Final Report

ECE 437: Computer Design and Prototyping

Stephen Bulley & Jun He

Chuan Tan & Jiangshan Jing

December 9, 2016

Overview

This report analyzes the performance improvements made by our cache design and our dual processor coherence control in comparison to our original pipeline design. Throughout this report, we will overview our whole design, design choices, and results.

After completing our pipeline design, we immediately began working on our cache design to improve our time when reading and writing to memory. The first thing that we implemented was a cache to introduce the idea of locality. Our cache holds recently used values and releases the least recently used ones. Our cache is much smaller than our ram which means that reading and writing to the cache shows a distinct timing advantage over r/w to ram at every opportunity. Of course, nothing is free and, to implement this speed upgrade, we required additional registers and blocks of logic. Fortunately, the payoff is clear and is a great improvement to our previous pipeline design.

After our cache, we took our overall design and began creating a dual core processor. The main motivation for a dual core processor is obvious. With two processors, you can accomplish twice as many instructions in the same amount of time. Unfortunately, this benefit also comes with a coherency issue. We would have to generate an algorithm to compensate for the fact that both processors may be reading and writing the same data at the same time. To implement two processors, we had to create a coherency controller to deal with the caveat. Again, however, the timing benefits far exceeded the extra work involved in generating this controller.

To demonstrate our improvements, we will use dual.mergesort.asm as a benchmark program to present the incremental transformation of our design. We will take note of the improvements in Clock Frequency, number of clock cycles, Instructions per Cycle, instruction latency, FPGA resources required, run-time, and overall speedup. These metrics will be used to measure our pipeline, pipeline with cache, and our dual pipeline with cache.

Design

Figure 1. Overall Dual Core Design

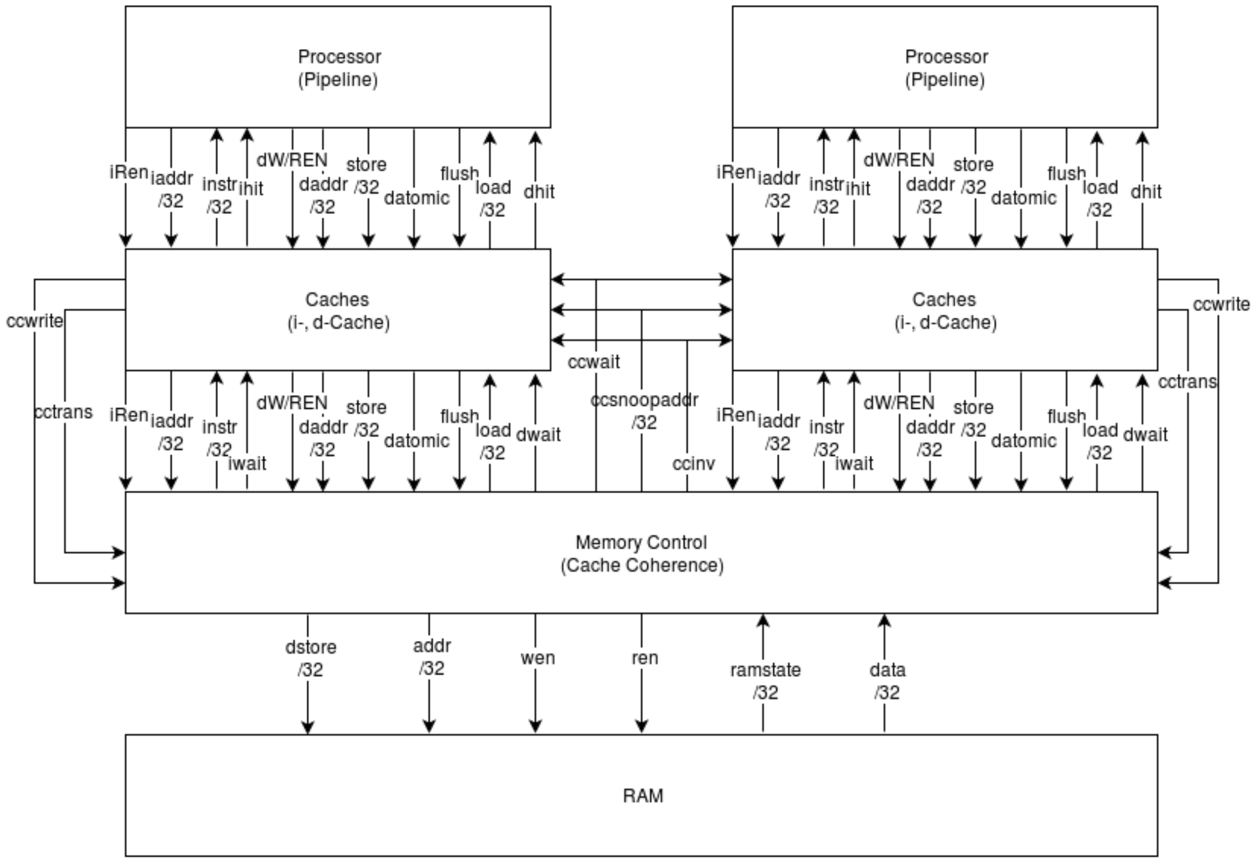


Figure 2. Processor (Pipeline) Design

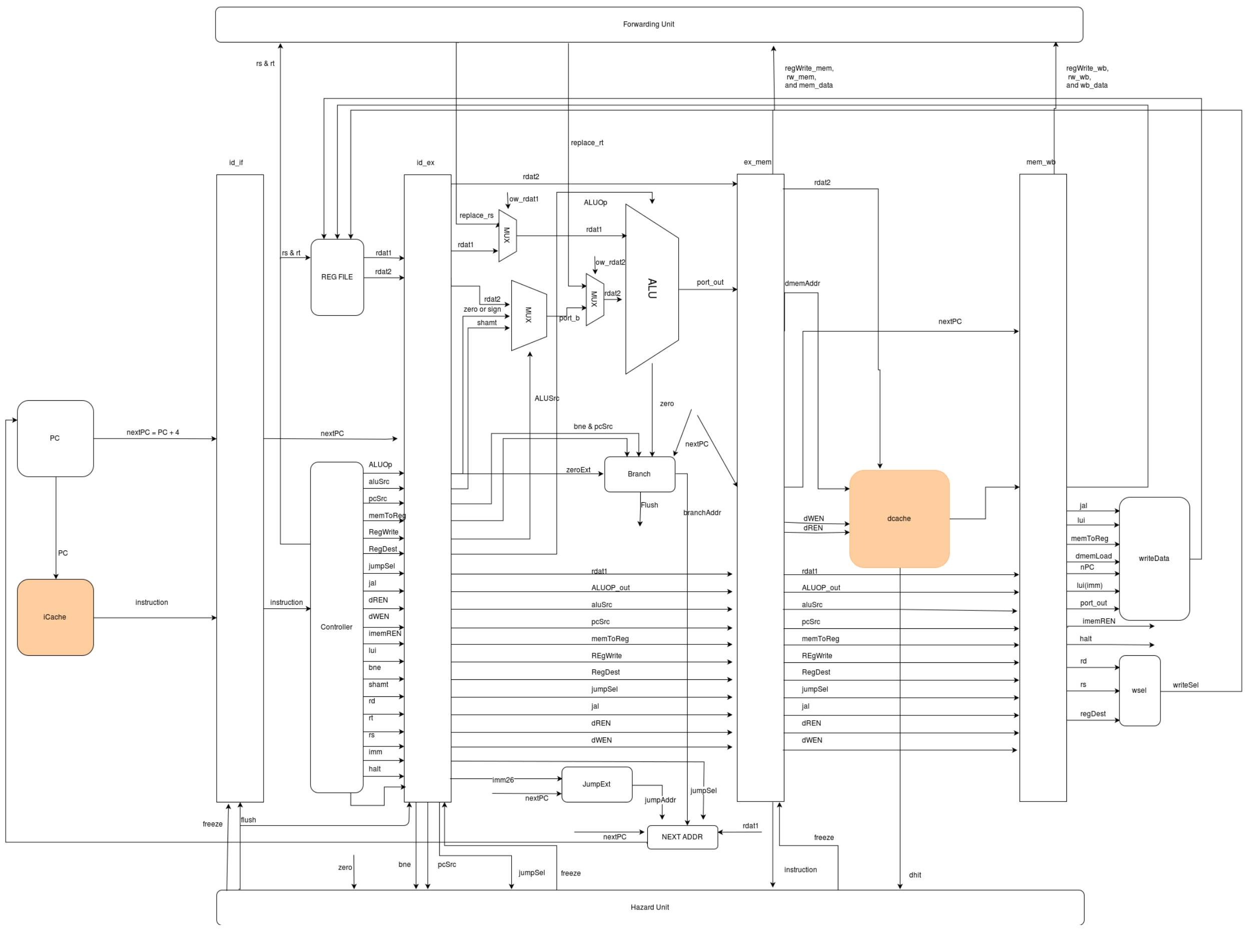
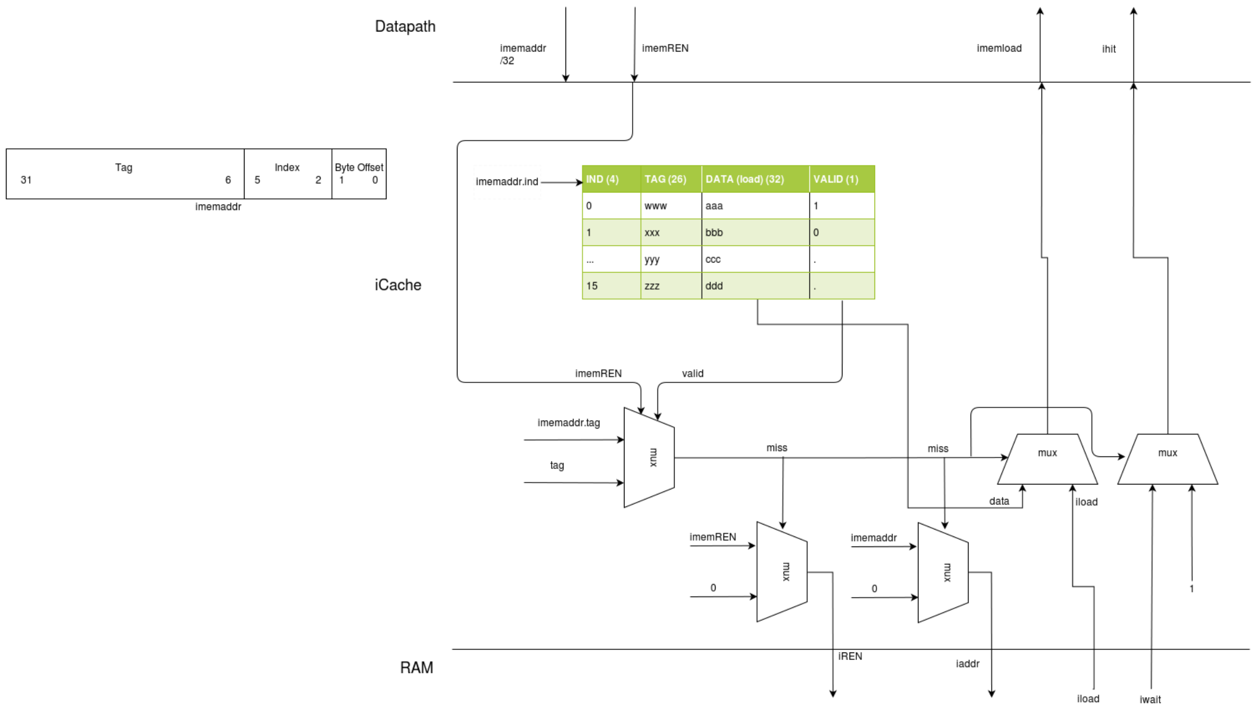
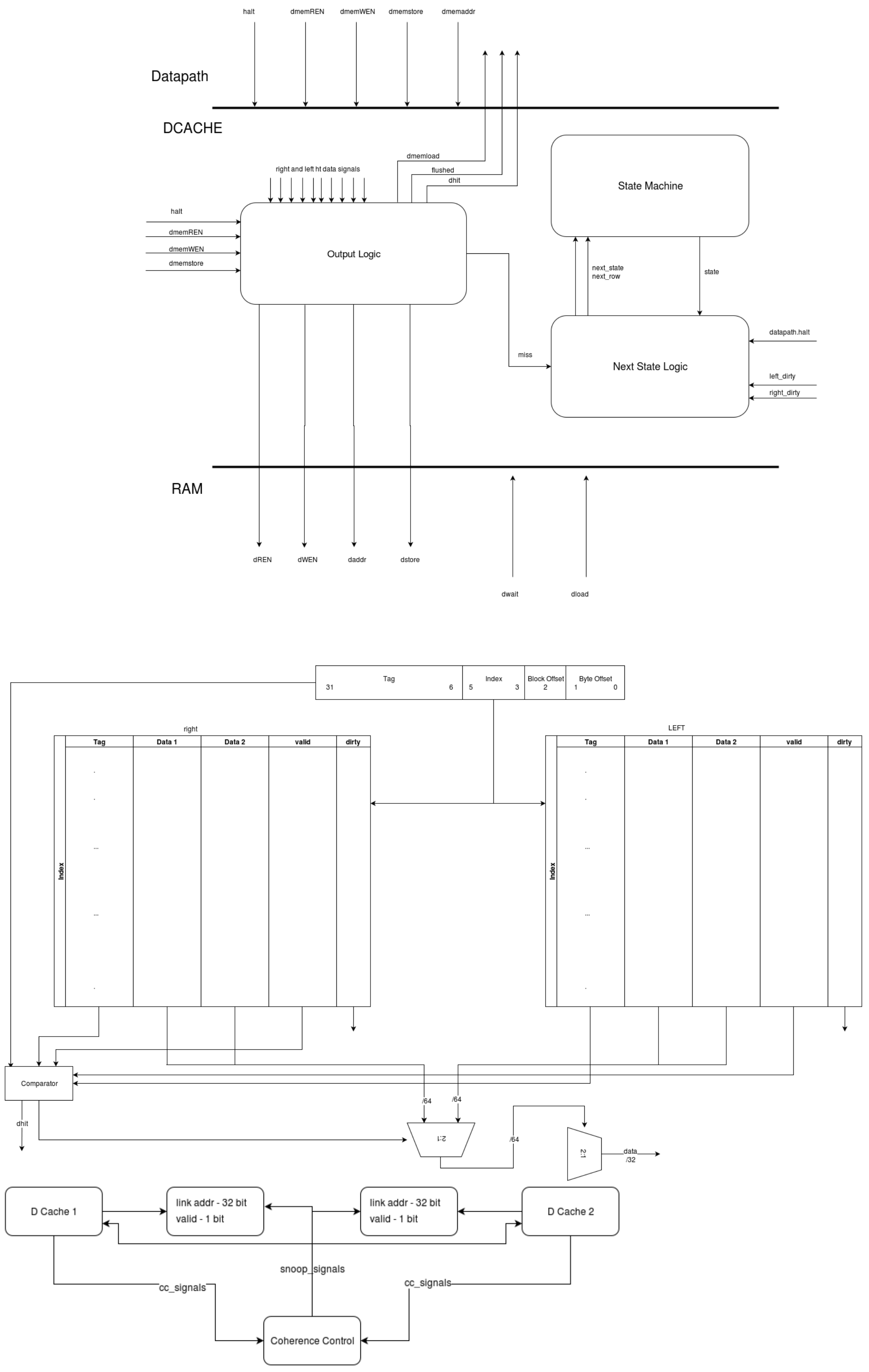


Figure 3. i-Cache Design



IND (4)	TAG (26)	DATA (load) (32)	VALID (1)
0	www	aaa	1
1	xxx	bbb	0
...	yyy	ccc	.
15	zzz	ddd	.

Figure 4. d-Cache Design (Linked Register) with State Diagram



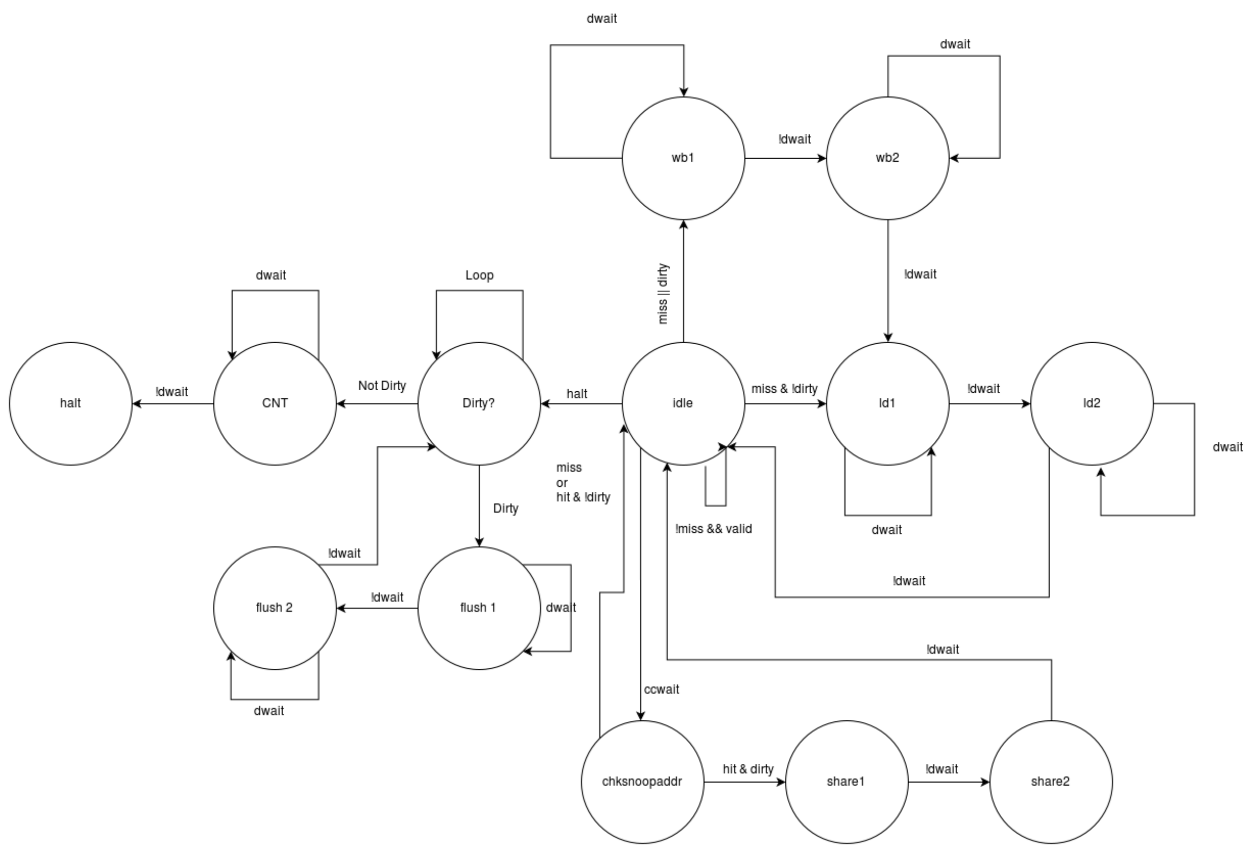
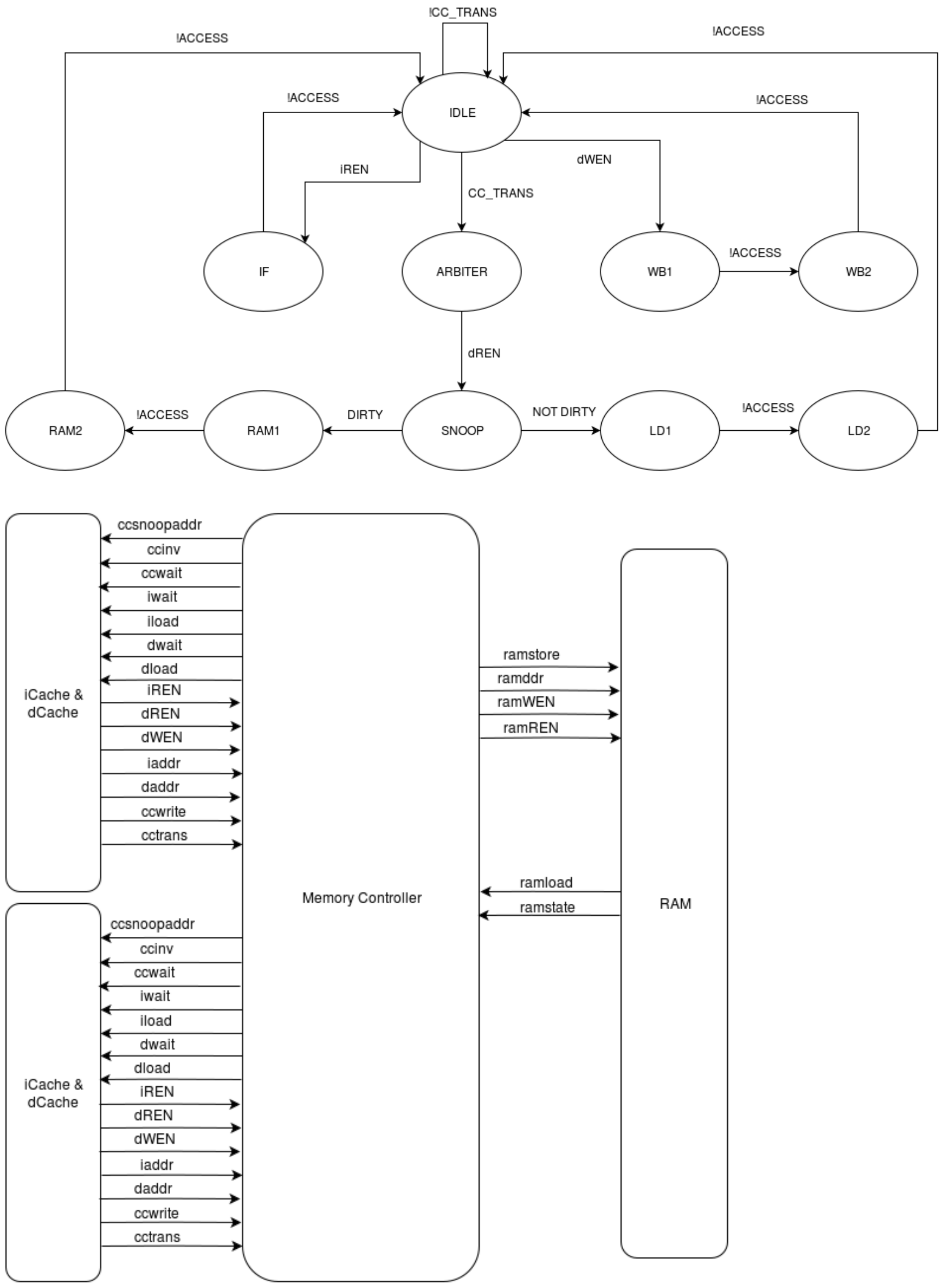

dwait
dwait
wb1
!dwait
wb2
!dwait
miss || dirty
dwait
Loop
halt
!dwait
CNT
Not Dirty
Dirty?
halt
idle
miss & !dirty
ld1
!dwait
ld2
dwait
miss
or
hit & !dirty
Dirty
!miss && valid
dwait
!dwait
flush 2
!dwait
flush 1
dwait
!dwait
!dwait
dwait
ccwait
chksnoopaddr
hit & dirty
share1
!dwait
share2

Figure 5. Memory Control Design with Coherence State Diagram



Results

The results of our testing and simulations are as follows (latency of 6 was used for results):

	Pipeline W/O Cache	Pipeline W/ Cache Stats	Multicore Stats
Frequency	45.52	40.07	29.61
Instructions	5399	5399	5414
Clock Cycles	17225	22759	18615
Instructions/clock Cycle	0.3134397678	0.2372248341	0.2908407198
Latency	1.10E-007	1.25E-007	1.69E-007
FPGA Resources	1856	7789	20891
runtime (seconds)	2.20E-008	2.50E-008	3.38E-008
Speedup	1.000	0.880	0.739

Conclusions

As can be seen from the data, despite our improvements in design, we managed to worsen our performance in the case of merge-sort. Unfortunately, this may be caused by our small sample size. The merge-sort algorithm that we implemented may not sort enough values to fully utilize the benefits of the cache that we implement. The number of cache misses may overshadow the number of cache hits and in turn may negatively affect the instructions per clock cycle and run time. Additionally, the overheads involved in cache coherence may outweigh the performance boosts of having two processors working on such a small sample set. The performance improvement would most likely become more evident as the number of instructions grow to infinity. We believe that these considerations constitute the lack of an overall speedup in comparison with the original pipeline W/O cache design.

The increase in FGPA resources is expected in the addition of the cache to hold all the values. Additionally, the dual core design should more than double the resources from the cache design and we witness that fact in the data above. Not only are we doubling the size requirement of one processor, but we are also adding wires and registers to hold the cache coherence protocol.

Overall, our design achieved what we predicted, however it shows that there is a long way to go. Our next step would be to improve the critical path of our coherence controller, implement a MOSI protocol to reduce write backs to ram, and to improve our overall pipeline. Additionally, we would like to realize an overall speedup on every program run on our dual core design, so whatever improvements need to be made to accomplish that feat would be on our to-do list.

Team Contributions

Stephen Bulley – Conceptualized and planned Cache and Multicore design, wrote test benches, helped debug, data collection for report, checked grammar and wording of final report

Jun He – Planned and coded the Cache and Multicore design, debugged whole design, wrote rough draft of the final report