ECE 437L Midterm Report

Pooja Kale (mg273)

Brian Rieder (mg267)

TA: Nick Pfister

October 14, 2016

1 Executive Overview

This report compares the performance of a single cycle processor and a pipeline processor on a merge-sort assembly (asm) file, and analyzes the results. The comparisons of the two processor designs will be based on the maximum clock frequencies, average instructions per clock cycle, the performance of the designs in MIPS and the FPGA resources required for each design. The comparisons will be further tested using variable latency RAM in order to ensure independence from memory timing. The single cycle processor is a simple design that takes in one instruction at a time to decode it and produce a result. This design does not have any instruction dependencies eliminating the possibility of instruction or data hazards. The pipeline processor is a more intricate design with five stages: instruction fetch, instruction decode, execute, memory, and write-back. This allows the design to process five instructions simultaneously within different stages of their execution.

The two designs will be tested using the merge-sort assembly file. The benefit of using the merge-sort assembly program, is it includes all possible assembly instructions that both processors can decode. The merge-sort file also includes multiple branches to test branch logic in both designs. The order of instructions in the program will test if the pipeline handles data hazards, and structural hazards with inserting stalls or bubbles in the correct places and flushing data at the right time. The execution results show that the pipeline processor is the favorable design because it increases the throughput (or decreases the execution time) as compared to the single cycle. While the single cycle handles one instruction at a time the pipeline processor design processes 5 instructions in different stages in one cycle.

2 Processor Design

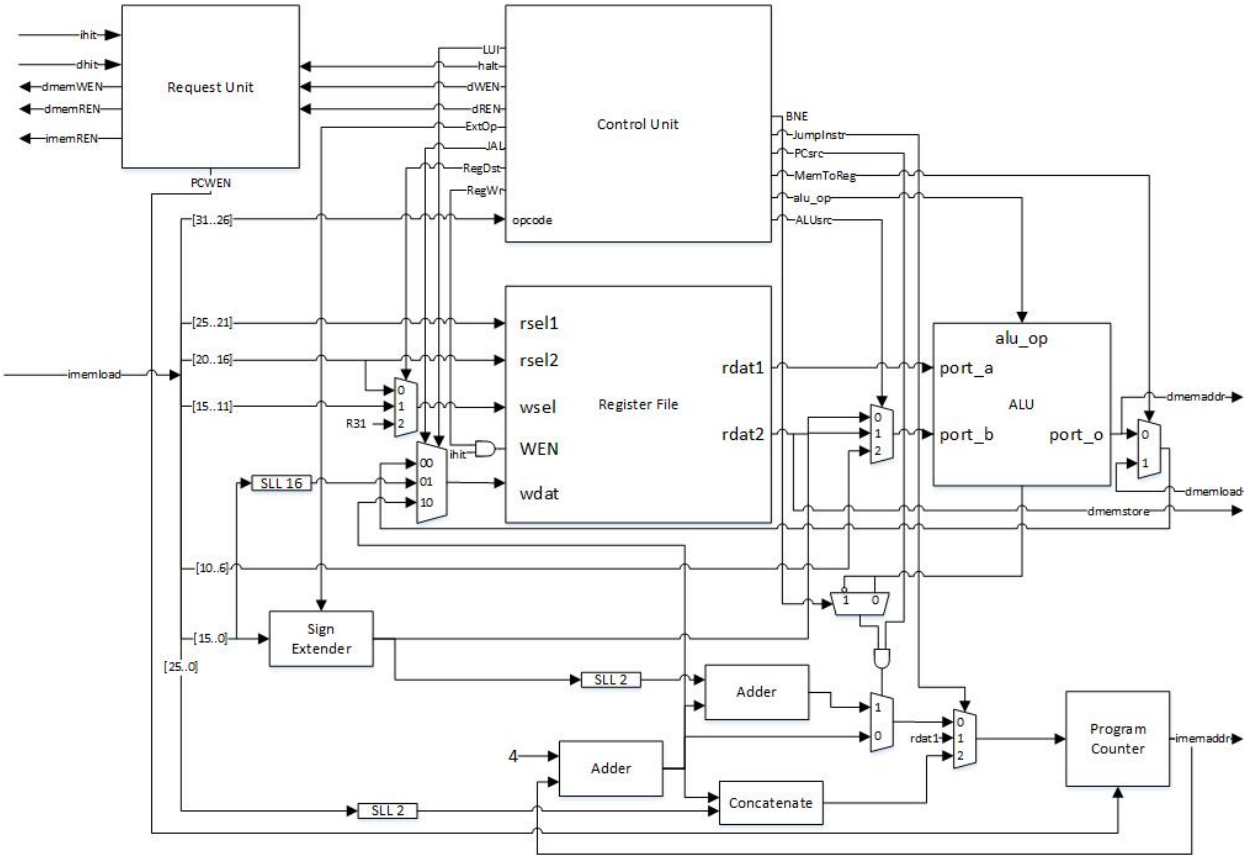

Figure 1: Single Cycle Processor Block Diagram

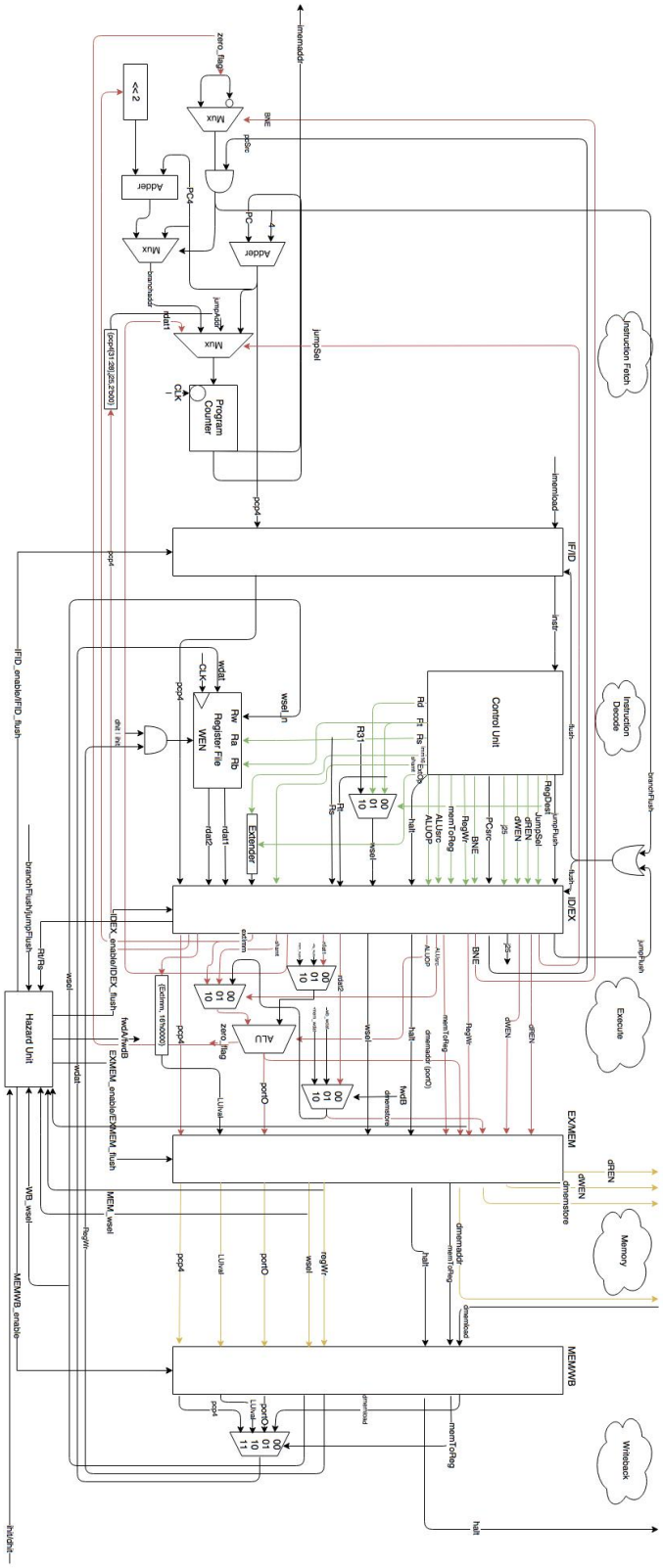

Figure 2: Pipelined Processor Block Diagram

3 Results

	Single Cycle	**Pipeline**
Maximum Possible Frequency (MHz)	37.24 (CLK/2)	48.88 (CPUCLK)
Highest Achieved Frequency (MHz)	55.56	100
Average Instructions per Clock Cycle (CPI)	1.277	1.608
Critical Path Length (ns)	26.939	20.774
Instruction Latency (ns)	26.85	102.29
Mergesort Number of Instructions	5,399	5,399
Mergesort Number of Cycles	13,791	17,365
MIPS	29.158	30.395
Total Utilized Combinational Functions	2,844	3,265
Utilized Dedicated Logic Registers	1,278	1,716

Table 1: Processor Specs

3.1 Sources of Result Values

- **Maximum Possible Frequency (MHz)** - Calculated by synthesis tools and retrieved from system.log
- **Highest Achieved Frequency (MHz)** - Determined through variation of PERIOD parameter within testbench/system_tb.sv
- **Average Instructions per Clock Cycle (CPI)** - Calculated using the total number of instructions retrieved from "sim -t" divided by half of the total number of cycles (displayed cycles are for the CLK instead of the CPUCLK) shown by "make system.sim"
- **Critical Path Length (ns)** - Calculated by synthesis tools and retrieved by system.log

- **Instruction Latency (ns)** - Calculated by multiplying period by a multiplier based on processor design - single cycle processors have a latency multiplier of 1 and a five-stage pipelined processor has a latency multiplier of 5 times
- **Mergesort Number of Instructions** - Determined using the total number of instructions output by "sim -t" when executing asmFiles/mergesort.asm
- **Mergesort Number of Cycles** - Determined using the total number of cycles output by "make system.sim" when executing asmFiles/mergesort.asm
- **MIPS** - Calculated using the inverse of CPI multiplied by the frequency of the processor
- **Total Utilized Combinational Functions** - Determined by synthesis tools and retrieved from system.log
- **Utilized Dedicated Logic Registers** - Determined by synthesis tools and retrieved from system.log

4 Conclusion

The performance results of running the single cycle processor and the pipeline processor on the merge sort algorithm show that the pipeline processor is the more powerful processor. The single cycle processor takes 1.2 million more instructions per second to produce the same result as the pipeline processor. The pipeline design also has a much shorter critical path of 20.77 nanoseconds as compared to single cycle processor which has a critical path of 26.939.

5 Contributions and Collaboration Strategies

The entirety of the pipeline design was done in tandem between Pooja and Brian. Early on in the process, the design was formulated together with all principles of the design being taken

into mutual consideration - all design decisions (for instance, where to resolve branches) were made together before any development began.

5.1 Individual Contributions

During development, certain tasks were partitioned between Pooja and Brian. Of the four latches that needed to be created for pipelining, two interfaces and their respective source files were generated by each team member - a direct split of all latch development. The process of developing the datapath was done together using a shared account and computer.

5.2 Collaboration Strategies

Account information was shared between team members at the beginning of the collaboration stages, but each team member did their development within their respective accounts and contributed to GitHub using their own account. Because of this Git usage and presence on GitHub, all contributions are documented in a quantitative manner within the repository.

5.2.1 Git VCS Usage

In order to compartmentalize development of features and to keep both contributors on the same page, an industry strategy of progressive-stability branching was employed and all development was done within respective branches and "work silos" - a master branch was maintained with the most recent functioning design, "singlecycle" and "pipeline" branches were maintained, and each feature were developed on an independent branch for that feature (some examples being "branchpredict" and "forwarding"). While this required more overhead in terms of planning and maintenance, the preservation of states and releases within the development process enabled ease of rollbacks, identification of changes, and the ability to develop without worrying about destruction of a prior working state.