Processor Prototyping Lab

Final Report

Course: ECE437

Name: Xinyuan Cai, Xiangyu Guo

Lab Section: 4

Teaching Assistant: Bhakti Vora, Jimmy Jin

December 11, 2022

1 Overview

In the final report, we will compare 5 different designs with three different latencies, 2, 6, and 10. The 5 designs are Singlecycle, pipeline without cache, pipeline with cache, multicore single-thread version, and multicore multi-thread version. The single cycle has the simplest implementation. The design of it is also relatively easy. The pipeline without cache prototype is based on Singlecycle since the most significant change is four extra latches. This modification will dramatically reduce the critical path and acquire a much higher clock rate. The pipeline with cache design is built upon the previous pipeline design. The 2-way associated d-cache and direct-mapped i-cache decrease the average memory access time. The multicore single-thread design combines two pipeline modules together. It could run two instructions with two cores at the same time. The final multicore multi-thread design implements the LL-SC based lock. As a result, the final design could deal with consistency problems in parallel programming.

To evaluate the performance of each design, we will focus on CPI, MIPS, latency, and execution time. We take the FPGA resources into consideration as well. The more FPGA resource used, the more complex the design is. The 5 designs explained above are in the sequence of oldest to latest. We expect an increase in performance from old to late design, which means a smaller MIPS and execution time. On the other hand, we expect an increase in FPGA resources from the old one to the late one because more function implementation inevitably increases the complexity of the design. In the rest of the report, we will first discuss the essential part of our new designs and provide block diagrams for different parts of the design. Then the resulting data will be provided to compare the performance and complexity of each design. In the end, we will discuss the result with our expectations. If the result does not meet our expectations, we will discuss the potential reason cause the problem.

2 Processor Design

The only modification from the previous Datapath is adding the atomic signal. It is generated from the control unit, indicating an SC or LL instruction is decoded.

We implemented the instruction cache and data cache separately. The i-cache is direct-mapped. It is 512 bits in size and one word per block. The d-cache is two-way associative, implementing the least recent use logic. It is 1 Kbits in size and two words per block. The d-cache also has special states for coherence problems. When receiving ccinv signal and ccwait signal at the same time, it will invalidate the matching frame in the cache.

The bus controller is the main part of the design. It could deal with different requests from each cache, including read-only, write-only, read-modify, i-transaction, and mem-flush. In the Idle state, the ccwrite signal represents the write request from caches, while in other states it represents the snoop hit. When one cache using the bus, the bus controller will send ccwait signal to the other cache. The other cache will be pulled back to the idle state and wait for the other cache to complete the transaction.

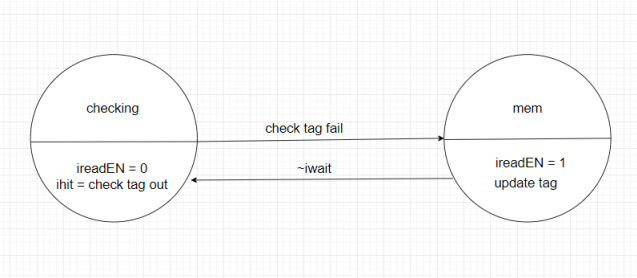

Figure 1: I-cache controller Block Diagram

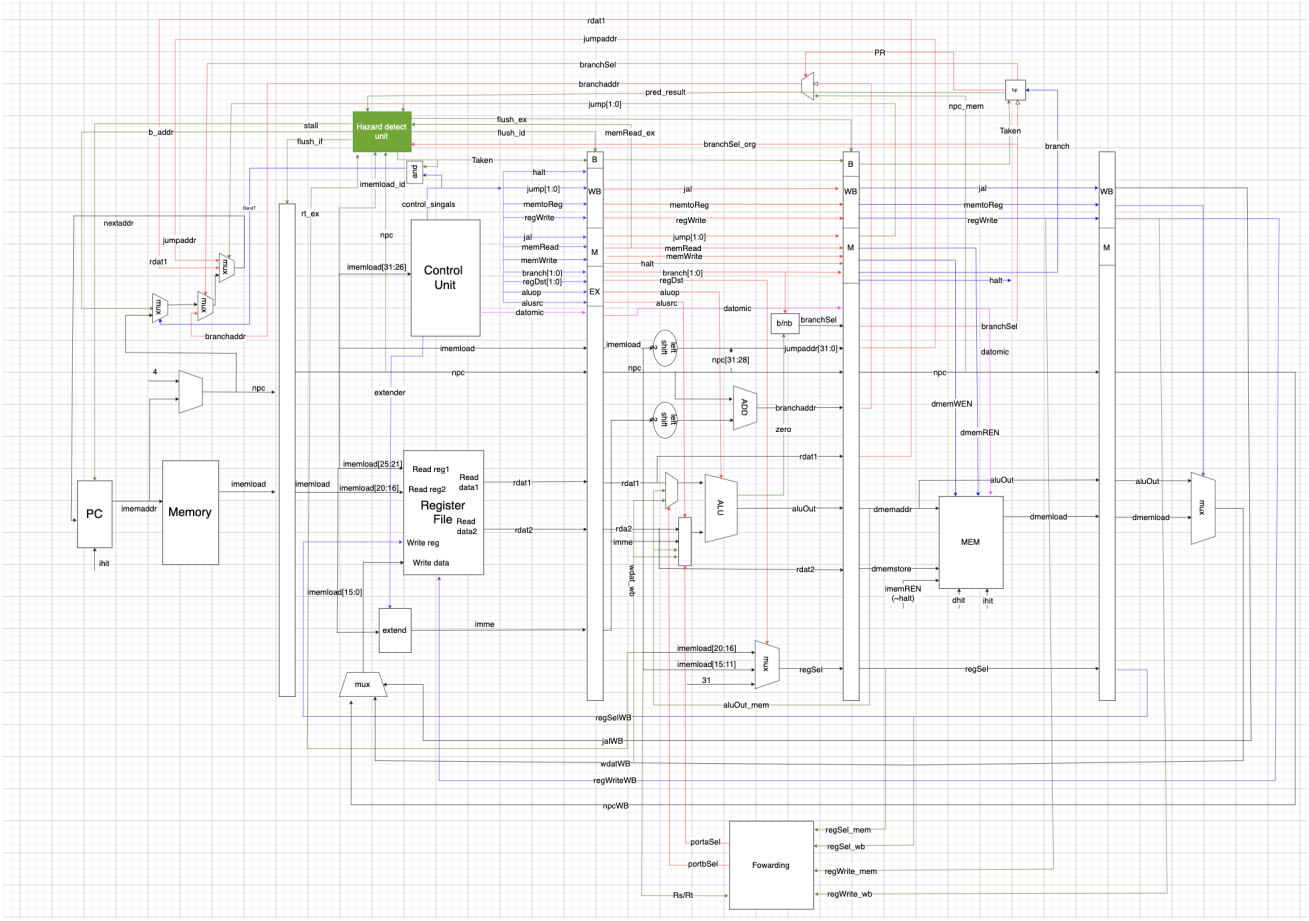

Figure 2: Datapath Diagram with datomic

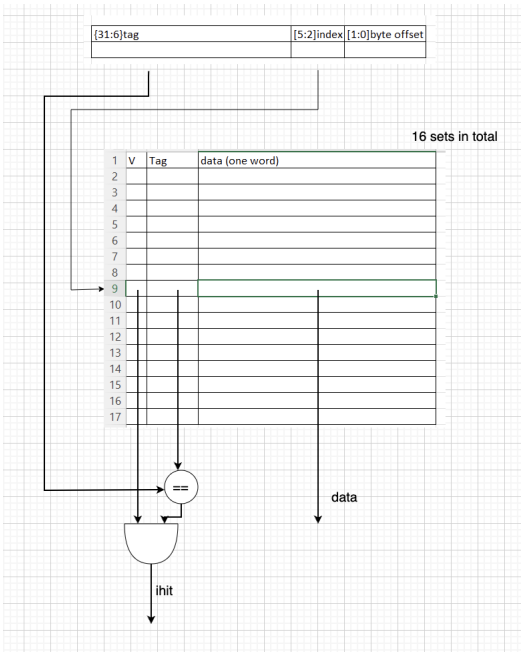

Figure 3: I-cache architecture Diagram

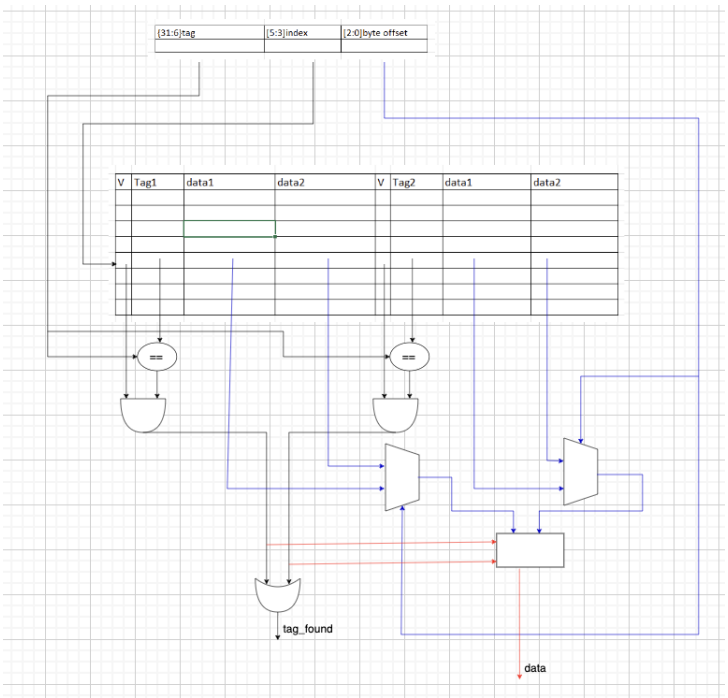

Figure 4: D-cache architecture Diagram

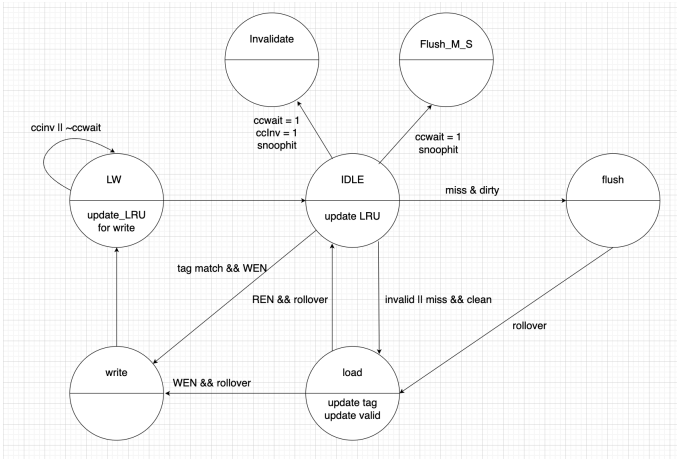

Figure 5: D-cache controller Block Diagram

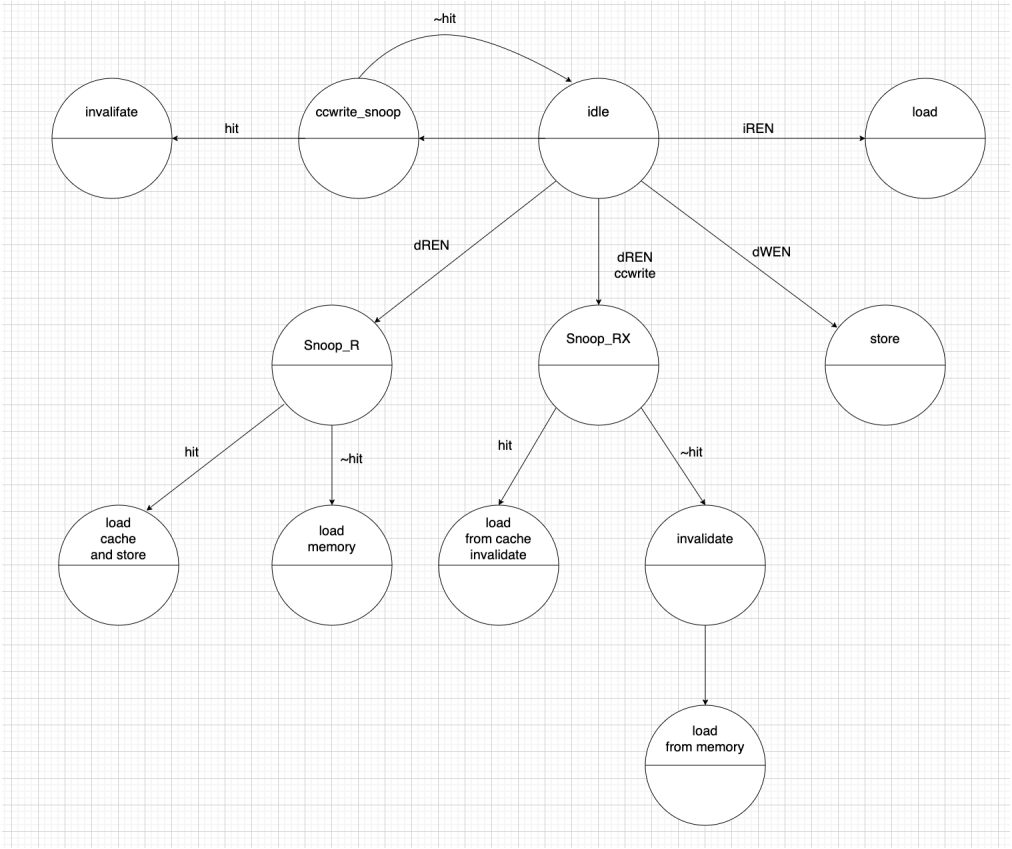

Figure 6: Bus Controller Block Diagram

3 Results

Criteria	Singlecycle	Pipeline without cache	Pipeline with cache	Multicore single-thread	Multicore dual-thread
Frequency(CPU)Hz	$3.94 * 10^7$	$7.15 * 10^7$	$6.11 * 10^7$	$4.31 * 10^7$	$5.79 * 10^7$
# of cycles	13802	20242	12863	20083	12320
# of instructions	5404	5404	5404	5021	5021
CPI	2.55	3.74	2.38	3.70	2.27
MIPS	$1.54 * 10^1$	$1.91 * 10^1$	$2.57 * 10^1$	$1.16 * 10^1$	$2.55 * 10^1$
latency(ns)	$2.54 * 10^{-8}$	$6.99 * 10^{-8}$	$8.18 * 10^{-8}$	$1.16 * 10^{-7}$	$8.62 * 10^{-8}$
execution time	$3.5 * 10^{-4}$	$2.83 * 10^{-4}$	$2.10 * 10^{-4}$	$4.66 * 10^{-4}$	$2.12 * 10^{-4}$

Table 1: Processoers performance when latency equal to 2

Criteria	Singlecycle	Pipeline without cache	Pipeline with cache	Multicore single-thread	Multicore dual-thread
Frequency(CPU)Hz	$4.01 * 10^7$	$7.01 * 10^7$	$6.29 * 10^7$	$4.36 * 10^7$	$5.61 * 10^7$
# of cycles	27604	40420	15025	25348	14538
# of instructions	5404	5404	5404	5021	5021
CPI	5.10	7.47	2.78	4.67	2.68
MIPS	$7.84 * 10^0$	$9.37 * 10^0$	$2.26 * 10^1$	$9.34 * 10^0$	$2.09 * 10^1$
latency(ns)	$2.50 * 10^{-8}$	$7.13 * 10^{-8}$	$7.94 * 10^{-8}$	$1.15 * 10^{-7}$	$8.90 * 10^{-8}$
execution time	$6.89 * 10^{-4}$	$5.76 * 10^{-4}$	$2.39 * 10^{-4}$	$5.81 * 10^{-4}$	$2.59 * 10^{-4}$

Table 2: Processoers performance when latency equal to 6

Criteria	Singlecycle	Pipeline without cache	Pipeline with cache	Multicore single-thread	Multicore dual-thread
Frequency(CPU)Hz	$4.02 * 10^7$	$7.28 * 10^7$	$5.99 * 10^7$	$4.23 * 10^7$	$5.66 * 10^7$
# of cycles	41406	60698	17187	28970	16723
# of instructions	5404	5404	5404	5021	5021
CPI	7.66	11.21	3.18	5.34	3.08
MIPS	$5.25 * 10^0$	$6.49 * 10^0$	$1.88 * 10^1$	$8.08 * 10^0$	$18.36 * 10^0$
latency(ns)	$2.49 * 10^{-8}$	$6.87 * 10^{-8}$	$8.33 * 10^{-8}$	$1.16 * 10^{-7}$	$8.83 * 10^{-8}$
execution time	$1.03 * 10^{-3}$	$8.32 * 10^{-4}$	$2.86 * 10^{-4}$	$6.70 * 10^{-4}$	$2.95 * 10^{-4}$

Table 3: Processoers performance when latency equal to 10

Criteria	Singlecycle	Pipeline without cache	Pipeline with cache	Multicore single-thread	Multicore dual-thread
Total logic element	3236/114480 (3%)	3846/114480 (3%)	7651/114480 (7%)	16145/114480 (14%)	18545/114480 (16%)
Total combinational functions	2957/114480 (3%)	3560/114480 (3%)	6581/114480 (6%)	14222/114480 (12%)	15893/114480 (14%)
Dedicated logic registers	1316/114480 (1%)	1791/114480 (2%)	4258/114480 (4%)	8322/114480 (7%)	8593/114480 (8%)

Table 4: FPGA resource when latency equal to 6

$$\text{CPI} = \frac{cycles}{instructions}$$
$$\text{execution time} = \frac{1}{Freq} * CPI * numberOfInstruction$$
$$\text{MIPS} = \frac{numberOfInstructions}{executiontime} * 1 * 10^{-6}$$
$$\text{latency(Pipeline)} = \frac{5}{Freq}$$
$$\text{latency(SingleCycle)} = \frac{1}{Freq}$$

4 Conclusion

After the pipeline design without caches, the number of cycles are increased. However, due to the reason that the frequency is improve hugely, the final performance and CPI are improved. What's more, to reduce the times of NOP to improve the performance, we use static branch prediction, which means that every branch will be predicted NOT-TAKEN. If the prediction is correct, we don't need to insert three times NOPs. If the prediction is incorrect, we can just flush the three wrong instructions right after the branch instruction and branch to the correct location after three NOPs. Therefore, the CPI will be closer to 1.

The frequency for pipeline with caches is lower, which is reasonable, because the circuit is becoming more complex. I think the latency setting for 2, 6, 10 is not enough to estimate the RAM latency at which Caches start helping performance. For latency 2, the execution time has already been lower than pipeline without caches. I believe when latency is equal to one or zero, it is likely that the execution time for pipeline with caches is lower than pipeline without caches. When latency is zero, because of the extra cycles to load block data(each block has two data to be loaded), it is likely to have longer time for pipeline with caches.

The multicore design (both single or dual thread) has lower frequency comparing to pipeline, because the design becomes more complex by adding bus controller and MSI. Comparing between single-threaded and dual-threaded, the execution time for dual-threaded is decreased. The reason for that is we break several critical path when our group is doing the dual-threaded design, which hugely improves the frequency.

The FPGA resource for the latency 6 shows the complexity of the design. For SingleCycle, the logic element and combinational functions are only around 3 percent. The complexity keeps increasing from SingleCycle to multicore dual-threaded. This is reasonable since we are keeping adding combinational blocks and registers every design. At the end, we got 16

5 Contributions

Xinyuan Cai:

Lab8: d-cache design, i-cache test bench

Lab10: Bus controller source code

Lab11: Modify bus controller

Lab12: Implement LL-SC

Xiangyu Guo:

Lab8: i-cache design, d-cache test bench, unit assembly file

Lab10: Bus controller diagram, bus controller test bench, unit test assembly file

Lab11: Modify cache controller

Lab12: Parallel algorithm assembly test

Draw and modify each RTL diagram

Together: Lab9, Design and Debug the overall process