Processor Prototyping Lab

ECE 437

Final Report

Abhijay Achukola

Geetha Prasuna Yarramneni

Lab section 2

Zhaoyu Jin

8th December 2024

1 Executive Overview

In Weeks 8–9, we enhanced our pipelined processor by integrating cache support to reduce memory access overhead. By storing frequently accessed data in on-chip memory (caches), we introduced additional hardware, including registers and logic units. We implemented separate Level-1 (L1) instruction and data caches to eliminate structural hazards in the pipeline's fetch and memory stages. For the data cache, we adopted a 2-way set associativity configuration to minimize conflict misses and employed an LRU (Least Recently Used) replacement policy for both caches. While adding caches slightly increases the miss penalty due to the additional cache lookup time, it significantly improves hit rate and access latency. This aligns with Amdahl's Law by optimizing the common-case performance, leading to reduced overall execution time.

To further enhance performance, we extrapolated the design into a multicore architecture. We integrated two pipelined cores with caches through a shared bus controller. To manage concurrent access to shared global memory, we implemented the MSI (Modified, Shared, Invalid) coherence protocol. This protocol ensures single-writer, multiple-reader consistency at any given time. The multicore setup offers significant advantages over a single pipelined processor with cache by enabling parallel instruction execution via parallel algorithms. While individual instruction latencies may increase due to bus protocol overhead, the overall throughput nearly doubles, thanks to the simultaneous operation of the two cores and the snoop-based cache coherence mechanism.

In conclusion, our multicore processor achieved up to a 2× throughput increase and reduced memory accesses during parallel algorithm execution, resulting in a highly efficient design.

2 Processor Design

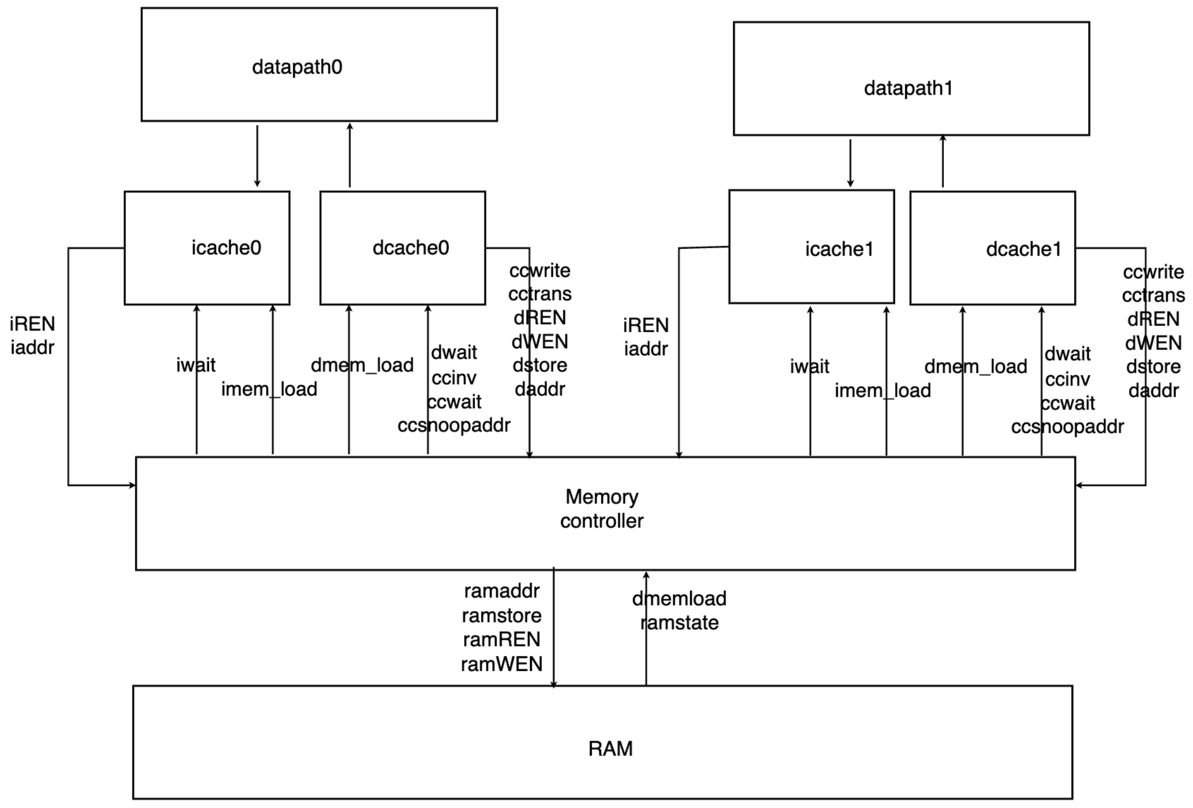

Figure 1: top level Block diagram of Multicore Processor

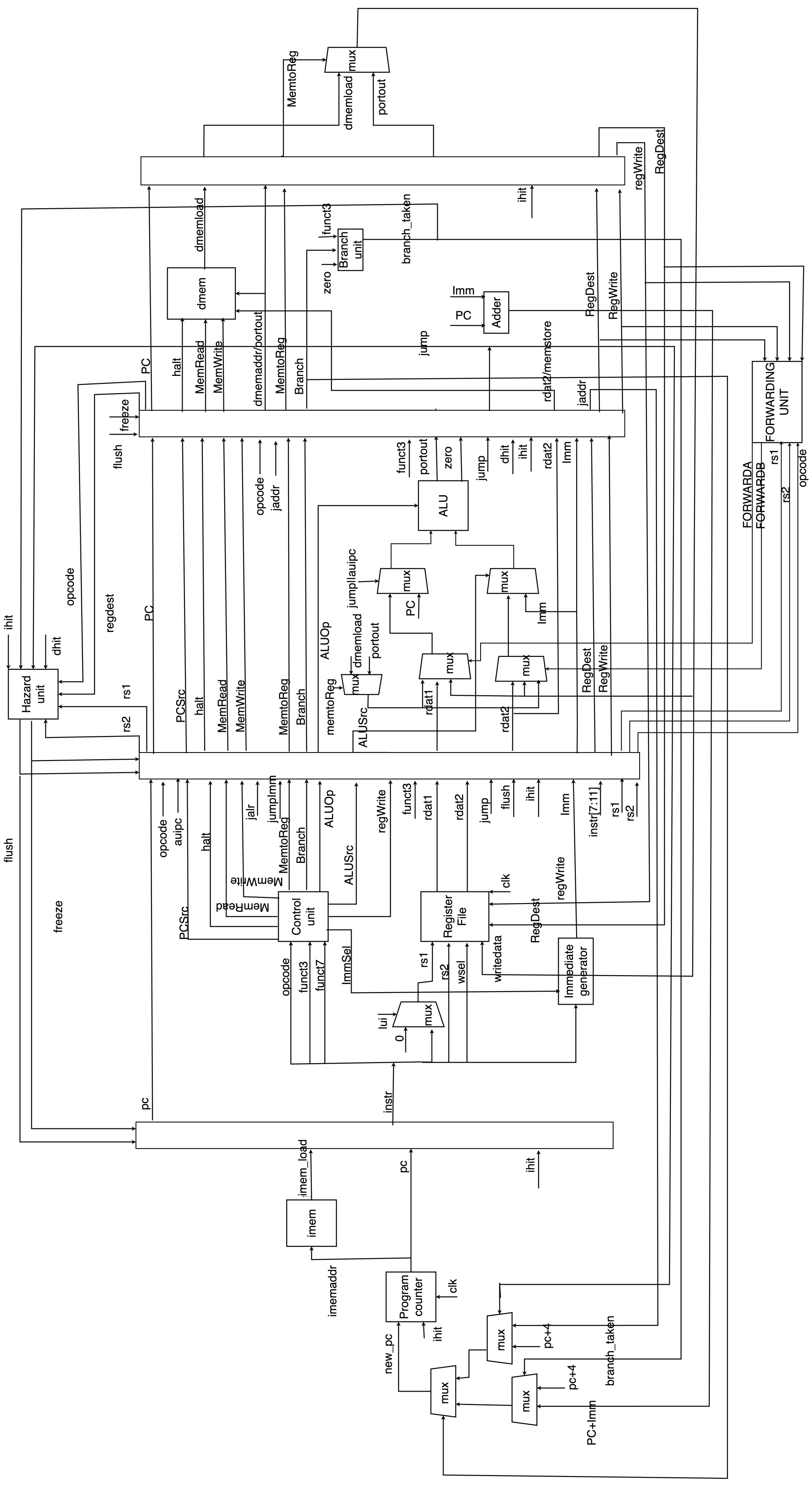

Figure 2: Datapath of Multicore Processor

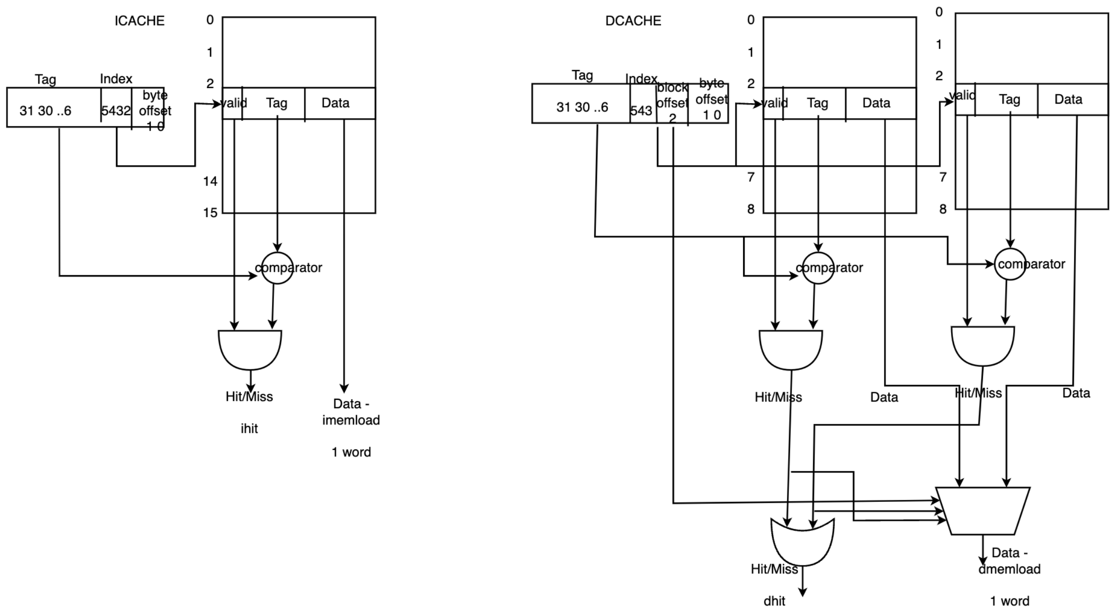

Figure 3: Block diagram of Cache in Multicore Processor

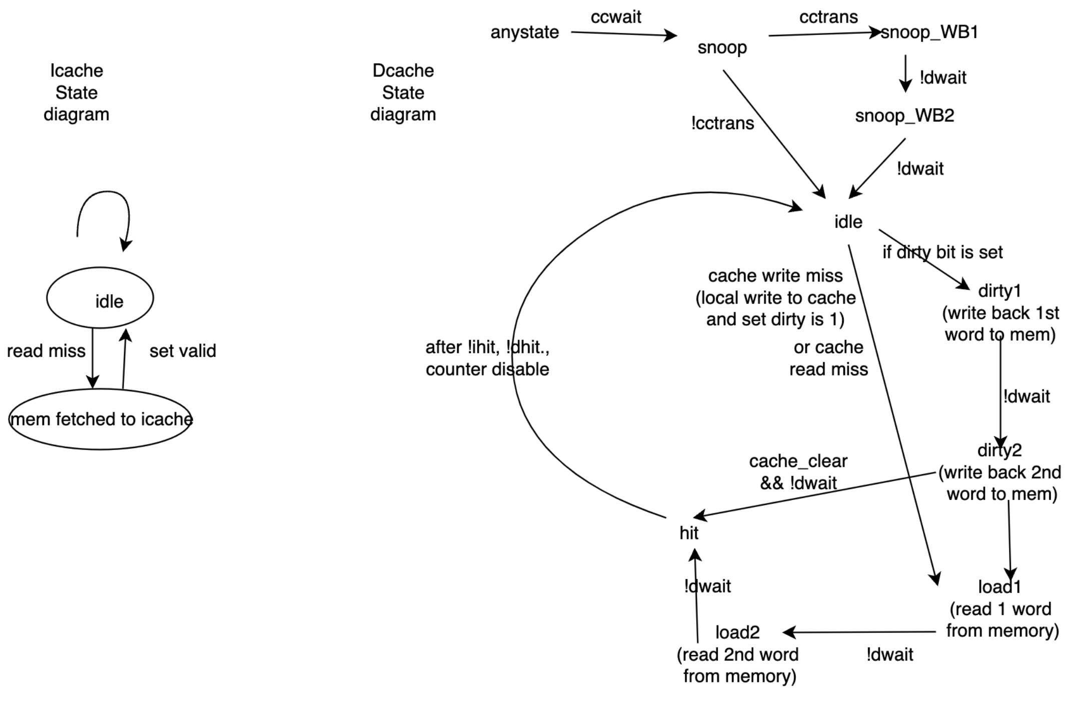

Figure 4: State diagram of Caches in Multicore Processor

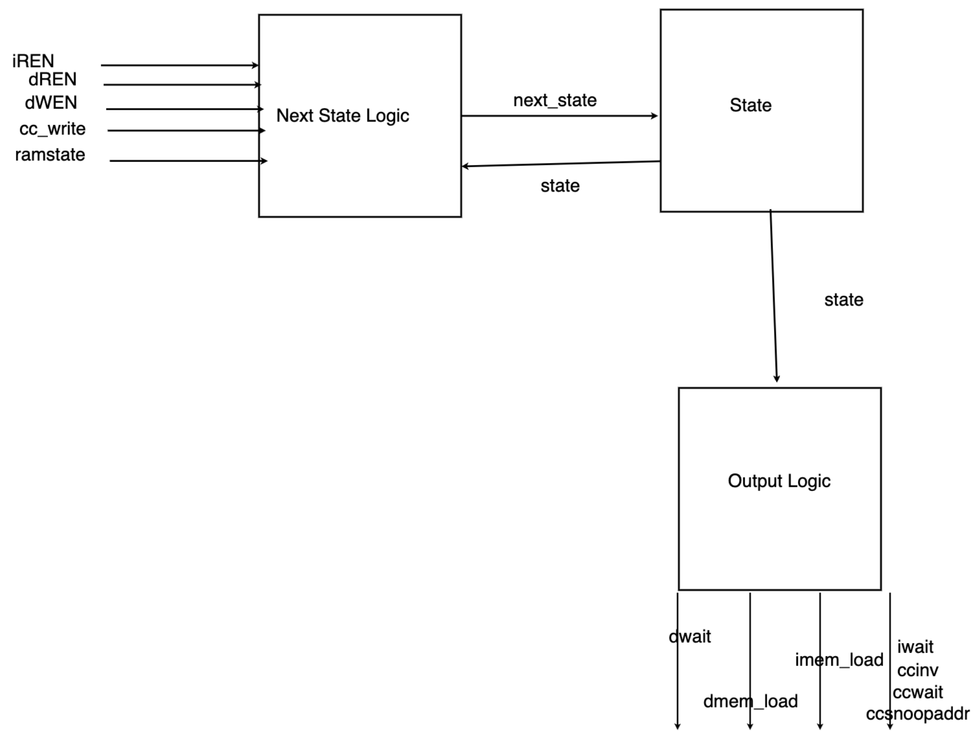

Figure 5: Bus controller in Multicore Processor

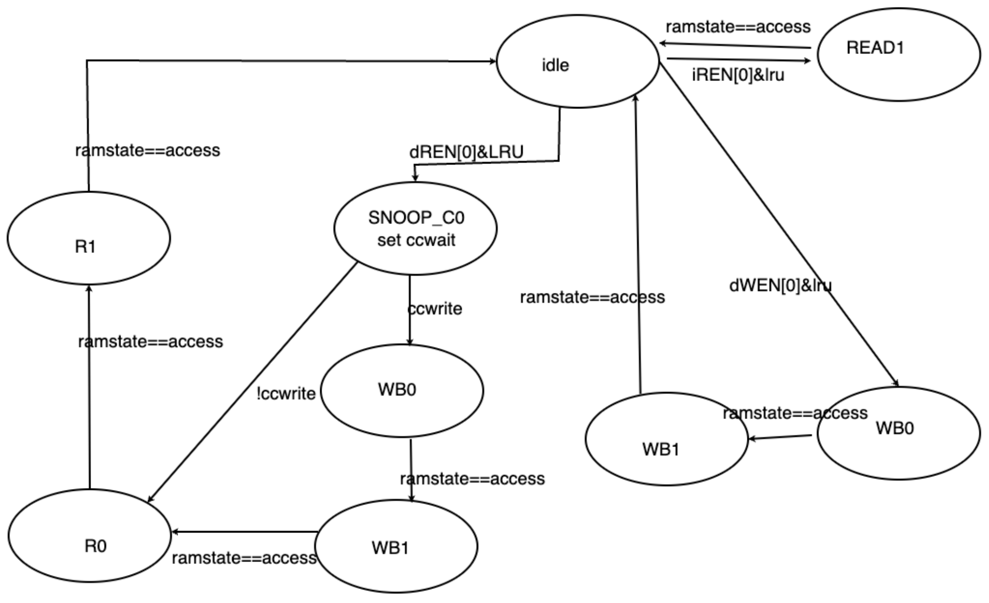

Figure 6: State diagram of Bus controller in Multicore Processor

3 Results

Testing both processor designs on the merge sort algorithm at various RAM latencies produced the max frequency of the CPU clock (F_{max}) and the number of cycles taken for each scenario within a log file. These values are depicted in table 1 below. The number of cycles taken in the log file represents the number of cycles of the testbench, so the values were divided by two before being used for other calculations, as the testbench clock is twice as fast as the CPU clock. The number of CPU Cycles taken is the value depicted in the table below. The FPGA resources needed for each design was also shown in the table below. Taking into account that the instruction count of the merge sort algorithm the processor was tested on is 5409 apart from dual thread mergesort which is 5429, we can calculate other important values to judge processor performance. To calculate CPI we used the following equation:

$$CPI = \frac{Total\ Number\ of\ CPU\ Cycles}{Total\ Number\ of\ Instructions}$$

Since the CPI, clock speed, and the number of instructions per program is known, the time per program can be calculated using the iron law:

$$Execution\ time\ per\ program = \frac{CPI * Number\ of\ Instructions\ per\ Program}{CPU\ Clock\ Speed}$$

From total execution time, the average latency of each instruction can be calculated using this simple formula:

$$Average\ Latency = Number\ of\ Stages * \frac{Execution\ time\ per\ program}{Number\ of\ Instructions\ per\ Program}$$

As mentioned before number of instructions per program will always be 5409 apart from dual thread which is 5429 and the total number of CPU cycles and the CPU clock speed for each processor and RAM latency can be found in the table. The number of stages will be one for single cycle and five for pipelined, as there are five stages in the pipelined processor. The calculated values can be seen below.

Metric	**Single Cycle**	**Pipelined**	**Pipelined with Caches**	**Multicore with single thread**	**Multicore with dual thread**
FPGA Resources Utilized					
Combinational Functions	2875	3297	8342	15027	15027
Registers	1287	1768	3972	8823	8823
RAM Latency = 0					
CPU Clock (MHz)	30.68	63.37	57.94	53.67	53.67
CPU Cycles Taken	6906.5	9480.5	10215.5	17600.5	10835.5
CPI	1.28	1.75	1.88	3.25	2.00
Latency (ns)	41.62	138.29	162.98	303.14	185.94
Milliseconds/Program	0.2251	0.1496	0.1763	0.3279	0.2019
RAM Latency = 2					
CPU Clock (MHz)	29.10	62.97	55.71	51.42	51.42
CPU Cycles Taken	13814.5	18897.5	11156.5	18739.5	11854.5
CPI	2.55	3.49	2.06	3.46	2.18
Latency (ns)	87.76	277.41	185.12	336.88	212.33
Milliseconds/Program	0.4747	0.3001	0.2002	0.3644	0.2305
RAM Latency = 6					
CPU Clock (MHz)	29.10	62.97	55.71	51.42	51.42
CPU Cycles Taken	27628.5	37729.5	13139.5	21017.5	14033.5
CPI	5.11	6.98	2.43	3.89	2.58
Latency (ns)	175.53	553.86	218.02	377.83	251.35
Milliseconds/Program	0.9494	0.5992	0.2358	0.4087	0.2729
RAM Latency = 10					
CPU Clock (MHz)	29.10	62.97	55.71	51.42	51.42
CPU Cycles Taken	41442.5	56561.5	15111.5	23295.5	16098.5
CPI	7.66	10.46	2.79	4.31	2.97
Latency (ns)	263.29	830.31	250.74	418.79	288.34
Milliseconds/Program	1.4241	0.8982	0.2712	0.4530	0.3131

Table 1: Comparison of Single Cycle and Pipelined Processors with/without Cache and Multicore Processor with single/dual thread program

4 Conclusion

Given the metrics calculated, the pipelined processor's performance was superior to the single cycle processor's performance. Across all RAM latency values, the pipelined processor was over 1.5 times faster. Clock speed doubled even though CPI went up from the single cycle processor to the pipelined processor. When we look at the pipelined processor with caches, it did decrease in clock speed compared to the plain pipelined model, and is slower at low latencies, the CPI is vastly decreased at higher latencies thanks to caching, which provides an immense speedup, 2.54 times faster at latency 6. However, due to some coherence delays that occur in the implementation of multicore, it is actually slower than pipelined with caches in dual thread programs. The single thread program is even worse, as it is simply a pipelined single core cpu with caches while still having the overhead of coherence.

In conclusion, the most significant improvement of the processor came in implementing caches which helped reduce the affect of ram latency on CPI, and is the best CPU model at the mergesort benchmark task we used. Overall, this project demonstrated why caching is very vital in modern CPUs and how multicore processors need to be optimized to give the best performance.

5 Contributions

The original single cycle processor design that was built upon for the pipelined design was created by Geetha Prasuna Yarramneni. Geetha created the hazard unit and forwarding unit test bench, while Abhijay Achukola made the forwarding unit and the hazard unit testbench. Abhijay created the pipelining latches and Geetha helped connect them to the datapath. Geetha created Icache and Dcache testbench while Abhijay created Icache testbench and Dcache. Block and State Diagrams of Icache are done by Geetha and Dache are done by Abhijay. Bus controller is created by Geetha and Bus controller testbench is done by Abhijay. Dcache integration with coherence protocol is done by Abhijay and corresponding testbench is created by Geetha.