ECE 437L Midterm Report

Course: ECE 437

Akshay Daftardar (mg226)

Shubham Rastogi (mg227)

Teaching Assistant: Mengchi Zhang

Due Date: October 14th, 2016

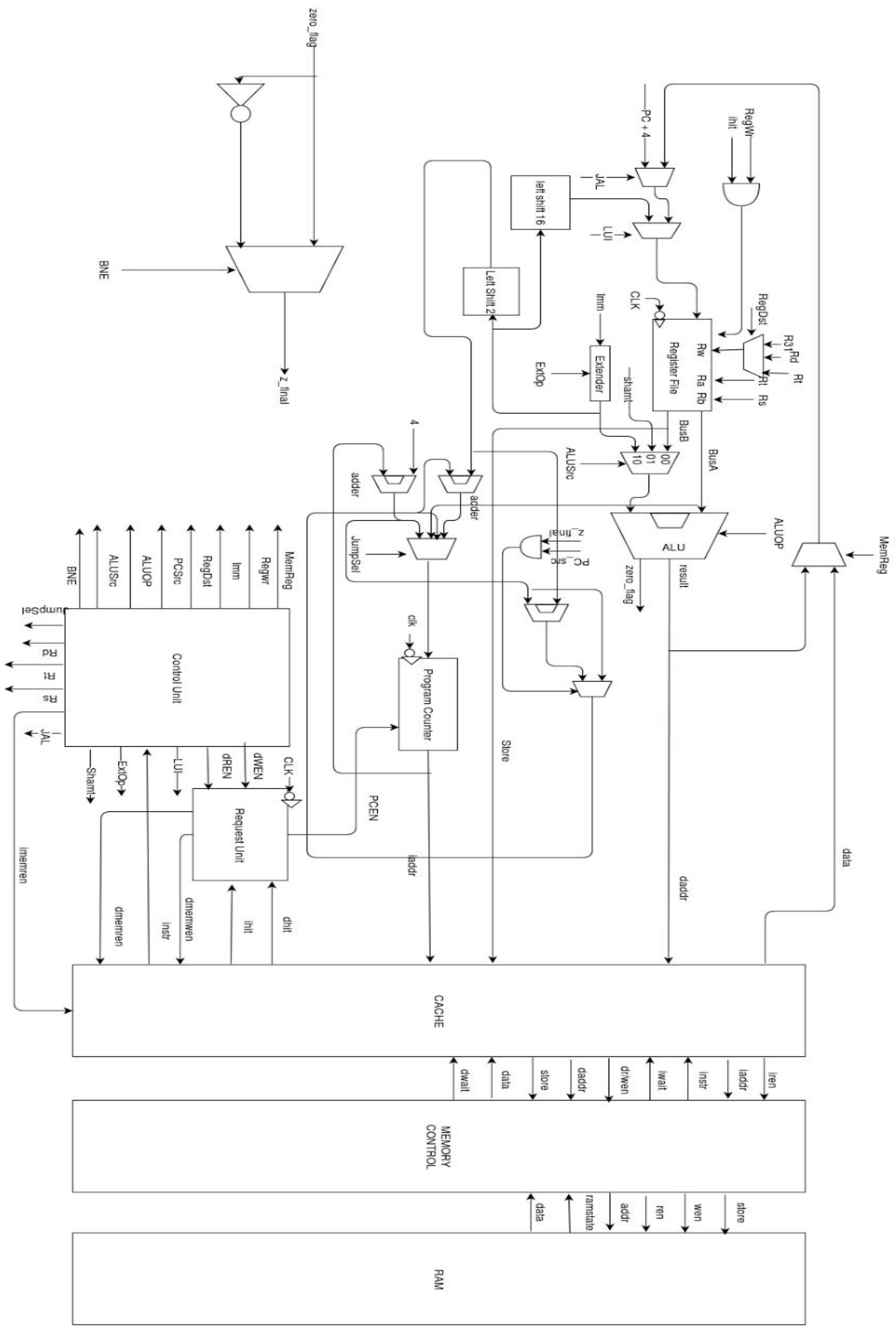
1. Overview:

Over the course of these two designs, we have been comparing the clock frequencies, latencies, critical paths, and MIPS of the processors. The benefits of the pipeline processor lie within its reduced latency and increased throughput, while the single cycle has a better CPI (Cycles Per Instruction) over a higher clock period. This allows the pipeline processor to have a higher clock frequency, making it comparatively faster than the single cycle. The single cycle processor has a less complicated datapath, and therefore has no risk of instruction overlap,. There is a need for more components in the pipeline design such as the pipeline registers and hardware to account for hazards, therefore increasing the overall area of the pipeline design.

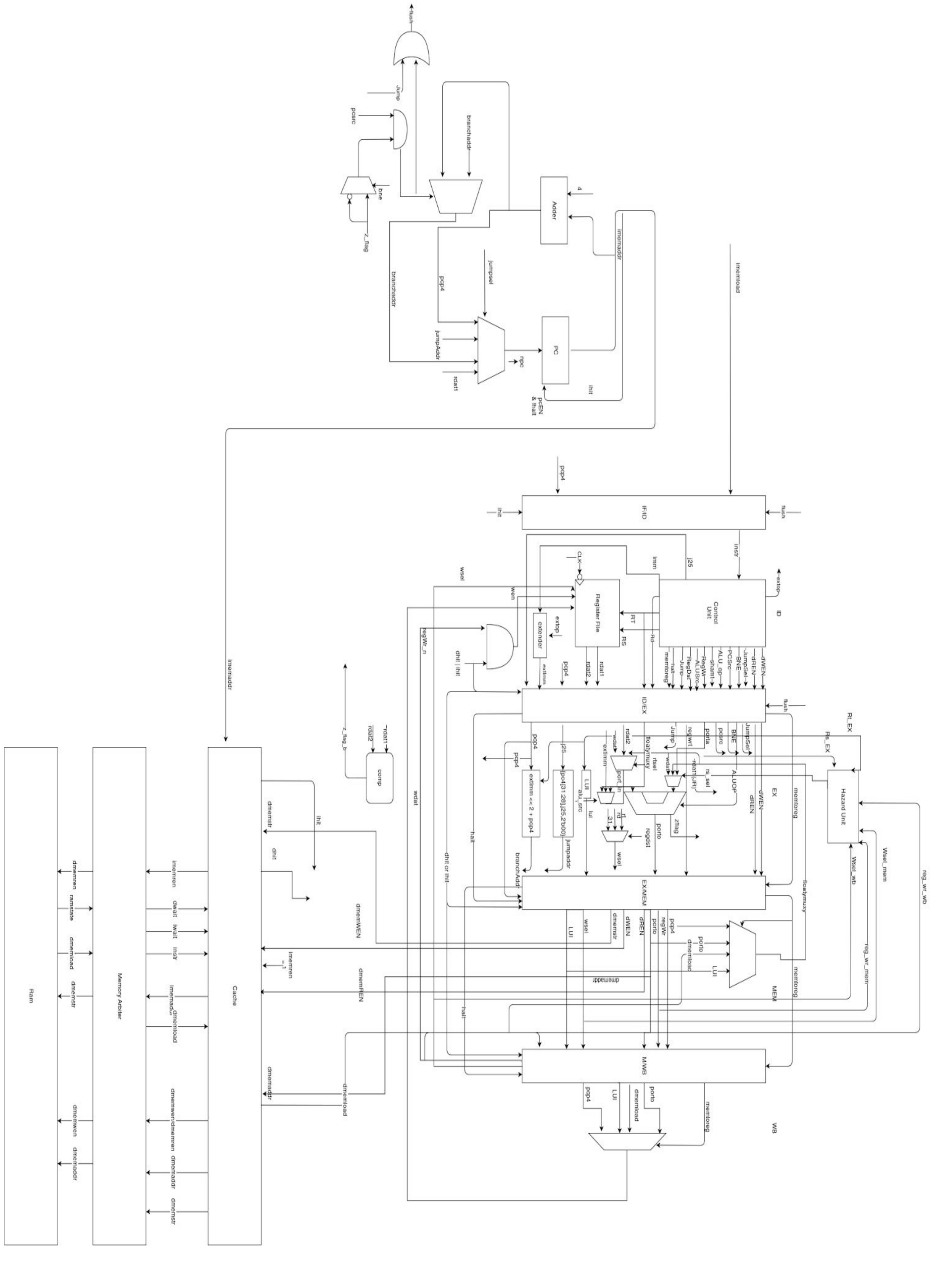
The benefit of the selected test program, mergesort.asm, is that it includes a myriad of instructions that test both the pipeline and the single cycle design to their fullest capacity. Mergesort is a complicated program with a mix of branches, jumps, calculations, loads, and stores in various combinations that thoroughly test a pipeline design. It contains a large amount of hazards that a pipeline design has to account for, and accounts for almost every kind of case that can arise while a pipeline processor is going to be used. This helps the designer of the pipeline processor test their design for any cases that they might have not accounted for. The single cycle processor is also tested for all kinds of instructions to make sure that it can handle any MIPS instruction without a problem. At the end of this study, the conclusion is that the pipeline processor is faster than the single cycle processor because of its increased throughput, decreased latency, and increased clock frequency.

2. Processor Design:

2.1 Single Cycle:



2.2 Pipeline:



3. Results:

	Pipeline	SingleCycle
Maximum Possible Frequency (MHz)	56.79 CPUCLK	37.2 CLK
Maximum Achieved Frequency (MHz)	100	55.56
Average Instruction per CLK Cycle	0.6223272434	0.782974402
Length of Critical Path (ns)	17.514	25.225
Latency of Instructions (s)	8.804e-8	2.68817e-8
MIPS of processor using Merge Sort	35.34	29.12
Total Logic Elements	2875	3189
Total Combinational Functions	2805	2890
Dedicated Logic Registers	1246	1310

Table 1: Single Cycle vs Pipeline

3.1 Result Source:

- **Maximum Possible Frequency (MHz):** This value was acquired from the system.log file after synthesizing both the pipeline design and the single cycle design.
- **Maximum Achieved Frequency (MHz):** To calculate this value, change the 'PERIOD' variable, in the system.tb file until the design breaks; which occurs when there is, either an infinite loop or the memory dump is incorrect.
- **Average Instruction per CLK Cycle:** To calculate this value, take the number of instructions from running the 'sim -t' command, and divide it by the number of CPU clock cycles.
- **Length of Critical Path:** This is a value that was acquired from the system.log file.
- **Latency of Instructions:** For the pipeline design, the latency is 5/Maximum Possible Frequency, because it takes 5 clock cycles for one instruction to go through the pipeline. The single cycle design has a latency of 1/Maximum Possible Frequency, because it takes 1 clock cycle for the instruction to get executed.
- **MIPS of processor using Merge Sort:** To calculate this value, divide the Average Instruction by the Maximum Possible Frequency in order to get the MIPS for each processor.

- **Total Logic Elements:** A value that was found in the system.log file.
- **Total Combinational Functions:** A value that was found in the system.log file.
- **Dedicated Logic Registers:** A value that was found in the system.log file.

3.2 Result Summary:

From this test program, one can tell that the maximum clock frequency for the pipeline design is greater than the single cycle design, therefore making the pipeline processor significantly faster than the single cycle processor. One can also see a reduced critical path and a reduced latency of instruction in the pipeline processor, which contributes to this speedup that is achieved by observing a higher MIPS for the pipeline design when compared to the single cycle design. Both designs were tested without any RAM latency.

4. Conclusion:

In conclusion, the pipeline design is faster when compared to the single cycle design. Theoretically, the pipeline processor should be 5 times faster than the single cycle processor, but, because of added combinational logic to account for hazards and the penalty suffered due to branches and jumps, the clock period slowed down or cycles were wasted. Comparing the critical paths of the pipeline and the single cycle, the pipeline has a critical path of around 17 ns, while the critical path of the single cycle is 25 ns. This shows that there is a larger latency in the single cycle design, which makes the clock frequency for this design lower, which results in the pipeline design being superior in terms of speed.

Because of the lower clock frequency of the single cycle design, The single cycle executes less intructions per unit time when compared to the pipeline design. The MIPS for the single cycle is 29.12, while the MIPS for the pipeline processor is 35.34. The pipeline can execute 6.22 million more instructions per second than the single cycle for the same program running on both designs. The reason for the speedup in the pipeline design is because the instructions are put through different stages of execution, therefore five instructions will be processed concurrently in different stages of execution during any clock cycle in a pipeline processor. This means that no hardware is idle, and along with a smaller critical path, results in faster instruction execution, when compared to a single cycle processor.

5. Contributions:

We started with our respective single cycle design and picked which design would be the best. We together then drafted what the pipeline design would look like, and we created the block diagram for it together. Shubham worked on the different register files and their respective interfaces, after which Akshay implemented the changes to the control unit and the datapath connections.

With hazards and forwarding, we decided to implement forwarding for lab 6, instead of writing the stall logic. Akshay redesigned the pipeline diagram and added the hazard and forwarding logic. Whereas Shubham implemented the hazard unit code and with Akshay worked on the forwarding logic.