ECE 43700: Computer Design and Prototyping Lab
Final Report

Vaibhav Ramachandran

Asheem Chhetri

Lab Section: 3

GTA: Manik V. Singhal

Due Date: April 29th, 2018

1 Executive Overview

This report aims to analyze the performance improvements that were achieved by our dual-core and single-core cache designs compared to our single-core pipelined processor without caches. It will also aim to reason the benefits of each design using the data obtained from the test program.

The initial pipelined processor design was observed to perform horribly when RAM memory latency was introduced. It's CPI and execution time were extremely high due to the fact that the test program involved several R/W memory operations. As such, overall performance was far below average, despite the design's high clock frequency. To resolve it, we implemented an L-1 cache for both instructions and data in order to cut down average memory access times. The caches took advantage of the concepts of spacial and temporal locality to provide a significant boost in performance over the previous design. It cut memory access times to about 1/3rd of the previous values while improving the CPI and execution times by a factor of 3. The only drawback of adding an L-1 cache hierarchy was the drop in maximum clock frequency, due to the introduction of longer critical paths. To achieve even greater performance, a dual-core processor, with each core having it's own dedicated cache hierarchy, was then implemented. The primary reason was to double the performance, since a dual-core design can complete twice as many instructions as a single-core processor, in the same time period. Unfortunately, it also introduced the issue of cache coherency. To rectify it, we altered the memory controller, which was previously a pass through controller, to implement an MSI cache coherence protocol. Despite the increased drop in maximum clock frequency, due to longer critical paths, the addition of a second core provided further performance improvements over our cached single-core design. With each iteration of our design more of the FPGA's resources were used, due to the larger number of registers and state machines required. However, execution time and overall throughput showed improvements.

The test program that was used was the mergesort ASM file thanks to it's coverage of the full MIPS ISA and it's high percentage of memory accesses. It helped provide a definitive view of the differences in performance between the different designs.

2 Processor Design

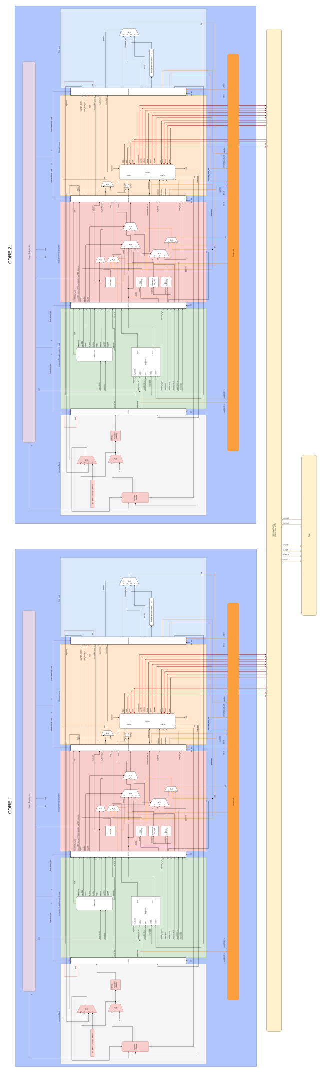


Figure 1: Multi-Core Block Diagram

Link to full-size image: Multi-core Block Diagram

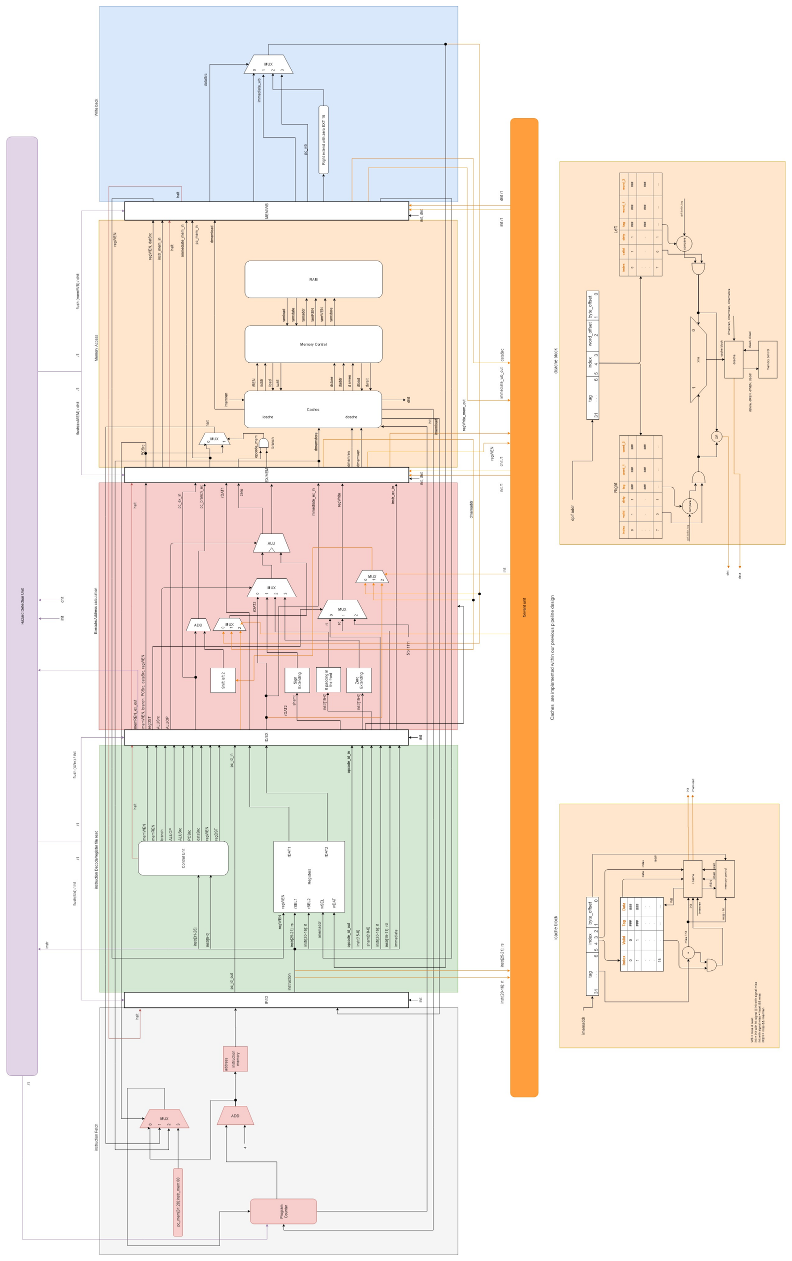
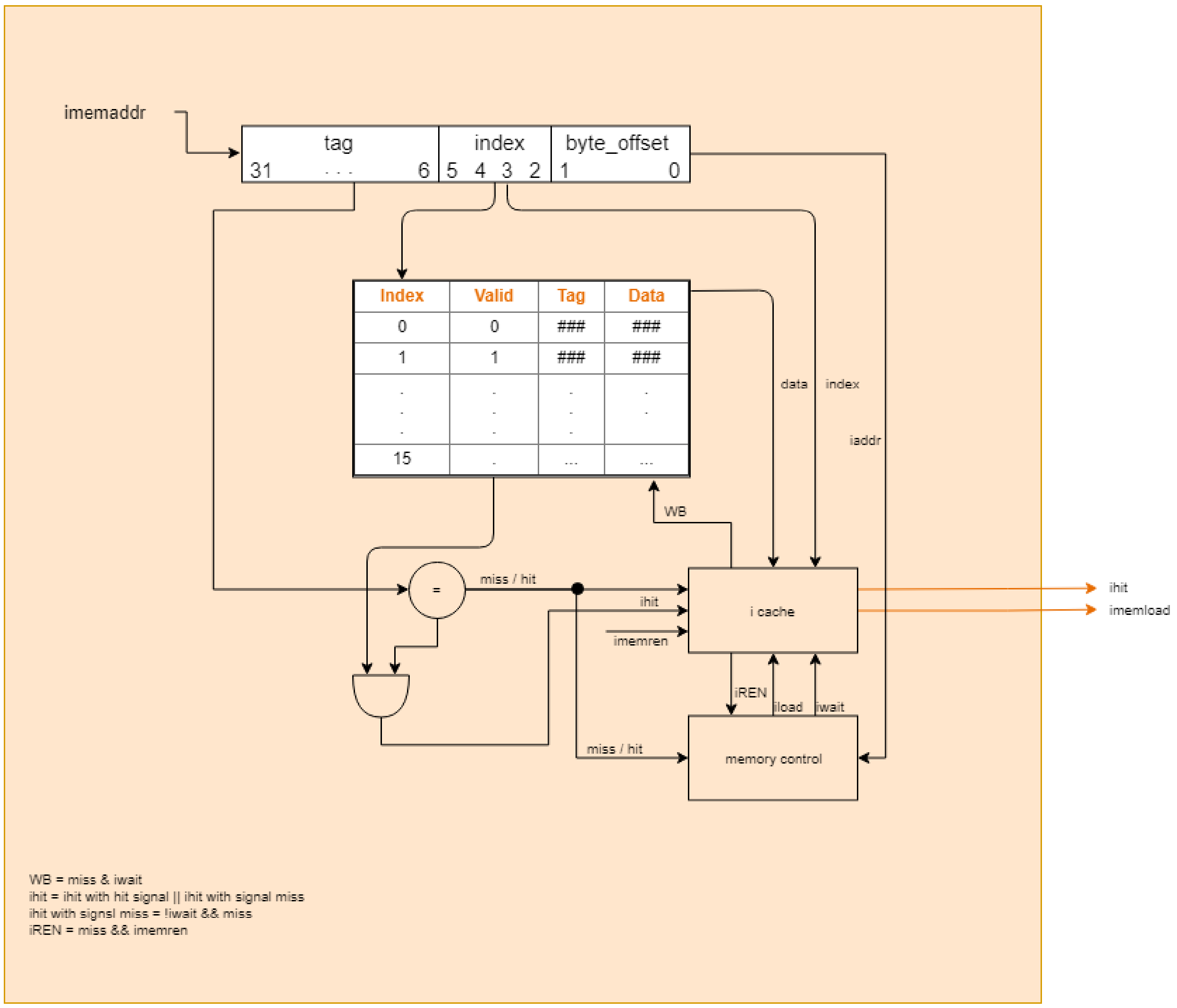


Figure 2: Single-Core Pipeline Block Diagram with Caches

Link to full-size image: Single-core Block Diagram



WB = miss & iwait
ihit = ihit with hit signal || ihit with signal miss
ihit with signsl miss = !iwait && miss
iREN = miss && imemren

Figure 3: I-cache Block Diagram

Link to full-size image: I-cache Block Diagram

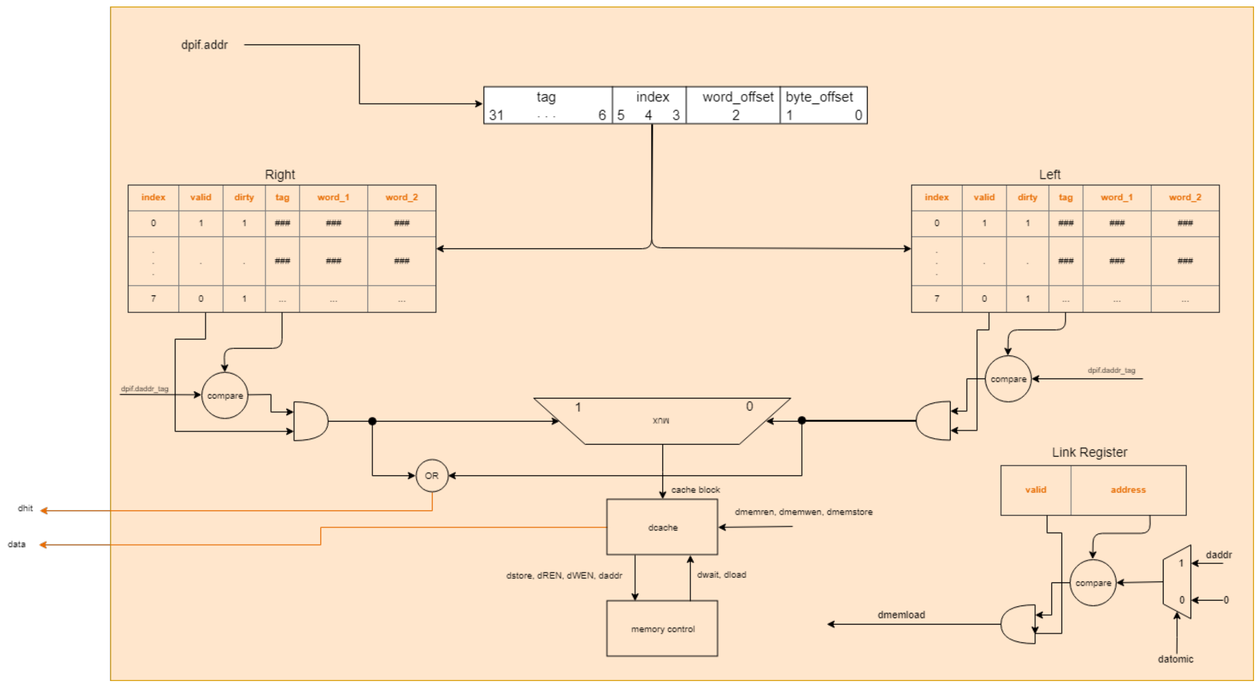


Figure 4: D-cache Block Diagram

Link to full-size image: D-cache Block Diagram

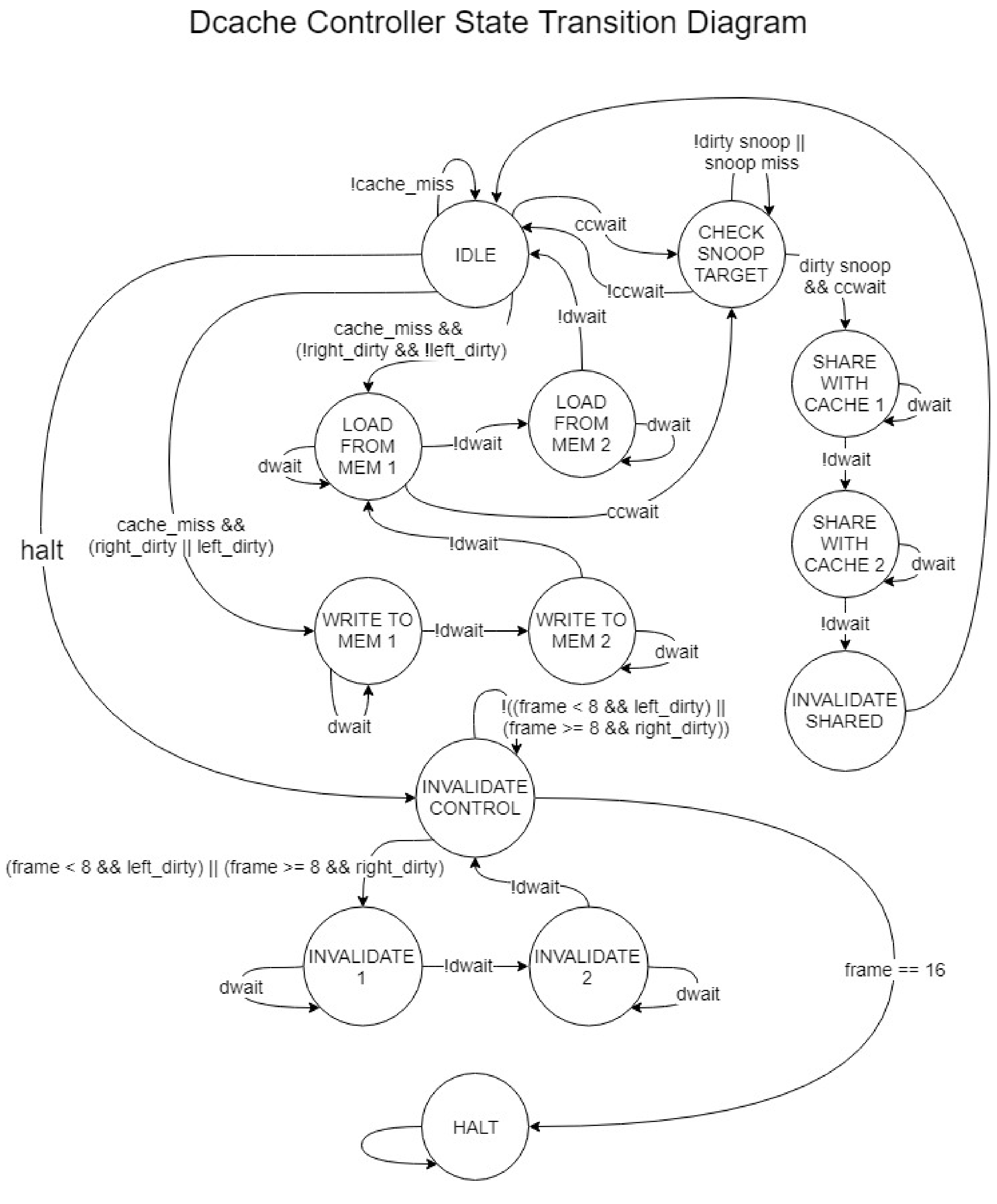


Figure 5: D-cache State Transition Diagram

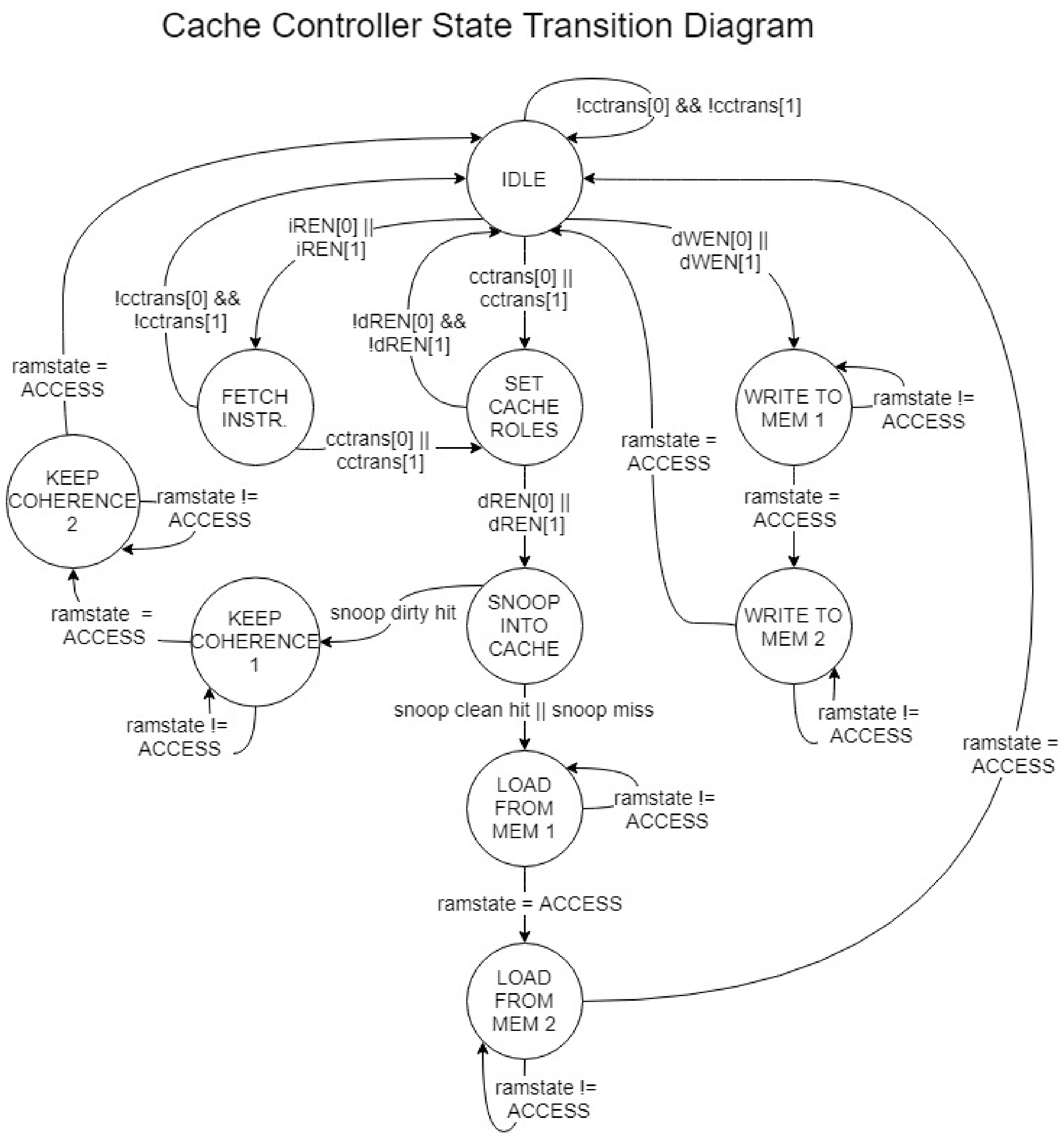


Figure 6: Cache Controller State Transition Diagram

3 Results

Following are the results of our comparison between the Pipelined processor with and without Caches, along with the Multicore design:

Memory Latency used to generate this table was: 5

ASM file used for the comparison: mergesort.asm / dual.mergesort.asm

Criterion	Pipeline without Cache	Pipeline with Cache	Multicore
Number of Instructions	5399	5399	5414
Execution Time(μs)	1275.564	468.384	455.96
Max Frequency(85c model)(MHz)	51.4	45.8	36.67
Number of Cycles	65564	21452	16720
IPC(Instr. / Cycles)	0.08235	0.2517	0.3238
CPI(Cycles / Instr.)	12.1433	3.973	3.088
MIPS	4.233	11.528	11.874
Critical Path(ns)	14.454	16.833	26.018
Latency per instructions(μs)	0.4864	0.5459	0.6818
Speedup	1.000	2.723	2.798
Registers (/114480)			
Total Logic Elements	4237	8332	21348
Total Combinational Functions	3853	7089	19135
Dedicated Logic Registers	2006	4477	8786
Total Registers	2007	4478	8786

Table 1: Synthesis Results

Formulas Used:

$$Instructions\ Per\ Cycle = \frac{\#\ instructions}{\#\ cycles} \tag{1}$$

$$Execution\ Time = \frac{1}{fMax} * \#\ cycles \tag{2}$$

$$MIPS = IPC * fMax \tag{3}$$

$$Latency = \frac{1}{fMax} * (Memory\ Latency) * 5 \tag{4}$$

$$Speedup = \frac{Execution\ Time}{Execution\ Time\ of\ Pipeline\ without\ Cache} \tag{5}$$

4 Conclusion

From the above table it can be seen that with each successive iteration, the execution time, CPI, max frequency and the number of cycles have reduced. Also, the throughput, critical path, the latency per instruction and the number of registers have increased. This is in accordance with what was expected once the improvements were made. The drop in execution time, CPI and number of cycles was primarily due to the introduction of caches which enabled much quicker execution of loads and stores by taking advantage of spacial and temporal locality of data. This helped reduce RAM accesses by a significant amount and thus contributed to an overall performance boost. The performance was further improved thanks to the introduction of a second core, though the improvement was not as significant as adding caches. This could be because of the huge increase in the critical path since a coherence protocol needed to be added between the two D-caches. Going through the system log file also revealed the presence of a combinational loop within the D-cache design. A cache-to-cache data transfer would therefore include two of these combinational loops. This likely was a huge factor in the sudden increase in the length of the critical path when implementing the dual-core design. It's likely due to this, that the maximum clock frequency dropped by such a significant amount. This in turn, could've caused the execution time, throughput, overall speedup and the CPI to improve only by an incremental amount, instead of the expected doubling of performance.

As expected, the number of registers used almost doubles/triples with each successive iteration, thanks to the introduction of I-caches and D-caches, a coherence protocol and a second core; each of which require a large number of registers and combinational blocks. However, our final dual-core design achieved what was expected; much better overall performance despite the reduced clock frequency, increased latency and increased critical path. This made it the most superior design compared to the other two.

Further improvements that could be made would be the removal of the combinational loop in the D-cache and the implementation of a MOESI coherence protocol to reduce write backs to RAM.

5 Contributions

Name	Source Files	Test Benches
Asheem Chhetri	D - cache(Lab 8) D - cache(Lab 11) LL/SC implementation	I - Cache test-bench (Lab 8) Bus Controller test-bench(Lab 10)
Vaibhav Ramachandran	I - cache(Lab 8) Bus Controller(Lab 10) Palgorithm.asm	D - Cache test-bench (Lab 8) D - Cache test-bench(Lab 11)

Table 2: Team Contribution

Asheem, designed the block diagram for D-cache, I-cache and Multicore CPU block diagram. While on design side he worked with D-cache for both Lab 8, 11 and implemented the LL/SC aspect of Multicore design. While on test-bench side, he worked with I-cache test-bench for Lab 8 and Bus Controller test-bench for Lab 10. He also helped debugging the Multicore design, specifically with D-cache design and finding bugs in the control_unit from previous labs, which was then resolved by Vaibhav.

Vaibhav, designed the state diagram for the D-cache block and cache-controller. He also updated the D-cache state diagram for Lab 12. Going further, he designed the palgorithm assembly file for testing the Lab 12 Multicore design. He also handled the source code implementation of I-cache for Lab 8, and Bus Controller for Lab 10. While on test-bench side he focused on D-cache testing for both Lab 8 and 11. Eventually, his thorough testing method picked errors, which were overlooked during design phase of D-cache. Also, interesting enough he found a subtle bug in the control_unit, which was left undetected until Lab 11.