Final Report

ECE437

Abdullah Khan, Neil Gupta

Nick Pfister

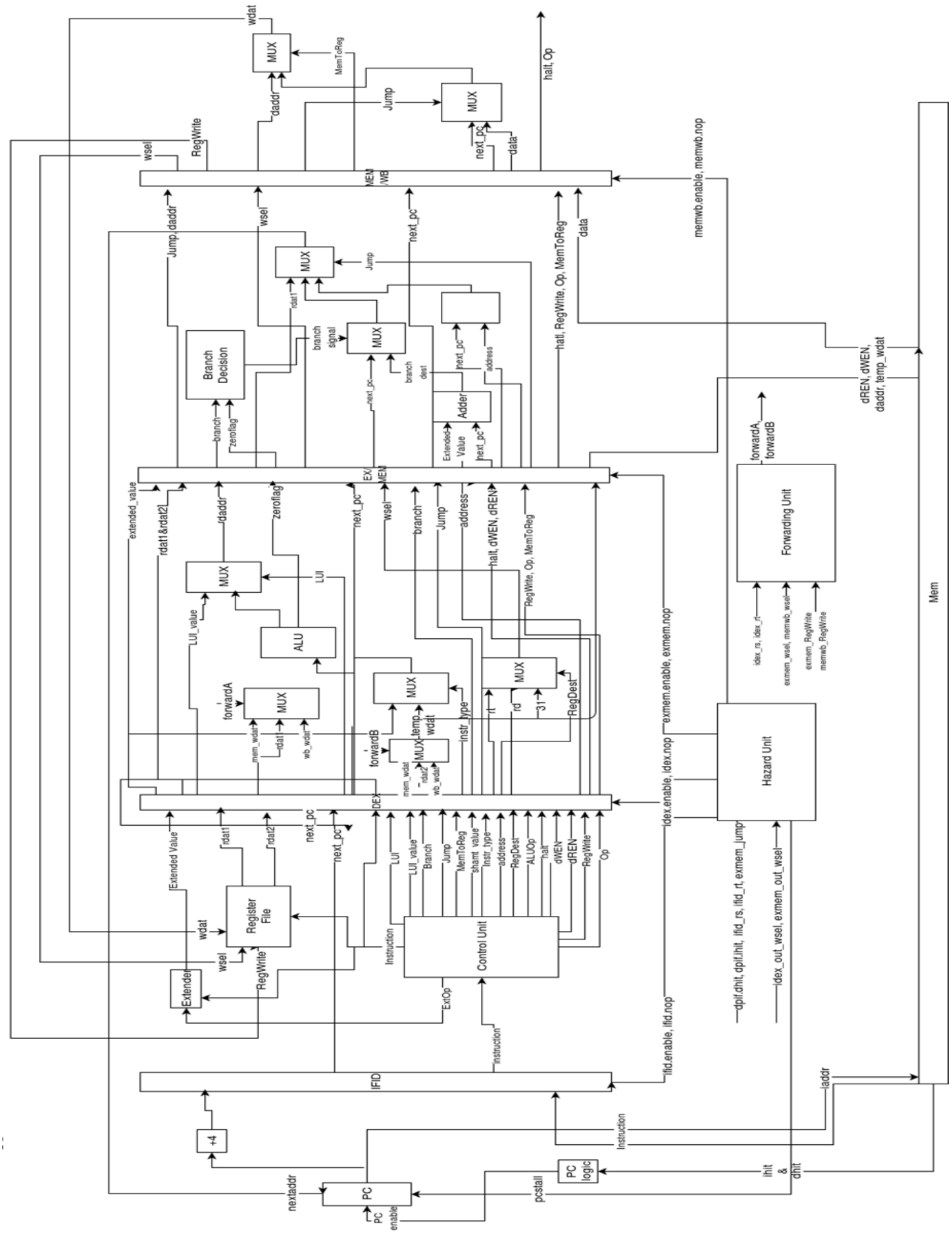
04/27/2017

Overview

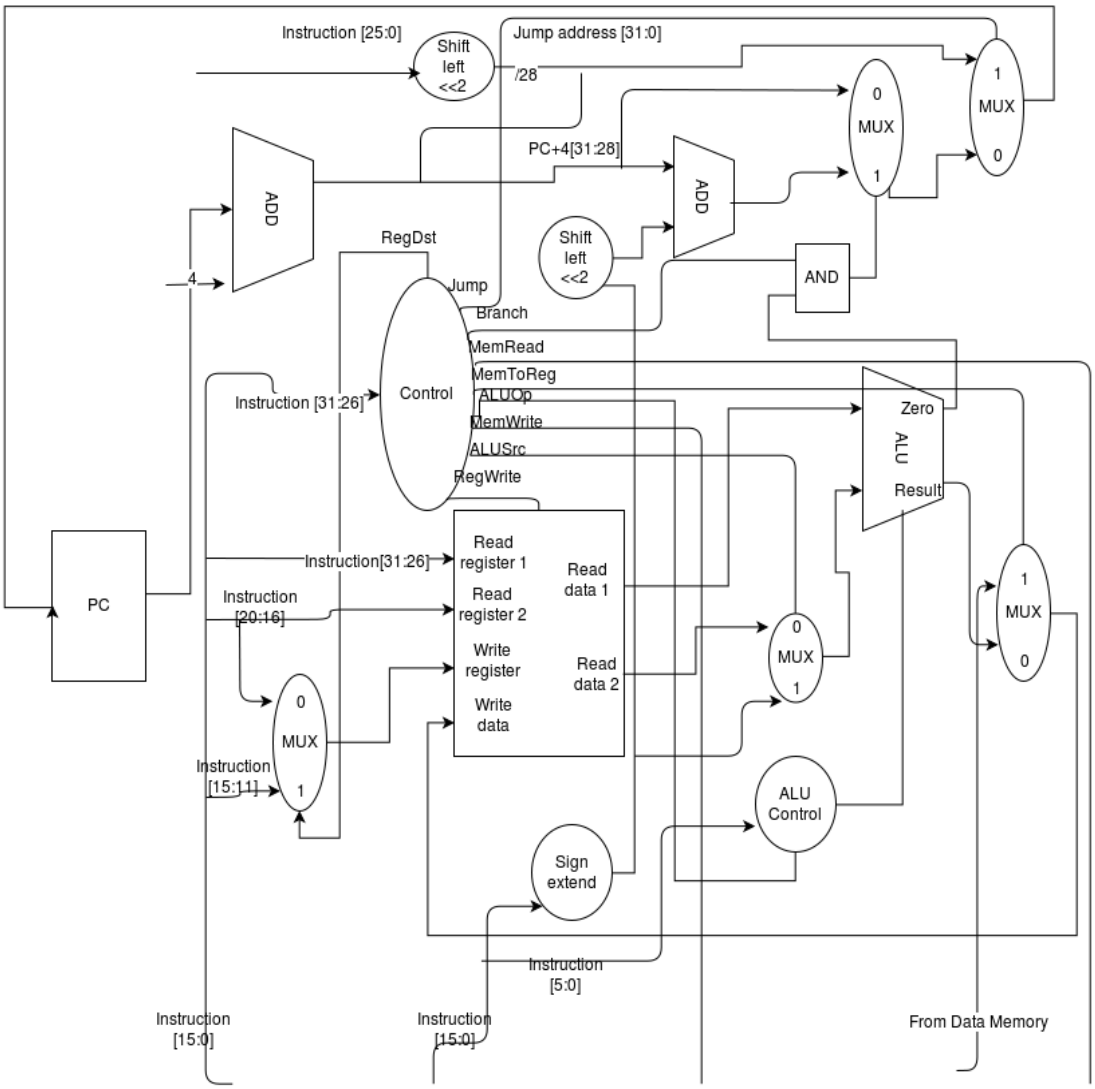
Over the course of this class we have designed, developed and implemented a Single cycle design, Pipeline design and a Multicore design. Some advantages of the Single cycle design are that it is simple to design and since there is less combinational logic it is easier to implement however compared to Pipeline design it lacks in efficiency and performance. This is because in Single cycle, every instruction regardless of how complex it is, will require the same amount of time. Due to this the processors clock is dependent on how slow the longest instruction takes. In comparison, with a pipeline processor you have a higher throughput because you can queue instructions and the critical path is lower because it is broken up into several stages. After we finished working on the pipelined processor we developed a cache interface. This was an important step in boosting the performance because constantly accessing memory slows down the system and with locality caches provide quicker access. Although it adds more logic blocks and requires more resources the pros definitely outweigh the cons. Moving on, after our pipelined processor with caches was implemented we started working on a multicore processor with shared memory. It makes sense to improve performance with two cores since improving the clock by a factor of 2 would increase the power usage by a factor of 8, but with two cores running at the same clock would only increase the power usage by a factor of 2. So with 2 cores we were able to deal with twice as many instructions as before which boosts performance and to make sure there are no synchronization errors between cores in the memory we had to develop a coherency controller.

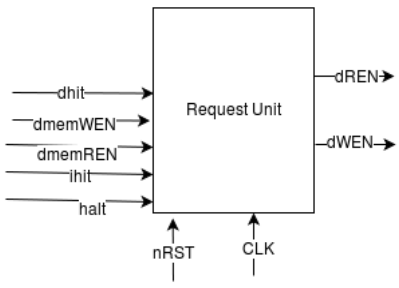
Finally, to benchmark the performance of the two processors we have selected to run dual.mergesort.asm as it will cover most metrics a processor should meet. There are a total of 5399 instructions and it tests plenty of I type, J type and R type instructions. Furthermore, dual.mergesort.asm introduces many hazards, some which have to be stalled and some which require forwarding, so it properly tests the functionality for the hazard and forwarding unit in the pipelined processors as well as cache coherency.

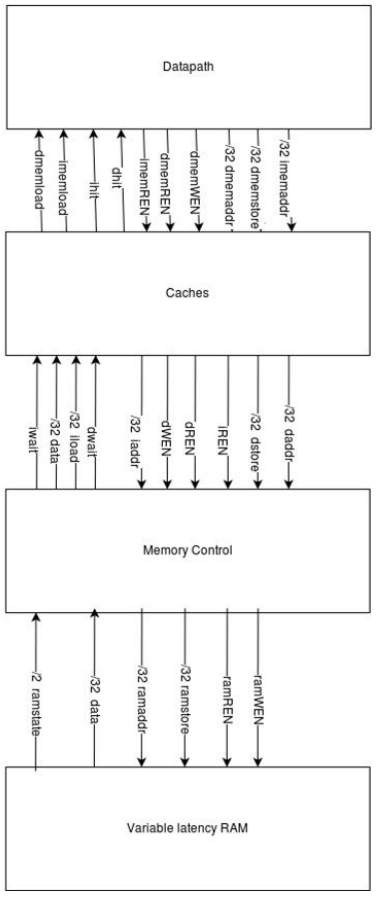
Processor Design – Pipeline (DP0 and DP1)



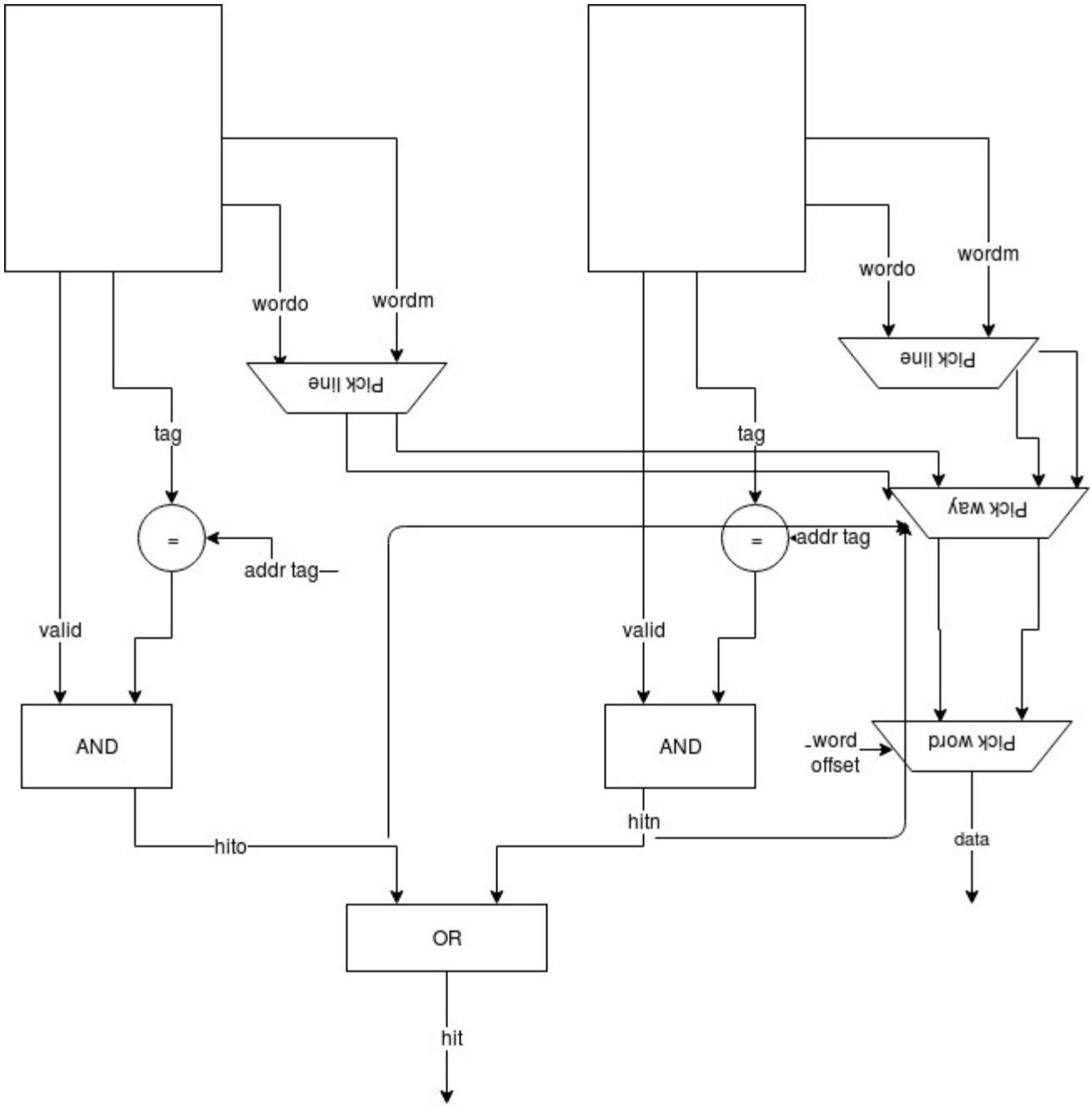
Processor Design – Single Cycle



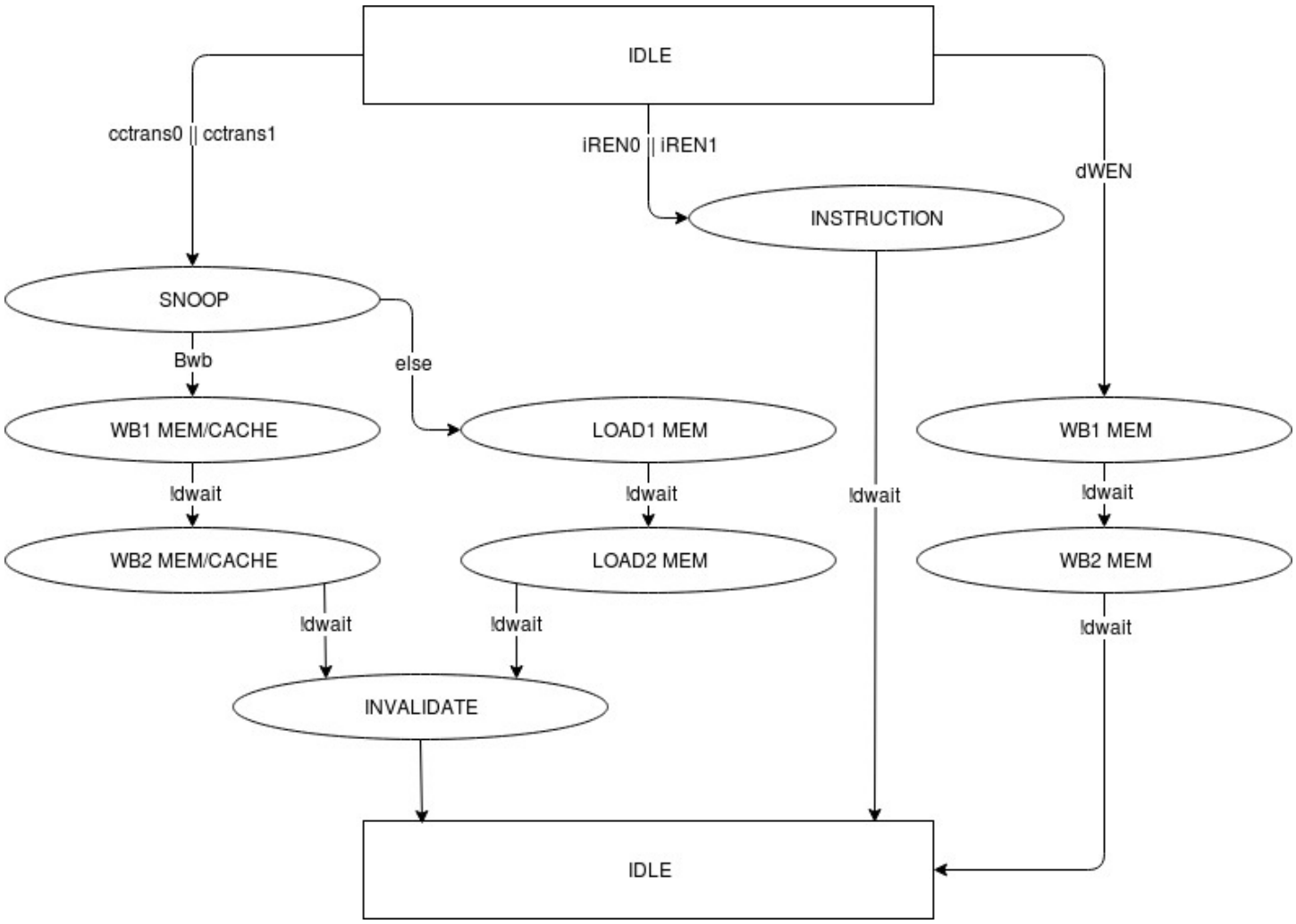




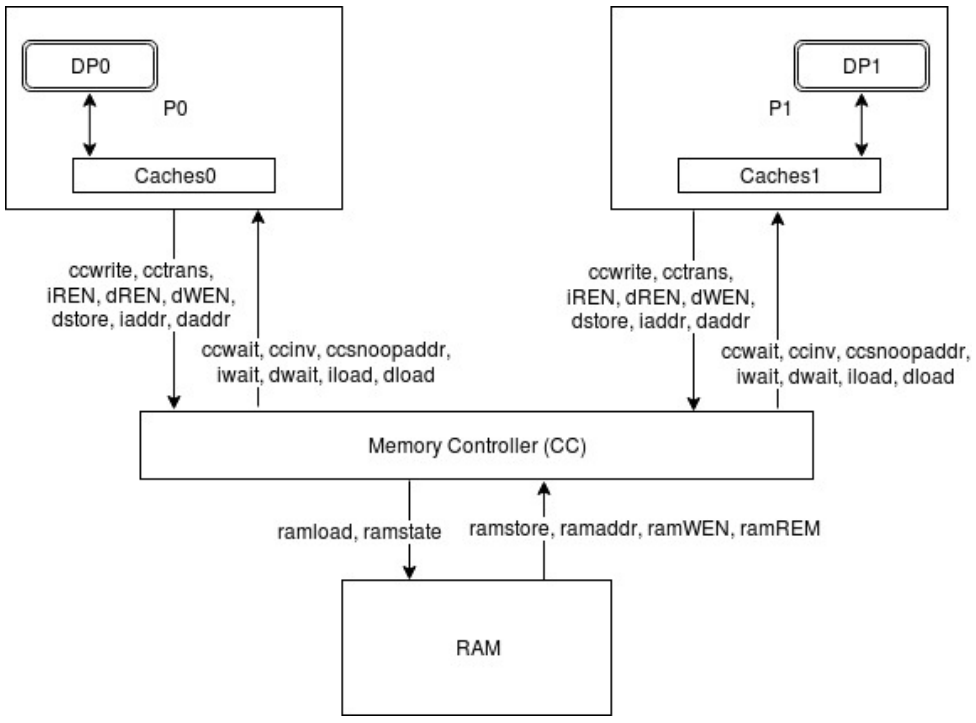
Dcache Expanded



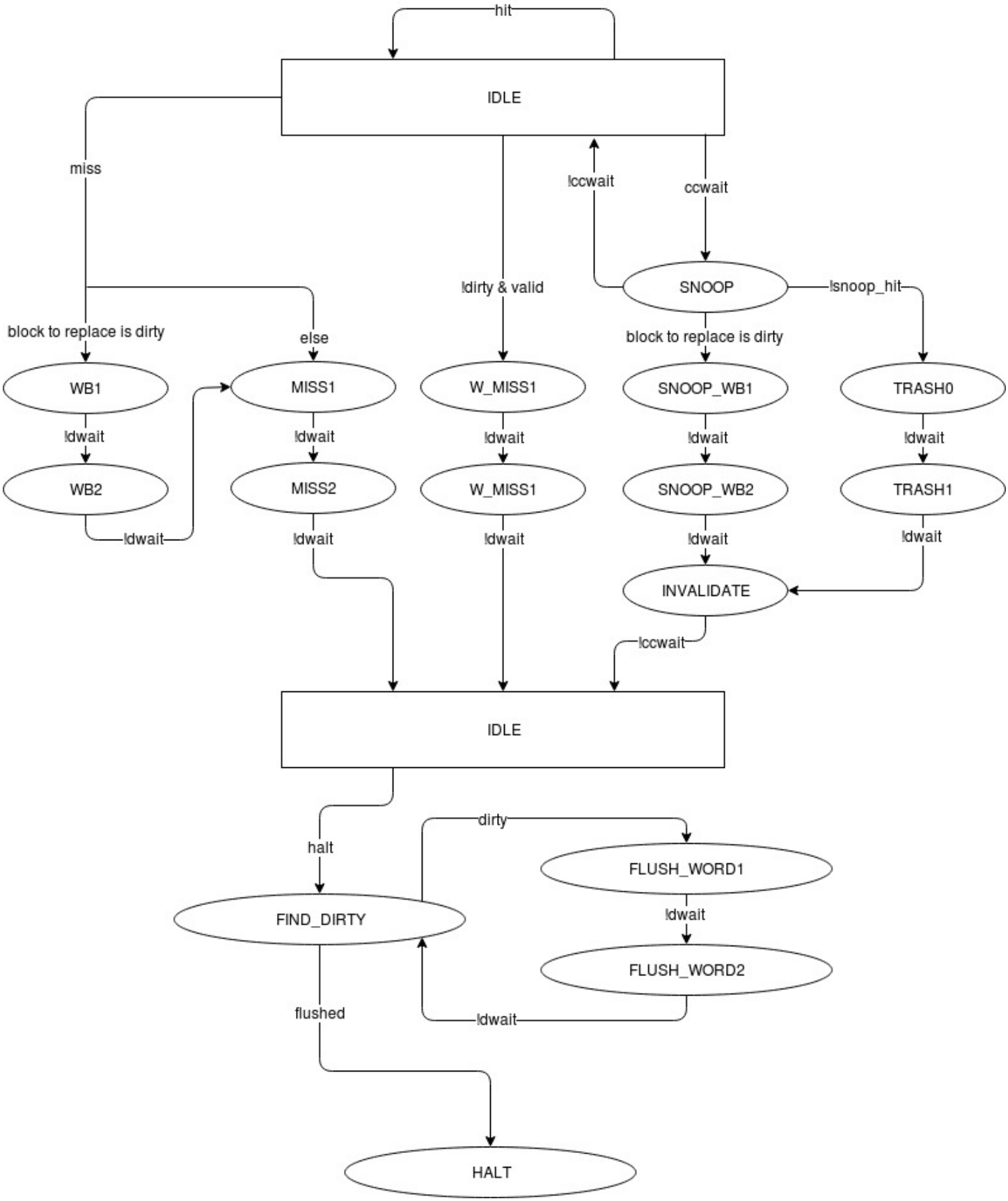
Dcache State Diagram



Multicore Top Level Diagram



Coherence State Diagram



Results

	Single Cycle	Pipeline	Pipeline w/Caches	Multicore
CLK Frequency	29.55 MHz	54.95 MHz	49.13 MHz	24.48 MHz
Length of critical path	33.84 ns	18.19 ns	20.35 ns	40.85 ns
Latency	33.84 ns	90.95 ns	101.75 ns	204.25 ns
Execution time	13791 cycles * 33.84 ns	17587 cycles * 33.84 ns	14625 cycles * 20.35 ns	367 cycles * 40.85 ns
Maximum Achieved Frequency	1/10 GHz	1/2 GHz	N/A	N/A
Breaking parameter period	10 ns	2 ns	N/A	N/A
Total Combinational functions	3,107	3,224	6,379	13,227
Total Registers	1279	1718	4,172	8,043
Average Instructions/clock cycle	5399 instructions / 33.84 ns = 159.54 x 10^9 instructions/cycle	5399 instructions / 18.19 ns = 296.81 x 10^9 instructions/cycle	5399 instructions / 20.35 ns = 265.31 * 10 ^ 9 instructions/cycle	5414 / 40.85 ns = 132.53 * 10 ^ 9 instructions/cycle

Speed up calculations

dual_mergesort.asm: 367 cycles * 40.85 ns -----> new execution time (MULTICORE)

mergesort.asm: 14625 cycles * 33.84 ns ---------> old execution time (PIPELINE W/CACHES)

speedup = 494910 / 14991.95 = 33.012

Conclusion

In conclusion, analyzing the results we can see that the CLK frequency for the Multicore processor got lower as compared to pipeline with caches. The reason for this might be because the overheads involved with cache coherence might have worsened performance with the dual.mergesort.asm. Also, with dual.mergersort.asm there might have been many cache misses so that in turn affected the CLK frequency. The results might get better with more instructions when there are less cache misses.

Moreover, the multicore processor requires significantly more FPGA resources as a lot more logic was added to implement cache coherence. We can see that the Multicore processor requires 8,043 registers and 13,327 combinational blocks as compared to 4,172 registers and 6,379 combinational blocks from before.

Overall we can see that the Multicore processor is able to handle a lot more instructions per cycle but to moving forward to boost performance we could implement the MOESI protocol which further improves the MSI protocol currently implemented.

Contributions

Abdullah Khan

- Prepared Block Diagram
- Designed and wrote pipelines
- Wrote datapath
- Debugged datapath
- Wrote code in forwarding unit
- Contributed to logic in writing code hazard unit
- Debugged forwarding unit
- Debugged hazard unit
- Wrote the coherence controller
- Wrote the caches interface
- Wrote testbenches
- Debugged entire project
- Worked on report

- **Neil Gupta**
- Prepared block diagrams
- Contributed to design and tinkering of final block diagram used
- Contributed to debugging pipeline code written
- Wrote hazard unit
- Contributed to logic and writing code in forwarding unit
- Contributed to logic in writing code hazard unit
- Debugged hazard unit
- Debugged forwarding unit
- Wrote cache interface
- Debugged entire project
- Worked on the report