Final Report

ECE 43700

Jordan Huffaker & Lucas Krull

Nick Pfister

12/9/16

1. Overview

The power of computation is becoming increasingly relevant in people's lives. The speed at which hardware can complete a set of commands in the MIPS instruction set is a foundational limitation to the speed at which an end user can achieve their desired goal. The traditional method, completing each instruction in a single clock cycle, is limited by the critical path of the longest instruction. We introduce two techniques that overcome this barrier. First, a pipelined processor that splits instructions into multiple stages and completes different stages of multiple instructions at the same time. Second, a multicore processor that allows for the execution of parallel programs.

We compared the effectiveness of a multicore and pipelined approach over the traditional method by comparing several metrics such as average IPC, MIPS, instruction latency, number of logic gates, and number of flip-flops. These metrics were gathered while completing a common task programs must complete for end users to use a computer: merge sorting an array. We found that the pipelined approach was the most effective in its MIPS performance; however, it came at the cost of average IPC, instruction latency, number of logic gates, and number of flip-flops over the traditional approach. We found that the multicore approach was the least effective in its MIPS performance, instruction latency, number of logic gates, and number of flip-flops; however, it was the most effective in its average IPC suggesting that a better implementation of the technique could lead to an overall improvement in MIPS performance.

2. Processor Design

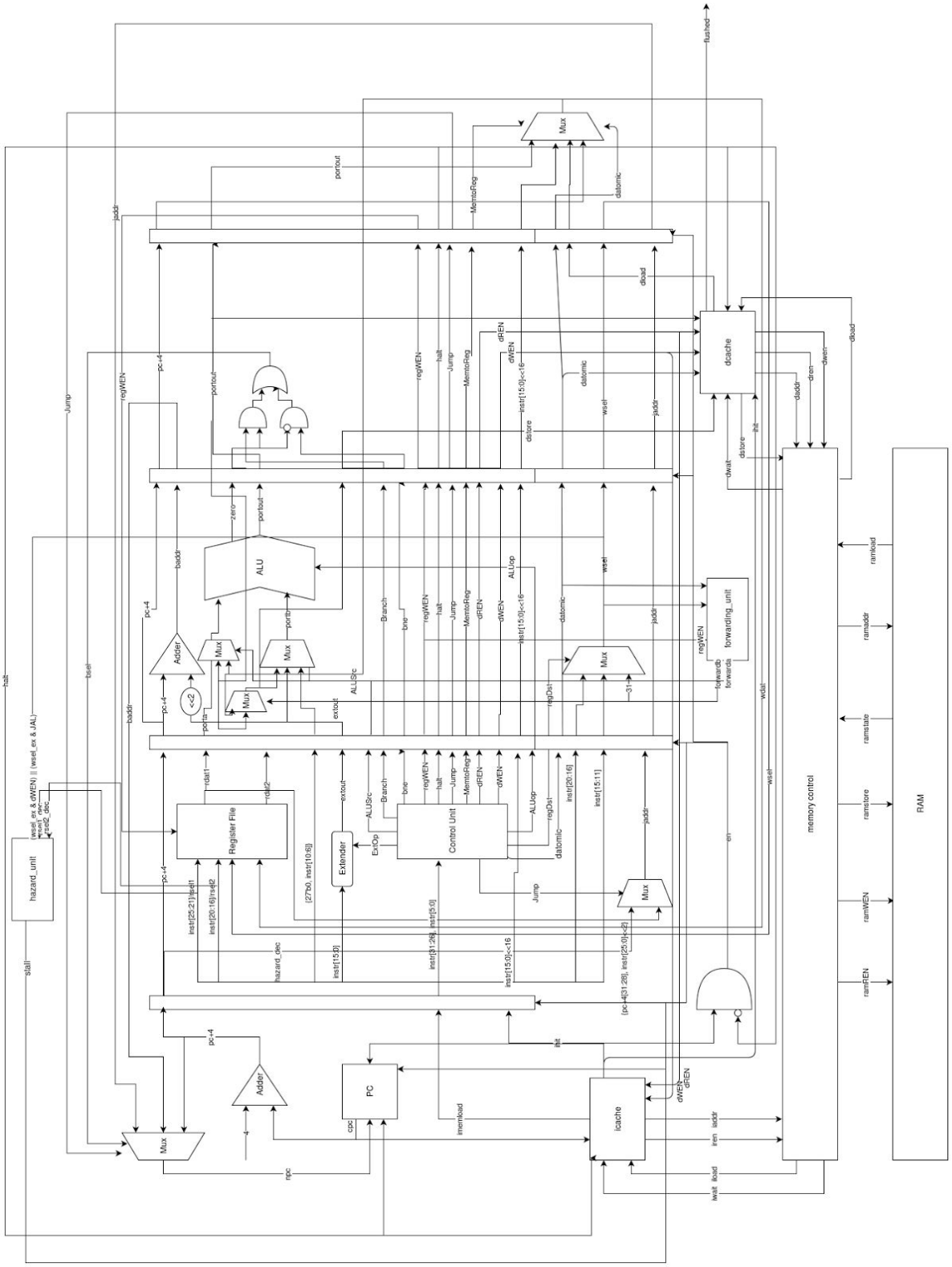

Figure 1: Pipelined Block Diagram with LL/SC

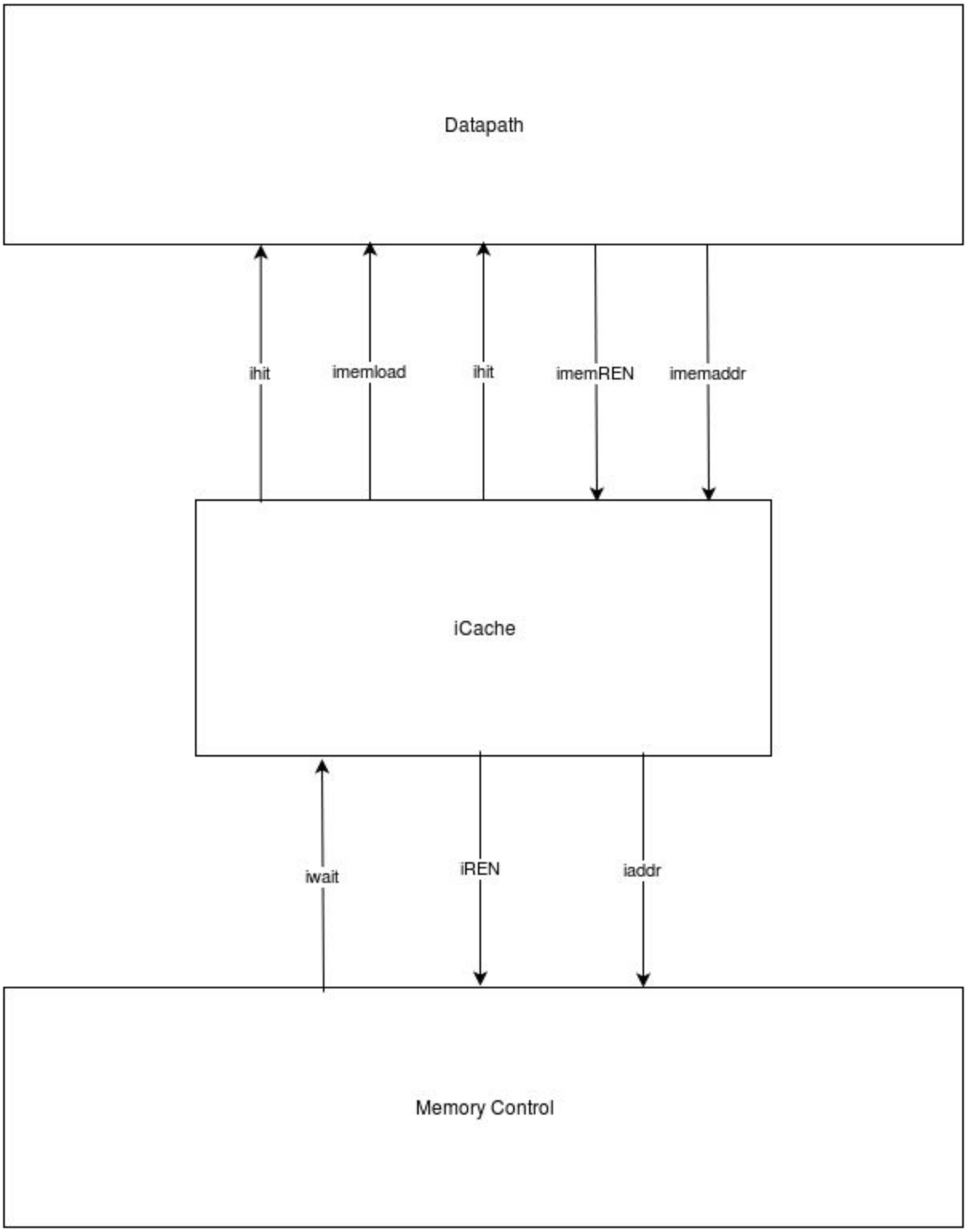

Figure 2. iCache Block Diagram

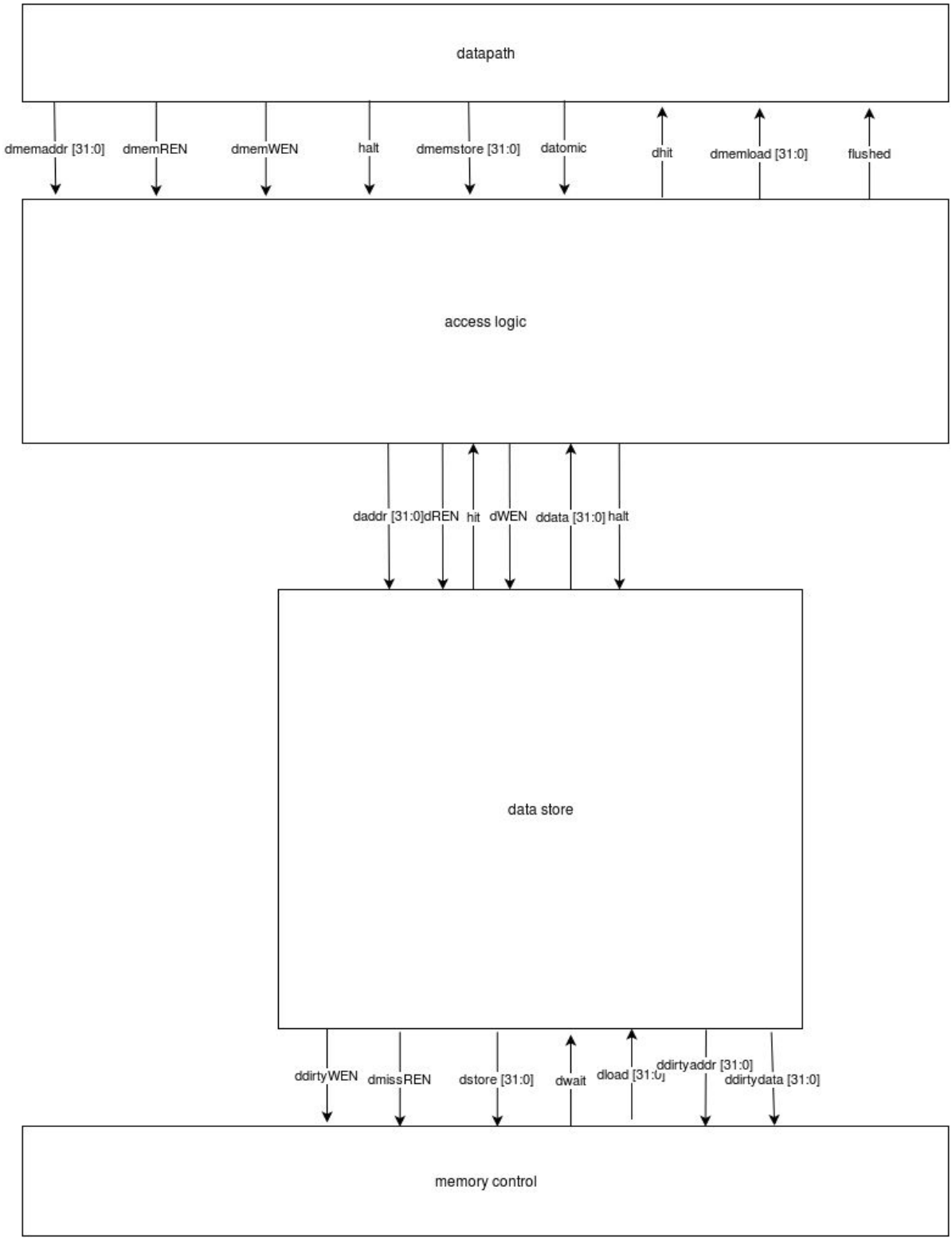

Figure 3. dCache Block Diagram

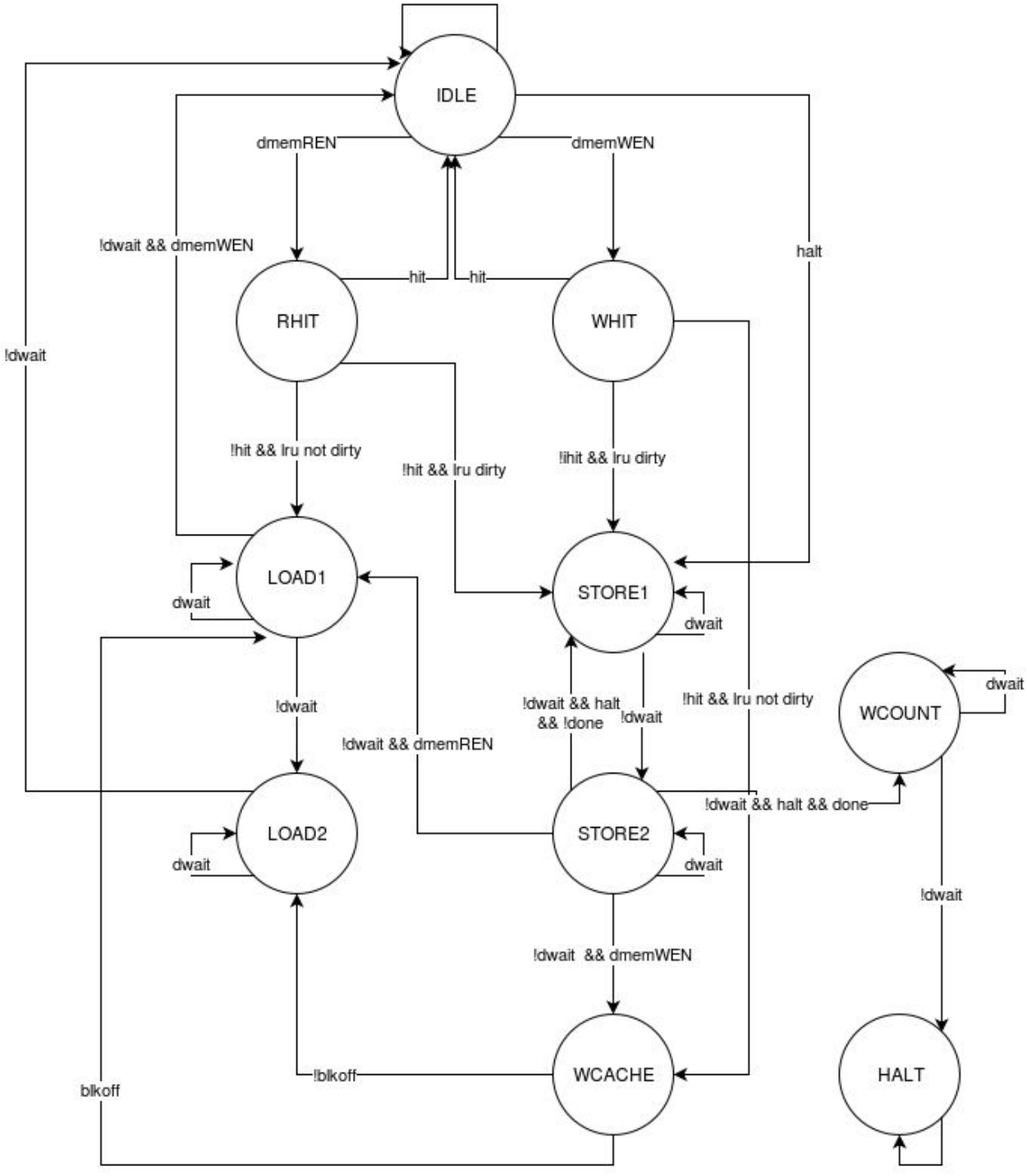

Figure 4. dCache State Diagram

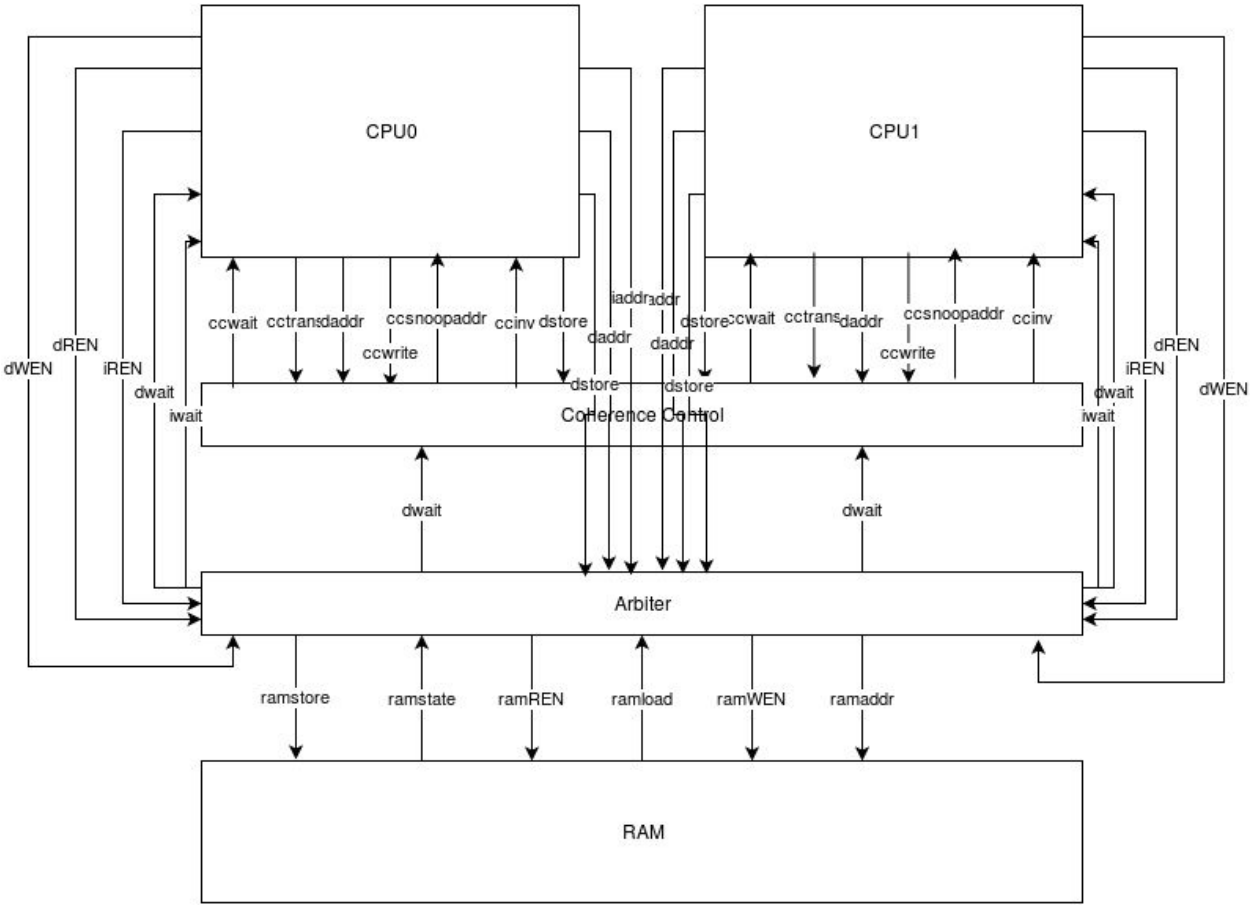

Figure 5. Coherence Block Diagram

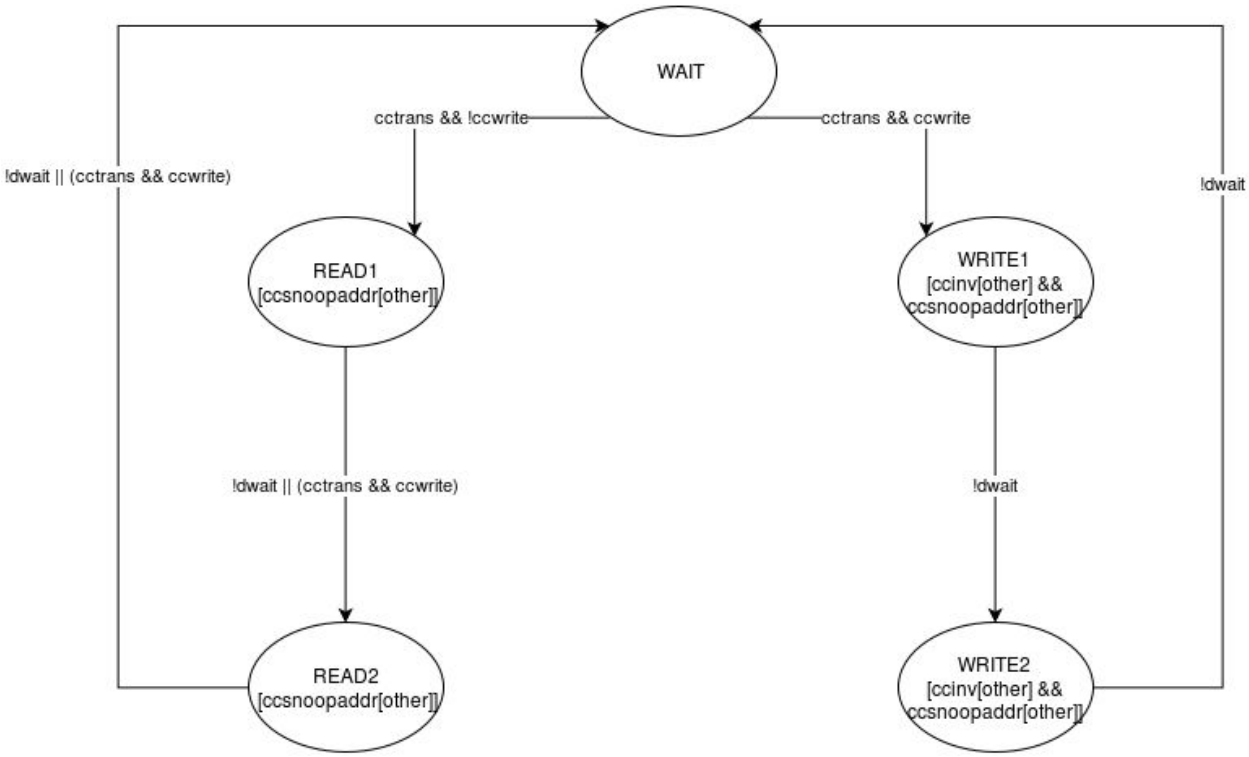

Figure 6. Coherence State Diagram

3. Results

Metric	SingleCycle	Pipelined	Multicore
Max Frequency (MHz)	31.71	66.64	20.94
Max Test Bench Freq. (MHz)	50	500	35
# of instructions	5,399	5,399	5,414
# of cycles	13,844	17,416	**11,000
Average IPC	0.39	0.31	0.49
*MIPS Performance	12.37	20.66	10.26
*Latency of One Instr. (ns)	31.5	75.03	23.88
# Logic Gates Required	3,153	3,497	8,674
# Flip-Flops Required	1,311	1,750	5,798
Speed-up	-	1.67	0.49

*uses the “Max Frequency” parameter in its calculation

**TA guided estimate for dual.mergesort.asm

- Max Frequency was calculated with the formula $min(CLK/2,\ CPUCLK)$ with CLK and CPUCLK obtained from the system.log files
- Average IPC was calculated with the formula $\frac{Total\ Instruction}{Total\ Cycles}$ with Total Instructions obtained from running “sim -t” and Total Cycles obtained from the report generated after running the synthesized simulation
- Latency of One Instr. was calculated with the formula $\#\ Stages * Period$ with Period calculated from Max Frequency in table
- MIPS was calculated with the formula $\frac{Average\ IPC * Max\ Frequency}{1{,}000{,}000}$ with Average IPC and Max Frequency from table

4. Conclusions

As was expected, our pipeline design achieved a significant performance improvement over our single cycle design. However, our multicore design did not achieve an improvement over our pipeline design. Pipelining allowed our design to more than double its maximum clock frequency while using two cores cut the maximum frequency by a nearly a third. This result can be mostly attributed to the long, slow cache coherence bus. These clock speed changes resulted in an improvement in MIPS performance of nearly 2x for our pipeline design over our single cycle design and a performance degradation of nearly 2x for our multicore design over our pipeline design.

However, some metrics did worsen due to pipelining and some did improve due to the use of two cores, though these were expected. The latency of a single instruction increased from our single cycle design to our pipeline design and from our pipeline design to our multicore design. These are affected by stalling and the number of stages, which for our pipeline and multicore designs was 5 and for single cycle was 1. The average IPC decreased from single cycle to pipeline designs. Although, the average IPC decreased due to the necessity for stalling, the overall increase in clock frequency more than compensates for this and results in an overall increase in performance. On the other hand, the average IPC increased from pipeline to multicore designs. This is due to the parallelization of the multicore design, which allows the processor to do more work per cycle. Additionally, our multicore design requires more FPGA resources compared to our pipeline design and our pipeline design requires more FPGA resources compared to our single cycle design. This is due to the necessity for pipeline registers and the forwarding and hazard units from single cycle to pipeline and the entire second core, caches, and coherence blocks from pipeline to multicore.

5. Contributions

Jordan Huffaker

- Updated block diagram
- Created coherence block diagram
- Created cache state diagram
- Cooperated on all coding and debugging
- Wrote overview, and aided in result compilation for report

Lucas Krull

- Updated pipeline block diagram
- Created caches block diagram
- Created coherence state diagram
- Cooperated on all coding and debugging
- Wrote conclusion, inserted block diagrams, and aided in result compilation for report