# Pipeline vs. Single Cycle Report

ECE 437

Javier Gonzalez Souto & Cyrus Sutaria

Yuhao

October 16th, 2015

## Overview:

The hardware that executes a program is called a processor; this report will contrast and compare the different types of processors that were implemented, single cycle and pipeline. Although, both have the same functionality, the main difference between them is noticeable in their performance, time taken to execute a program, and the complexity of each of them.

Processors fetch instructions from an executable program, they decode the information (read the registers and the data in them), execute basic operations (add, subtract, shift, and, or, etc), read from memory, and lastly, write the computed value into registers that way other instruction can gather the information computed on a previous instruction.

As stated above, both types of processors compared in this report have the same functionality; however, the single cycle processor does all of the steps for one instruction at a time. It fetches the instruction, decodes it, executes it, if it needs to read data from memory, it loads it or stores it, and lastly it writes the final result into the register destination. On the other hand, the pipeline processor is much faster as it splits the single cycle into different stages. While one instruction fetches, another may be on the decode stage, one in the execute stage, one in the memory and one can be on the write back stage. However, this type of processor has a few hazards that need to be taken care of. The only hazard taken care of is Read After Write (RAW), as the next instruction may not read a register when the previous one is writing to it. Other hazards are taken care of by the architectural design. Therefore after the execute stage, the output from that stage becomes the input to the other. For testing we decided to use the mergesort algorithm which uses most of the instructions available with and without dependencies which cause pipeline hazards. We chose a list size of 64 to ensure that even with many elements, the right computation takes place and it is loaded from and stored correctly in memory.

Furthermore, the pipeline processor is much faster and more complex than the single cycle.

## Processor Designs:

Single cycle:

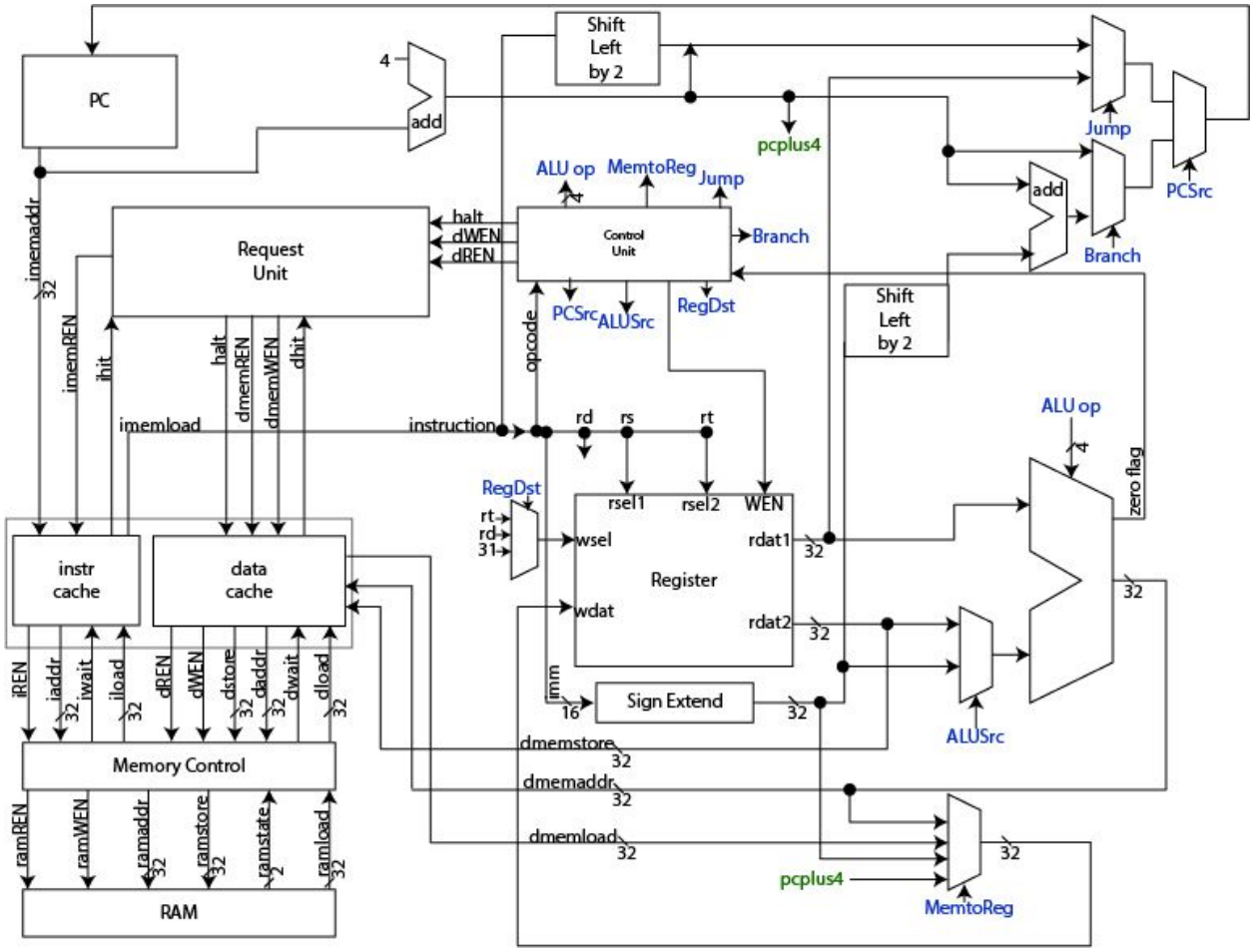


Fig A

Pipeline:

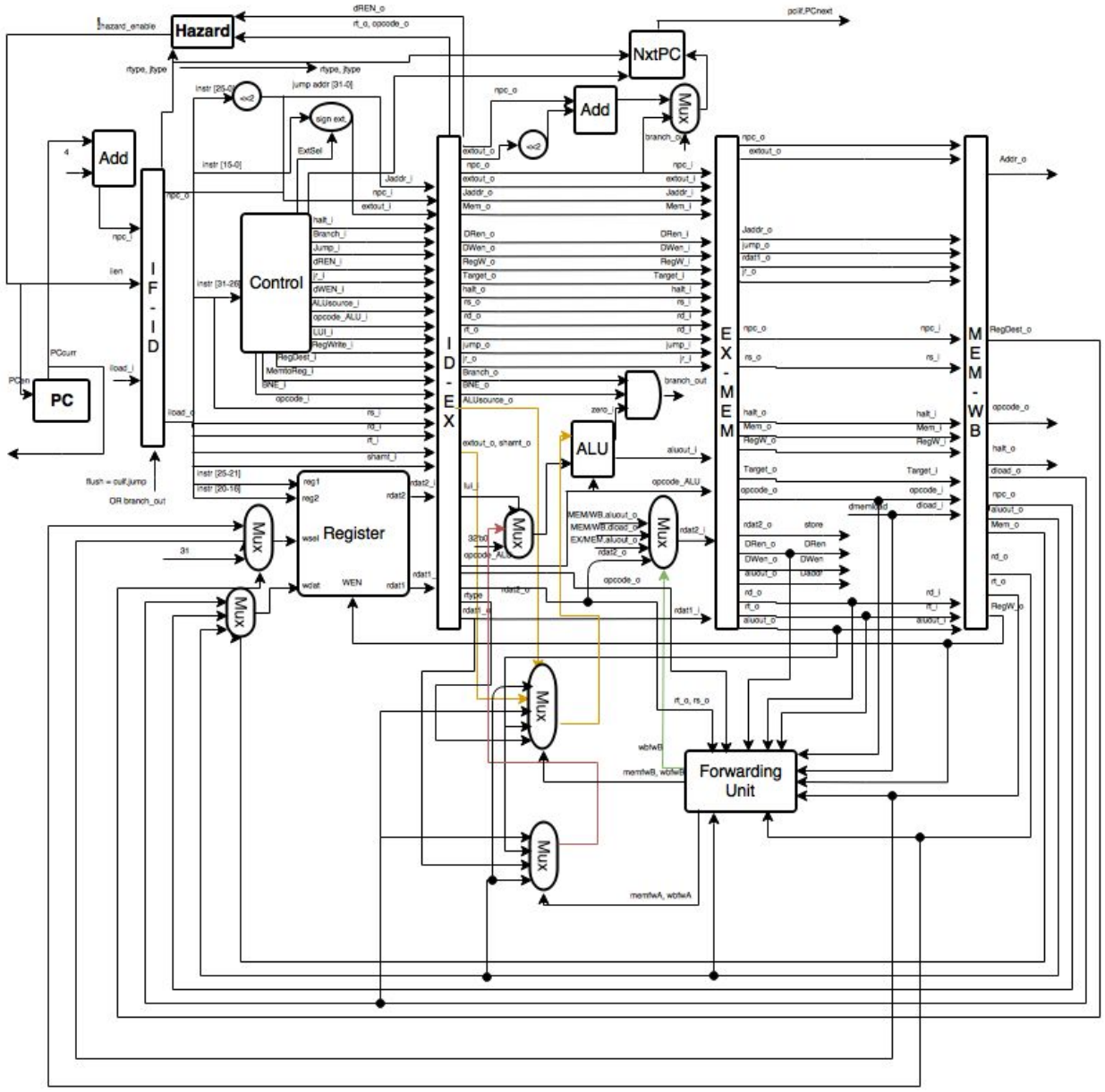


Fig B

## Results:

Single cycle: CPUCLK Fmax = 36.08 MHz, RAMCLK Fmax = 65.54 MHz

Pipeline: CPUCLK Fmax = 47.82 MHz, RAMCLK Fmax = 74.07 MHz

| | **Single cycle design:** | **Pipeline design:** |
|---|---|---|
| **Fmax** | 32.77 MHz | 37.035 MHz |
| **CPI** | 2.554 | 2.907 |
| **Latency** | 14 ns (calculated using CPUCLK Fmax) | 55 ns (calculated using CPUCLK Fmax) |
| **MIPS** | 13.98 (calculated using CPUCLK Fmax) | 15.48 (calculated using CPUCLK Fmax) |
| **Critical Path Setup Slack Worst Case** | 26.596 ns<br>from ram_block3a54~porta_we_reg<br>to pcif.address[13] | 8.494 ns<br>from idex.rt_o[0]<br>to pcif.PCcurr[26] |
| **Design Resources Used** | Total combinational functions: 2,826<br>Total registers: 1,278 | Total combinational functions: 3,193<br>Total registers: 1,714 |

## Conclusions:

There are a few details we can observe by looking at the results. The reason the calculated maximum frequency in the table is the minimum between the CPUCLK and half of the RAMCLK is that if the component has a lower maximum frequency than the other it would not be able to support the higher clock speed. The reason for dividing the RAM clock speed by two before comparing them is that the clock input to the RAM is at full speed and is divided by two for the CPU part of the design. The pipeline processor can be run at a higher frequency, and with the same workload can execute more instructions per second despite having a much higher latency. It has a shorter critical path despite being more complicated and using more logic than the single cycle design. These changes can all be explained fairly simply. Because the combinational logic for the pipelined processor is divided into stages separated by the pipeline latches we get shorter critical paths, and therefore our clock speed can be higher (as the period can be shorter). So then why is latency so much higher for the pipeline version of the design? Remember that with the single cycle processor there is one stage with one instruction executed at a time. The pipelined processor has five stages with one instruction in each, and each instruction passes through every stage. This means to find latency of an instruction we multiply the period by five.

The pipeline design was an increase in complexity from its single cycle counterpart. While the single cycle design used 2,826 combinational function sections of the fpga and 1,278 total registers, the pipeline increased these to 3,193 combinational function sections and 1,714 total registers. The single cycle design finished running the program in 193088 ns while the pipelined design only took 156940 ns. Despite the pipelined design having a higher complexity, CPI, and latency for each instruction, the increased throughput more than makes up for the difference. The pipelined processor ends up running the mergesort test program faster overall.