Midterm Report

ECE 437: Computer Design and Prototyping

Stephen Bulley & Jun He

TA: Chuan Tan & Jiangshan Jing

October 14, 2016

Overview

This midterm report analyzes the performance of our our single-cycle processor and pipelined-processor designs. Both of these designs attempt to implement a simplified MIPS instruction set. Although these designs attempt to accomplish the same tasks, the pipeline design attempts to complete these tasks in a lower execution time than the single cycle implementation, but at the risk of introducing structural and data hazards. The idea of the pipeline is to increase the throughput of the processor without sacrificing CPI by utilizing different hardware in different steps as they are needed. The compartmentalization of the hardware increases the throughput, but also creates the risk of having incorrect data being used and data being overwritten when reading from RAM. Our design overcomes these obstacles by two units known as the Hazard Unit and the Forwarding Unit. These modules take care of different sorts of hazards that become apparent in our pipeline design.

The metrics that the report use to measure the relative performance of the processors are the maximum clock frequency, instructions per cycle, instruction latency, and so on. A merge sort algorithm was run on each design to acquire the above metrics. The merge sort algorithm is fairly complex and requires the use of most of the MIPS instructions that we attempted to implement as well as has the highest potential to invoke structural and data hazards. Finally, the looping nature of the algorithm gives it a very long run-time which will yield the best contrast in effectiveness of the two implementations. After looking at the data that we collected, our conclusion is that our pipeline design improves upon our single cycle design, but still leaves a lot of room for improvement. The pipeline design improves on the maximum clock frequency, CPI, runtime, instruction latency and the critical path. Unfortunately, the number of registers increased by over 600 to achieve this solution. In a space restricted environment, the pipeline processor may not be the best choice.

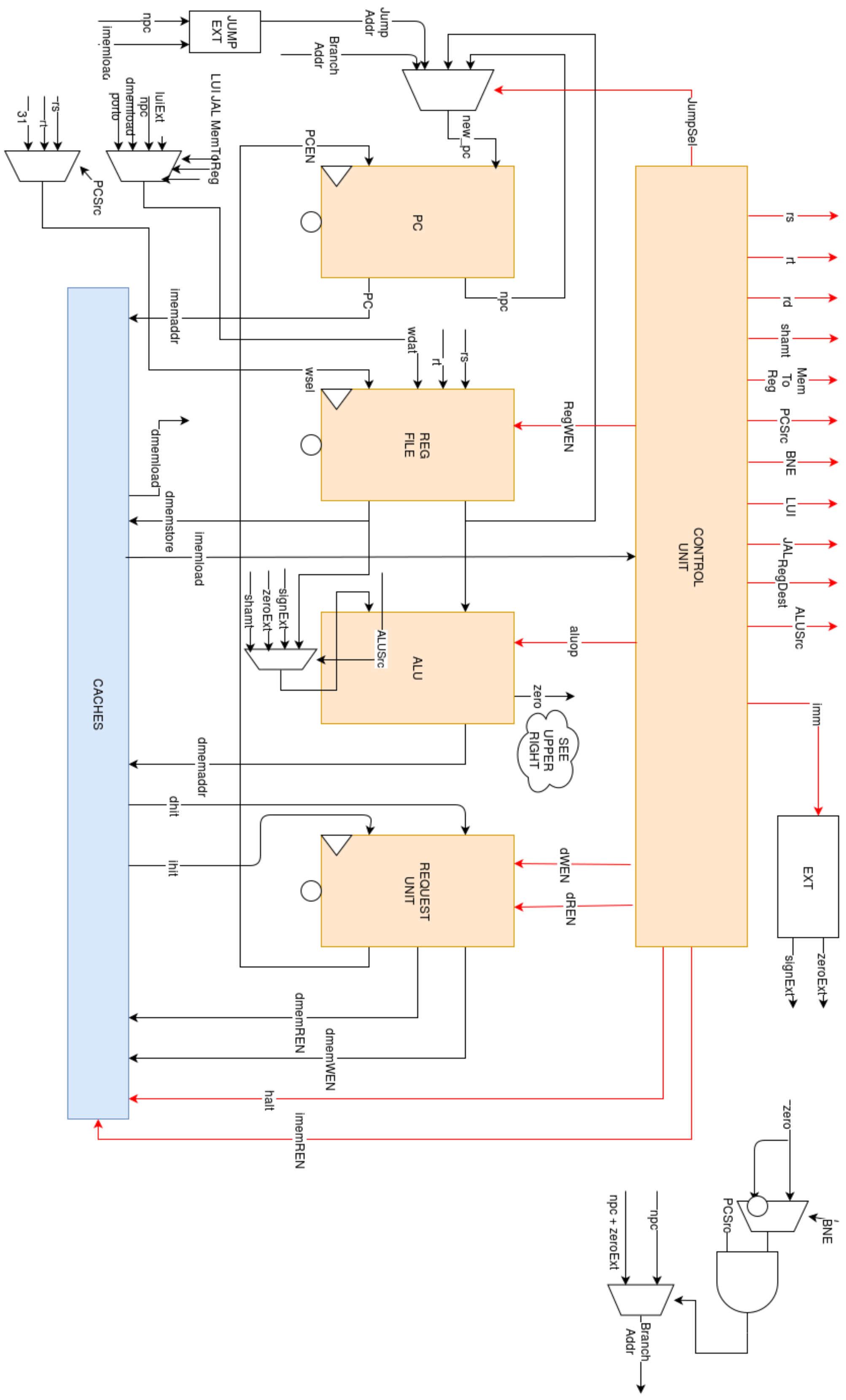

Figure 1: Single Cycle Schematic

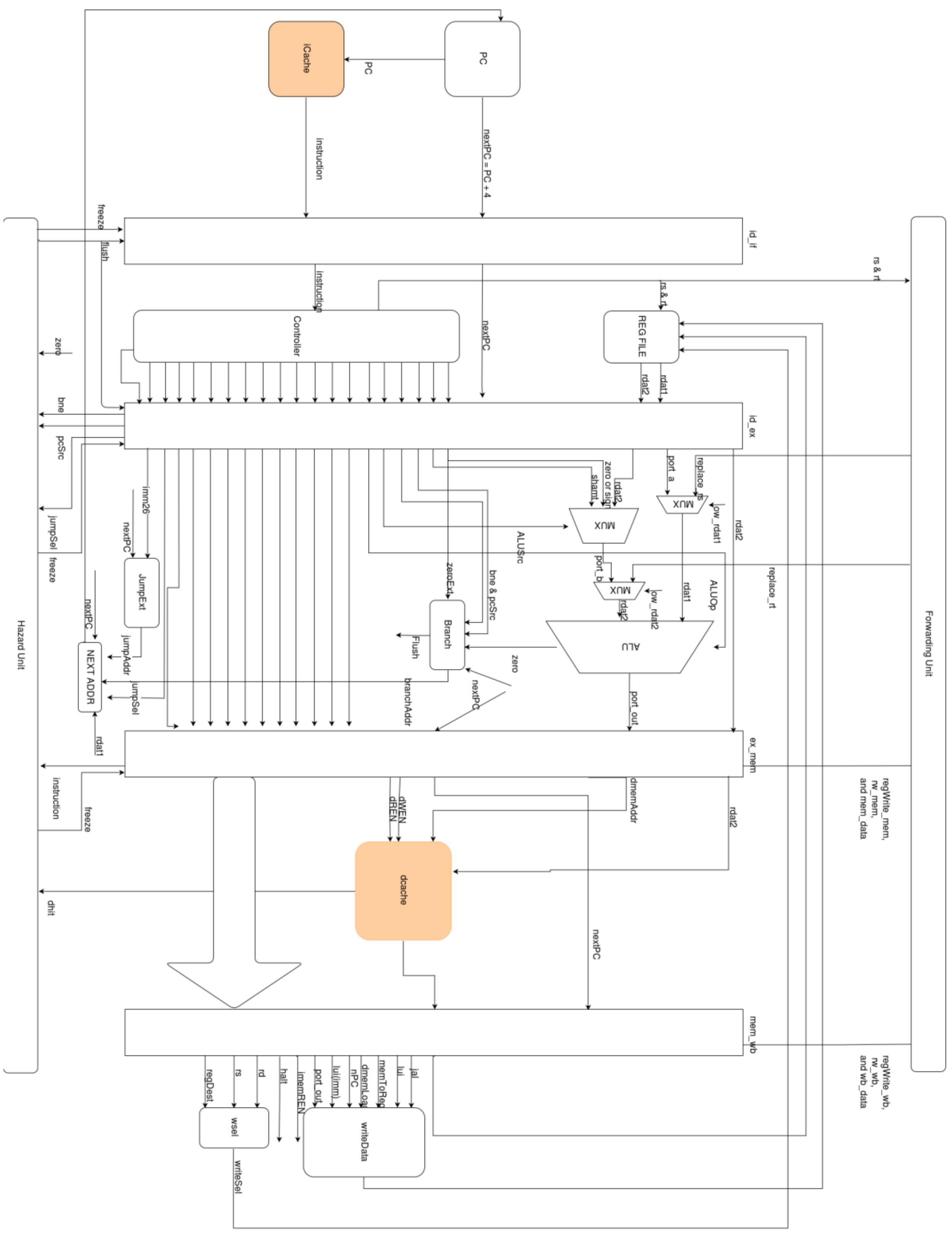

Figure 2: Pipeline Processor Schematic

Results

The results of our testing and simulations are as follows:

	Single Cycle	Pipelined	Units
Max Clock Frequency (logs)	36.625	45.52	MHz
Max Clock Frequency (tb)	50	83.33	MHz
IPC	0.130489426	0.3134397678	Instr./Cycle
MIPS	6.5243136118	15.6710786021	Millions of instr./sec
Latency of Instructions	27.3037542662	109.841827768	ns
Length of Critical Path	24.385	21.904	ns
FPGA Resources	1283	1856	Registers
Runtime	827520	344520	ns
Cycles	41375	17225	-
Instructions	5399	5399	-

Our max clock frequency was found by looking at the system.log file as well as testing different periods in the system testbench. The IPC was calculated simply by dividing the number of instructions by the number of cycles in one successful execution of the program. The MIPS was calculated as follows: (Instructions/Cycles) * (Cycles/(Runtime * 10^6)) * 10^9. This results in the Millions of instruction per second. The instruction latency was simply the inverse of the max frequency from the log file multiplied by 10^9. The critical path and the FPGA resources were acquired from the log file and the runtime, cycles, and instructions were found in the .syn simulation.

Conclusions

Overall, the pipeline's increase in speed is obvious. Every single category shows an improvement in the pipeline design except for FPGA Resources. This increase in required registers makes sense due to our more complicated design and our need to watch for hazards. Not only does our pipeline require a register for each signal moving into each next stage of the processor, but our design also has two additional modules: hazard and forwarding units. This sacrifice in required space is more than rational considering the huge increase in execution time. Although the number of registers grows by 1.5x the single cycle, the execution time drops by almost 60%. This increase in speed is a huge improvement. The improvement though can be attributed to a combination of factors. First, the maximum possible frequency increased as well as the instructions per cycle. This alone will increase speed. Additionally though, the critical path decreased by 2.5 ns. This is a huge percentage comparatively speaking.

In addition to the required registers, one might say that the instruction latency might be an issue, however, upon a closer look and a better understanding of a pipelined design, this increase is expected. The whole idea of the design is to increase throughput but compartmentalizing the processors resources. By breaking up the design, this will slow down the time it takes for each instruction to be processed. The individual delay however is negated due to the increase in clock speed and the splitting of resources.

Overall, our pipeline achieved what we wanted, however it shows that there is a long way to go. Our next step would be to build a branch predictor to better predict branches and whether or not to load instructions from the branch location or from the next chronological location. Additionally, parallel processing provides the potential to speed up our design even more. Just like the pipeline however, the likelihood of hazards increases exponentially with this design choice.

Team Contributions

Stephen Bulley – Conceptualized and planned pipelined design, wrote hazard unit and testbench, debugged, data collection for report, checked grammar and wording of midterm report

Jun He – Planned and coded part of the pipeline design, wrote forwarding unit and testbench, debugged forwarding unit, wrote branch prediction, wrote rough draft of the midterm report