Final Report

ECE 437 Processor Prototyping Laboratory

By: Jhen-Ruei Chen, Yue Yu

Section 3

TA: Abhishek Bhaumick

10/17/2021

Design Overview

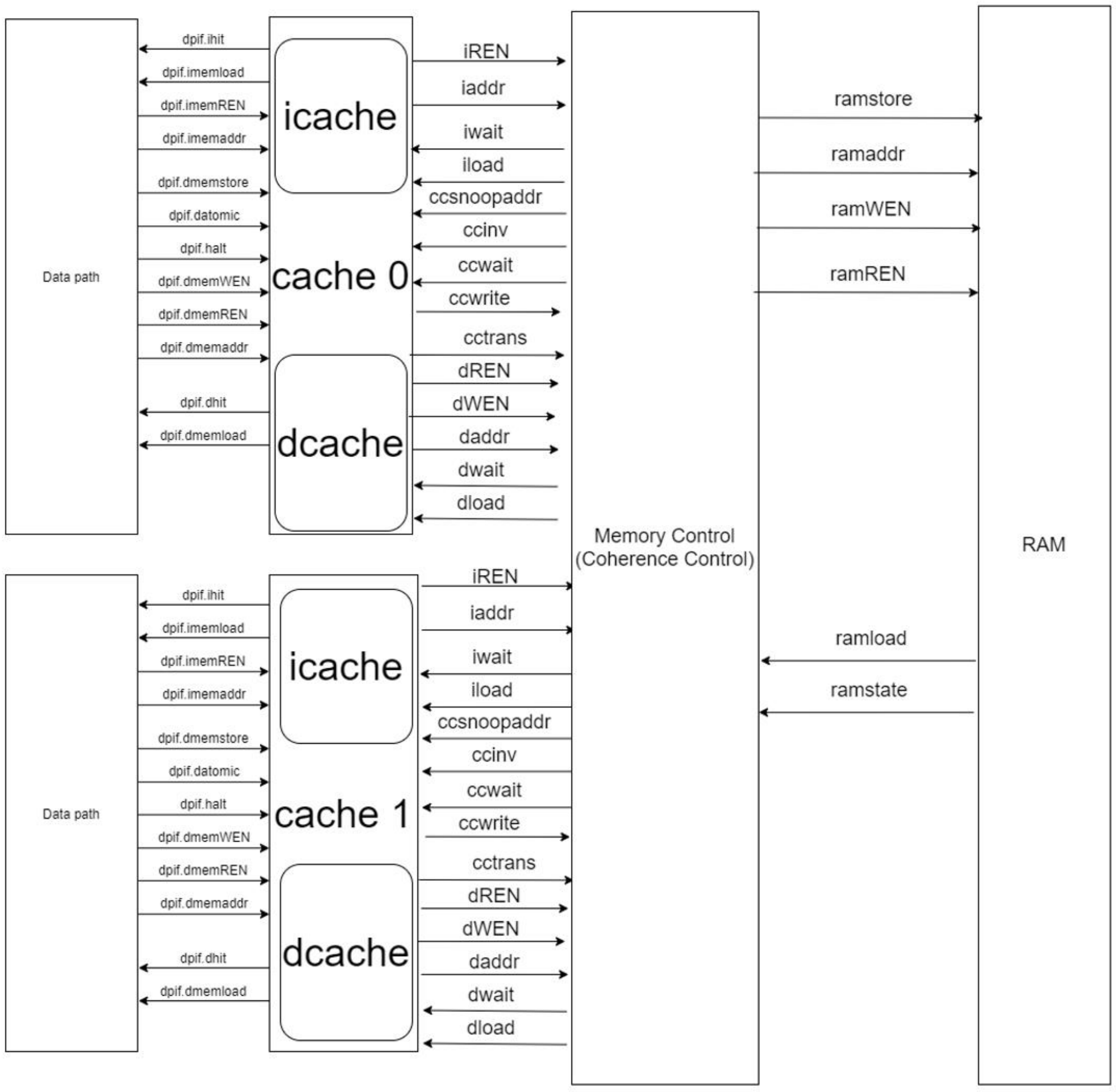
Throughout this report, we analyzed and compared the performance of our pipelined processor with and without a cache hierarchy and the pipelined processor with cache and a multi-core processor. We will run both processors on MIPS instructions.

The pipelined processor with cache hierarchy is significantly faster than without a cache hierarchy because of the reduction of Read & Write time. We implemented the concept of locality where we put the most recently used instructions and data in the cache to reduce search time because the size of the cache block is significantly smaller than memory. Our design used LRU(least recently used) replacement policy, "allocate on miss" allocation policy, and 'write-back" write policy. We ran mergesort.asm as a benchmark to compare the performances.

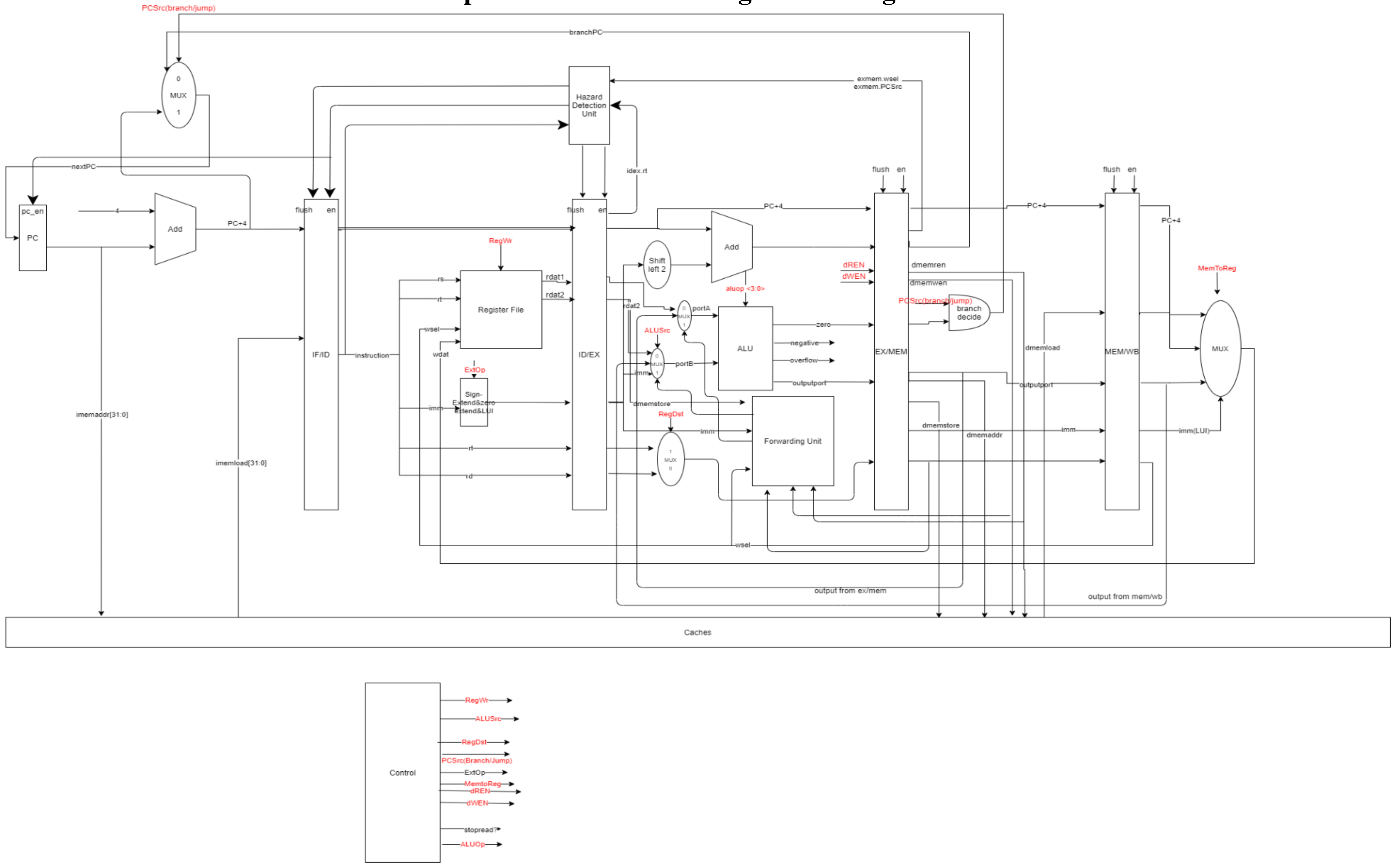
The multicore processor design is theoretically faster because it can complete twice as many instructions in the same amount of time. We implemented the MSI protocol to maintain cache coherence and we applied parallel algorithm to maintain data coherence between two cores. However, on the last stage of our final lab, we realized our MSI protocol had structural issues where our design couldn't cover some of the state transitions which makes our multicore design impossible to run palgorithm.asm and dual.mergesort.asm. Therefore, we ran test.coherence2 as our benchmark standard test.

In order to compare the performance between both processors, we will check on the maximum clock frequency, the average instructions per clock cycle, the latency of one instruction, the performance of the designs in MIPS, and the FPGA resources.

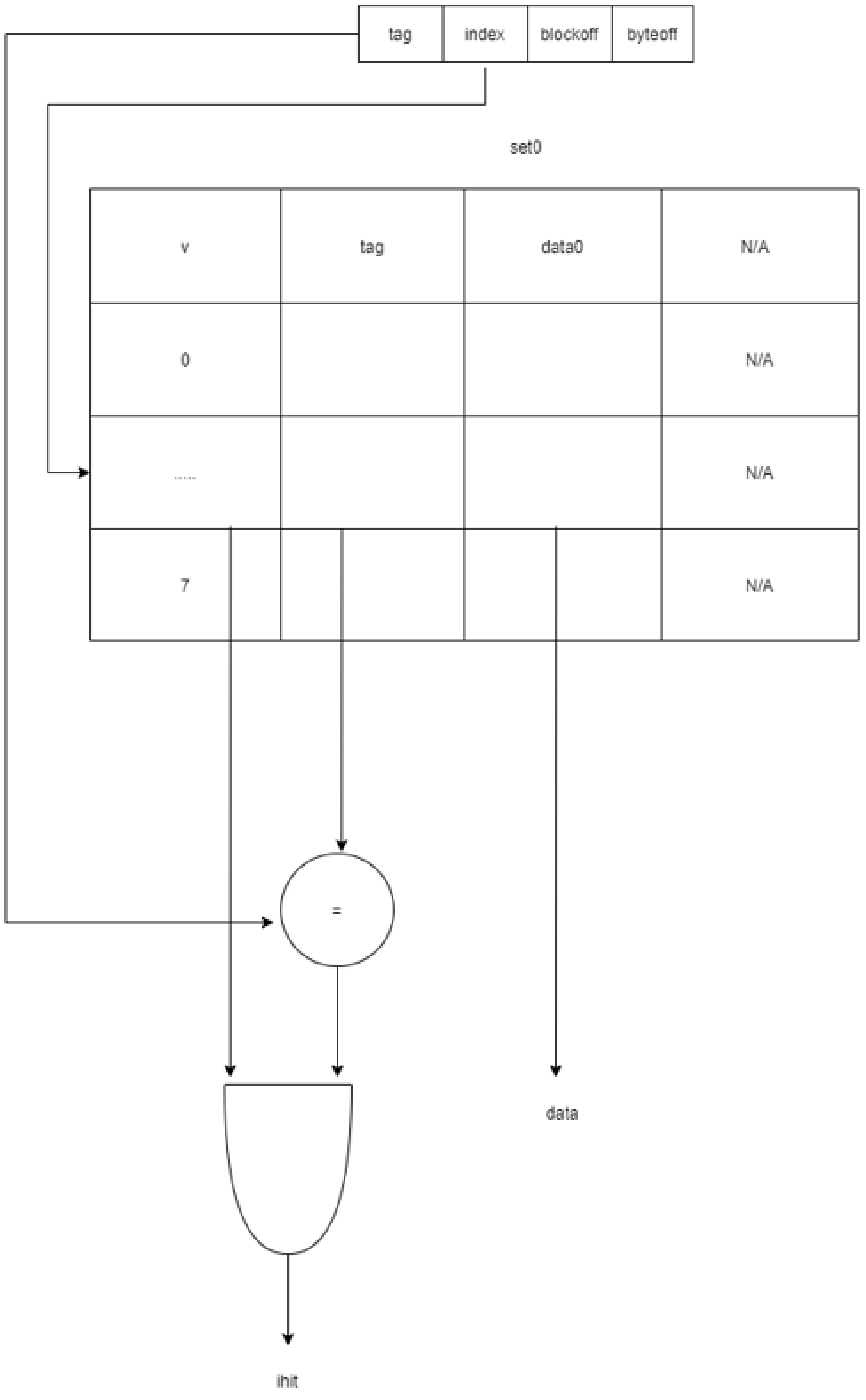
Multicore Processor Overall Design Diagram



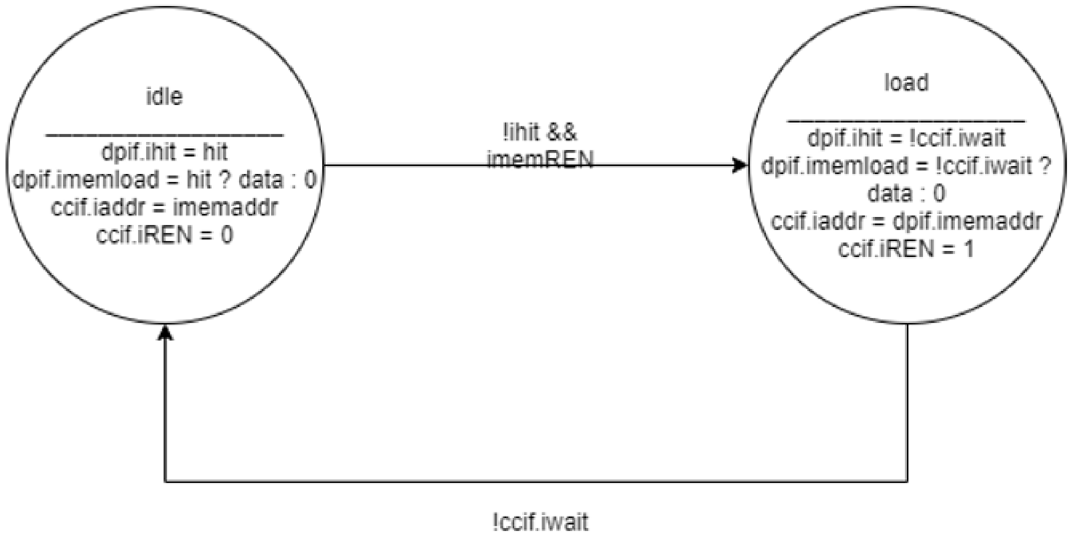
Pipelined Processor Design RTL Diagram



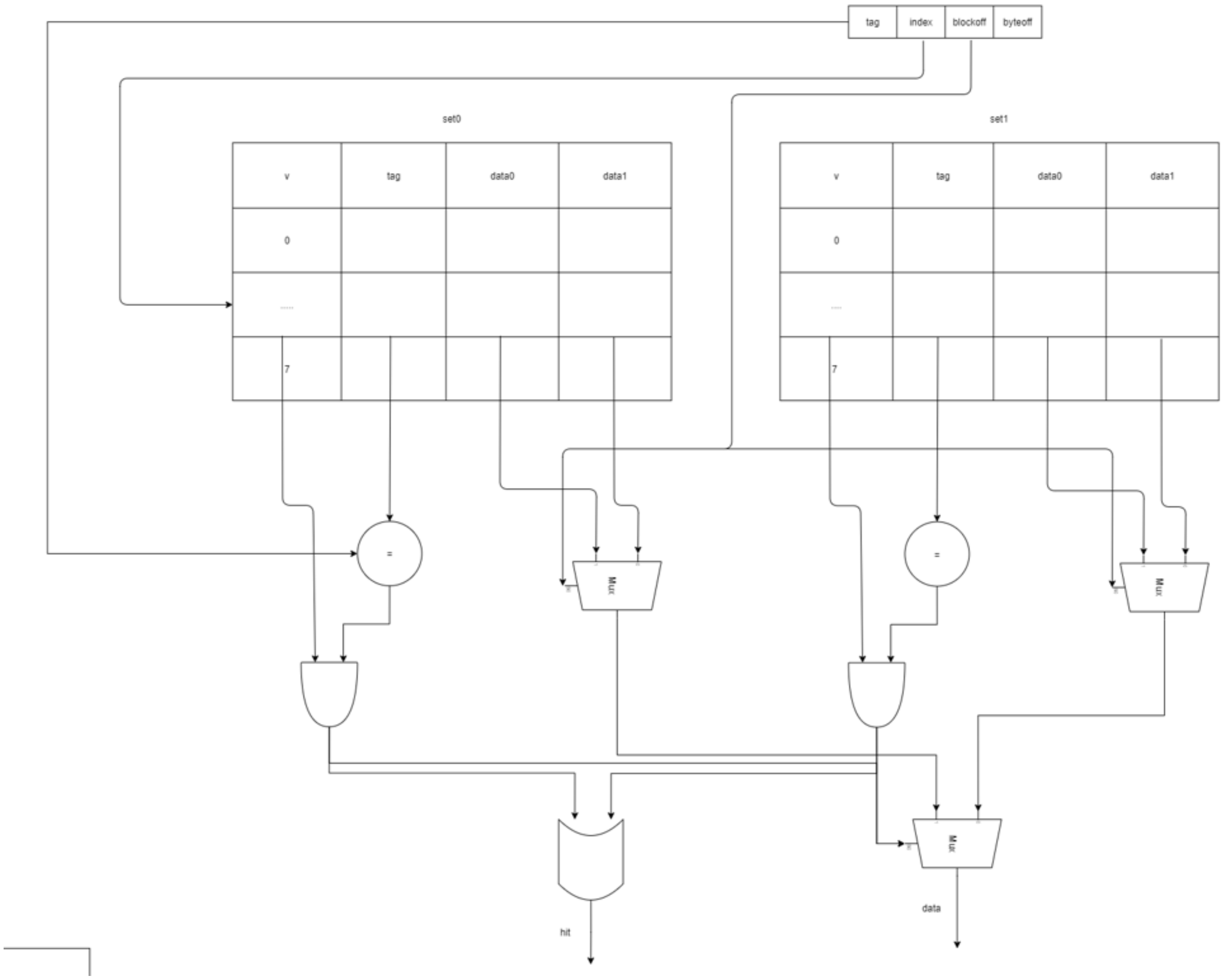
I-Cache Design



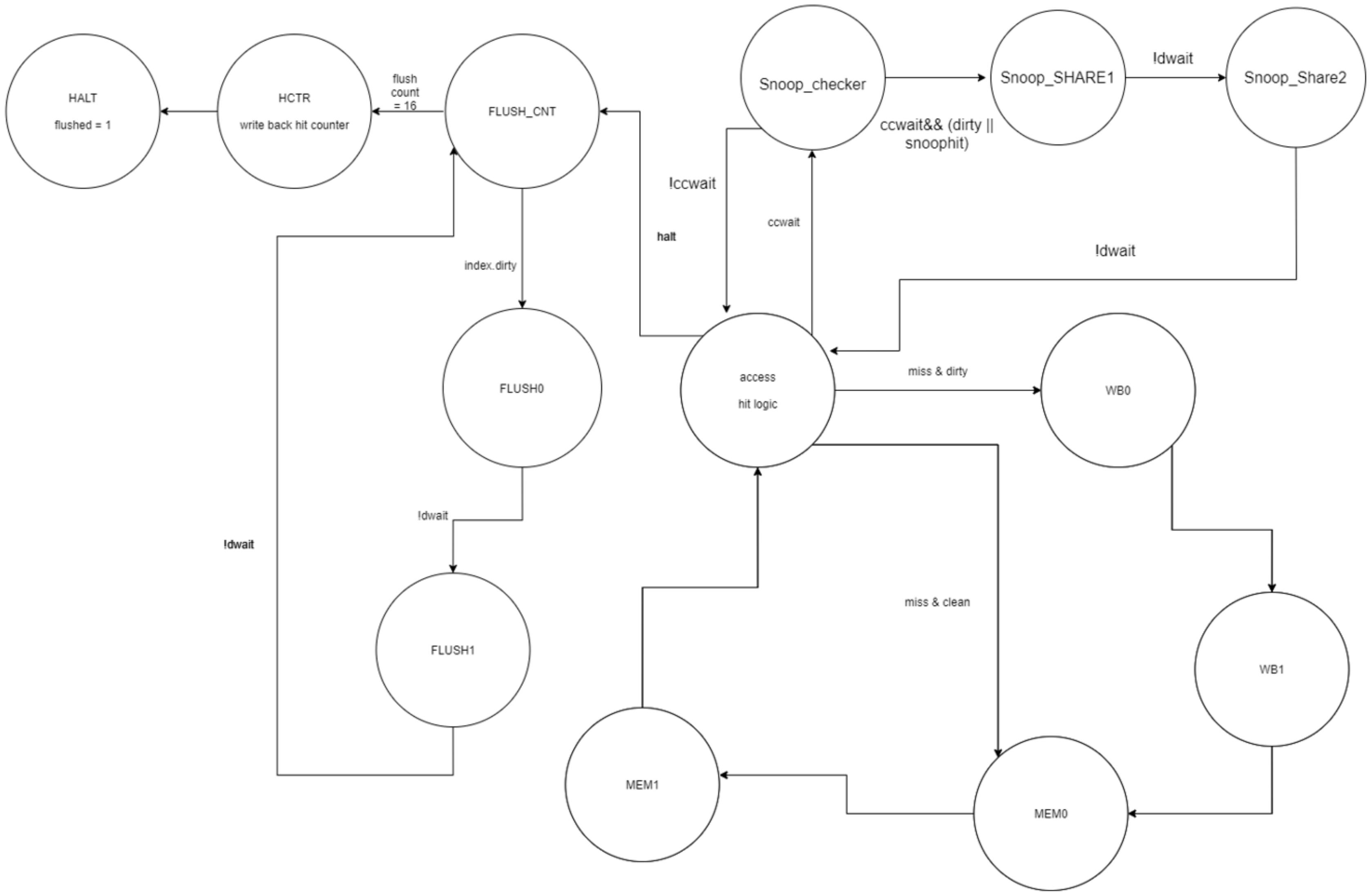
I-Cache state transition diagram



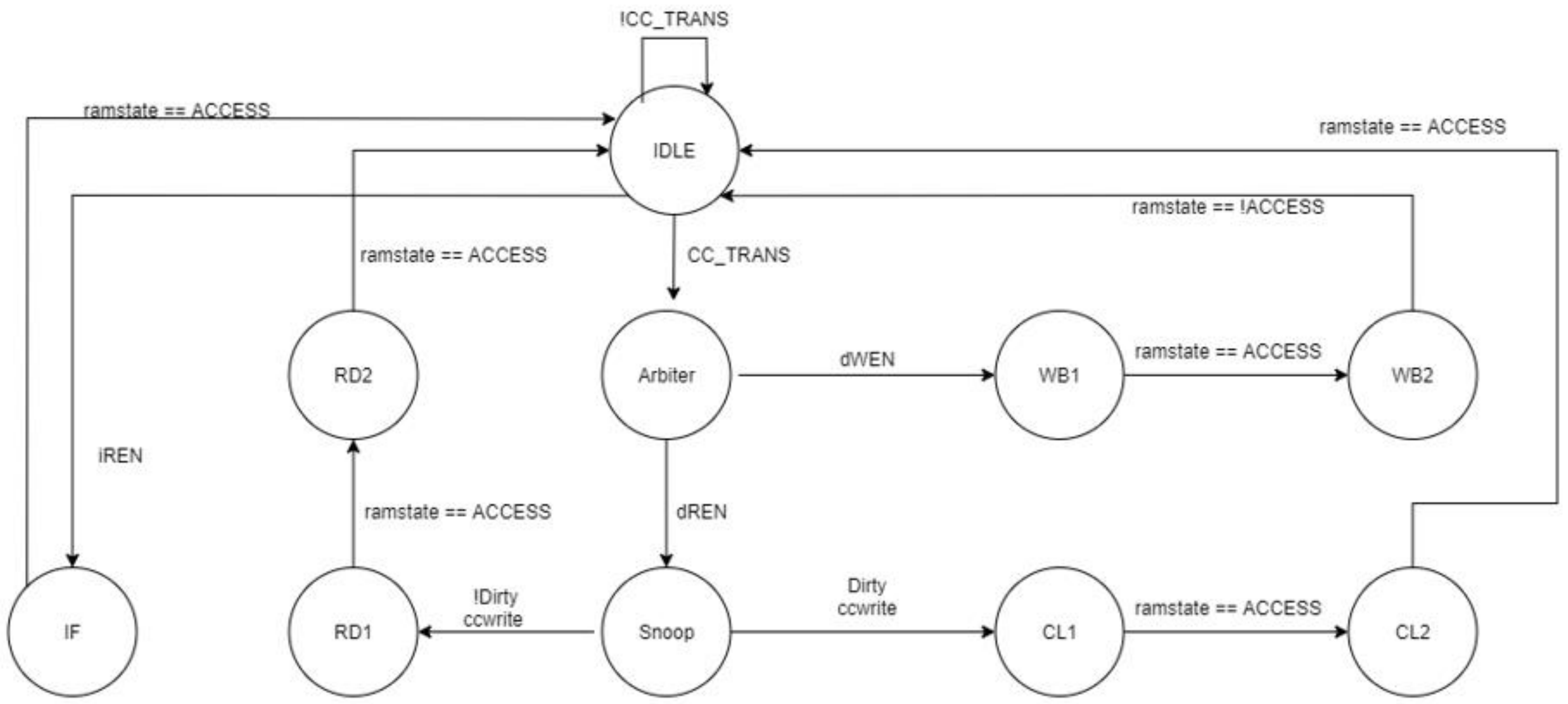
D-cache design



D-Cache state transition diagram with coherence control modification



Bus controller state transition diagram



Results

Throughout this lab, we ran the mapped version by applying the command “synthesize -t -f 200 system” to enable timing and to set an effort of 200 MHz for Quartus. All data collected was running under latency = 0.

Formulas to Calculate different parameters

1. **Max Frequency:** From “system.log” and Fmax = minimum of CLK/2 and CPUCLK
2. **Number of instructions:** From running “sim -t”
3. **Number of cycles:** From running “system.sim”
4. **Latency:** a) Single-cycle: 1 / Fmax

   b) Pipelined: 5 / Fmax
5. **Performance:** clk period * number of cycles

Result Table:

	Pipelined processor without cache	**Pipelined Processor with cache**	**Multicore processor**
Max Frequency (MHz)	**53.37 (CPUCLK)**	**51.41 (CPUCLK)**	**27.05 (CPUCLK)**
Average Instructions per clock cycle (Instructions / Cycles)	**5404 instructions / 6900 cycles = 0.783**	**5404 instructions / 6900 cycles = 0.783**	**20 instructions / 161cycles = 0.12**
Latency of one instruction (ns)	**11.5**	**15.2**	**4.57**
Performance (µs)	**176.93**	**134.22**	**5.97**
FPGA resources	**Total Logic Elements: 3790 / 114480 (3%) Total Registers: 1820 / 117053 (2%)**	**Total Logic Elements: 7893 / 114480 (7%) Total Registers: 4257 / 117053 (3%)**	**Total Logic Elements: 17313 / 114480 (15%) Total Registers: 8347/ 117053 (7%)**

Performance Sweep tables

singlecycle							
LAT	CLK	RAM CLK	CPI	Instr/cycle	latency/instr(ns)	MIPS	total time(us)
0	39	69.7	1.2769245	0.783131657	20.0802989	49.80005551	176.9358974
2	38.66	67.47	1.277063286	0.78304655	20.25689749	49.36590118	178.4919814
4	37.47	66.78	0.957785899	1.044074673	20.90023104	47.84636103	184.1606619
6	37.29	70.67	0.851360104	1.174591099	21.00111711	47.61651462	185.0496112
8	38.49	60.29	0.798147206	1.252901711	20.34636677	49.14882402	179.2803326
10	39.54	66.71	0.766219467	1.305109101	19.80606113	50.48959474	174.519474

pipeline							
LAT	CLK	RAM CLK	CPI	Instr/cycle	latency/instr	MIPS	total time(us)
0	53.37	139.82	1.628330866	0.614125803	11.50694777	86.90401832	164.8772719
2	60.15	159.59	1.953136566	0.511996968	10.20990528	97.94410159	146.2926018
4	56.45	140.71	1.556416543	0.642501523	10.87911076	91.91927739	155.8813109
6	55.82	149.63	1.186146836	0.843065943	11.00189542	90.89342894	157.6406306
8	53.81	145.33	1.112092894	0.899205458	11.41285639	87.6204839	163.5290838
10	58.34	155.21	0	#DIV/0!	10.52666785	94.99682272	150.8313336

pipeline with caches							
LAT	CLK	RAM CLK	CPI	Instr/cycle	latency/instr(ns)	MIPS	total time(us)
0	51.41	129.89	1.2769245	0.783131657	15.23306083	65.64668856	134.224859
2	50.93	123.66	1.277063286	0.78304655	15.37662786	65.0337648	135.4898881
4	49.7	125.57	0.957785899	1.044074673	15.7571762	63.46314767	138.8430584
6	51.3	115.92	0.851360104	1.174591099	15.26572431	65.50622687	134.5126706
8	50.13	125.23	0.798147206	1.252901711	15.6220159	64.0122252	137.6521045
10	51.25	116.35	1.532438934	0.652554551	15.2806177	65.44238064	134.6439024

multicore							
LAT	CLK	RAM CLK	CPI	Instr/cycle	latency/instr(ns)	MIPS	total time(us)
0	27.05	126.21	8.075	0.123839009	4.578151915	218.42875	5.970425139
2	22.17	124.38	5.7125	0.175054705	5.585882241	179.02275	7.284618854
4	28.36	125.36	3.69375	0.27072758	4.366678748	229.007	5.694640339
6	33.38	121.6	2.9875	0.334728033	3.709976312	269.5435	4.838226483
8	26.12	117.38	2.715625	0.368239356	4.741156558	210.919	6.183001531
10	32.8	122.06	2.4975	0.4004004	3.775579551	264.86	4.923780488

Conclusion

According to the result we got from all three processors, we can conclude that, by adding cache hierarchy to the pipelined processor design will improve the overall performance. Although the latency per instruction has increased, but due to the decrease in read & write time of cache hits, the total runtime has decreased about 28%. This is a significant improvement on the overall performance and we are happy about the result. Our Max Frequency for cached-pipelined processor is less than 5% of the pipeline processor without cache.

However, the data we collected for the multicore processor has some problems and we cannot use it to compare the performance with single-core. Our MSI protocol design has significant flaws on date replacement logic so it cannot perform correctly under palgorithm.asm and dual.mergesort.asm. We found this problem after we finished adding the parallel algorithm. We didn't have time to fix it because we basically need to rewrite the entire lab10~lab11 content. As a result, we used test.coherence2.asm as our benchmark instead. However, the test program is too small (20 instructions in total) and we cannot obtain any usable performance data. The only useful data was the logic elements and total register used. We can see that a multicore processor used almost twice as much resources as the single-core processor because we double the datapath registers and cache registers used.

Overall speaking, we have a successful pipelined processor design with cache hierarchy, but unsuccessful multicore processor design. Although we have correct parallel algorithm implemented, but the lack of complete testing on our MSI protocol caused coherence issue on certain cases.

Contribution

- Jhen-Ruei Chen: I-cache design, D-cache testbench, datapath cache implementation, bus controller testbench, parallel algorithm, Report data collecting
- Yue Yu: D-cache design, I-cache testbench, Bus controller design, modified D-cache design, LL/SC implementation, Report Writing