ECE 437L Final Report

By: Kyle Rakos and Kshitij Jain
TAs: Nitin Rathi and Alec Funke
April 28, 2017

Overview

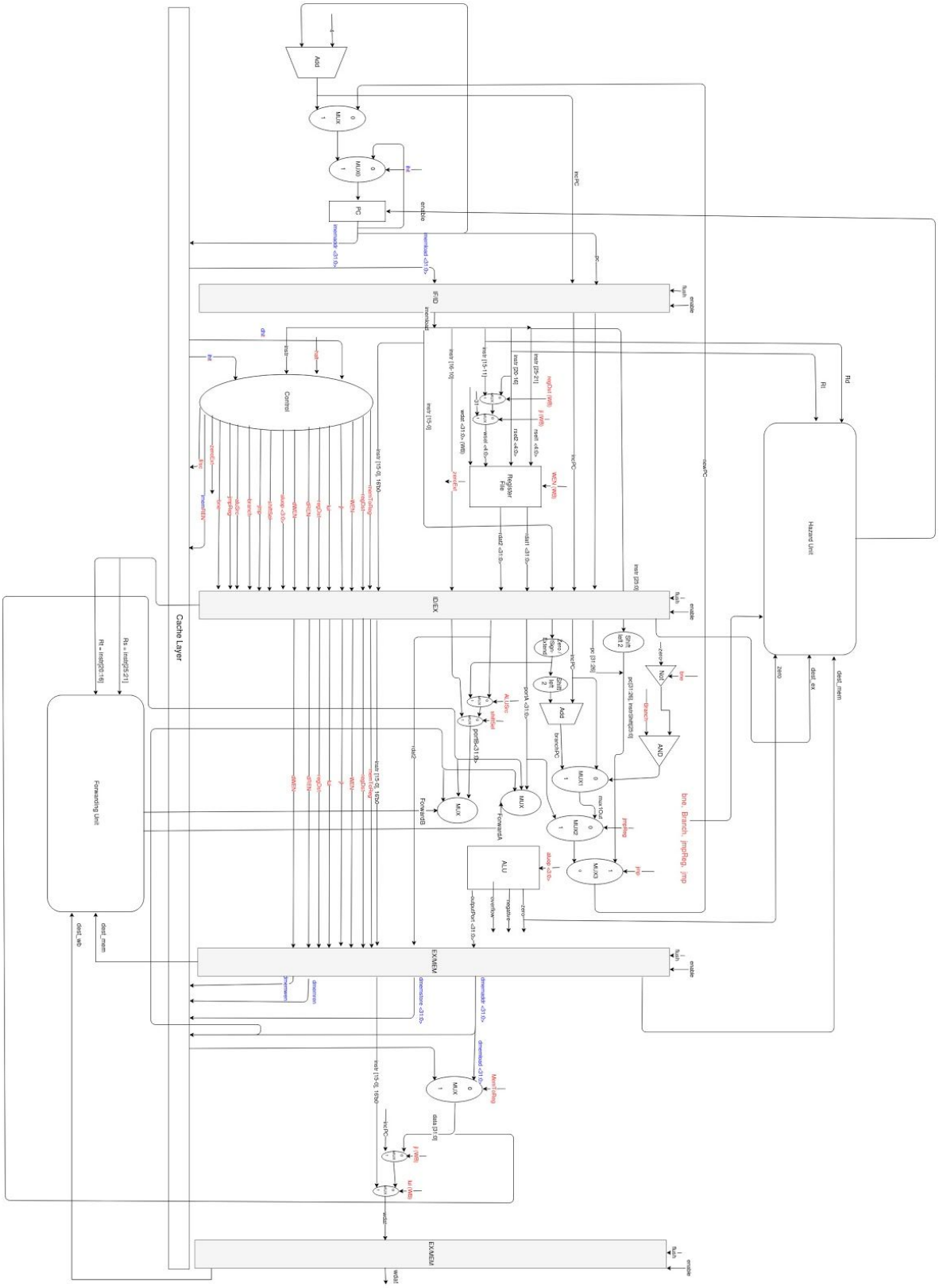
In this report we have compared the performances of three different kinds of processors - pipelined processor without caches, pipelined processor with caches and multi-core processor with caches. We have used the mergesort program as a standard to gauge the efficiency and speed of the three different processors listed above.

A pipelined processor, is faster than a single-cycle processor as it increases the throughput. A pipelined processor with caches is better than a pipelined processor without caches as it reduces the number of bus transactions with the memory and leverages the spatial and temporal locality of the data used by a program. A multi-core processor with caches is theoretically even better than a pipelined processor with caches as we have two processors working on the same program. However, in some programs the multicore overhead is greater than the parallel speedup.

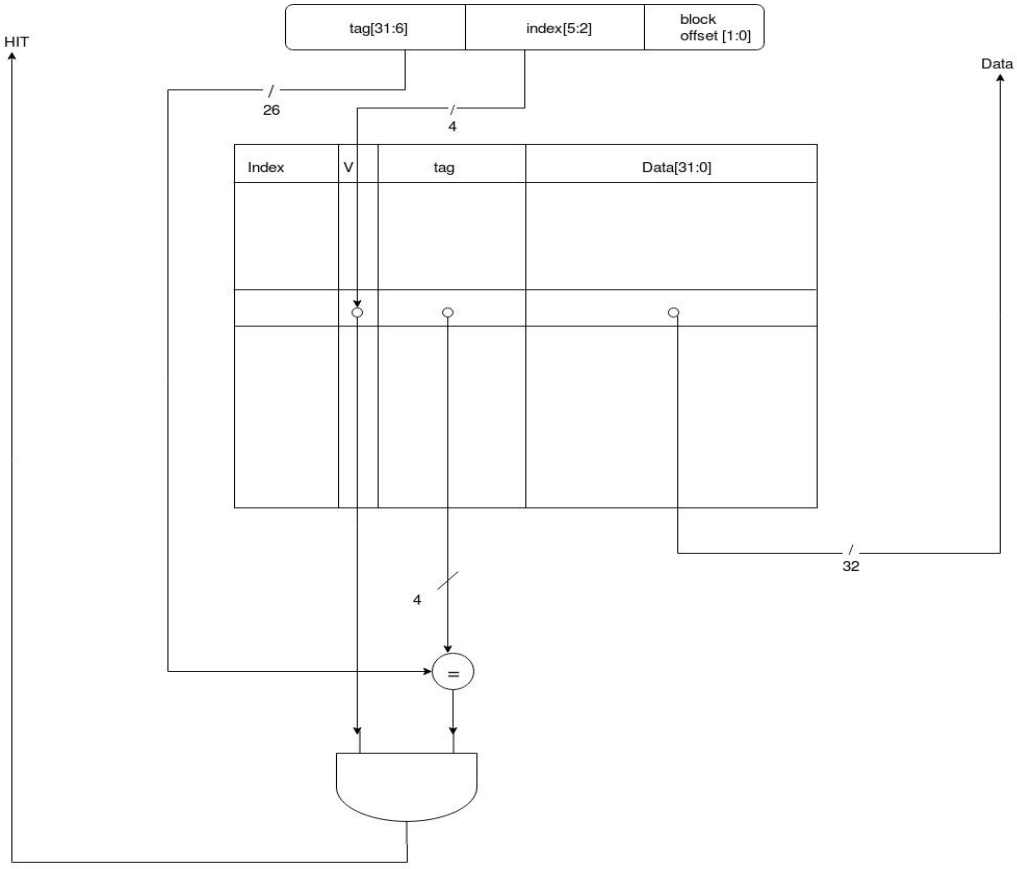
For comparing the three processors we will be analyzing multiple parameters but will mainly be concerned with three parameters - average instructions per clock cycle, latency of one instruction and total execution time. In the rest of the report we have included detailed diagrams for our i-cache, d-cache, coherency controller, pipelined processor and a comparison table for the performances of the above mentioned processors.

Processor Design

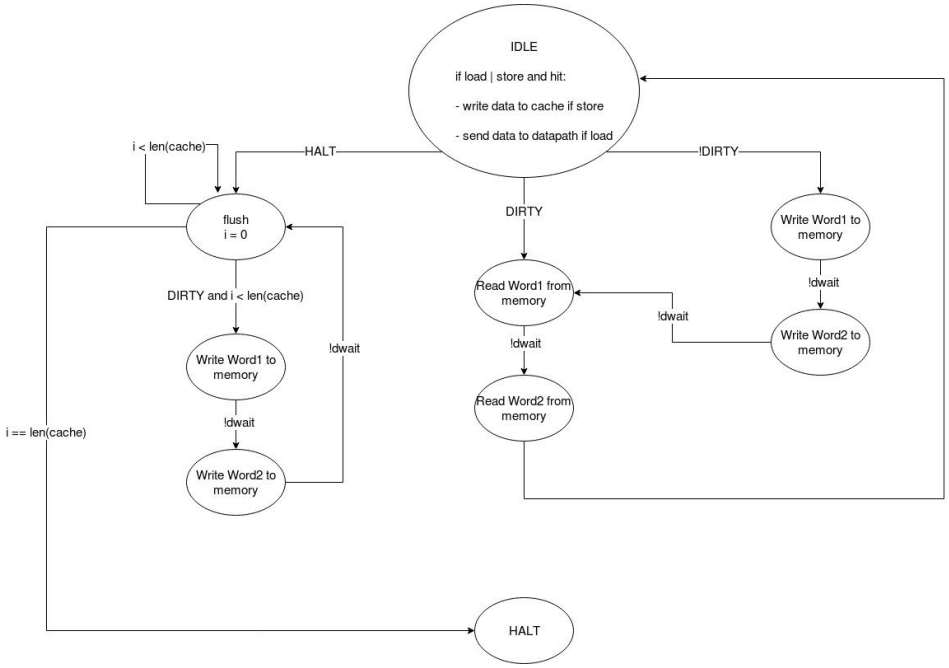
Multicore/Pipeline with Cache (only difference in pipeline is missing of ll/sc signal)



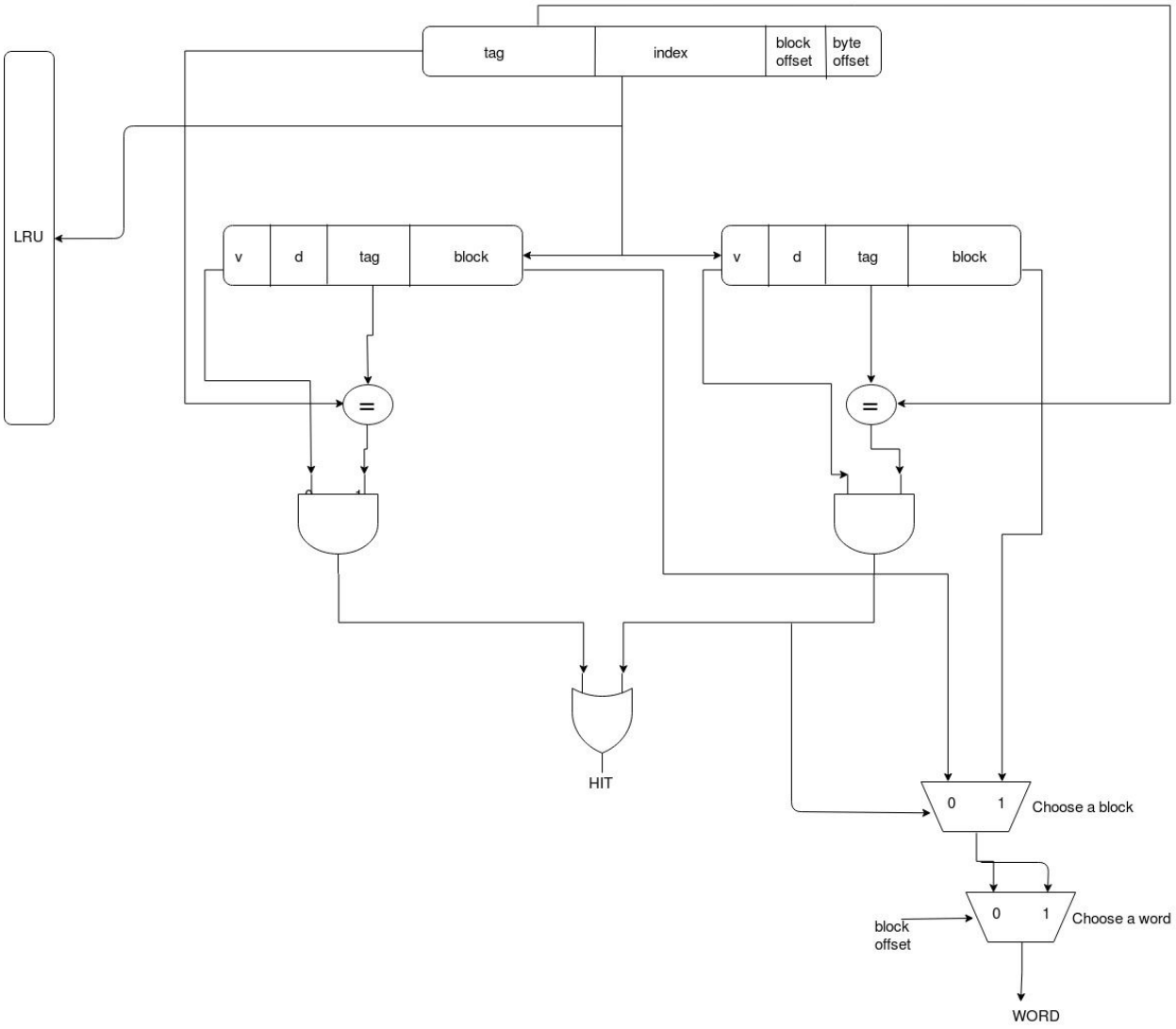
DCache



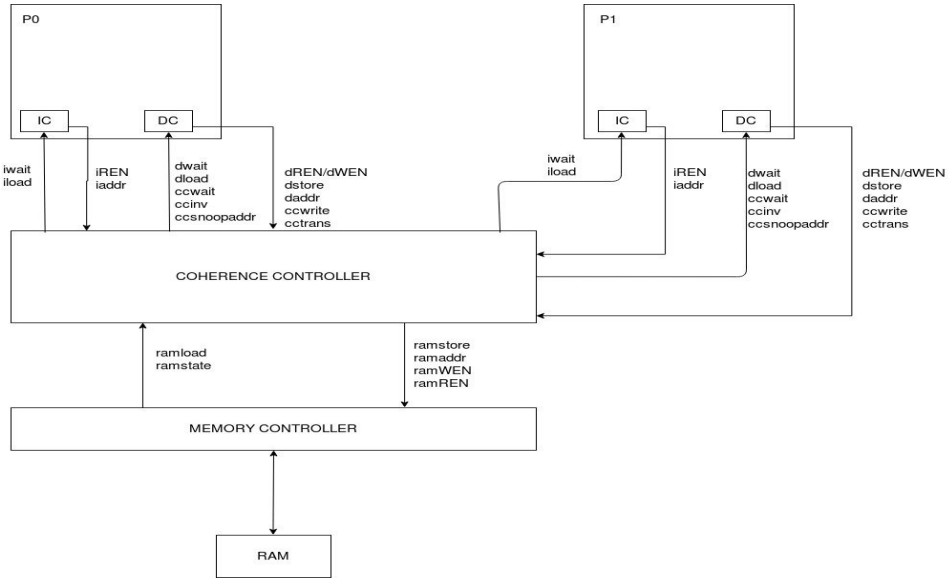
Dcache State Diagram



Icache


tag
index
block offset
byte offset
LRU
v
d
tag
block
v
d
tag
block
=
=
HIT
0
1
Choose a block
0
1
Choose a word
block offset
WORD

Coherence Controller Block Diagram



Coherence Controller State Machine

Results

Results for synthesizing the various processors and running the mergesort.asm program. Mergesort program size is 4 (due to issues at running at higher numbers in multicore). RAM latency is 7ns

	Pipeline (no Cache)	**Pipeline (Cache)**	**Multicore (cache)**
Max Frequency (logs)	45.22 MHz (CPUCLK value)	50.81 MHz (CPUCLK value)	50.93 MHz (CPUCLK value)
Max Frequency (testbenches)	50 MHz	50 MHz	50 MHz
Avg Instructions per CLK Cycle	0.060(107 instrs / 1791 cycles)	0.064 (107 instrs / 1,685 cycles)	0.042 (122 instrs / 2,909 cycles)
Latency of One Instruction	100ns (5*1/50MHz)	100ns (5*1/50MHz)	100ns (5*1/50MHz)
Total execution time	35.8ms (1791cycles * 1/50MHz)	33.7ms (1685 * 1/50 MHz)	58.18 ms (2,909 * 1/50 MHz)
FPGA Resources Required	Total Logic Elements: 4,021 / 114,480 (4%) Total Logic Registers: 1,822 / 117,053 (2%)	Total Logic Elements: 8,262 / 114,480 (7%) Total Logic Registers: 4,307 / 114,480 (4%)	Total Logic Elements: 17,476 / 114,480 (15%) Total Logic Registers: 8,471 / 114,480 (7%)

Cycles: Obtained from running system.sim command
Total instructions: Obtained from running sim
Sequential to Parallel speedup: Theoretically a parallel program could approach up to twice as fast as a sequential program. However, this assumes a just as fast processor cores and coherency not slowing down the processors.

Conclusion

As seen from the table, the performance of pipeline with caches is better than the performance of pipeline without caches, which is expected. The average instructions per clock cycle is 0.064 for pipeline with caches vs 0.060 for pipeline without caches. The latency of both processors is the same since caches do not affect latency. Finally, the execution time of pipeline with caches is

less than that of pipeline without caches due to the reduced number of bus transactions with the memory.

The multicore processor with caches was actually the slowest processor with only 0.042 instructions per cycle. The likely reason that this is the case is the overhead of a multicore processor due to coherency operations. If a longer running program could be tested it is expected the two cores would make up for the coherency operations and then the multicore would be quiccker.

Contributions

Kshitij did almost all of the caches and debugging. He also wrote the coherency controller for multicore and wrote the dcache testbench. Kyle helped with some cache debugging and wrote the dcache modifications and coherency controller test bench for multicore. Both partners worked together for debugging the multicore design.