Processor Prototyping Lab

Midterm Report

Joshua Hom

Section 1

Sharath Shivakumar

10/15/2023

1 Executive Overview

In this report, we are comparing a single-cycle and a pipelined processor. The major difference between the two processors is that the pipelined processor can take up to five instructions at a time due to it being split into fives stages. Whereas the single-cycle processor can only take one instruction at a time. In rough terms of the design, the pipelined version will be similar to the singlecycle except with stage latches, a hazard unit, a forward unit, and no request unit. These differences allow the pipelined processor to increase its instruction throughput, allowing for faster execution times. However, it will have a high CPI than the singlecycle and is not as easy to debug.

For comparing the designs, we used the mergesort program. This program was ideal because it tested all the possible hazards and combinations of instructions that would be handled differently between the two designs. To obtain data from the designs, we first synthesized them and ran the mergesort program on different latencies. From the simulation, we gathered the amount of cycles, the clock speed, and the instruction count. From the data, we can calculate the CPI and the execution time and compare.

2 Processor Design

Datapath

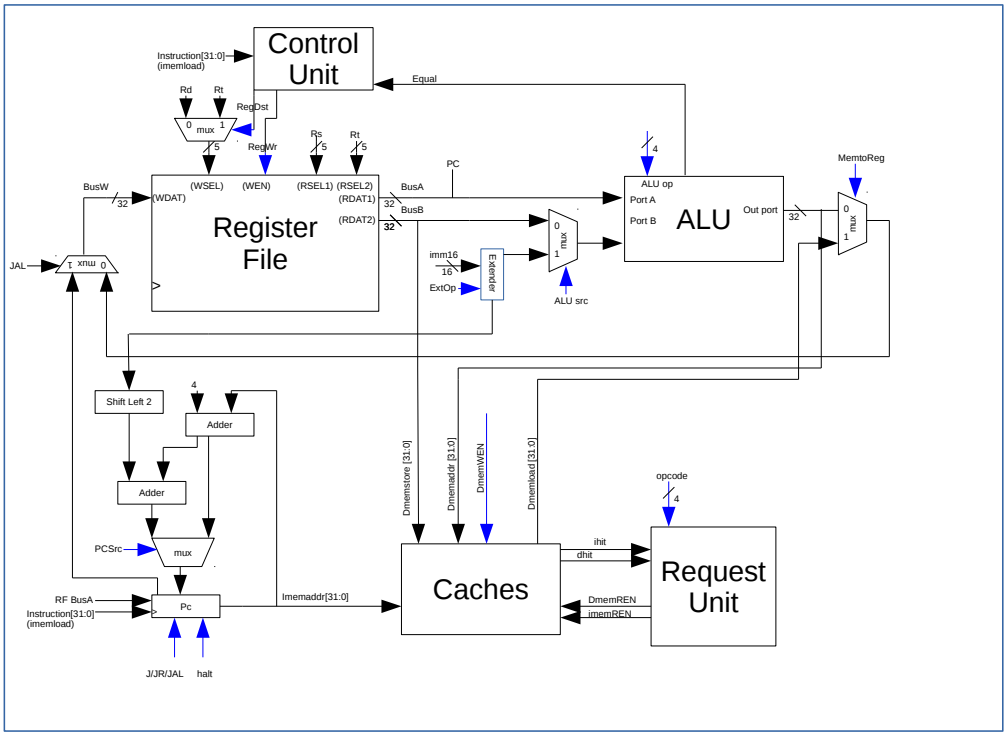

Figure 1: Single Cycle Block Diagram

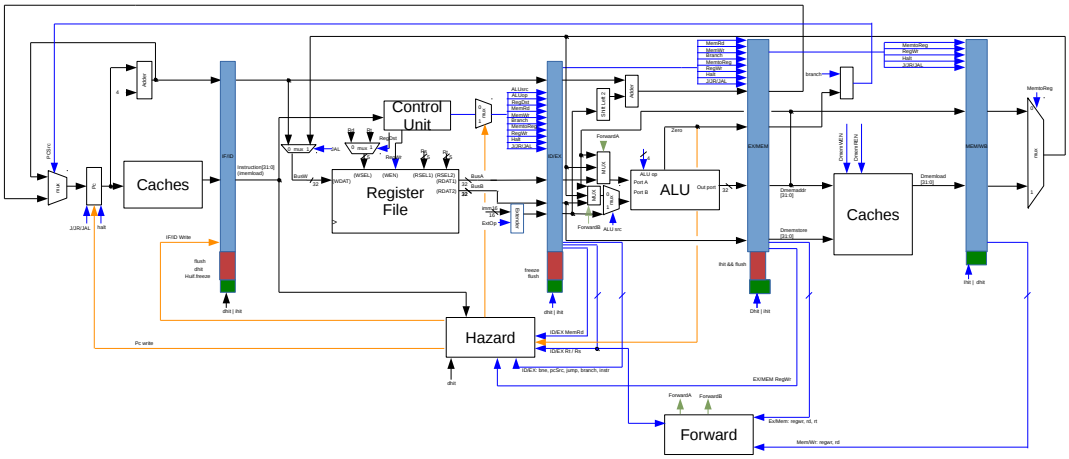

Figure 2: Pipeline Block Diagram

There are some important notes for pipelined design. Jumps and branches are executed in the memory stage. Forwarding is from writeback and memory to execute stage. Lastly, each latch moves on ihit or dhit with flush signals adjusted accordingly.

3 Results

RAM Latency	Max Frequency	CPI	Instruction Latency (µs)	Execution Time (µs)
0	30.935	1.277	0.0413	223.064
2	31.980	2.554	0.0799	431.598
6	32.475	5.108	0.1573	850.023
10	33.510	7.662	0.2287	1235.646

Table 1: Single-Cycle Results

Combinational Functions	Registers	Pins	Memory Bits
2,794	1,284	102	524,288

Table 2: Single-Cycle FPGA Resources

RAM Latency	Max Frequency	CPI	Instruction Latency (µs)	Execution Time (µs)
0	65.1	3.314	0.0418	275.092
2	65.46	3.830	0.0493	316.155
6	65.72	7.537	0.0944	619.753
10	65.98	11.244	0.1467	920.939

Table 3: Pipeline Results

Combinational Functions	Registers	Pins	Memory Bits
3,337	1,686	102	524,288

Table 4: Pipeline FPGA Resources

In order to obtain the max frequency, both the CLK/2 and CPUCLK frequencies were used. The final frequency would be the minimum of the two. The CPI was calculated by dividing the number of instructions by the amount of cycles which were divided by two. The instruction count and the number of cycles were obtained by running sim on the mergesort program. The execution time was calculated using the Iron Law. Multiplying the CPI by the Instruction count and dividing by the max frequency which was also divided by two. Finally, in order to obtain the instruction latency, we divided the execution time by the instruction count. The FPGA resources were from the "system summary" file after synthesis.

4 Conclusion

From the results we can see that in terms of execution time, the pipelined design performed better. It had about double the max frequency of the single-cycle design. However,

the piplined design did have a higher CPI, which is expected. This observation is evident throughout all latencies. We can also see that it required more combinational functions and registers. We reason this is because it is made of more components and logic than the single-cycle.

As you will notice from the processor specs, when transitioning to our pipelined design, we improve our throughput of instructions while not improving the latency. This means we increase the amount of instructions we process at the same time, creating an overlap. Although this design takes more cycles, were able to start and finish processing the instructions earlier, giving the appearance of a faster execution time.

5 Contributions

For the contributions from my partner, we used his single-cycle datapath design for the pipeline implementation. He created the instruction fetch/decode and decode/execute latch. He also created the rough draft code for the hazard unit and test bench for the fowarding unit.

My initial contributions for the pipelined design were the execute/memory and the memory/writeback latch. I also created the testbench for the hazard unit and the rough draft code for the fowarding unit. In addition to my inital contributions, I had to rewrite the pc, flush, hazard, and latch logic. All waveform debugging was also done by me.