Superscalar MIPS Pipeline Processor

CpE 404/ECG 604 Final Project
May 9, 2019

Team # 2
Jett Guerrero
Chris Barr

Abstract

The purpose of a superscalar pipeline processor is to improve the execution time of a program. With superscalar processing, the processor is able to issue two instructions on the same clock cycle. Other advanced processors have multi-way processing such as 4-way or 6-way processing. This superscalar processor focuses on two-way processing which fetches two instructions at a time. The two instructions will be processed in parallel between the two lanes. Most of the components from the original pipeline processor are doubled to create the two lanes for processing. The hazard unit has been expanded to handle hazards from each lane and hazards between the two lanes.

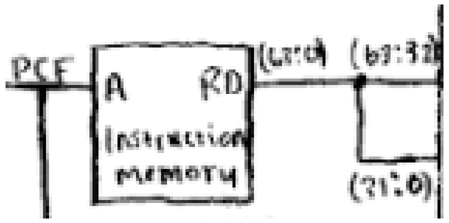
1. Introduction

The objective of the final project is to create and implement a two-way superscalar processor on the MIPS pipeline. With a superscalar MIPS pipeline, the throughput should increase since the processor can execute two instructions per clock cycle. Since the processor can process two instructions at the same time, the execution time becomes faster. With a superscalar processor, there are more components involved to create the two lanes for processing. To create the second lane for processing, the components from the first lane are doubled. The hazard unit needed to be expanded to handle the normal forwarding operations on both lanes and cross forwarding between the two lanes. The superscalar processor also will implement jump and branches. The DE2 board will be used to implement the superscalar processor onto hardware and provide verification that the processor performs correctly.

2. Implementation

Fetching Two Instructions:

To create a two-way superscalar, the Fetch stage needs to fetch two instructions at the same time. To fetch two instructions at the same time, the dmem module needed to be modified to have read data to be assigned RAM[a] as well as RAM[a+1] to capture the second instruction's address.



Since read data now holds two instructions addresses, read data needed to be expanded from 32 bits to 64 bits.

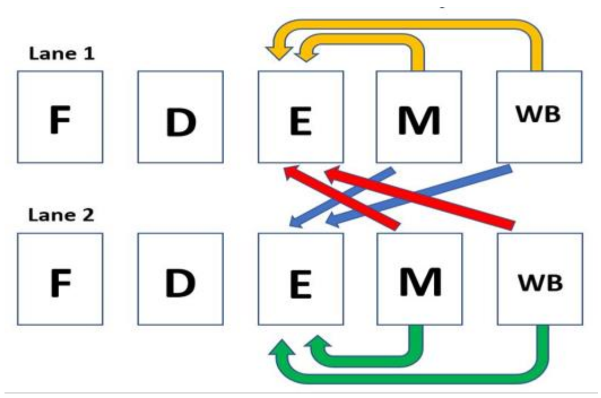
Components:

The majority of the pipeline data path needed to be doubled to accommodate the second instruction. There are now two Control Units, two sign-extend blocks, two shift left logical 2 blocks, two ALUs, eight forwarding multiplexers and the wires have doubled as well. Originally, there were four pipeline registers however, for modularity, it also has been doubled so there are now eight pipeline registers. There are four pipeline registers for the first lane and four pipeline registers for the second lane. In mipssingle_final.v, the code has been expanded by doubling the variables. Each variable such as memtoregD or writeregD for example, were duplicated and named to either 1 or 2 – memtoregD1 and memtoregD2. With the two Control Units, Control Unit 1 takes in the op code of bits 63:58 and Control Unit 2 takes in the op code of 31:26. Control Unit 1 takes in function code of bits 37:32 and Control Unit 2 takes in function code of bits 5:0. Control Unit 1 has seven outputs while Control Unit 2 only has six outputs because Control Unit 2 is not implementing branch.

Although most components have been doubled, a few of the other components were not doubled such as the components in the Fetch stage. There is still only one Instruction Memory block, one Register File block and one Data Memory block.

Data Dependency Hazards:

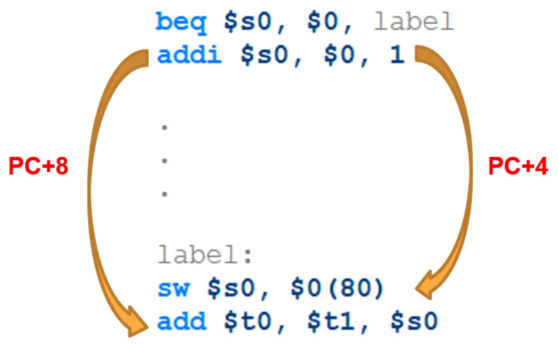
Since the processor has been super scaled, there will be new data dependency issues. To handle the new dependency issues, the hazard unit needed to be modified and expanded. Normal forwarding such as writeback stage to memory stage and writeback stage to execute stage still applies however, it also needed to be implemented on the second lane. Since forwarding can happen on each lane independent of the other lane, there are now a total of four forwarding multiplexers in the execute stage. Originally the forwarding multiplexers were three to one multiplexers, however, because of cross forwarding between the two lanes, it has been modified to be five to one multiplexers. Cross forwarding is necessary in a superscalar processor because depending on how the assembly program was written, there might be dependencies that need to be taken account for.



For example, if the value inside the destination register of the second instruction is needed for the execution of the first instruction of the next pair of instructions, then that value inside the destination register needs to be forwarded to the next pair of instruction's first instruction. Another data dependency example that the hazard can handle through cross forwarding is when the first instruction's destination register is needed by the first instruction of the next pair of instruction.

Branching and Jumps:

The superscalar processor handles branching and jump instructions. As the compiler, branching and jump only occurs as the first instruction. Only Control Unit 1 is programmed to handle branching and jumps while Control Unit 2 does not. For branching, when RD1 and RD2 are equal which is determined by the equal comparator, it is ANDed with BranchD1 to check if there is in fact a branch. If there is a branch, the output of the AND gate which pcsrcD1 is set to one. PcsrcD1 selects for the branching multiplexer and selects the pcbranchD wire which has the address to where we'll be branching to. So, if a 32-bit instruction comes into the decode stage, we will take the first 16 least significant bits, and sign extend it back to 32-bits. Afterwards, we will shift left logically by 2, because the last 2 bits are not imperative. Finally, we add 4 to this address which will then complete our pcbranchD wire. In the original pipeline, the default to pcsrc (when pcsrc is '0') will use pc+4, and the branching would also use the +4.



```
beq $s0, $0, label
addi $s0, $0, 1
.
.
.
label:
sw $s0, $0(80)
add $t0, $t1, $s0
```

However, in the superscalar model we use pc+8, and for the branching we still need to use pc+4. Branch needs to follow the convention for the MIPS architecture. So, it's naturally PC+4, which is why we can't just default it to our PC+8.

DE2 Hardware:

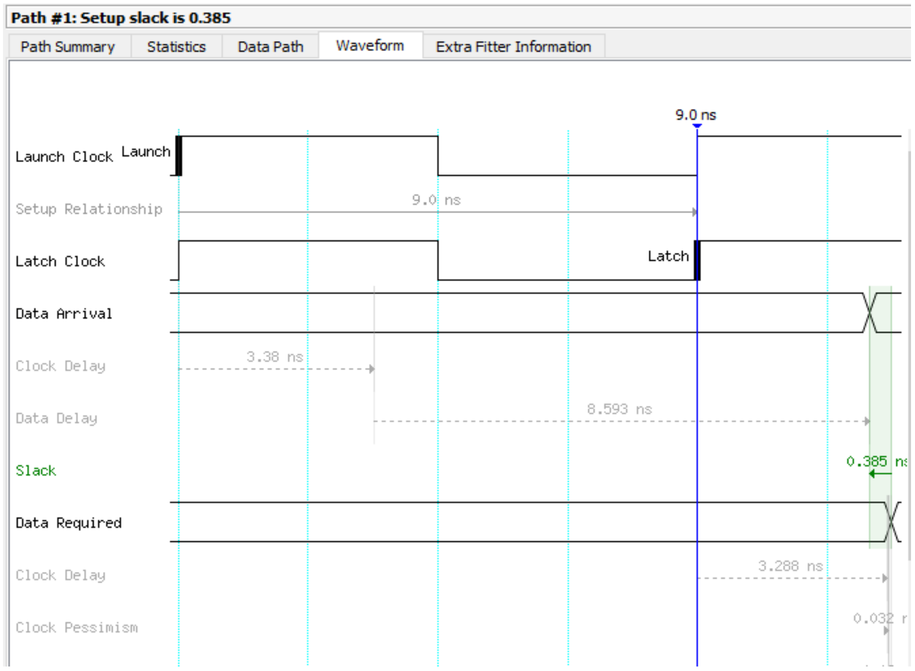
The superscalar processor program is compiled on Quartus and loaded onto the DE2 board. Before loading the superscalar processor program onto the DE2 board, Pins are assigned using the Pin Planner to assign the clock and reset to the switches and ResultW1 as well as ResultW2 to the red LEDs and green LEDs respectively. The superscalar processor program is then programmed onto the DE2 board. Once the DE2 has the program uploaded, the board can be tested following the ModelSim waveform simulation. The red LEDs display the result value of lane 1 while green LEDs display the result value of lane 2.

3. Experimental Evaluation / Results

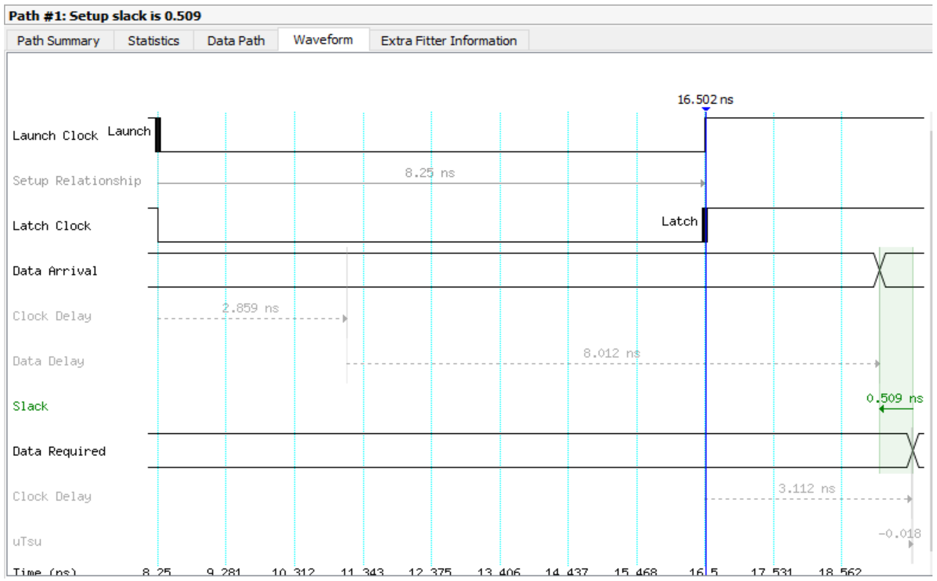
Timequest Timing Analysis:

We evaluated the systems power consumption by using Intel's Time Quest Analyzer on the Quartus II 13.1 software. From there, we were able to determine the frequency (or period) required for both the regular pipeline processor, as well as the superscalar pipeline processor.

Original Pipeline Processor:



Superscalar Pipeline Processor:



This is the result from Time Quest for both the original and superscalar processors. This shows us using a 9.0/16.5 ns period for the clk, and it gives a positive slack between the data arrival and data required. We can use these periods to calculate for the power consumption.

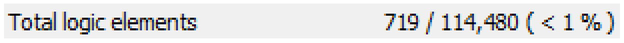
Power Performance:

First, we find the frequency for our superscalar system:

$$Frequency = \frac{1}{Time\ period}$$

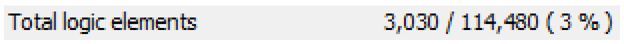
From the compilation report from Quartus, the area of the FPGA that is used is as follows:

Original Pipeline Processor:



Total logic elements	719 / 114,480 (< 1 %)

Superscalar Pipeline Processor:



Total logic elements	3,030 / 114,480 (3 %)

	ORIGINAL	SUPERSCALAR
Period (ns)	9.0	16.5
Frequency (MHz)	111.0	60.6
Transistor Blocks	719 (<1%)	3030 (3%)

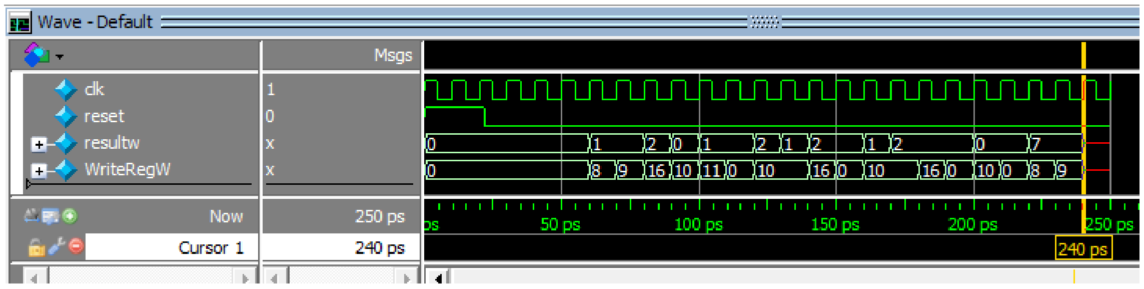
Now, we will find the comparison of the power consumption from the original processor to the superscalar one. We'll use the dynamic power equation:

$$\frac{P_{dynamic_S} = \cancel{\tfrac{1}{2}}C\cancel{V_{DD}}^2 f}{P_{dynamic_O} = \cancel{\tfrac{1}{2}}C\cancel{V_{DD}}^2 f} = \frac{Cf}{Cf} = \frac{3030 * 60.6}{719 * 111.0} = \text{2.3x more power}$$

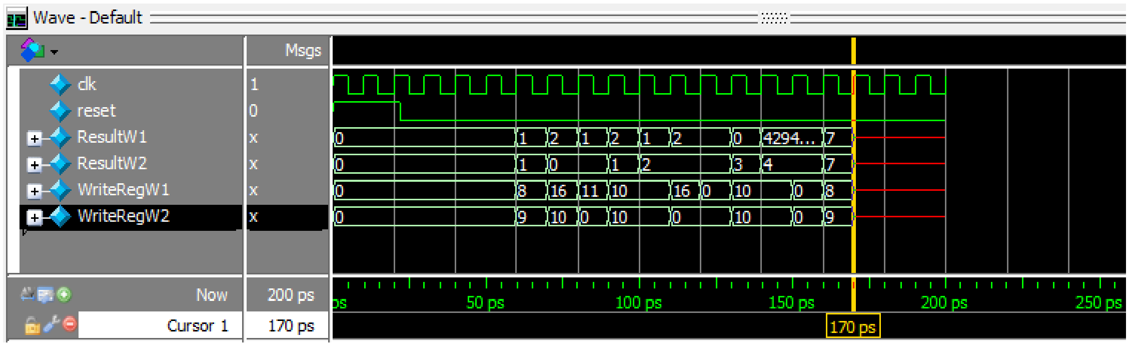
ModelSim Simulation:

To evaluate the performance of the superscalar pipeline processor, it can be compared to the original pipeline processors clock cycles per instruction (CPI). The CPI can be analyzed by running a simulation through ModelSim to test how many picoseconds (ps) it will take to run the same program.

Original Pipeline Processor Simulation:



Superscalar Pipeline Processor Simulation:



Total Run Time (Original) = 240 ps

Total Run Time (Superscalar) = 170 ps

Superscalar is 1.41x faster than the original pipeline processor

Execution Time = # instr x CPI x T_c

Execution Time (Original) = (12) * (1) * (9 ns) = 0.108 ms

Execution Time (Superscalar) = (12) * (0.5) * (16.5 ns) = 0.099 ms

DE2 Hardware:

Using the MIPS Assembly Code in the appendix, the board used four red/green LEDs to show the values of ResultW1 and ResultW2 in the pipeline. This proved that the program succeeded.

4. Conclusion

The superscalar processor developed as it was designed to do and did in fact make the execution time faster. Between the original pipeline processor and the superscalar processor, the original pipeline processor used less than one percent of the FPGA area of the DE2 board while the superscalar pipeline processor used about 3 percent of the FPGA area. Since the superscalar processor has twice the number of components compared to the original processor, the superscalar processor uses about 2.3 times more power by the dynamic power equation. The hazard unit also managed to handle the cross-forwarding data dependencies.

The main differences between the proposal and the final design is that the memfile that was originally intended to be used wasn’t. Because the original memfile used too many dependencies that weren’t going to be covered by the superscalar processor. Otherwise, the proposal matched the final design well.

Working and programming the superscalar processor allowed to be more familiarized and understand the pipeline processor. It’s a lot clearer on how instructions are handled with the program counter as well as the register file.

Future improvements would handle all types of data dependencies hazard - implementing methods such as Out of Order Processor and Register Renaming.