FINAL DESIGN REPORT

Hao Wang, Siyi Cai
TA:Nich Pfister
DUE at 12/9/2016 11:59 PM

Overview

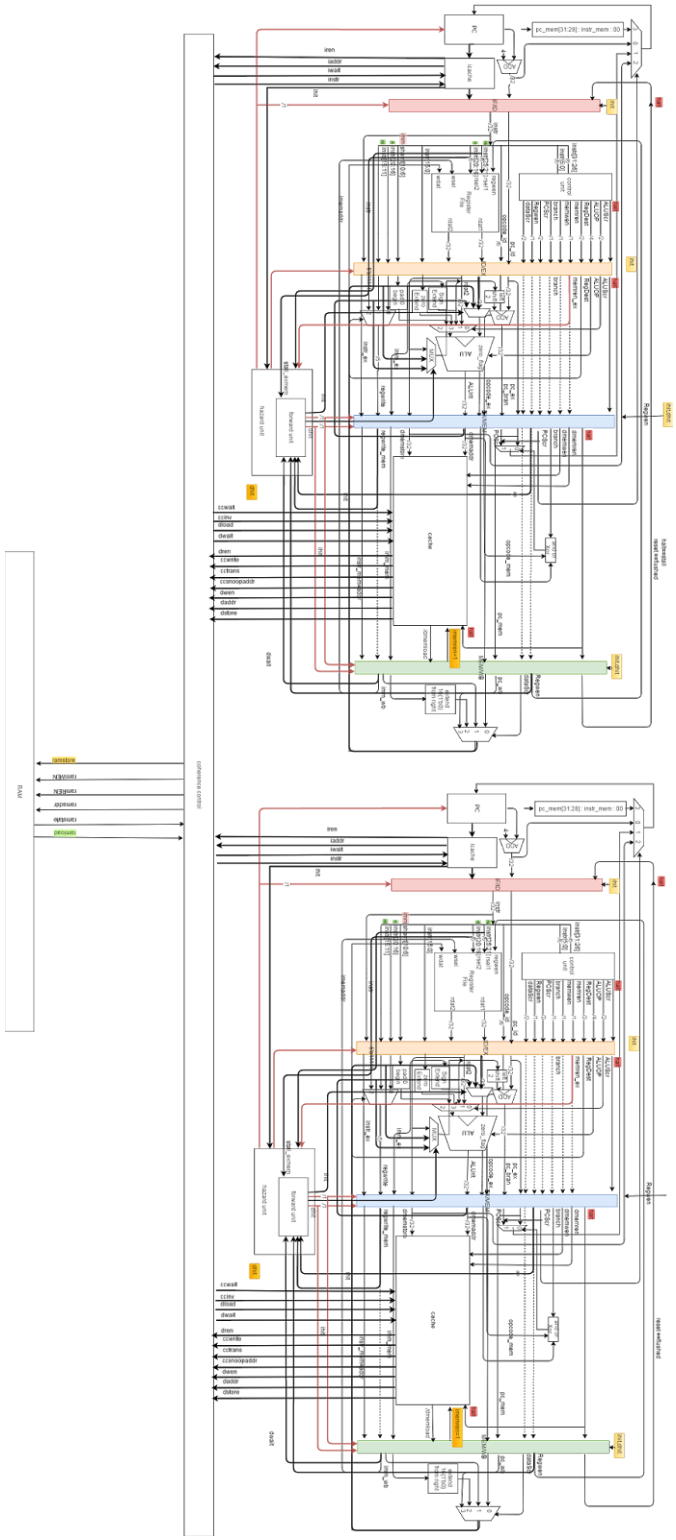
Up to now, we finished single cycle, pipeline, pipeline with cache, and multi-core design. Obviously the second phase of our design (pipeline with cache, and multi-core) made great improvement on data processing and the speed of the processors. At the first phase, we kind of unsatisfy with the performance of either single cycle or pipeline. Both of them will process longer than expecting on the situation with heavy loops or repeated data. Pipeline with cache we designed solving this situation. We designed two caches, one for instruction (icache), and another one is for processing data (dcache). The purpose of instructions cache is to reduce the cycles when the program has loops, and try to read the same instruction from the same address. Meanwhile, data cache decreases huge amount of cycles, while the program has loop as well as manipulation on memory. As a result of this cache implementation, we had a significant performance boost on cases such as mergesort.

As well, although pipeline has made great optimization on the speed compared to single cycle, multi-core will make will make processors work more efficiently with two pipeline cores. Since these two cores with cache respectively are sharing one memory, this implementation won't improve the memory reading or writing. Multi-core design, however, means there are two pipeline systems with caches inside and it is able to run two instructions at the same time. As the implementation of instruction cache, multi-core design saves a lot time instead of cycles because it just allocates those instructions to two cores at the same time.

All the data collected in this report is based on running mergesort.asm file and dual.mergesort.asm which designed for multi-core. We will compare the following data for each of the three design: estimated synthesis frequency, average instructions per clock cycle, latency of one instruction, FPGA resources and total execution time.

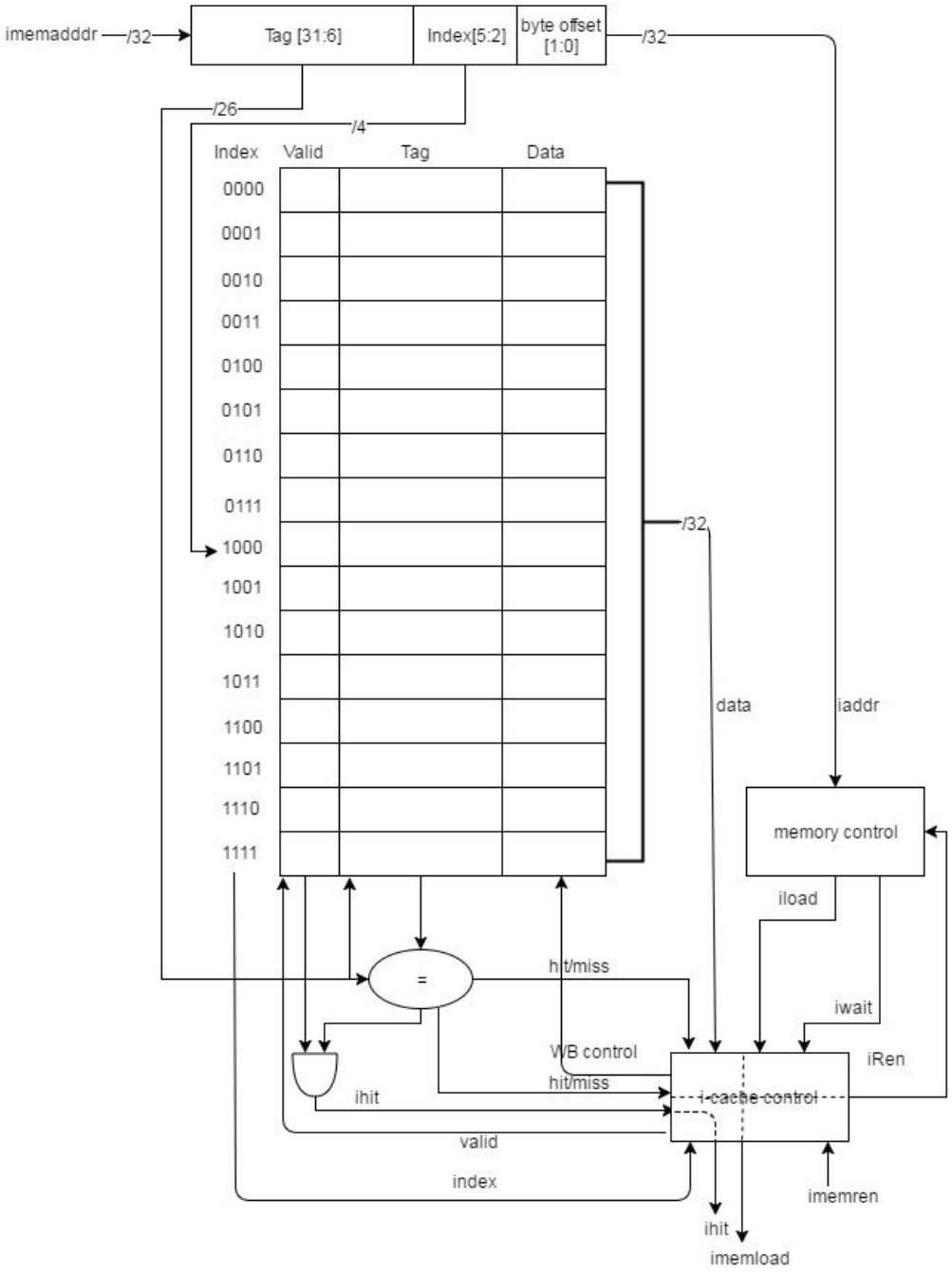
Design

- Multi-core block diagram



- Icache block diagram

i-cache



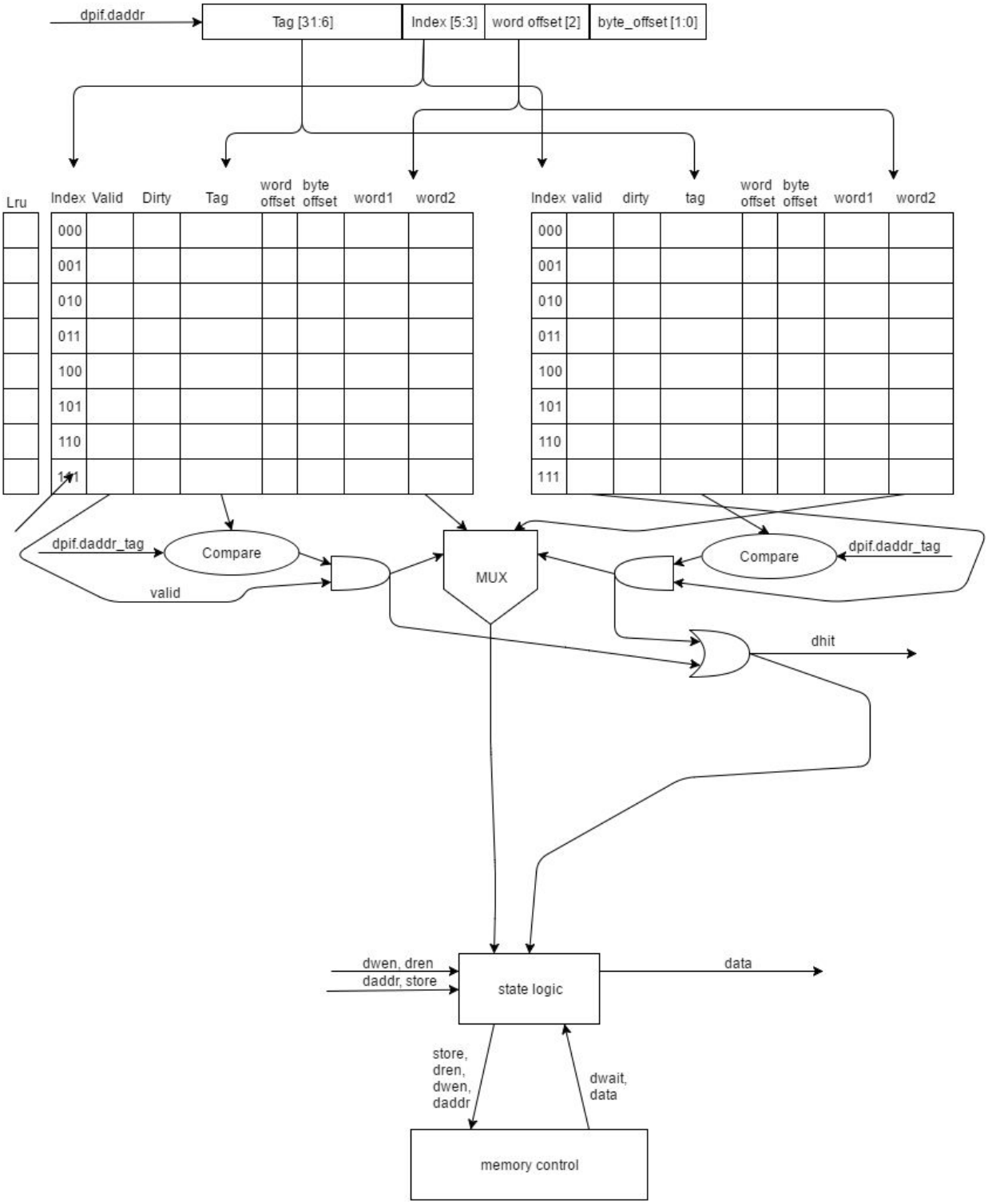
WB control = miss && !iwait

ihit = ihit when hit || ihit when miss

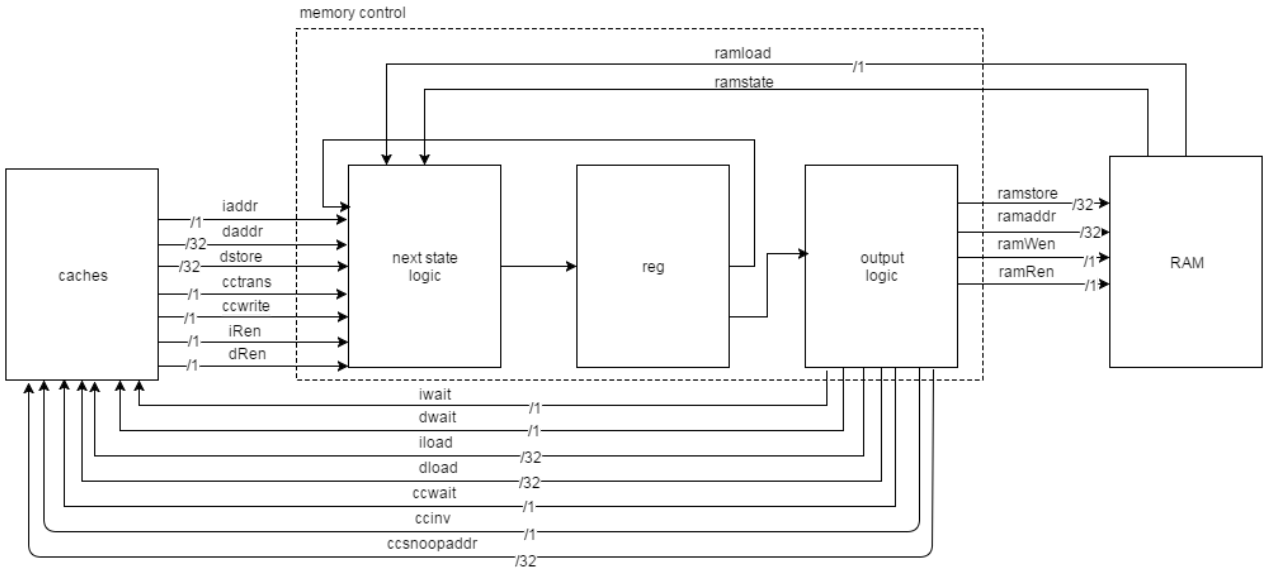
ihit when miss = ! iwait && miss

iRen=miss && imemren

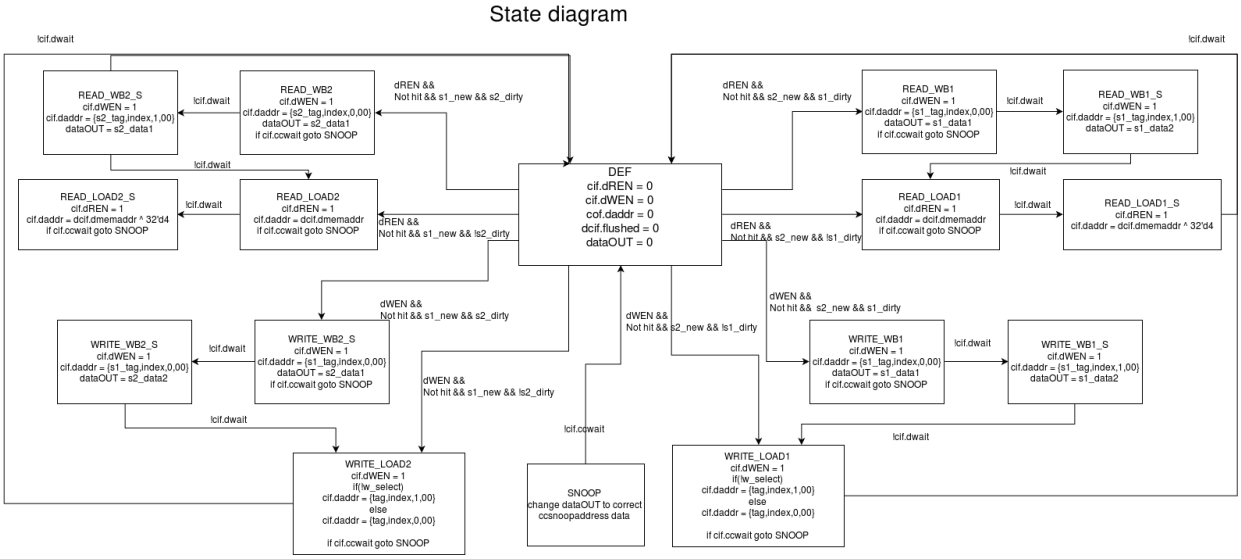
- Dcache block diagram



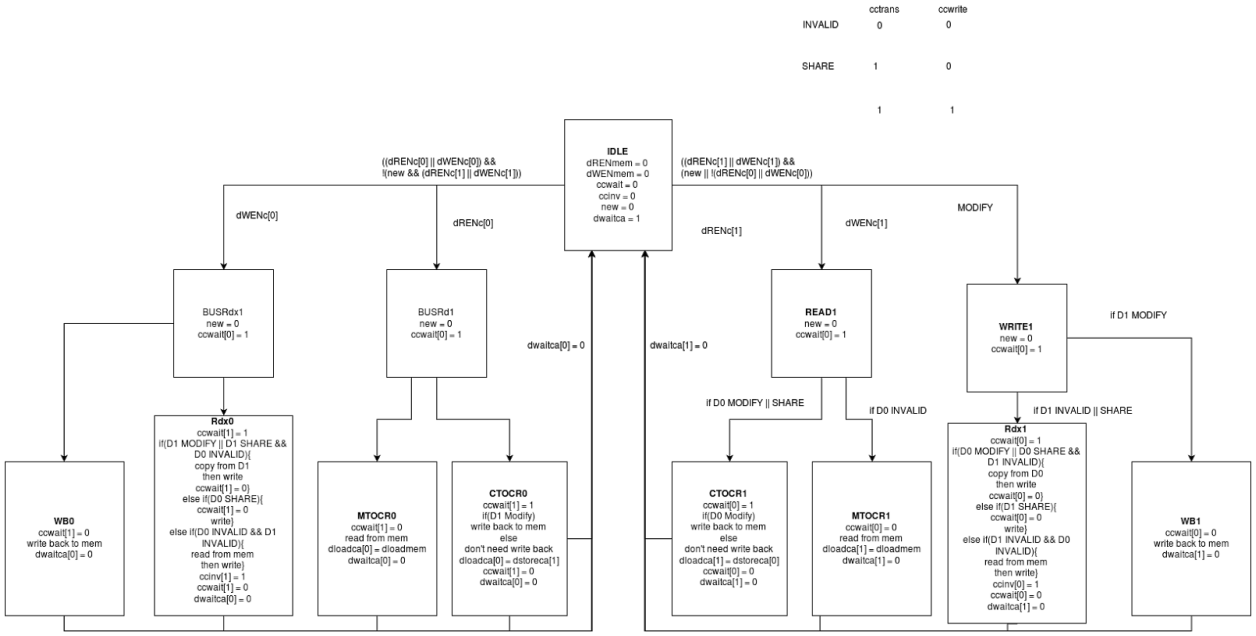
- Coherence block diagram



- Dcache state diagram



- Coherence state diagram



Result

Ran with variable latency = 2

	Pipeline	Pipeline (with Cache)	Multicore
Fmax	66.98 MHz	68.66 MHz	70.36 MHz
Instructions/cycle	1 / 8.6	1 / 8.9	1 / 4.9
Latency/instruction	74.65 ns	72.82 ns	71.06 ns
Registers Total logic elements Total combinational functions Dedicated logic registers	 3927 3598 1800	 7809 6661 4268	 17124 14626 8332
Total execution time	874780 ns	958640 ns	533440 ns
Speedup (compared with pipeline without cache)	0	-1.095	1.39

- Fmax comes from system.log.
- Instructions/cycle = instructions (comes from result after running sim) / cycles (after gate level simulation)
- Latency/instruction = (1/Fmax) * 5
- Registers comes from system.log
- Total execution time is from running gate level simulation
- Speed up = abs(time for first processor - time for second processor) / time for first processor + 1

Conclusion

The results in the table shows that the pipeline with cache has basically the same performance comparing to pipeline without cache. We believe this is because the file we chose to ran between these three processors is mergesort, which does not utilize the cache very efficiently. Our pipeline with cache design has slightly higher maximum frequency and lower latency/instruction. We believe with a different test file that utilize the cache design better the pipeline with cache design will show more significant advantage over the pipeline without cache design.

The multi-core design is apparently the best design among all three designs. With two processors running together, It has the highest maximum frequency, lowest latency/instruction and fastest speed, which shows a 1.39 speedup compared with pipeline without cache design. The only downside is it has significantly more registers than the other two. Even with the additional coherence logic between processors' caches and memory it is still the fastest and most superior design.

Contributions

Name	Contribution
Hao Wang	Working on dcache, debugging cache, working on coherence control with Siyi and debugging multicore.
Siyi Cai	Working on icache, and debugging cache, and working on coherence control and synchronization together with Hao and debugging multicore.