CMPE-260 Laboratory Project Part One

Pipelined MIPS Processor

By submitting this report, I attest that its contents are wholly my individual writing about this exercise and that they reflect the submitted code. I further acknowledge that permitted collaboration for this exercise consists only of discussions of concepts with course staff and fellow students; however, other than code provided by the instructor for this exercise, all code was developed by me.

Chris Abajian
Performed March 28, 2019
Submitted April 18, 2019

Lab Section 01L3
Instructor: Amlan Ganguly
TA: Piers Kwan
Sabrina Ly
Andrew Bear

Lecture Section 02
Professor: Richard Cliver

Abstract

A Full MIPS Pipelined processor was structurally assembled using previously created components. The five stages of a MIPS Pipeline were wired in Xilinx Vivado Design Suite in this order: Instruction Fetch, Instruction Decode, Execution, Memory, and Writeback. To test the processor, a series of independent instructions were loaded into the Instruction Memory and simulated. The results of the behavioral waveform proved correct functionality of the processor. To further test more complex, dependent instructions, a Fibonacci sequence generator was derived in RTL and the corresponding instruction set was implemented in Instruction Memory. The program was simulated, and the data memory module was analyzed for the Fibonacci sequence. The MIPS processor correctly produced the first 11 Fibonacci values and stored them in memory. The Full MIPS Pipeline worked as expected as evident by simulation results.

Design Methodology

This laboratory exercise sought to combine the MIPS pipeline stages previously created into one complete MIPS processor. The five stages are as follows: Instruction Memory Stage, Instruction Fetch Stage, Execution Stage, Memory Stage, and Writeback Stage. To test the processor, a set of pre-determined instructions must be hardcoded into the Instruction Memory component for execution. This exercise tested the general functionality of the processor by performing simple register arithmetic, as well as creating a Fibonacci sequence program. Table 1 shows the Register Transfer Language Fibonacci program pseudocode.

Table 1: Fibonacci Sequence RTL

Register Transfer Notation	
	R1 ← #1
	[R1] ← #1
	R1 ← #0
Jump	R2 ← [R1]
	R3 ← [R1 + #1]
	R4 ← R2 + R3
	[R1 + #2] ← R4
	R1 ← R1 + #1
	PC ← Jump

As shown in Table 1, the program is only 9 instructions long. This is before optimization, as well. An example of such optimization would be using a different register for counting other than R1 as it would already be set to zero, and the third instruction would be redundant. Either way,

because the Instruction Memory can hold up to 255 instructions, there was no need to further optimize the program.

Results and Analysis

The Full MIPS Pipeline was created in VHDL using modules previously created in earlier laboratory exercises. Because a MIPS processor only has two inputs (a clock and a reset signal), a testbench cannot be used to iterate over test cases and validate the results. Instead, the test cases must be provided in the instruction memory in the form of instructions. The instruction memory was populated with a sample set of instructions to verify the structural composition of the VHDL code. Figure 2 shows the behavioral simulation results for this instruction set.

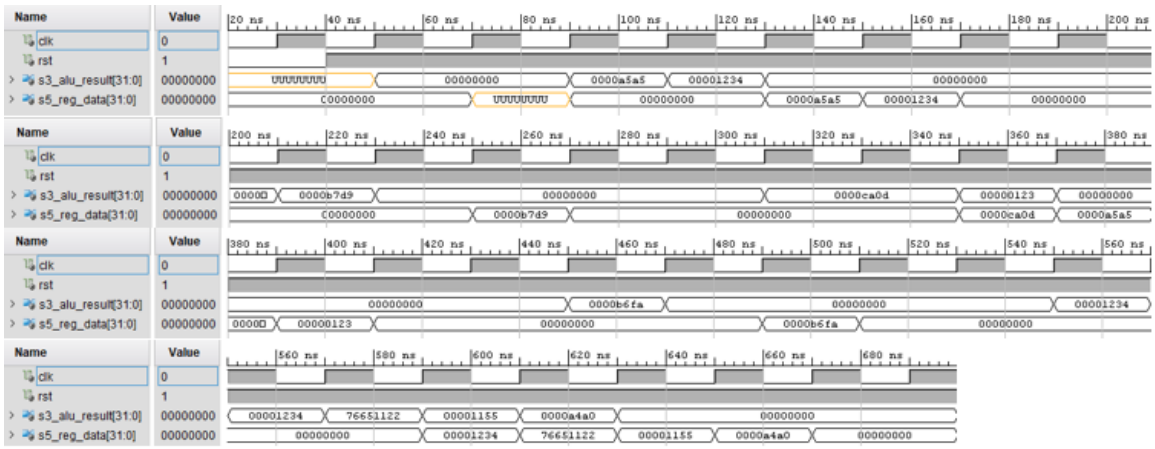

Figure 2: Testbench behavioral simulation waveforms.

As displayed in Figure 2, the Full MIPS Pipeline worked as expected. Verifying this involved looking at the components within every stage in the processor as instructions are fetched and observing the response. Because the stages are synchronous, this debugging process was made easier by stepping through the simulation in intervals of the clock period.

The Fibonacci program instruction set was placed in the Instruction Memory and simulated. Figure 3 in the Appendix shows the resulting waveforms. As seen in the `alu_result` and `reg_data` signals, the Fibonacci values were correctly computed. The first 11 values were correctly stored in memory, as well, as seen in the resulting array signals. When creating the instructions for the Instruction Memory, it was important to assign each address carefully as some instructions required an entire pipeline cycle to run before its result can be utilized. Specifically, any instruction that utilized registers must be spaced four clock cycles after the last modification of said registers. This is due to the Register File being the first stage after an instruction is fetched. If an instruction calls for a change in register value, it must compute this value and store the result back in the register file, which takes four clock cycles to update. This phenomenon is observed in Figure 3 where the value of the `alu_result` and `reg_data` signals are 0.

Conclusion

The Full MIPS Pipeline was pieced together using individual components of previously implemented stages. To verify correct functionality of the processor, a sample set of generic operational instructions were loaded into Instruction Memory and simulated. The resulting waveforms were observed and verified by the ALU output and Writeback stage signals and showed correct functionality of the processor. To further test, a Fibonacci sequence creator was derived and placed into the Instruction Memory. After simulations, it was verified that the correct values were produced and stored by the processor. The Fibonacci sequence program also allowed further investigation into timing requirements, such as waiting four clock cycles before a register value is updated.

Appendix

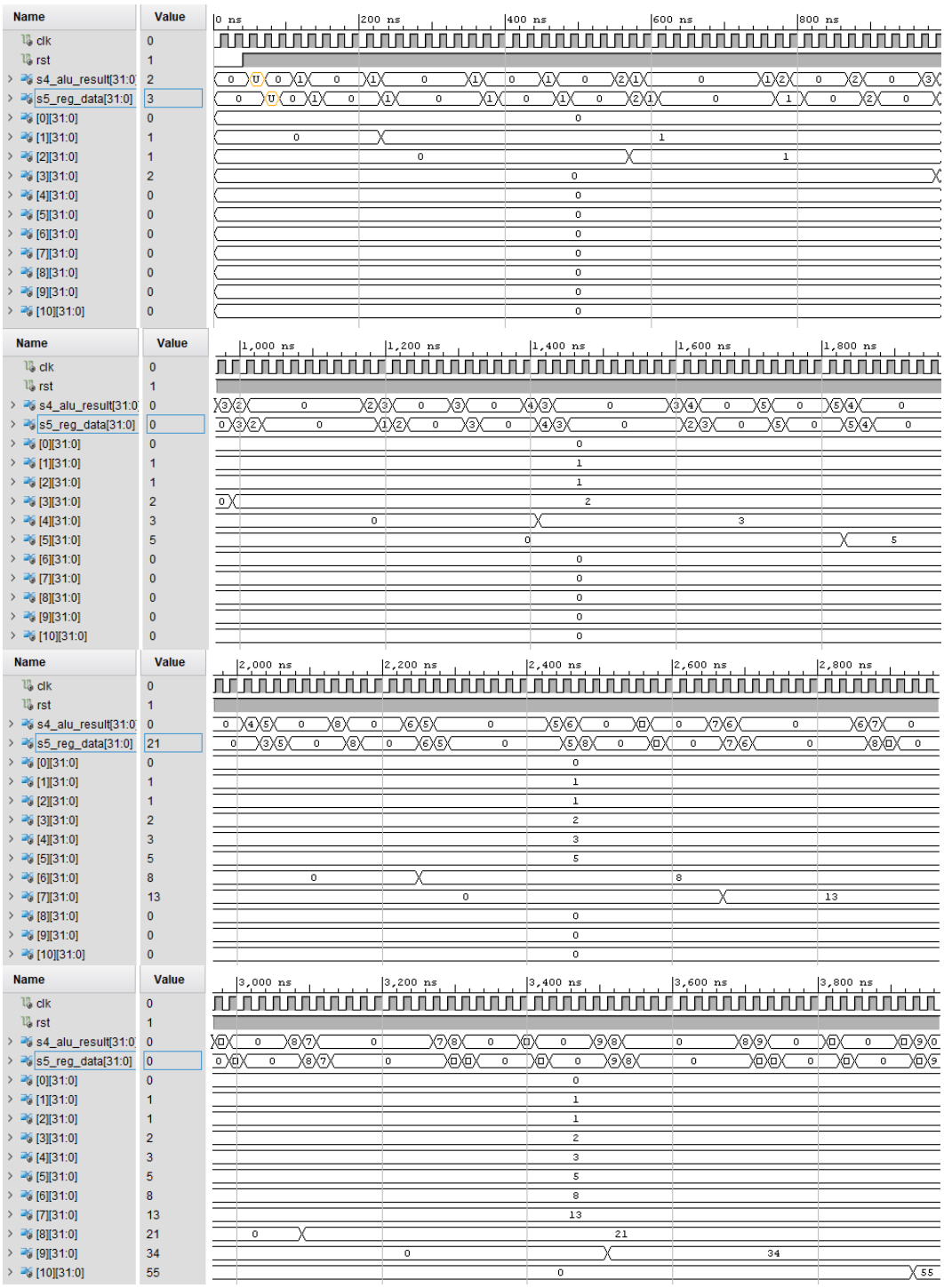

Figure 3: Fibonacci behavioral simulation waveforms.

Exercise 1: Pipelined MIPS Processor

Student's Name: Chris Abajian Section: 03

PreLab		Point Value	Points Earned	Comments
PreLab	Fibonacci Pseudo-Code	10	10	PL 3/28

Demo		Point Value	Points Earned	Date
Demo	"Basic Testbench (Part A	25	25	SL 4/11
	Part B	25	≥5	SL 4/11

To receive any grading credit students must earn points for both the demonstration and the report.