figure 4. CP2 Cache and Arbiter design

iii. Checkpoint 3: Advanced Design - L2 Cache

For this checkpoint, we realized that our cache was not working properly when testing with the CP3 code. We spent the majority of time debugging the cache and/or integrating the given cache with the CP2 and CP3 code. Unfortunately we were unable to implement very many advanced features. We plan on implementing the rest until the final checkpoint.

Advanced Design Options

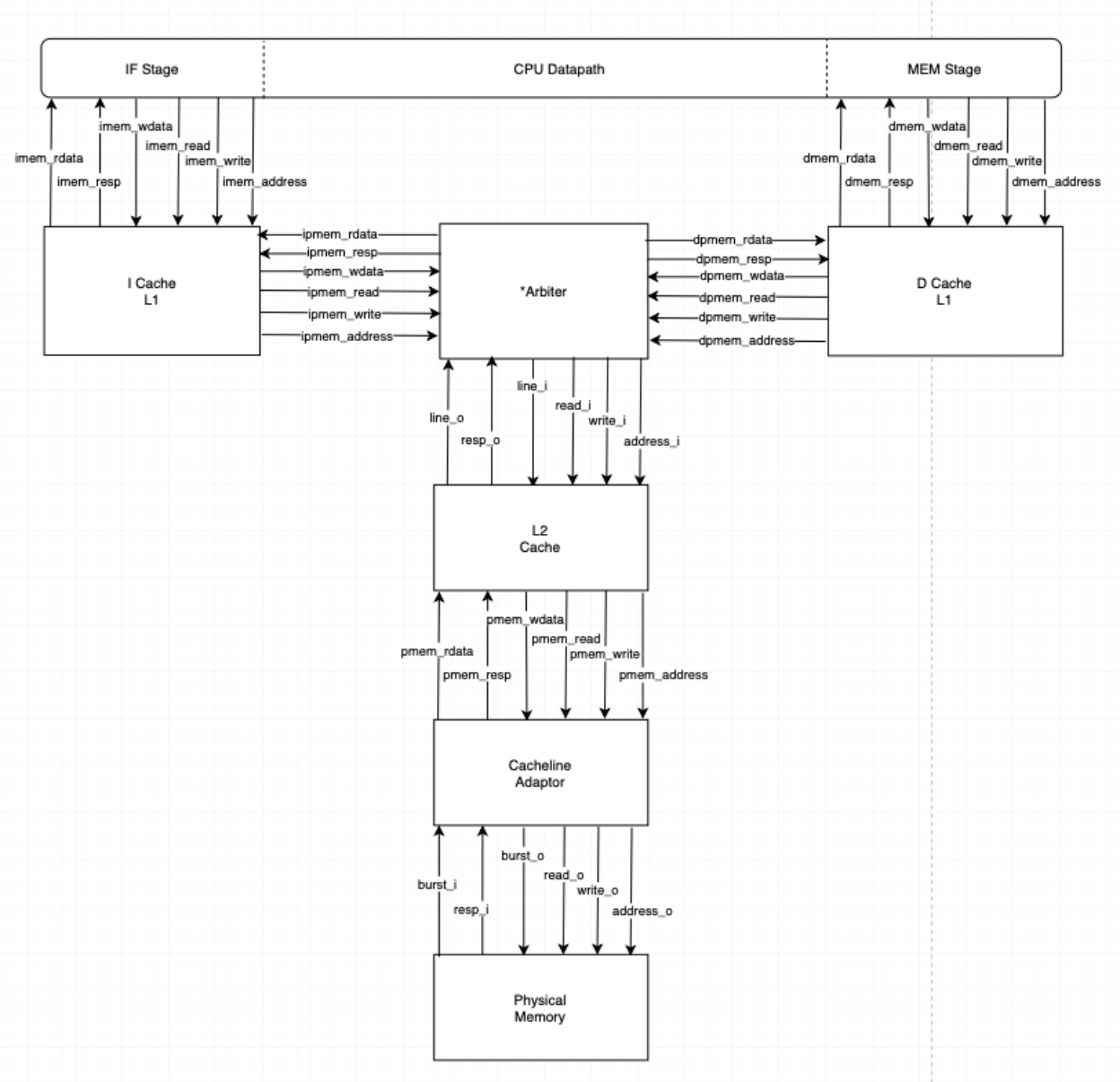

Figure 5. CP3 L2 Cache design

We added a 2 way set associative L2 cache with 32 sets in between the L1 cache and the physical memory of our baseline (checkpoint 2) processor to reduce its execution time. The L2 cache reduces the number of physical memory accesses because on an L1 cache miss, the memory data can be fetched from the larger L2 cache instead of the physical memory. Accessing

the L2 cache takes a much shorter time than accessing the physical memory. Adding an L2 cache can only slow down the system if there are always cache misses but that is very unlikely to happen. We first tried using a 2 way L2 cache with 8 sets, the same size as the L1 cache but it performed worse on the checkpoint 3 code than the baseline processor. As we increased the size of the L2 cache from 8 sets to 32 sets, it reduced the runtime of the processor on the checkpoint 3 code. We decided to keep the maximum size of the L2 cache at 32 sets because increasing the size of the L2 cache would increase the number of registers used in our processor and the power consumption of our processor.

L2 Cache	Comp1	Comp2	Comp3	Average
Time (ns)	654115	1772015	959845	
Power (mW)	1021.31	1015.88	905.93	
Score	2.86E-10	5.65E-09	8.01E-10	1.09E-09

Fmax (MHz)	94.07
Registers	28780
Pins	165

Baseline	Comp1	Comp2	Comp3	Average
Time (ns)	590545	3802125	3405605	
Power (mW)	621.81	514.69	481.27	
Score	1.28E-10	2.83E-08	1.90E-08	1.58E-08

Fmax (MHz)	92.7
Registers	10824
Pins	165

Figure 6. Performance Analysis of L2 cache and baseline

The L2 cache improved performance on the competition 2 and 3 code by wide margin. The execution time of comp2 code decreased from 3.8 million ns to 1.77 million ns and the execution time of comp3 code decreased from 3.4 million ns to 959845 ns.

However, it slowed performance on the comp1 code. The L2 cache version delayed branching to a new instruction address from the execute stage to the memory stage. This increased the amount of time required to jump to the next instruction during static branch prediction as a new nop had to be inserted in the execute stage along with the nops in the fetch and decode stage which slowed the runtime on the comp1 code. In addition, adding the L2 cache doubled the power usage of the L2 cache included processor compared to the baseline processor.

Conclusion

The objective of our design was to create an RV32I ISA processor with pipelined registers. In later checkpoints we added more features such hazard detection, forwarding, and an

arbiter with split caches (instruction and data). Finally, before optimizing the design we introduced the L2 cache as a feature to our processor. From our results above, we could tell how pipelining dramatically improves the performance of a multicycle processor and adding an L2 cache can dramatically improve the performance on top of that pipelined processor. On the other hand, we also observed that adding more complex designs results in utilizing more registers and power. Through this project, we could review what we accomplished throughout the course and gain the ability to develop a sophisticated computer architecture on our own.

References

1. Computer Organization and Design: The Hardware/Software Interface, 5th Edition, David Patterson and John Hennessy [p. 329]