Processor Prototyping Lab

Final Report

1 Executive Overview

This report compares several major processor designs: single-cycle, pipelined without caches, pipelined with caches, multicore single-threaded, and multicore dual-threaded. The single-cycle processor executes one instruction per clock cycle, and its benefits are simplicity and hazard avoidance. The five-stage pipelined processor allows instructions to be in different stages of execution, allowing for higher throughput. It also allows for higher clock speeds because latches reduce the critical path. Adding split instruction and data caches to the design significantly reduces the amount of slow memory accesses, which speeds up the processor. Multicore processor designs allow parallel processing and task handling, increasing throughput.

The five designs are compared using several metrics: maximum clock frequency, execution time, cycles per instruction (CPI), instruction latency, FGPA resources, and speedup. The metrics are acquired by running the 'mergesort.asm' program on non-multicore designs and 'dual.mergesort.asm' on multicore designs. The data is analyzed by calculating the metrics listed above and comparing them across the five different processor designs. It is expected that the results will show successive increases in processor performance when progressing from the single-cycle to the multicore dual-threaded design. The caches are expected to provide a significant performance increase, especially at higher RAM latency.

The rest of this report discusses the caches and multicore processor design as well as the results from comparing all five processor designs. Section 2 covers the designs, and section 3 details the results. In addition, section 4 is a conclusion, and section 5 lists the contributions of the group members.

2 Processor Design

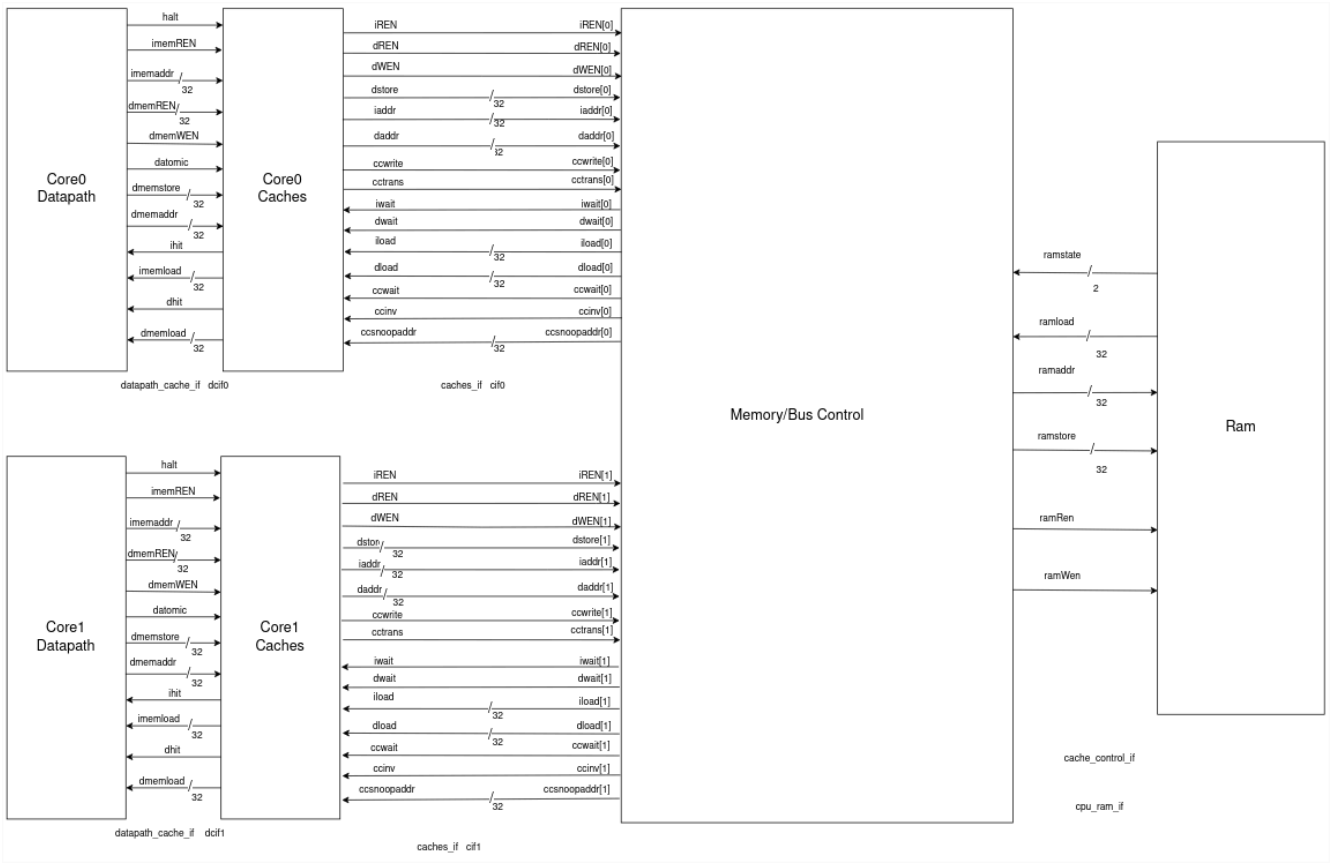

Figure 1: Multicore CPU Top Level Block Diagram

Figure 1 shows the top-level architecture of the multicore processor design. Each core has its own datapath and caches, and all necessary signals are sent to the memory/bus controller from both caches. For multicore, new coherence signals were added to the caches for cache coherence, and a ‘datomic’ signal for indicating a LR or SC instruction was added to both datapath and caches. The memory/bus controller arbitrates the requests from the cores and ensures that coherence protocol is followed. It handles data writebacks, cache-to-cache data transfers, ram-to-cache data transfers, snooping, and simple instruction fetching.

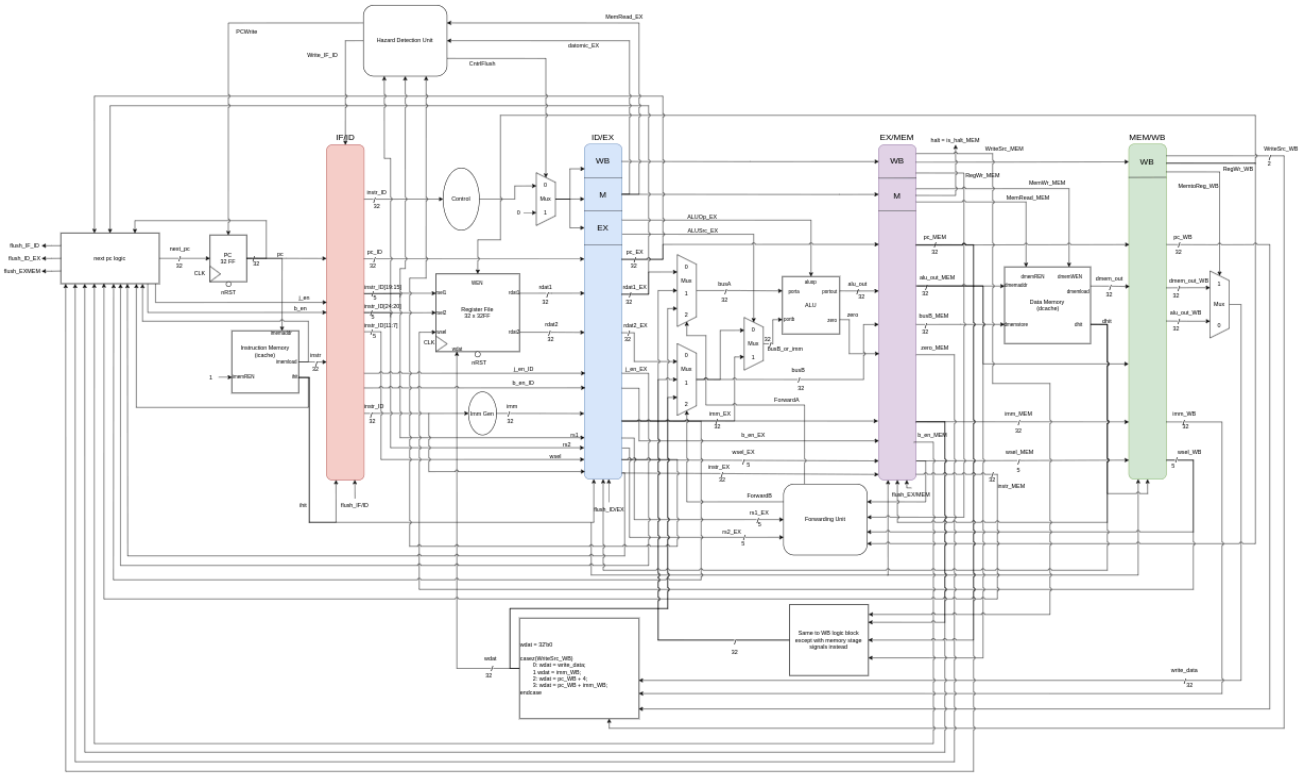

Figure 2: 5-stage Pipelined Datapath Block Diagram

Figure 2 shows the updated datapath for handling LR and SC instructions. The next PC logic had to be updated to handle the new instructions, and the control unit had to be updated to output a new ‘datomic’ signal that would be high when the instruction is LR or SC. The ‘datomic’ signal propagates through the design until the memory stage, at which point, it is sent to the dcache. In addition, when the ‘datomic’ signal is in the execute stage, it is sent to the hazard detection unit so that hazards introduced by LR and SC can be avoided. Those were the main changes required for the datapath since LR and SC can use a lot of the same hardware and control signals as load and store.

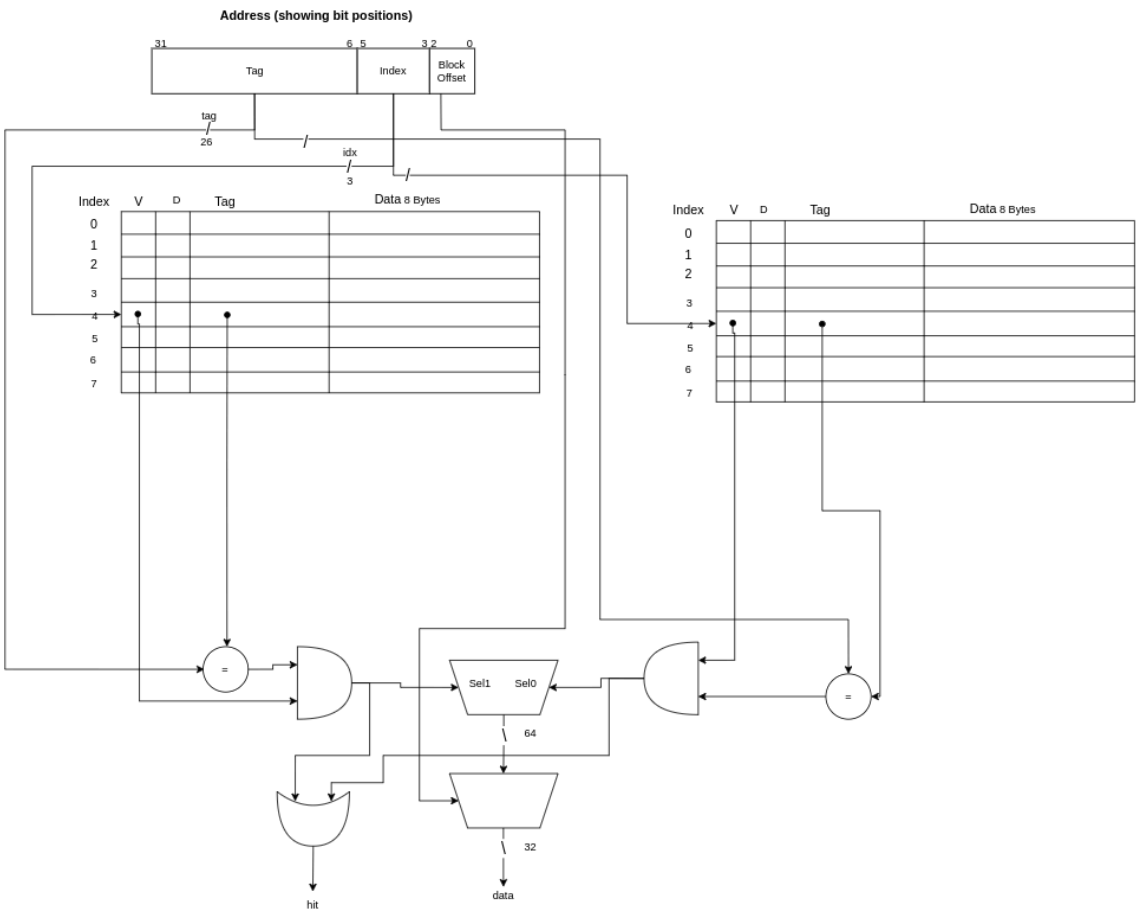

Figure 3: dcache Block Diagram

As shown in Figure 3, the dcache is two-way associative. It is 128 bytes in size, has two words per block, uses a write-back policy, allocates on a miss, and employs a least recently used replacement policy. Not shown in Figure 3, the dcache also has duplicate tags for snooping so that both the processor and memory/bus controller can access the information at the same time. The duplicate snoop tag is used for setting the ‘cctrans’ signal. The dcache is also responsible for handling the LR and SC instruction memory operations. Figure 5 shows the dcache state transitions. The states of SNOOP, WRITE5, and WRITE6 are a part of the snooping and cache-to-cache transfer implementation. WRITE3, WRITE4, and FLUSHED handle writing back dirty data in the cache on a halt. WRITE1 and WRITE

2 handle writing back dirty data on a miss, and ACCESS1 and ACCESS2 are for retrieving data from the bus. All hits are handled within the IDLE state. Lastly, Figure 4 shows the icache block diagram. The icache is a 64 byte, direct mapped cache that has one word per block and allocates on a miss. The icache is simple and only has two states of operation, an IDLE state for hits and an ACCESS state for retrieving data from the bus.

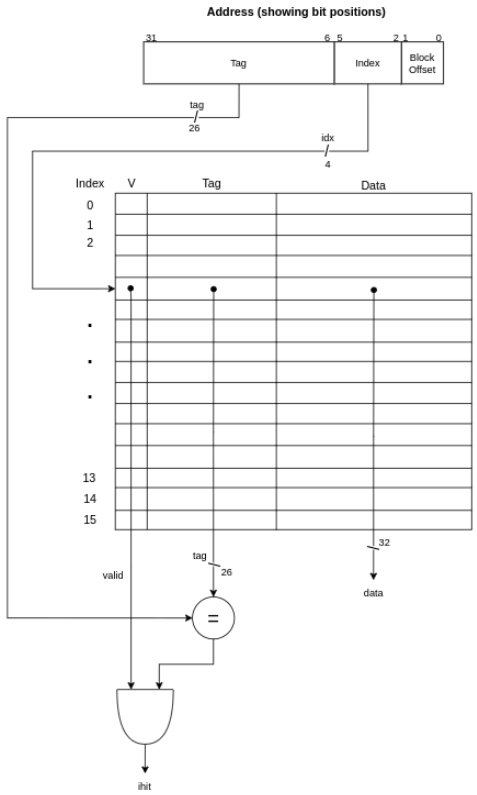

Figure 4: icache Block Diagram

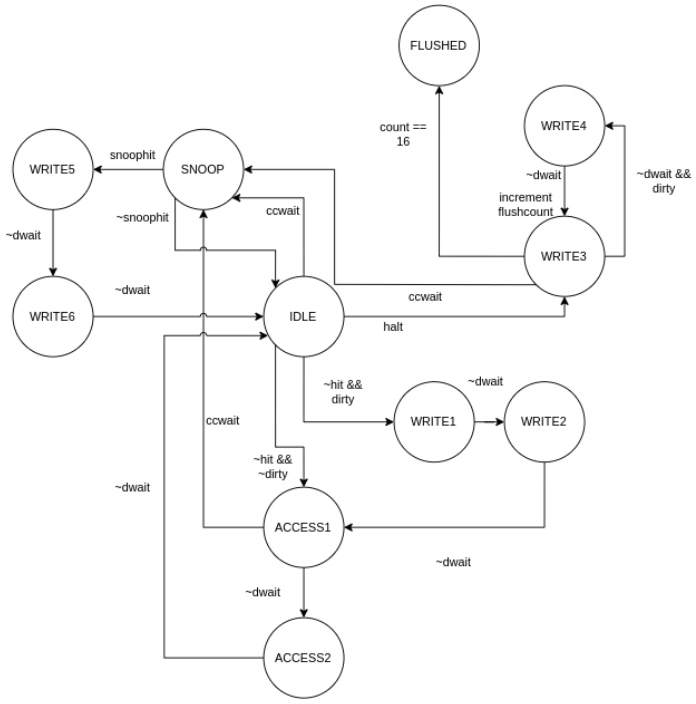

Figure 5: dcache State Transition Diagram

3 Results

As mentioned in the overview, the test program ‘mergesort.asm’ was used to collect performance data. Using the command ‘bash exec_data_sweep.sh’, which employed ‘mergesort.asm’, data was collected for both designs on ram latency zero, two, six, and ten. For the multicore processor, ‘dual.mergesort.asm’ was used instead. The command produces a sweep report file that contains clock frequency and number of cycles at each latency value which is shown in Table 1. The improvement in execution time of the processor, as it was periodically improved, is also visible in Table 1. Note that the max clock frequency for both designs is CPUCLK because it is the minimum of CPUCLK and CLK/2.

@ Lat 6	**Single Cycle**	**Pipelined (no caches)**	**Pipelined (caches)**	**Multicore (single-threaded)**	**Multicore (dual-threaded)**
Synthesis Freq (MHz)	34.43	63.6	63.39	62.38	62.38
CPI	5.108	6.455	1.949	2.186	3.542
Execution Time (s)	0.000802	0.000549	0.000166	0.000189	0.000128
Instruction Latency (s)	2.967E-8	5.075E-7	3.685E-7	4.201E-7	2.839E-7
Num Logic Elements	3197	3730	7721	16027	16027
Num Comb. Functions	2975	3594	6966	14647	14647
Dedicated Logic Registers	1293	1739	4371	8536	8536
Total Registers	1294	1740	4372	8536	8536
Speedup (adding caches)	—	—	230.18%	—	—
Speedup (seq. to parallel)	—	—	—	—	29.81 %

Table 1: Processor comparison at LAT 6

LAT	Single Cycle	Pipelined (no caches)	Pipelined (caches)	Multicore (single-threaded)	Multicore (dual-threaded)
0	0.000199	0.000153	0.000139	0.000163	0.000103
2	0.000401	0.000281	0.000146	0.000165	0.000107
6	0.000802	0.000548	0.000166	0.000189	0.000128
10	0.001203	0.000815	0.000185	0.000213	0.000149

Table 2: Execution Time (s) of each processor at LAT 0, 2, 6, and 10

To further compare performance of the two processor designs, execution time, cycles per instruction (CPI), instruction latency, and instruction throughput were calculated using the following locations:

Execution Time = Cycles / CPUCLK
Instruction Latency = (Execution Time / Number of Instructions)
CPI = Cycles / Number of Instructions
Instruction Throughput = 1 / Instruction Latency
Speedup % = ((Execution Time Slower / Execution Time Faster) - 1) * 100

The number of ‘mergesort.asm’ instructions, which is 5409, was obtained by running the ‘sim’ command after running ‘asm asmFiles/mergesort.asm’ and ‘make system.sim’. The number of ‘dual.mergesort.asm’ instructions was 2256 and was obtained the same way. The performance metrics calculated using the above equations are shown for the single-cycle design in Table 3 and the pipelined design in Table 4.

The results from the processor comparison show that architecture and design choices have a significant impact on performance, especially as memory latency increases. At LAT 6, the multicore dual-threaded design achieves the best execution time of 0.000128 seconds, with a CPI of 3.542, showing it can maintain high throughput although latency is compromised. In contrast, the single-cycle processor, while having the lowest instruction latency, performs the worst overall with an execution time of 0.000802 seconds and a CPI of 5.108. The pipelined design without caches performs worse than expected, with a higher CPI than even the single-cycle design, likely due to pipeline stalls and the absence of memory optimizations. However, pipelined design with caches significantly improves performance, reducing execution time to 0.000166 seconds at LAT 6 and achieving a speedup of nearly 4.8 times over the single-cycle processor.

As memory latency increases from LAT 0 to LAT 10, all processor types show increased execution time. However, the designs with caches and parallelism, such as pipelined with caches and multicore dual-threaded, are much less sensitive to this increase. For instance, while the single-cycle processor's execution time grows from 0.000199 seconds at LAT 0 to 0.001203 seconds at LAT 10, the dual-threaded multicore only grows from 0.000103 seconds to 0.000149 seconds over the same range. This indicates that cache usage and multithreading significantly mitigate the negative effects of memory latency.

In terms of hardware usage, the multicore designs use the most resources, with approximately 16,000 logic elements and over 8,500 registers. Despite the same hardware footprint, the dual-threaded version of the multicore processor outperforms the single-threaded version, highlighting the efficiency of using hardware multithreading. In summary, adding caches and multithreading capabilities results in substantial performance gains, making the multicore

dual-threaded design the best overall in terms of speedup and latency resilience, provided that hardware resources are available.

4 Conclusion

Through the results section it is clearly visible that adding an instruction and data cache to the processor speeds it up significantly. For latency 6 a 230.18% speedup is achieved. This is largely due to the fact that the long 6 cycle wait time to perform memory operations can be avoided using the temporal and spatial locality that the caches provide once a 32-bit value is already stored in the cache. Implementing a multicore processor and dual threading the mergesort algorithm also results in a 29.81% increase which is significant enough to warrant the implementation.

Another noteworthy detail is the increase in CPI when implementing the multicore processor. For latency 6, the CPI increased from 1.949 to 3.542. This is an acceptable loss since there are two cores executing instructions at once. The throughput of the processor is greatly increased although the CPI takes a hit. This lowered the overall execution time when compared to the single core processor.