Single Cycle vs. Pipeline Processor Comparison

ECE 437

Michael Baio(mg259)
Lucas Goedde(mg228)
Mengchi Zhang

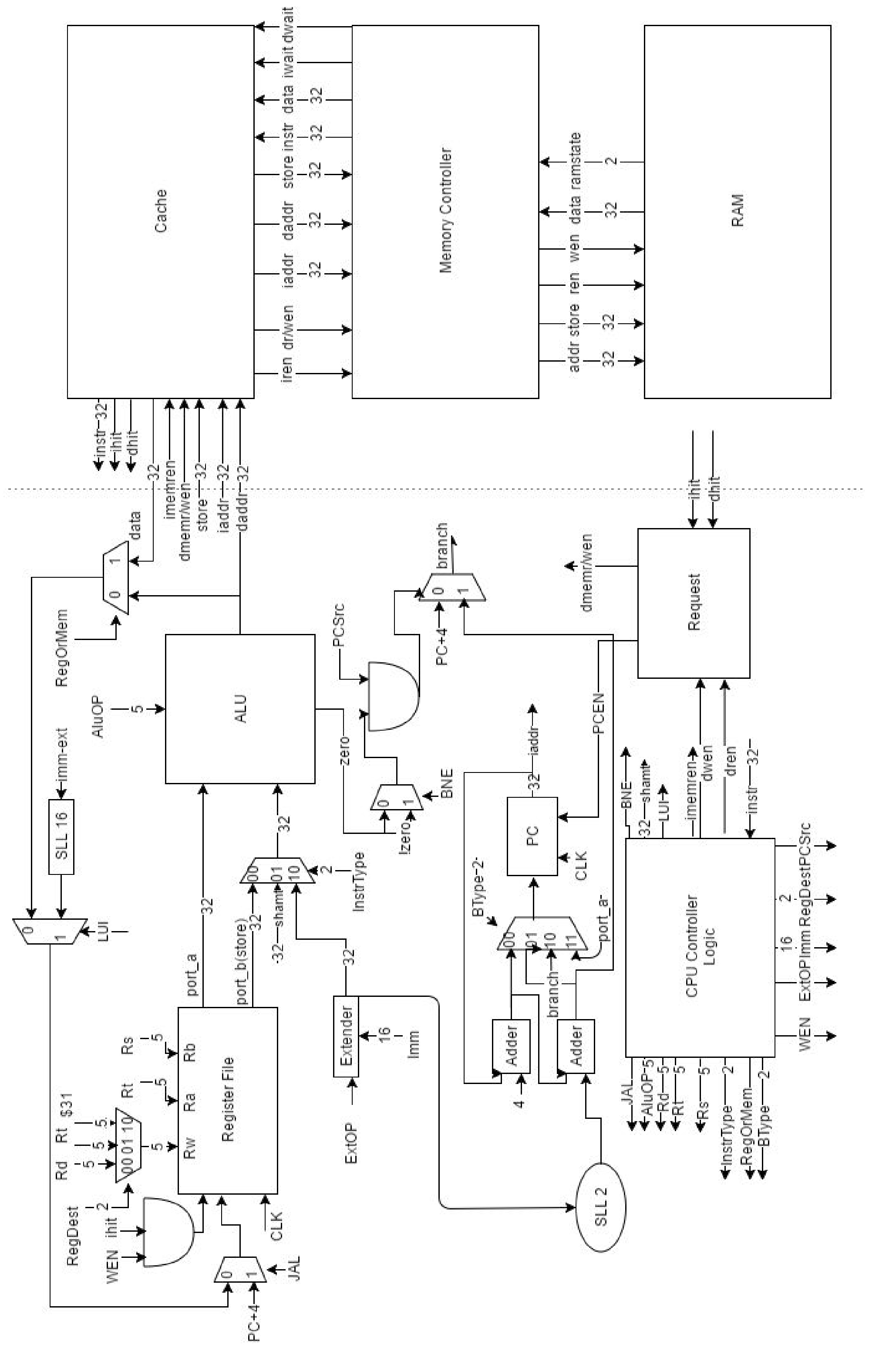
Friday October 14, 2016

1.0 Overview

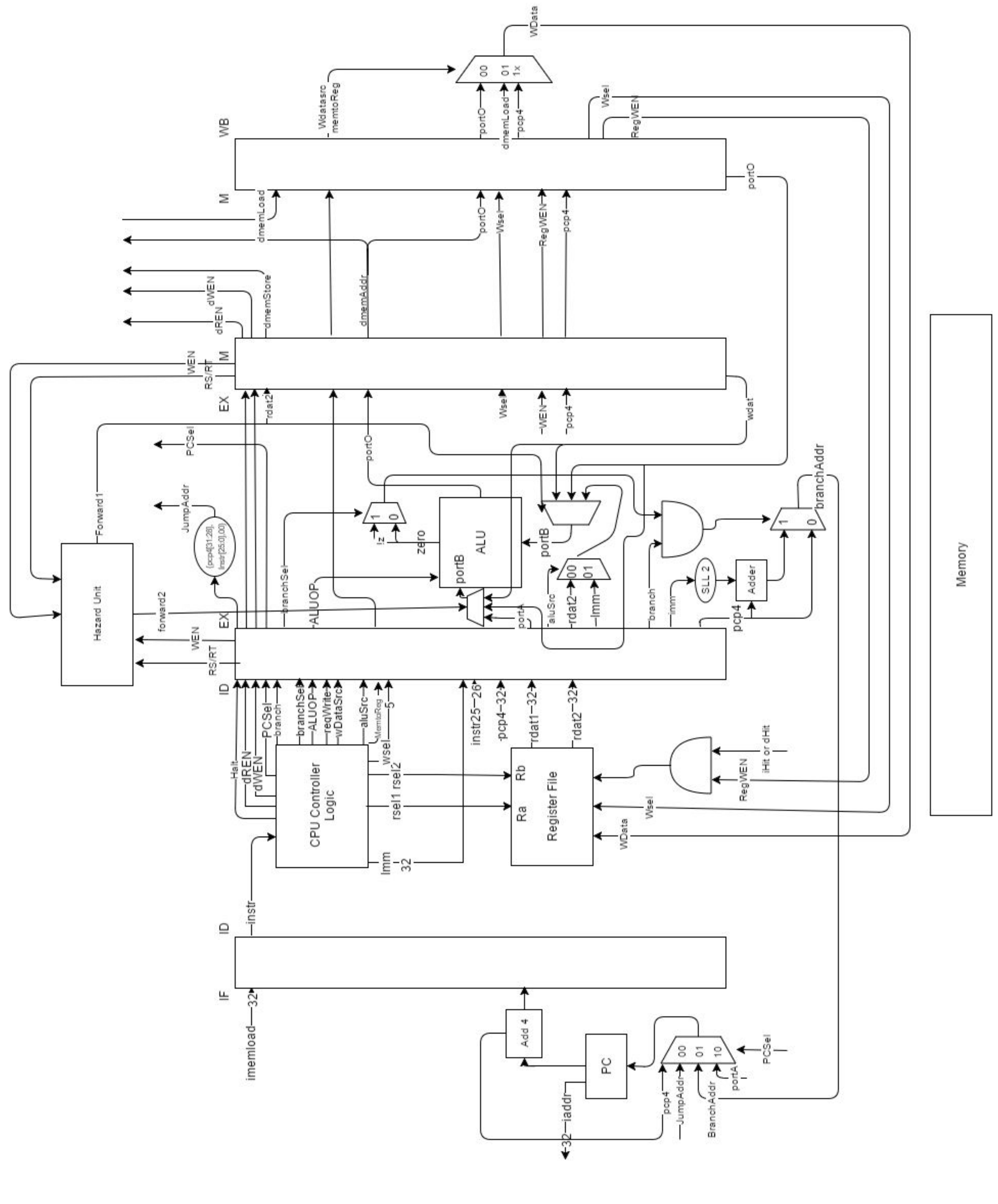
The purpose of this report is to compare various performance metrics for two different MIPS processor implementations: single cycle and pipeline. Both processors will run the same set of instructions, a mergesort algorithm, to compare clock frequency, period, hardware resources, critical paths, and a few others. Single cycle, with an inherently slow clock due to everything being executed in one long cycle, benefits from the fact that the overall implementation is simpler to program than pipeline. There is no need to have latches or additional control flow such as a hazard or forwarding units. However, pipeline uses more complexity to increase the throughput of the design. The pipeline design can take the long critical path from single cycle and break it up into smaller faster paths using pipeline latches.

The benefit of mergesort.asm test program is that it is complex enough to hit all of the longest critical paths and implements most of the functions of the processor instead of just doing one instruction over and over. After running the processors using the same mergesort program, data is gathered from the output of the simulation and from the log file that is generated when the processor is synthesized. Two of the main metrics to consider are MIPS and max frequency for the designs. From the table provided in the following pages, it can be concluded that the pipeline is marginally slower than single cycle. This seems counterintuitive at first but because of the way branch prediction works, the pipeline is slightly slower. The conclusion below will go into more detail about these comparisons.

2.0 Processor Design



Single cycle processor - Lucas Goedde



Pipeline processor - Lucas Goedde and Mike Baio

3.0 Results

Metric	**Single cycle**	**Pipeline**
Max Frequency - Possible(from log) - Achieved(in testbench)	 36.33 MHz(CLK/2) 50.00 MHz	 49.84 MHz(CPUCLK) 100 MHz
Instructions per Cycle (IPC)	.39	.233
Latency	27.52 ns	100.32 ns
MIPS	14.22	11.61
FPGA Resources - Combinational - Logic registers	 2,864 1,278	 3,351 1,677
Minimum CLK period from tb	20 ns	10 ns
Critical Path	26.546 ns	20.001 ns
Number of Instructions	5,399	5,399

3.1 Calculation Details

Memory Latency for both designs = 0

Max Frequency = min(CPUCLK, CLK/2) **Documentation is wrong**

IPC = # of Instructions / # of Cycles

Latency = period * 1 (for singlecycle) / period * 5 (for pipeline)

MIPS = IPC*(Cycles/sec)/1e6

3.2 Calculation Data

Max possible frequency - taken from system.log file created during synthesis

Max achieved frequency - derived from altering the PERIOD parameter in system_tb and using lowest possible value before design broke

FPGA resources - taken from system.log file created during synthesis

Critical Path - taken from system.log file created during synthesis

Number of instructions- taken from output of sim -t

Cycles and execution time - taken from output of make system.sim

4.0 Conclusions

The goal of the pipeline processor was to increase the throughput while also dealing with an increase in latency. From the table above, one can see that the latency definitely increased from 27 to 100 ns. The increase in clock frequency is also expected because the critical path is shortened, thus allowing the clock to run at a higher frequency.

Although pipelining is supposed to increase throughput of the design, it can be concluded that the pipeline processor is actually slower than single cycle. Currently the pipeline handles branching by flushing the first two registers when the branch is taken as the pipeline always assumes not taken before the branch is resolved in the execute stage. These flushes add up to a significant increase in cycles particularly because the test program, mergesort, contains a high amount of branches. The pipeline implementation would be able to reduce these flushes if it included a branch prediction table and utilized a strategy to reduce the number of misses. This reduction will reduce the number of cycles, and increase the overall performance of the pipeline. Although the clock speed did increase, the increase was not large enough to compensate for the increase in CPI due to branch miss penalties.

5.0 Contributions

Both Lucas and Michael contributed significant portions to the completion of the pipeline processor. Since Lucas' single cycle processor was slightly faster than Michael's the team decided to use that as the basis of the design. When beginning pipeline, Michael noticed that it would be easier to use his datapath and control_unit because they contained fewer overall signals to be passed between the pipeline latches. The rest of the design including ALU, register_file, memory_control, etc are from Lucas' single cycle. Both teammates split the work directly on lab 5 by having Michael create the latch interfaces and having Lucas tie all of the interfaces together through the datapath. Because lab 6 and 7 built on top of lab 5, the team worked more independently on these parts. Michael implemented the main part of hazard unit while Lucas added forwarding and debugged issues with correctness on the FPGA.