Final Report : Pipeline vs. Multicore (and Caches)

ECE 437 : Computer Design and Prototyping

Katie Bauschka

Apoorv Gaur

Chuan Yean Tan

Friday, April 29, 2016

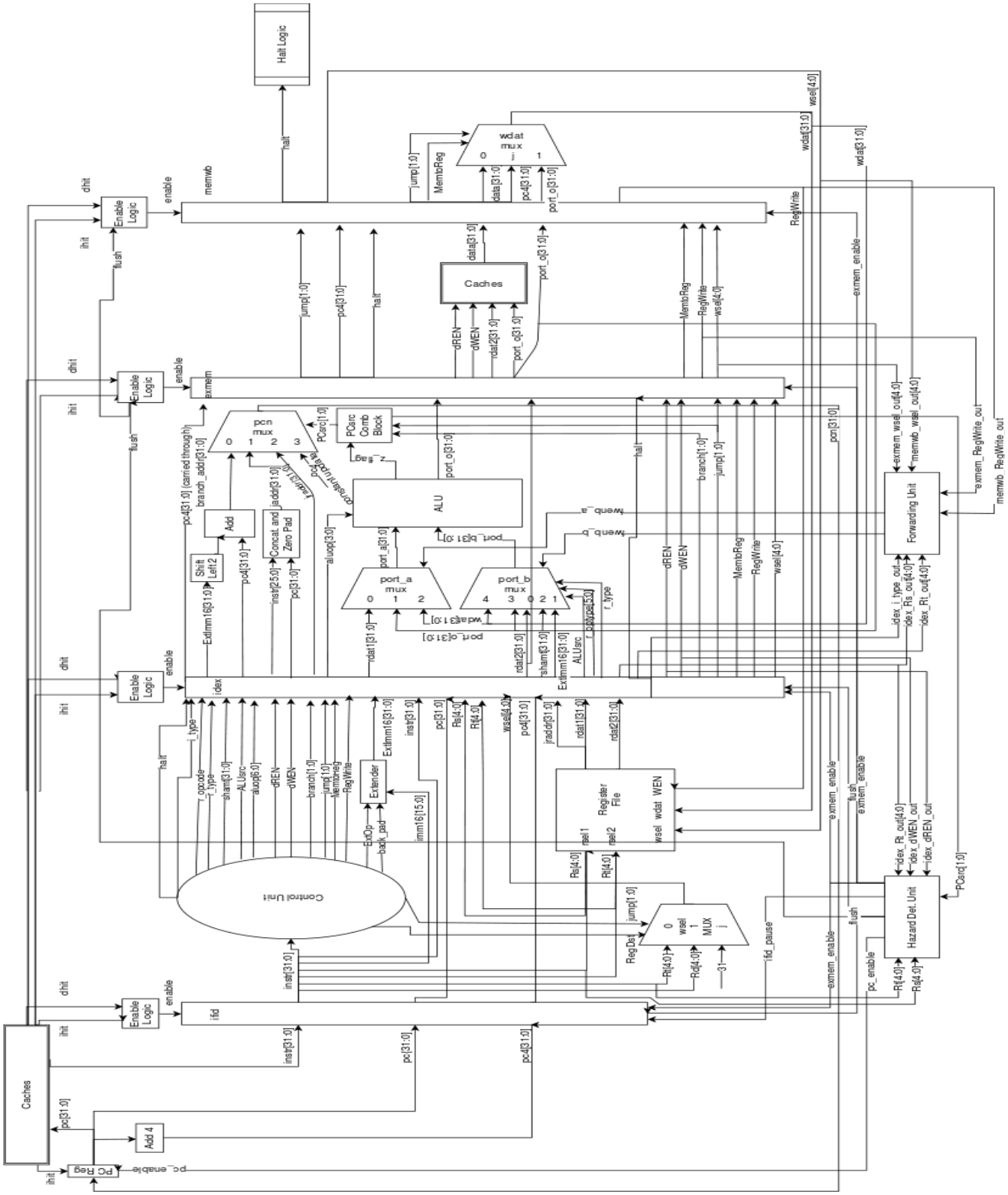
Overview

Performance is the main feature being compared between the various designs: pipeline without cache, pipeline with cache, and multicore. The first design is the most simplistic of the three. The second design adds caches, largely reducing the amount of time it takes to access memory, therefore speeding up the performance of the overall design. The final design is multicore, which includes caches. The performance goal of multicore is to lighten the load on a single processor, by mimicking a second one. A multicore processor might not necessarily have a faster performance than a pipeline with cache processor.

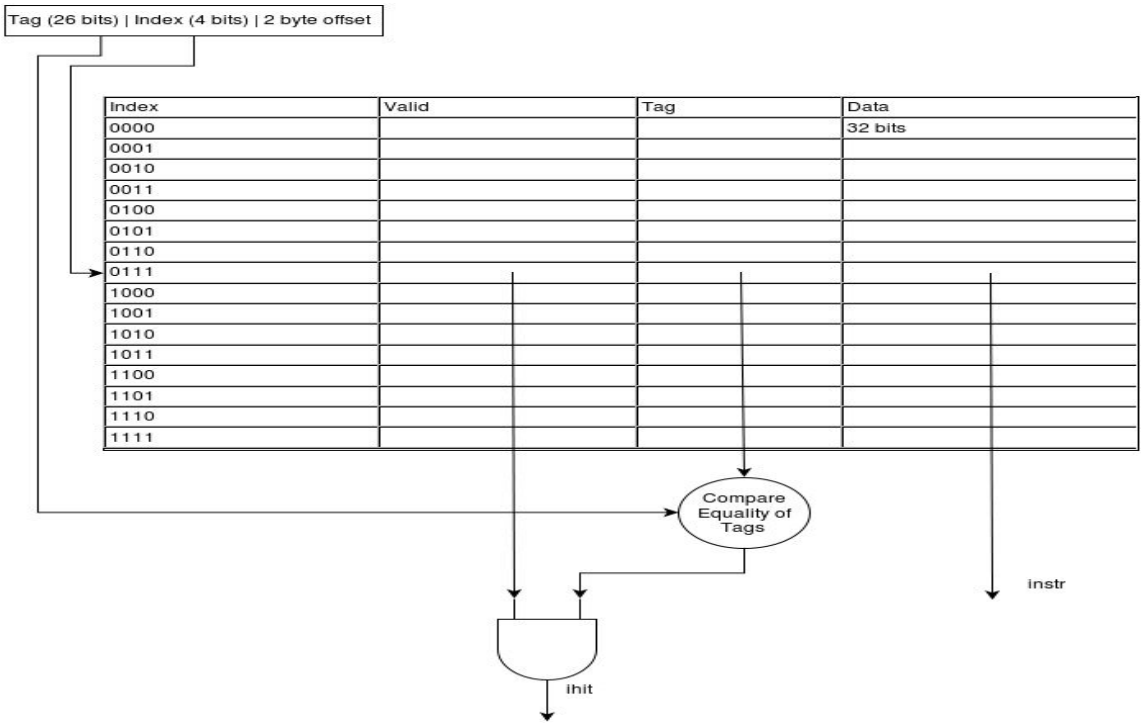
We are using mergesort.asm to test the two single core processor designs, and dual.mergesort.asm for the multicore processor. These programs will test all the types of instructions, and memory accesses. For the two single core designs, it shows the advantage of including the caches. For the multicore design, it shows the improvement in throughput. At the end of the testing, we expect to see a shortening in the clock frequency from pipeline w/out cache to pipeline w/ cache. Pipeline w/ Cache and multicore should have similar speeds, depending on coherence, but multicore should have the higher throughput. The remainder of this paper will further explain the designs through pictures, and report the results. A conclusion will give final remarks on the three designs.

Design

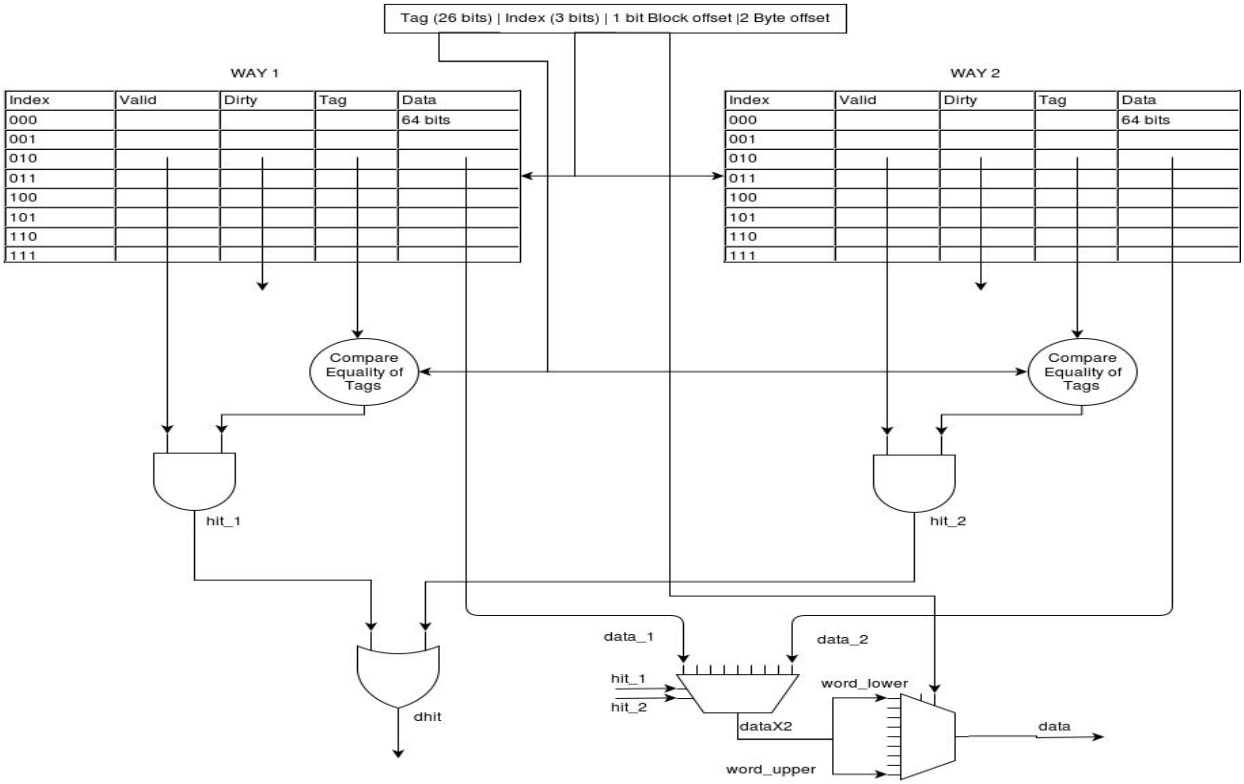
Pipeline



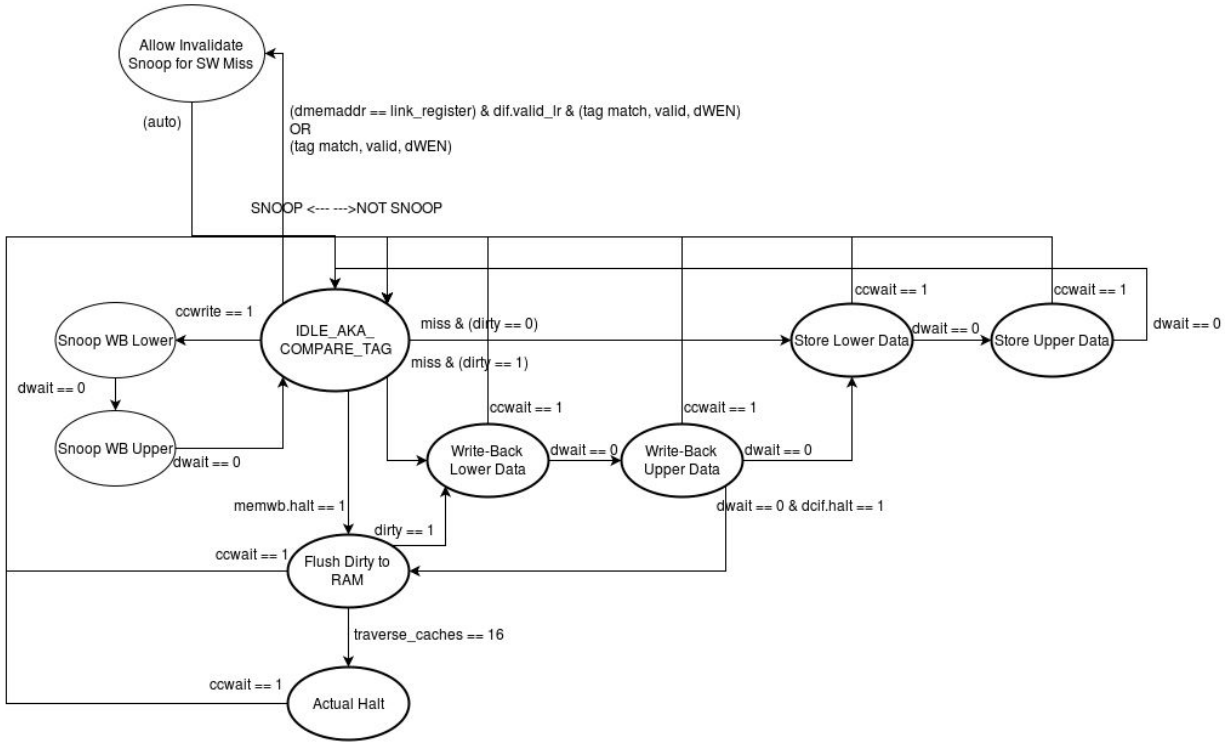
I-Cache



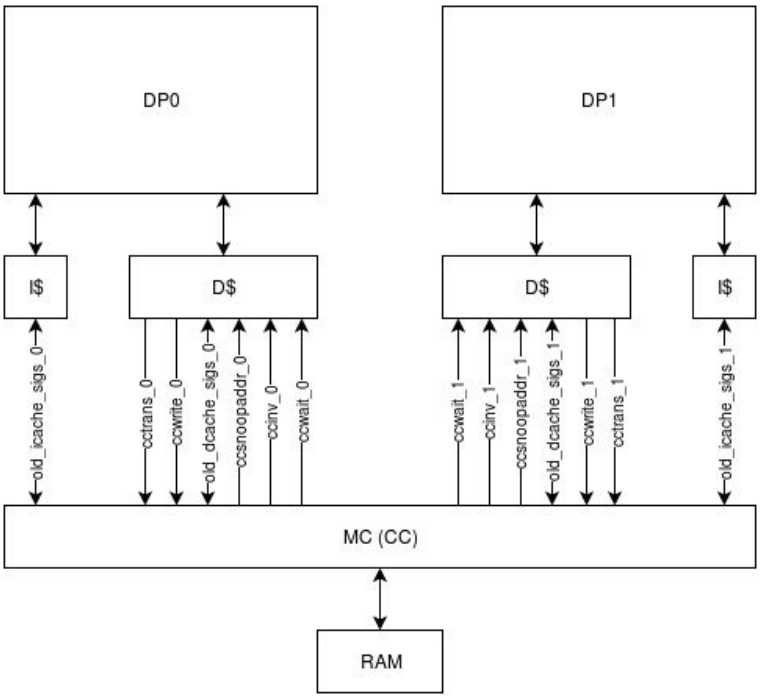
D-Cache



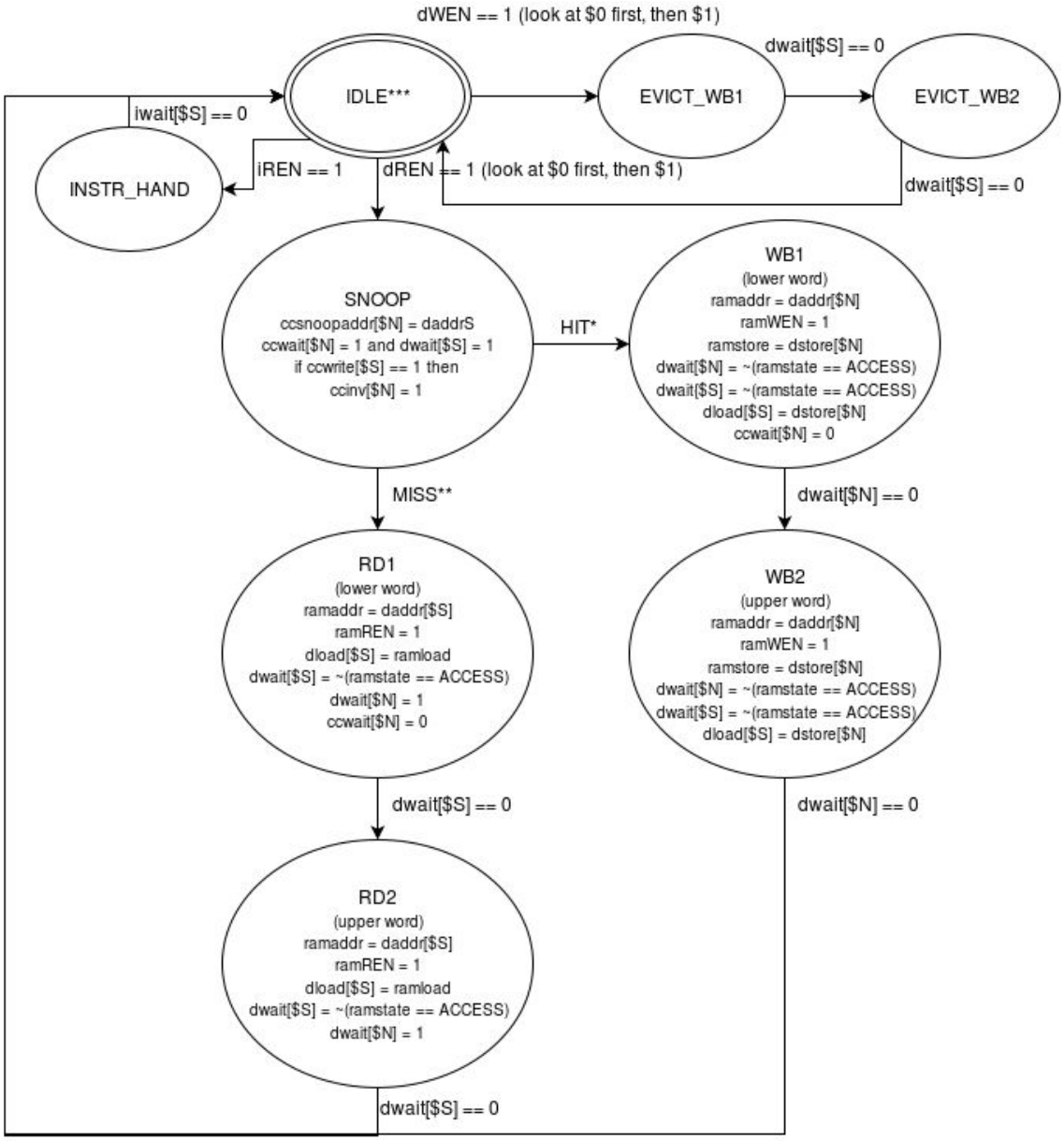
D-Cache State Transitions



Cache Coherence (a.k.a. Memory Control)



Cache Coherence State Transitions



Results

Pipeline w/out Cache Processor

Frequency (maximum possible)	min(72.98, 49.03) = 49.03 MHz (CPUCLK) *
Instructions per Clock cycle	5399 / 71901 = 0.075 IPC
Latency (of one instruction)	5 * (1/49.03MHz) = 101.98 ns

Original Execution Time	5399*(71901/5399)*49.03MHz = 3,525 GHz **
FPGA Resources	
Total Combinational functions	3,399
Total Registers	1,729

Pipeline w/ Cache Processor

Frequency (maximum possible)	min(56.89, 49.40) = 49.40 MHz (CPUCLK) *
Instructions per Clock cycle	5399 / 24199 = 0.22 IPC
Latency (of one instruction)	5 * (1/49.40MHz) = 101.21 ns
Original Execution Time	5399*(24199/5399)*49.40MHz = 1,186 GHz **
Speed Up from Previous Design	Speed up by factor of 2.97 ***
FPGA Resources	
Total Combinational functions	10, 140
Total Registers	4,155

Multicore Processor

Frequency (maximum possible)	min(61.91, 50.29) = 50.29 MHz (CPUCLK) *
Instructions per Clock cycle	(3164 + 2250) / 15415 = 0.35 IPC ****
Latency (of one instruction)	5 * (1/50.29MHz) = 99.42 ns

Original Execution Time	5414*(15415/5414)*50.29MHz = 755 GHz **
Speed Up from Previous Design	Speed up by factor of 1.57 ***
FPGA Resources	
Total Combinational functions	21,227
Total Registers	8,208

* The values being compared are the clock going into the RAM (divided by 2) and the CPU clock. The minimum frequency is used and the upper bound to the design.
** The execution time is calculated by multiplying the number of instructions times the CPI times the clock frequency.
*** The speedup is calculated by dividing the old execution time by the new execution time.
**** The instructions from Core 1 and Core 2, respectively, are added together and then divided by the total number of cycles.

Conclusions

A version of mergesort was run on all three processors, with a RAM latency of 6. As seen from the tables above, a multicore processor can operate at a higher frequency because more instructions can be executed per cycle. It has the highest throughput. The number of cycles needed to complete the program in the pipeline design with cache is much lower than without caches, since the processors does not need to waste so much time accessing RAM at each read and write. This lowering in cycles happens again in the multicore design, since it can split the work up into the two cores, which work in tandem.

The speed up going from one design to the next is also significant. By only adding caches, the design achieved a speedup factor of 2.97. By adding a second core, a speedup factor of 1.57 was achieved. In this case, multicore proves to be the best

processor design. However, should the program end up dealing with lots of coherence issues, the multicore design will begin to slow down.

Contributions

Pipeline Processor

- Block Diagram - Katie
- Final Design
    - Hazard Unit - Katie and Apoorv
    - Forwarding Unit - Katie and Apoorv
    - Datapath - Apoorv and Katie

Caches

- Block/State Diagrams - Katie
- Final Design
    - I-Cache - Katie and Apoorv
    - D-Cache - Apoorv and Katie
    - Memory Control - Katie and Apoorv

Multicore

- Block/State Diagrams - Katie
- Final Design
    - Add Coherence to Memory Control - Apoorv and Katie
    - Add Coherence to D-Cache - Apoorv and Katie
    - Add LL/SC to D-Cache - Apoorv and Katie
- LL/SC Parallel Program - Katie