Final Report for Multicore Design

ECE 437L

Michael Baio(mg259)
Lucas Goedde(mg228)
Mengchi Zhang

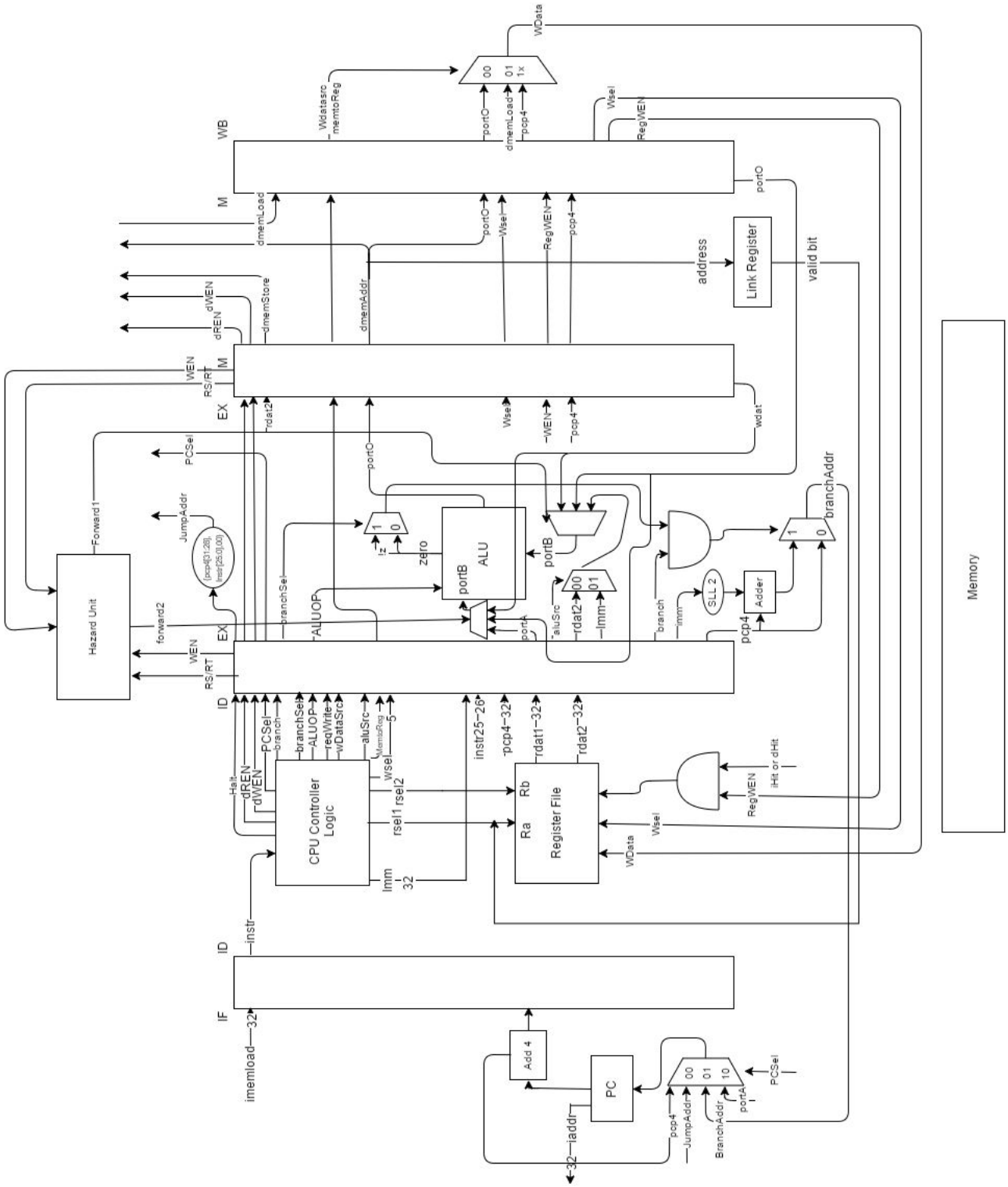
Friday December 8, 2016

1.0 Overview

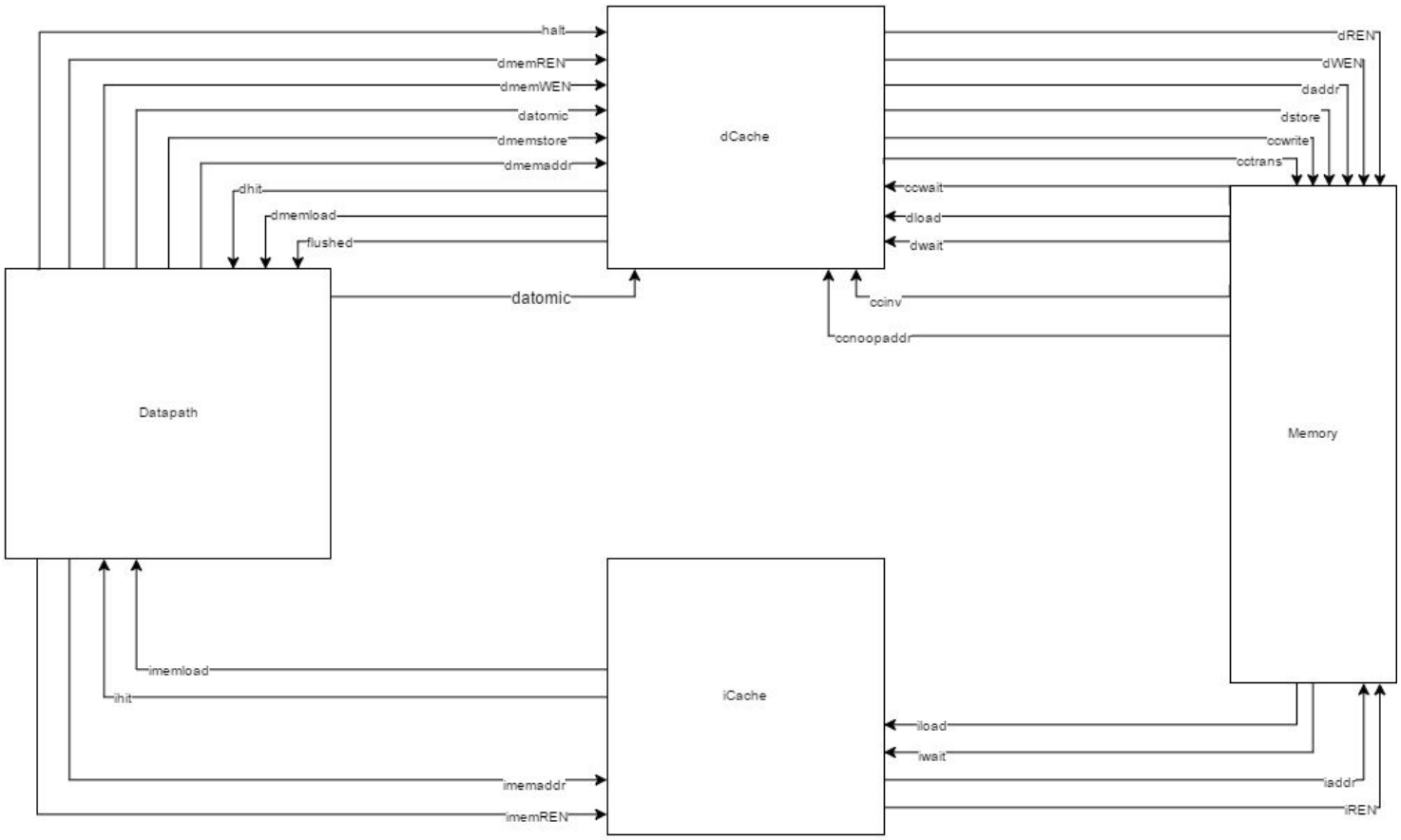
The purpose of this report is to provide a comparison of the pipeline processor with and without a cache hierarchy and the pipeline processor with cache and a multicore processor. Both processors will run a similar set of instructions, a mergesort algorithm for the non-multicore processor and a dual-mergesort implementation meant to run on a multicore processor. This will serve to compare clock frequency, hardware resources, processor speedup as well as a few others. One caveat is that the processors will only sort 6 items because the teams multicore processor would not run mergesort for lists of numbers greater than 6. The team attempted to debug the issue but ran out of time before the submission deadline.

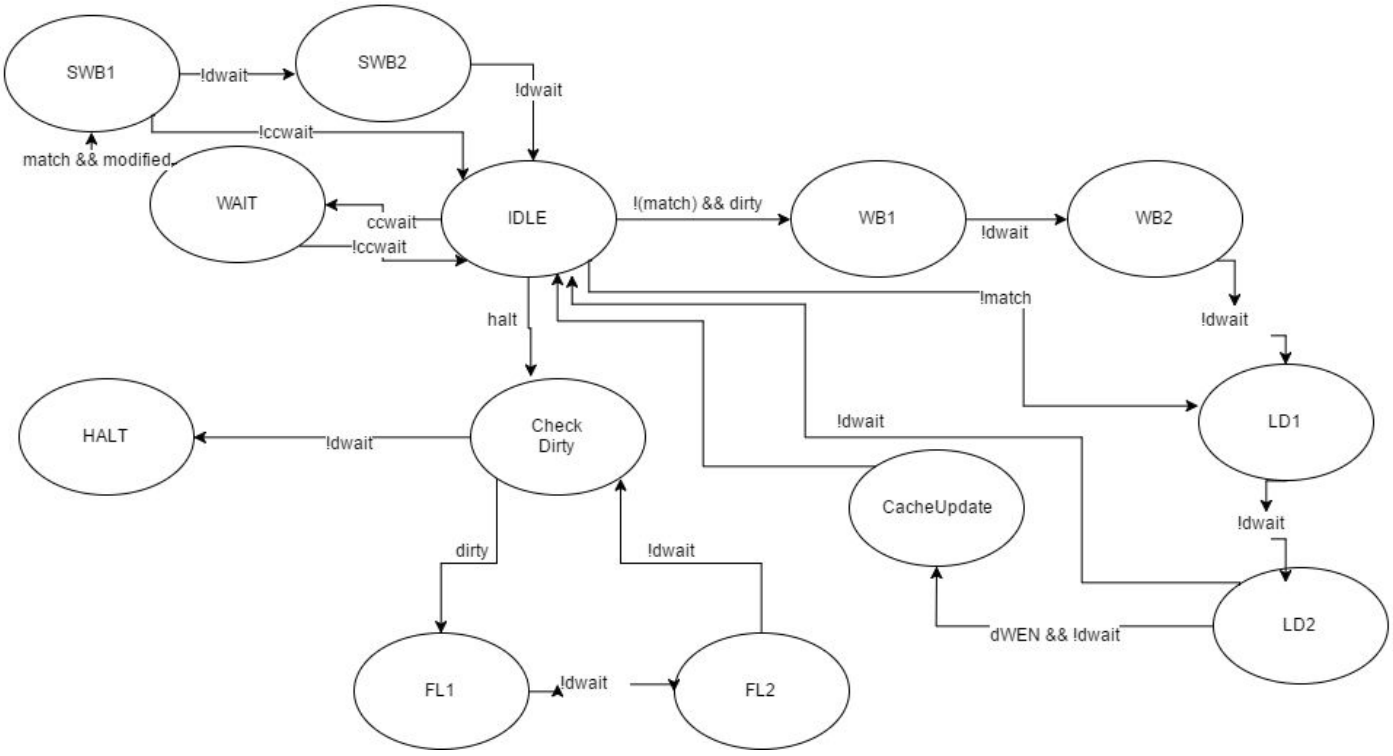
The benefit of mergesort.asm test program is that it is complex enough to hit all of the longest critical paths and is computationally intensive enough to stress the designs. After running the processors using a similar mergesort program, data is gathered from the output of the simulation and from the log file that is generated when the processor is synthesized. Two of the main metrics to consider are max frequency and program speedup for the designs. From the table provided in the following pages, it can be concluded that the multicore processor is marginally faster than the pipeline processor. Because of the limitation of only being able to sort 6 items, the multicore processor doesn't get to show it's full potential compared to the pipeline, but this will be explained more in the report below.

2.0 Processor Design

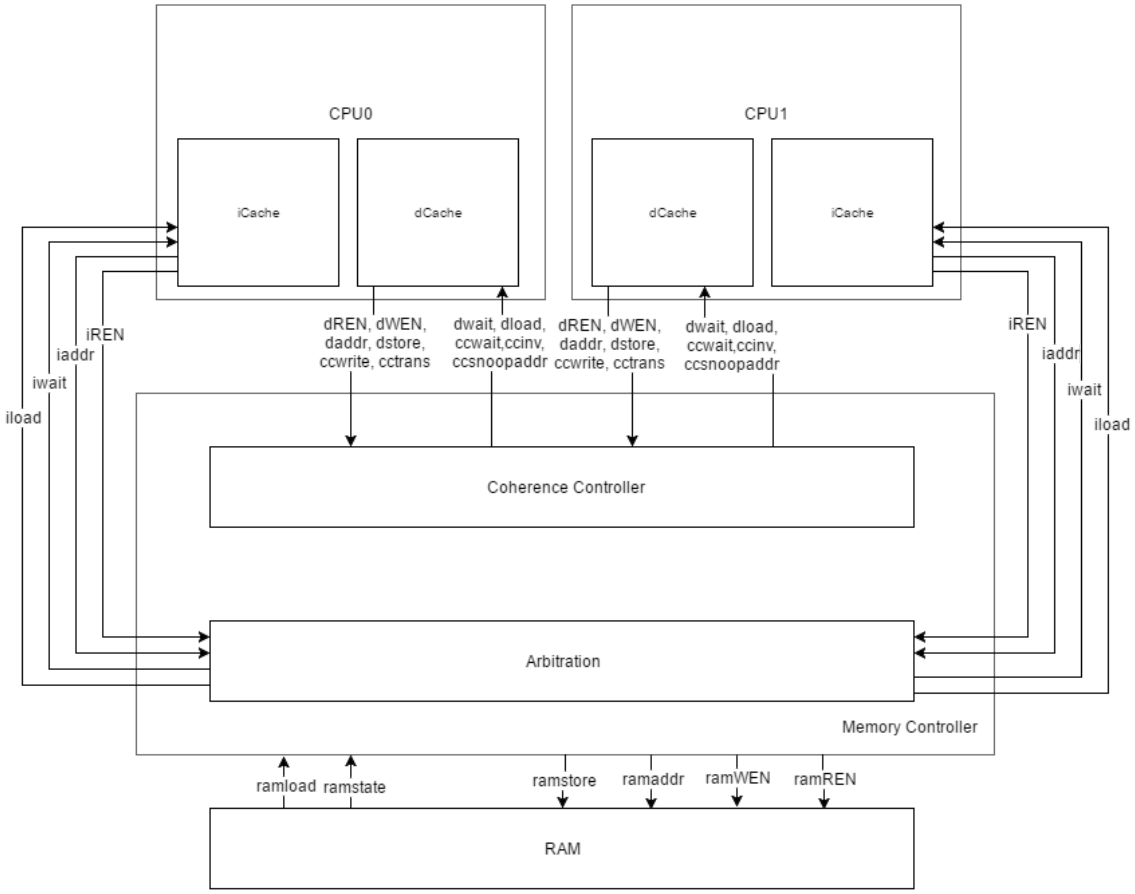


Pipeline processor w/ Link Reg - Lucas Goedde and Mike Baio

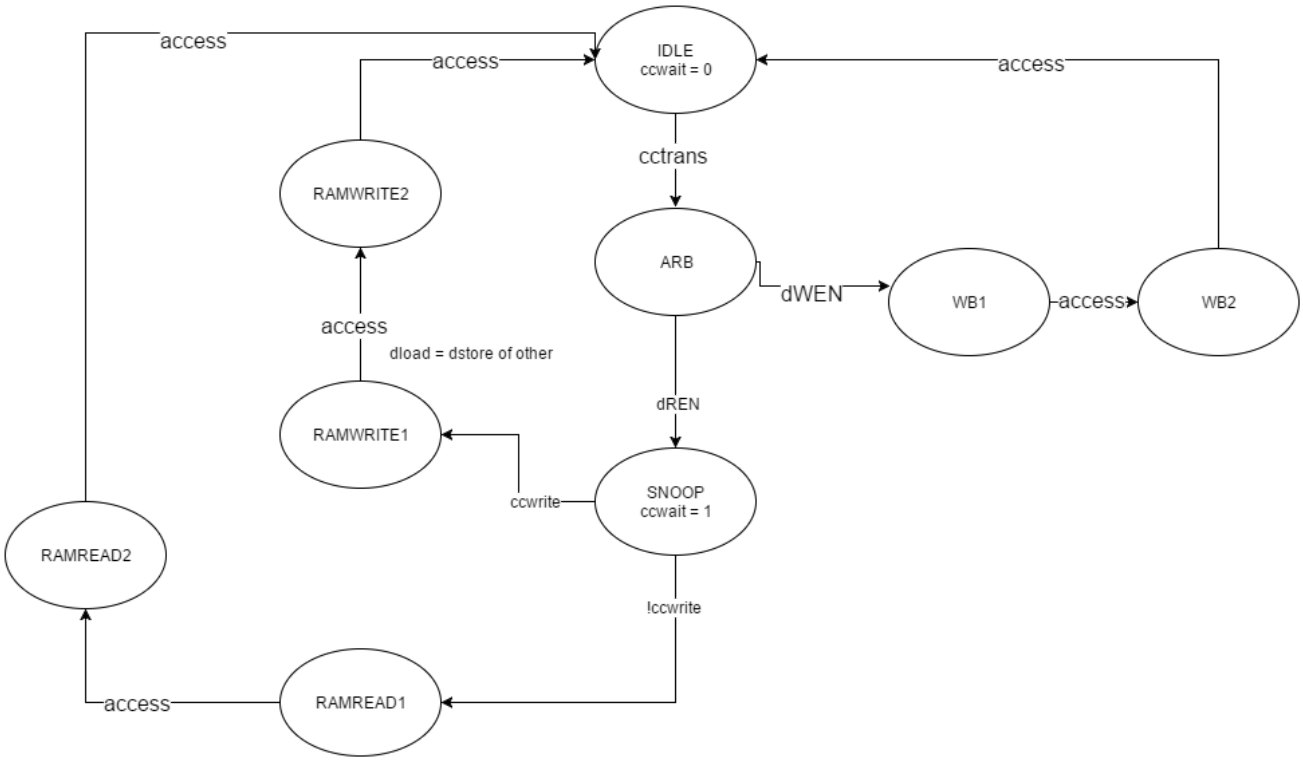




Cache Block Diagram and State Machine



coherence STD



Coherence Block Diagram and State Machine

3.0 Results

Metric	Pipeline w/o Cache	Pipeline w/ Cache	Multicore
Estimated Synthesis Frequency	49.79 MHz	43.63 MHz	35.20 MHz
Instructions per Cycle (IPC)	.046	.075	.051
Latency of 1 instruction	100 ns	115 ns	142 ns
FPGA Resources - Combinational - Logic registers	 3,342 1,680	 6,643 4,184	 14,082 8,134
Speedup from Sequential to Parallel	--	--	.8% speedup
# of Instrs(Core1/Core2/Total)	167/--/167	167/--/167	135/47/182

3.1 Calculation Details

Memory Latency for all designs = 7

Estimated synthesis frequency - taken from system.log file created during synthesis

IPC = # of Instructions / # of Cycles

Latency = period * 5 (for all)

Speedup = pipeline time / multicore time

3.2 Calculation Data

FPGA resources - taken from system.log file created during synthesis

Number of instructions- taken from output of sim -t

Cycles and execution time - taken from output of make system.sim

4.0 Conclusions

From the results presented in the table above it can be seen that the multicore design is just slightly faster than the pipeline without caches and a bit slower than the pipeline with caches. As mentioned earlier, the designs had to be tested using a version of mergesort that only sorted 6 numbers instead a list of 64 numbers. At first this may not seem like an issue until the overhead required for multicore is considered. This overhead adds complexity and slows the processor, evidenced by the lower clock speed in the table and thus will have a “startup” period where it could be seen as slower than the pipeline design. Because only 6 numbers were sorted and not a higher number, the full advantage of multicore does not offset the overhead associated with the design leading to only the slight speed up in the results. Dealing with this issue is the same as the concept of filling and emptying the pipeline versus a single cycle design. The pipeline adds overhead compared to the single cycle but once you have a larger number of instructions beyond the start up period then the real benefits of pipeline show up.

Besides the speed up from sequential to parallel, the multicore design uses over 2 times as many resources as the pipeline design and takes much longer to synthesize. An obvious consequence of synthesizing two complete cores instead of a single core, one has to take this into consideration if they would be designing in real life scenarios. A more complex design may be faster but it could also strain hardware resources and cost more money in a production situation.

5.0 Contributions

For caches and multicore labs, work was split as evenly as possible between Michael and Lucas. During labs 8 and 9, Lucas was responsible for the design of the icache and part of the dcache table structure while Michael implemented some of the dcache logic as well as created testbenches to test and debug our caches. Lucas created the initial implementation of the coherence controller while Michael created the block diagrams necessary to implement the code and the testbench to ensure coherence was performing as expected. Michael also had to make changes and bug fixes as he found issues during his testing. For the last two labs, the team kept the system of splitting the work of creation and debug. Lucas added LL/SC functionality which wasn’t very involved so he also debugged the test programs, such as example.asm and dual.llsc, alongside Michael who created the team's version of palgorithm.