Final Report

ECE 437: Computer Design and Prototyping

Karim Itani, Milind Shyam

Lab Section 3

April 29, 2018

Overview

The purpose of this report is to compare the performance of our implementation using the cache design and multicore to the original pipeline design. In the report, we will show our overall design, with all the completed diagrams throughout the semester and the results.

Once we were done with pipeline stage, we started the cache design to be able to improve read and writes to memory. The cache is smaller than RAM and allows faster access to data that are used regularly which increased the overall speed of our program. First, the cache checks if it contains the data we are looking for and a hit occurs if it is found, otherwise it is a miss. It will have to go get it from main memory and put it back in the cache. It is constantly updated with new values where the least recently used values are replaced (LRU). Obviously, with the addition of the cache design, we had to use extra resources (registers and logic blocks) which increases cost but also increases speed and performance.

Once the cache design was implemented, we started working on multicore design to take advantage of parallel programming and get even better performance. Having two processors allows to achieve twice as many instructions during the same amount of time. They can run different instructions at the same time if the coherency is taken care off obviously. A coherency controller was used to deal with the issue that the two processors might read and write the same data at the same time. Once again, the multicore design gave us a better IPC as more instructions were ran during same clock cycle.

Regarding the results analysis section, we used the dual.mergesort.asm file to get the data shown in the table below. The main characteristics used to compare the performance of the three designs were the Frequency, Instructions per clock cycle, Latency, MIPS, FPGA resources and finally Speed up.

Design

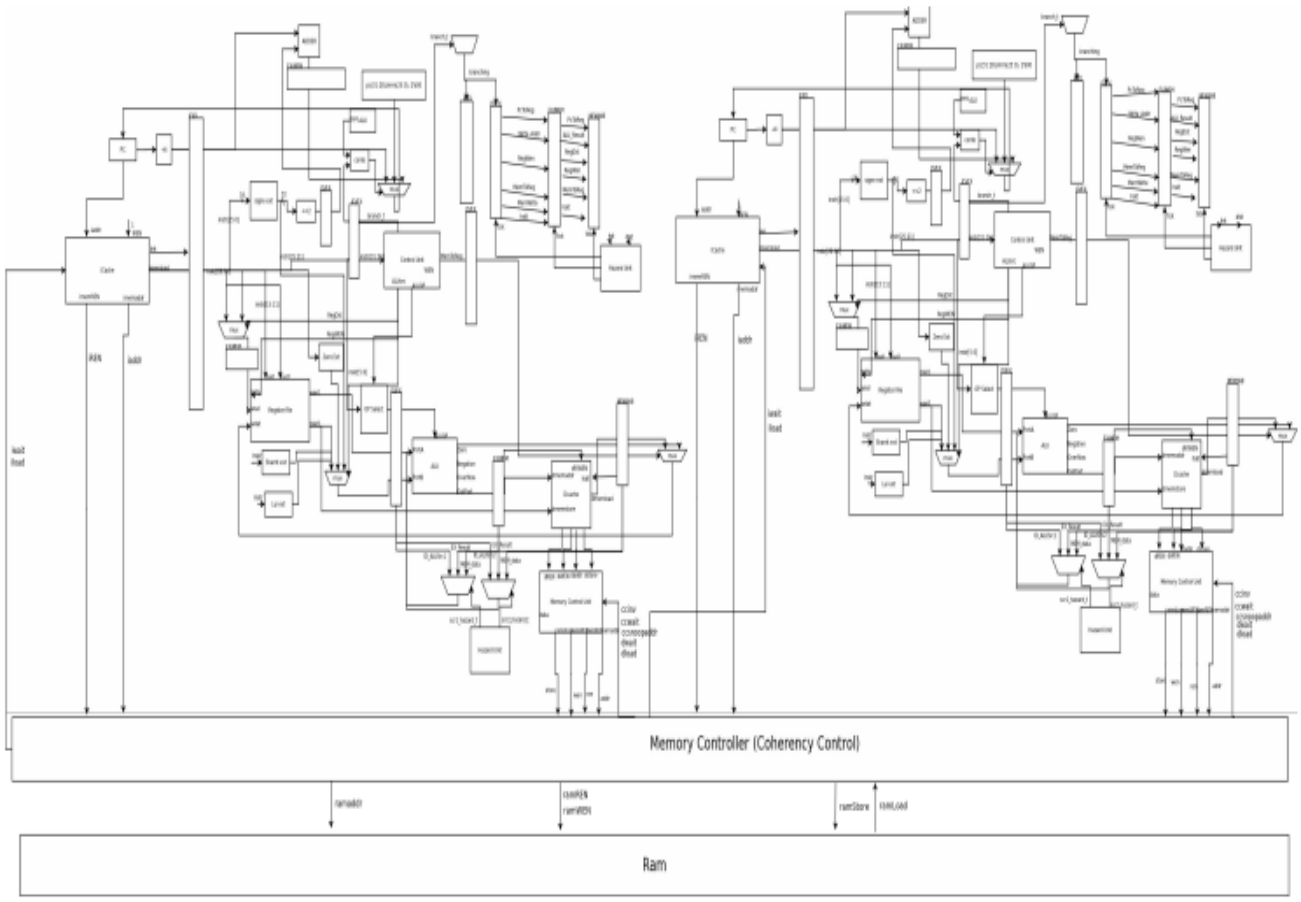

Figure 1: Multicore CPU Block Diagram

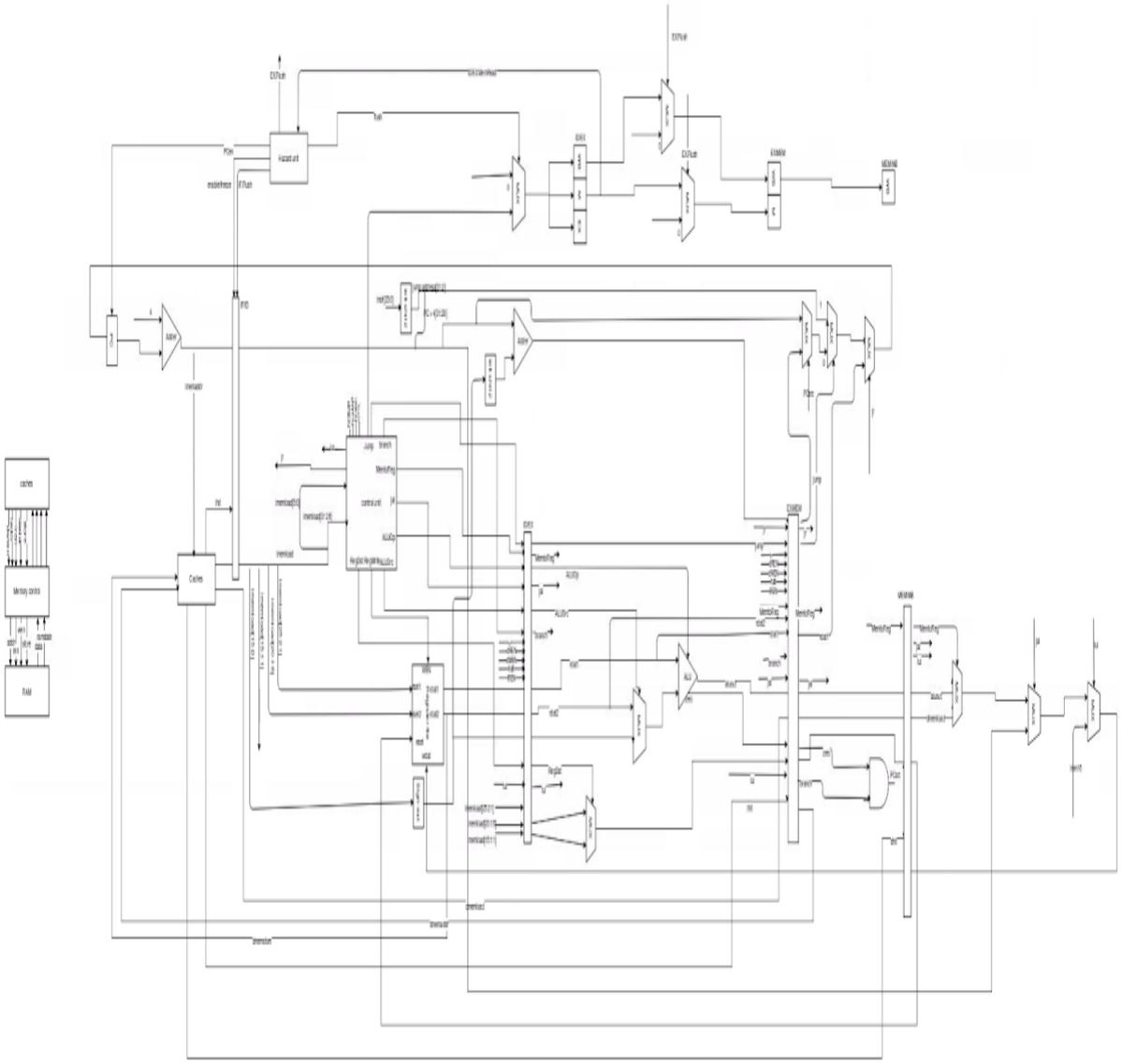

Figure 2: Pipeline Processor Schematic

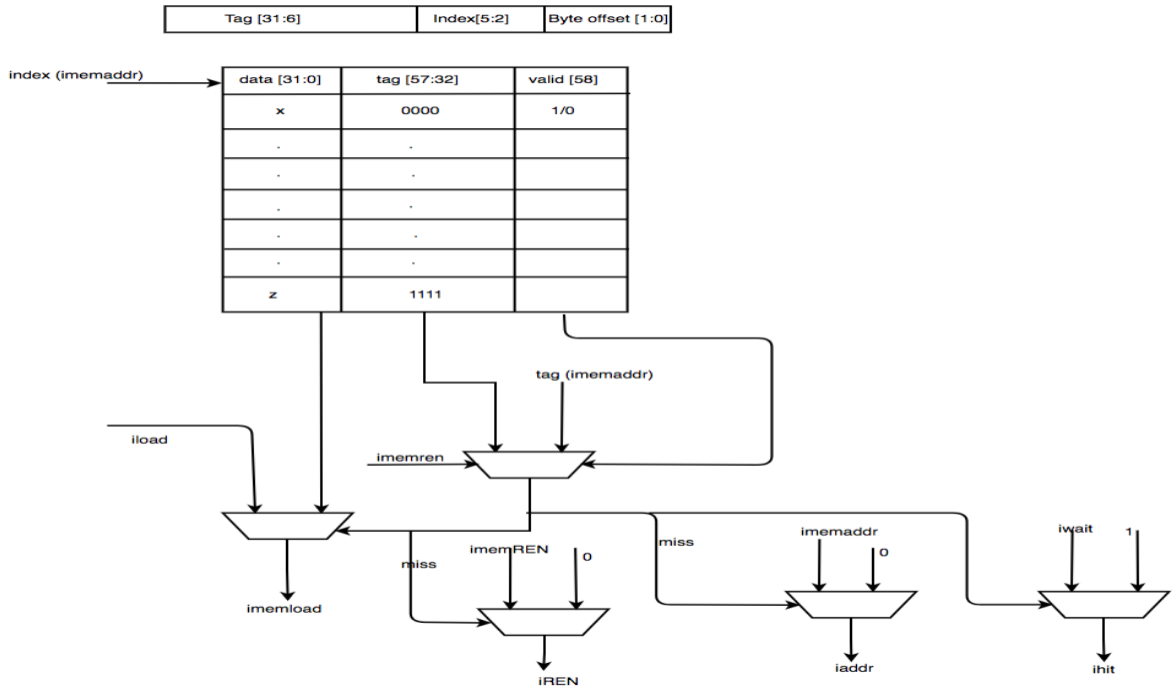

Figure 3: Icache Block Diagram

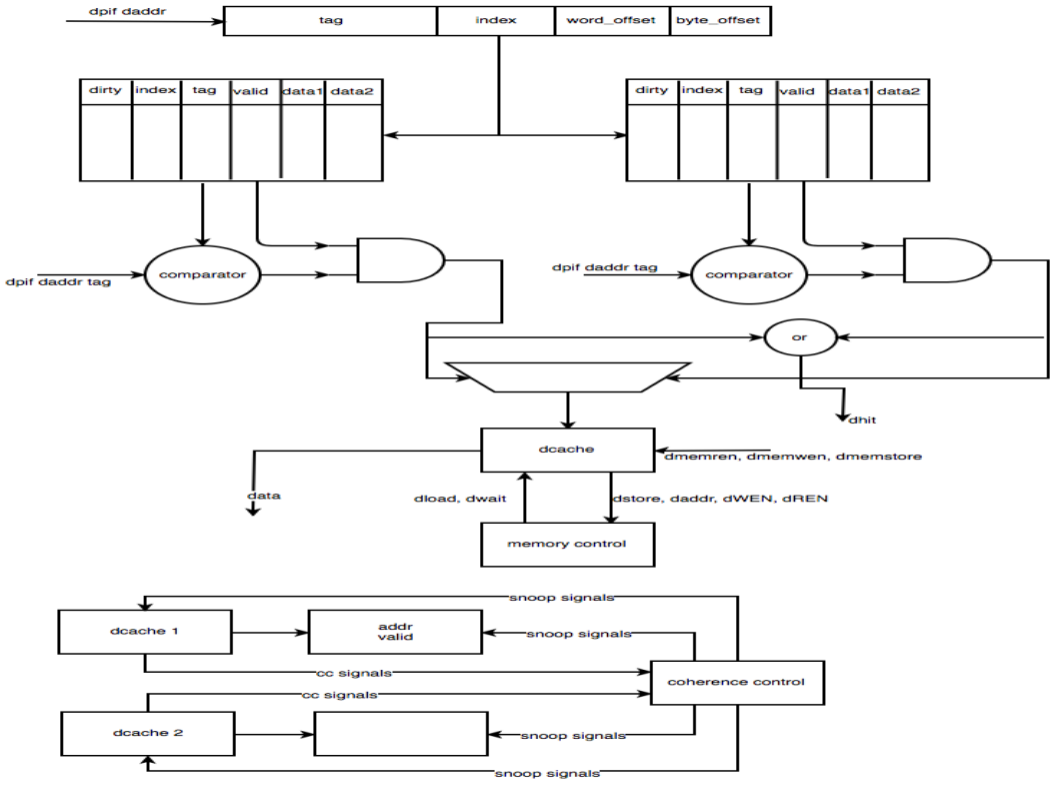

Figure 4: Dcache Block Diagram

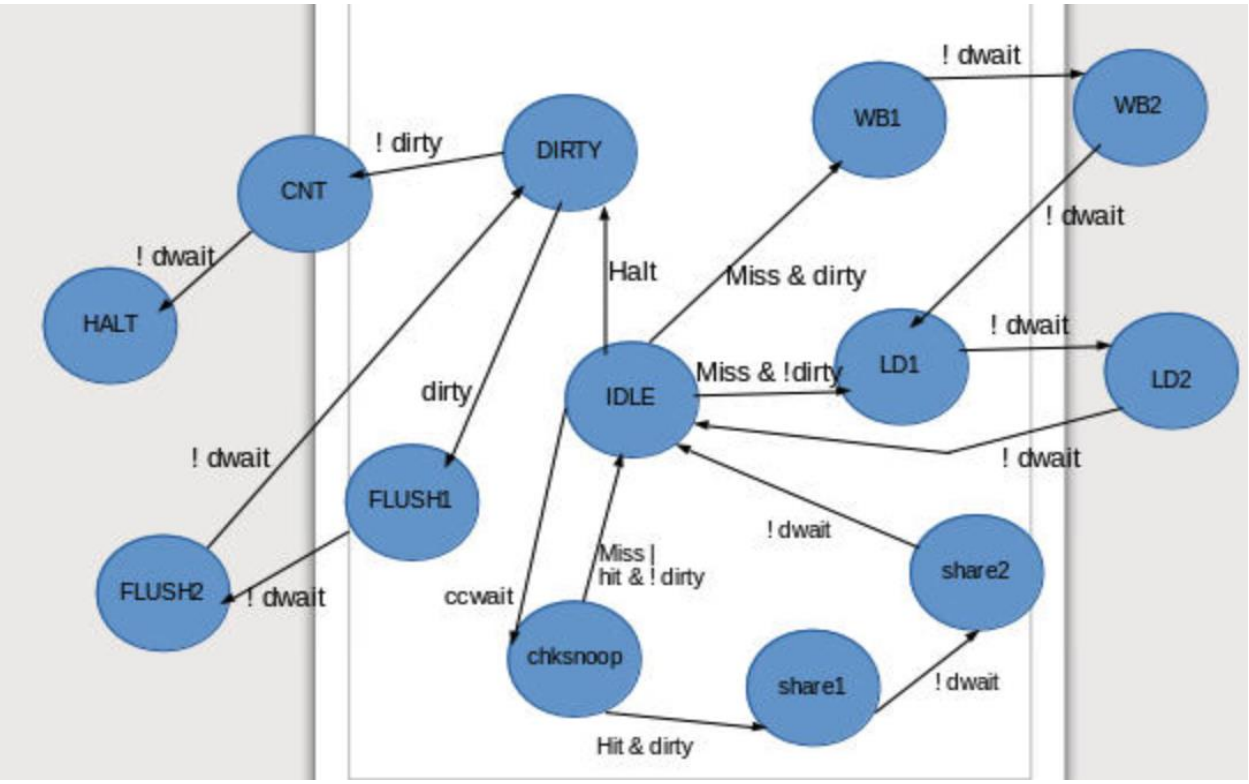

Figure 5: Dcache State Transition Diagram

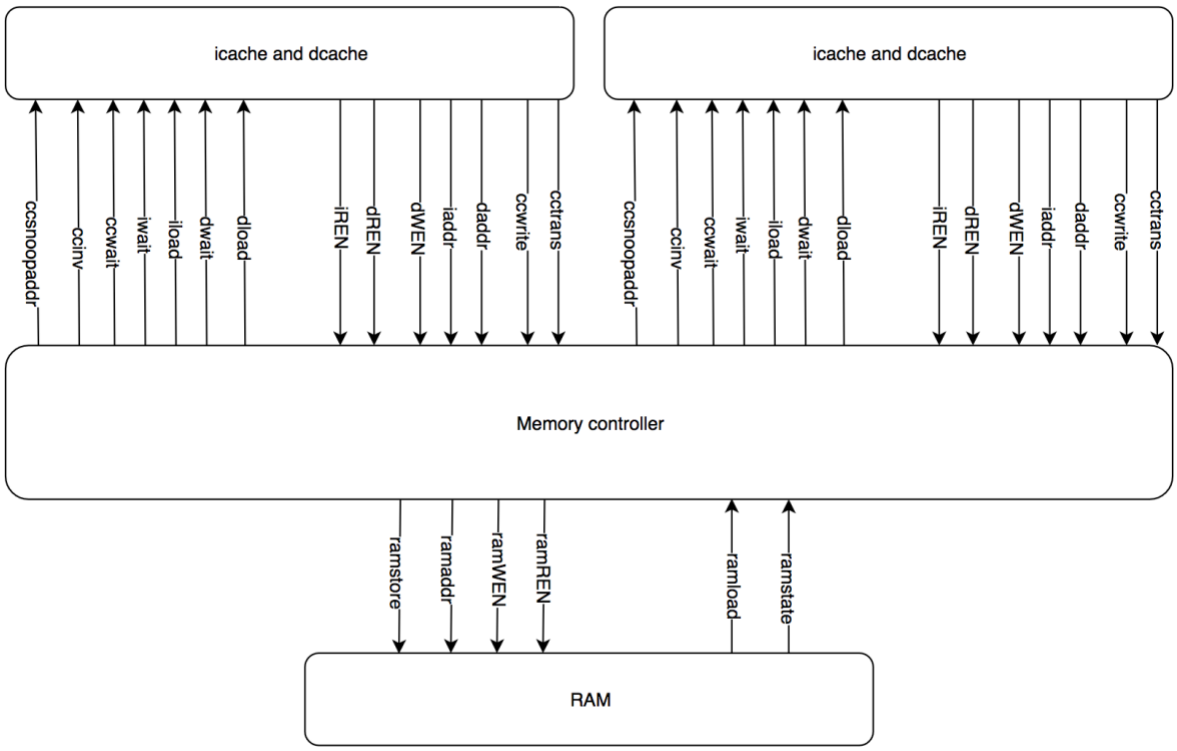

Figure 6: Coherency Block diagram

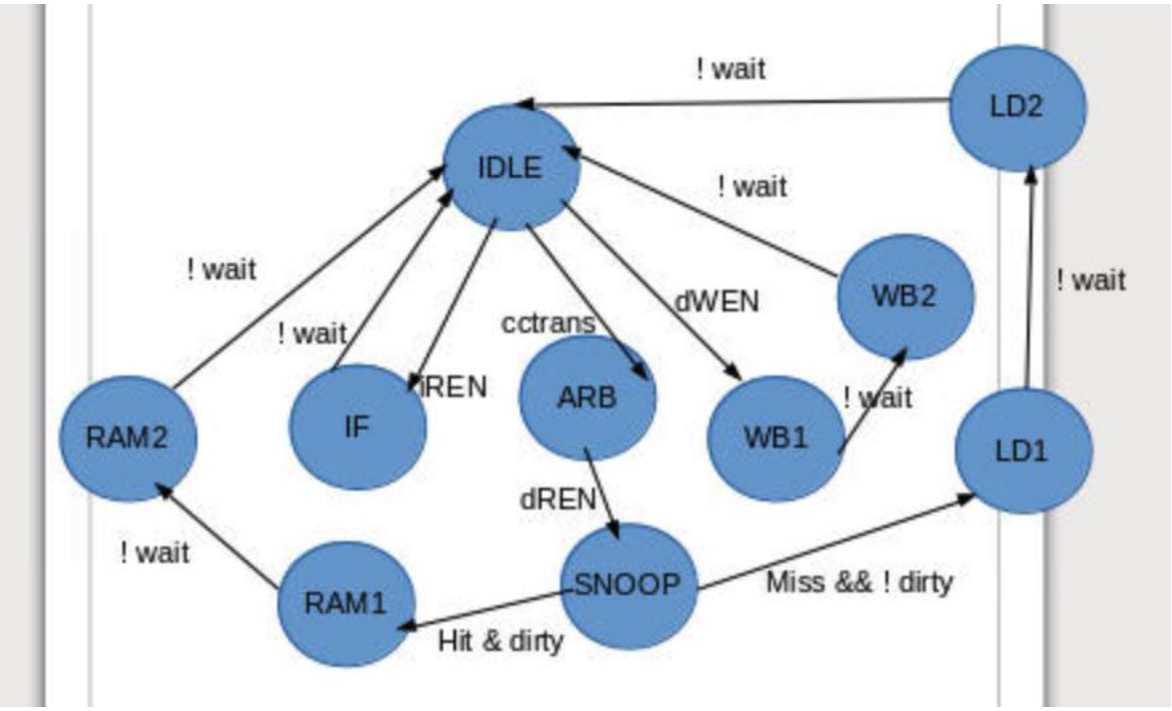

Figure 7: Coherency state transition diagram

Results:

	Pipeline	Caches	Multicore
Frequency (MHz)	43.09	37.82	32.94
IPC (instr/clk)	0.325	0.386	0.438
Latency (ns)	116.04	135.51	189.36
MIPS (minstr/sec)	15.12	18.67	21.03
FPGA (regs)	1735	5329	13852
Speedup	1	0.79	0.619

Conclusion

Our performance for merge sort slightly worsened even after improvements in our design, possibly due to our small sample size. Cache hits may be overshadowed by cache misses which would affect negatively the run time and the instructions per clock cycle. Even with two processors, the overheads involved with cache coherence will most probably outweigh the performance related to the processors because of the small sample size. As the number of instructions increases, the performance improvement should be more visible and relevant.

Our FPGA resources increased due to adding caches. From the results above we can see that the resources pretty much doubled from caches to our multi core design. This is something we had expected. This is because we are adding an additional processor and also there are additional wires, registers and logic blocks for the cache coherence protocol.

Finally, our design achieved what was expected from it, even though there is always scope for improvement. We can improve the critical path of our coherence controller and improve our overall pipeline design. We can also implement a MOSI protocol in order to reduce write backs to RAM. We could probably also improve the overall speedup for the program run on our dual core processor design.

Team Contributions

Karim Itani– Wrote Icache source code, worked on Dcache source code, worked on memory control source code. Drew coherency state transition and Dcache state transition diagram. Wrote the introduction and worked on the results section

Milind Shyam – Wrote Icache testbench, worked on Dcache source code, worked on memory control testbench. Drew caches diagram, coherency diagram and Multicore CPU diagram. Wrote the conclusion and worked on the results section