CMPE 260 Laboratory Exercise 5
Memory & Writeback Stage

Glenn Vodra
Performed: March 27, 2023
Submitted: March 30, 2023

Lab Section: 3
Instructor Sutradhar
TA: Ian Chasse
Cole Johnson
Robbie Riviere
Jonathan Russo
Colin Vo

Lecture Section: 1
Professor Cliver

[illegible]

Abstract

The purpose of this exercise was to design the memory and writeback stages of a MIPS processor. The exercise involved designing an alternate form of memory, data memory, as well as implementing control logic that handles writeback to the general purpose registers. Several pieces of hardware description code were provided and were translated from Verilog to VHDL. These were data memory, the writeback stage, and the write-back test bench. A "Memory stage" that created an instance of the provided data memory was designed, as well as a test bench to test this stage separately from the writeback. These two stages make up the last two pieces of the MIPS processor design. With them, values can be held in longer-term storage, and the processor can decide what to store in registers. Once these designs were tested, the exercise proved to be successful.

Design Methodology

The two main components, data memory and the write-back stage of the processor were provided in Verilog. Some details of the first design are as follows. At the top, there is a module declaration, which is very similar to an entity declaration. It contains all of the signals, listed as inputs or outputs, and also declares signals that will be written to in an always block as a register. This design also has parameters that define values such as the address length and memory location size. A few of the signals provided as inputs and outputs like the input address reference these parameters. Some of the other signals are control logic, as well as data buses.

The rest of the design defines three aways blocks, that are triggered on the rising edge of the clock signal. These are similar to processes in VHDL that are sensitive to a rising clock signal in addition to an if statement following it, to check the clock status. Each of these always blocks does something different. The first checks if the write enable is high. If it is, it will take the incoming data and store it in the address pointed to by an address signal. The second always block will write the output seven segment, which displays four numbers when the address is 1032 and the write enable is high. The final always block will check for an address 1031 and return four hex zeros concatenated with the value of a few switches.

The other design file was translated, and the writeback stage was interpreted in a similar style. The top module was translated to an entity with five input and three output signals. The always block, provided with this code is sensitive to several signals, indicating this utilizes combinational logic. These three signals are the read data, memory to register, and ALU result. The If-Else code that follows creates a single mux, with memory to register as the select line. The stage of the design will choose the result of the ALU or the memory read is stored in the registers. In this stage there is also a control signal, register write that is passed through to determine if the write enable of the registers will be set at all. For some MIPS instructions, writing data back to the registers is not necessary.

Results and Analysis

To demonstrate the proper functionality of these two processor stages, behavioral simulations were performed on both. Additionally, the memory stage also passed a post-implementation timing simulation. The results of the behavioral simulation for the memory stage are shown in the following figure, figure one.

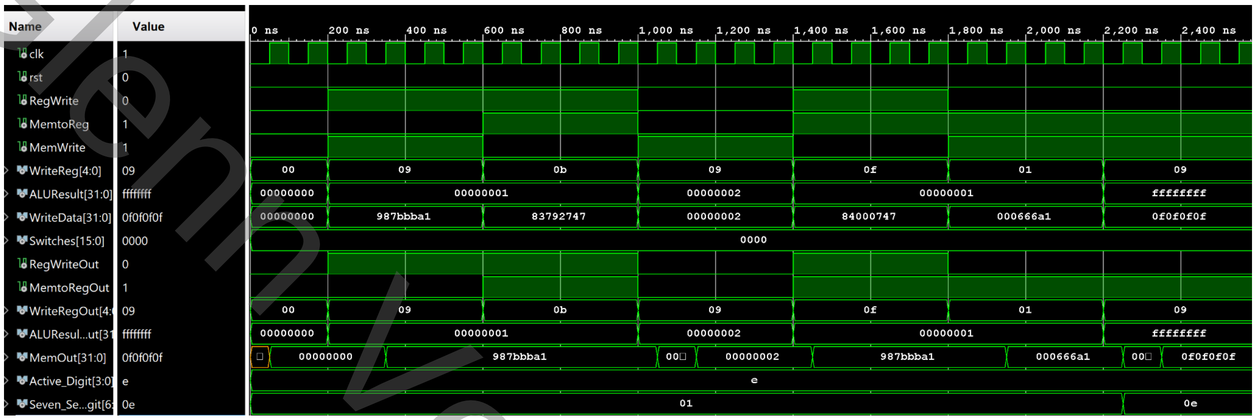

Figure 1: *Memory Stage Behavioral Simulation*

This self-checking test bench design performed six tests on the design. Since the design read and stored values on the same rising edge, it was important to ensure signals were assigned before these edges, as well as running the clock a few times before ensuring the value was written to memory. Many of the test cases were designed to write a value to memory and then, after ensuring it was properly stored, assert that it was added successfully. There were also several test cases where a value wasn't written. In these cases, the test checked if a value was still stored in the correct location, after other operations. The first test case, starting at 200 ns, in figure one, shows the value "0x987BBBA1" being written to memory and then checked. This design was also simulated with a post-implementation timing simulation, shown in figure two.

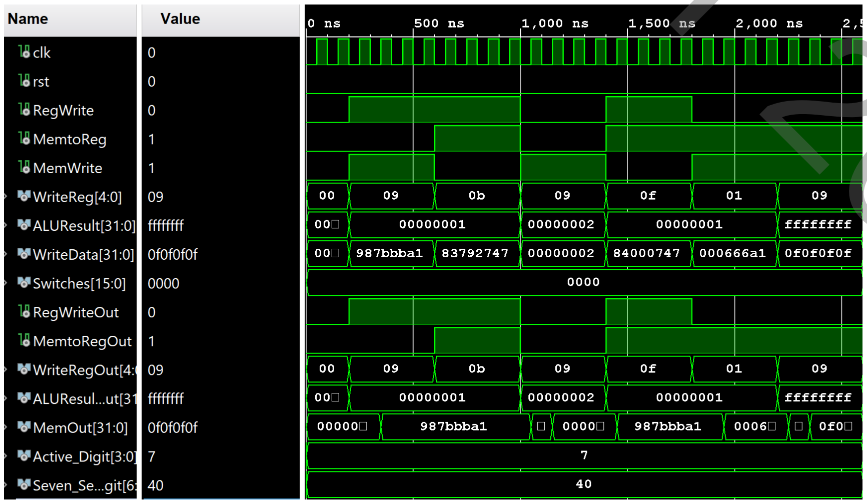

Figure 2: *Memory Post-Implementation Timing Simulation*

In this design simulation, the same test cases are performed and the simulation results are as expected. In figure two, test four checks the value in memory address 1, and assumes it is the value that was previously written. After a clock cycle, the correct output "0x987BBBA1" is read out. It is also important to note that some of the signals seen in figures, one and two are only present for the hardware implementation of this design. These are mostly related to a seven-segment display used for checking memory values.

In addition to the simulations performed on the memory stage, tests were also performed on the write-back stage.

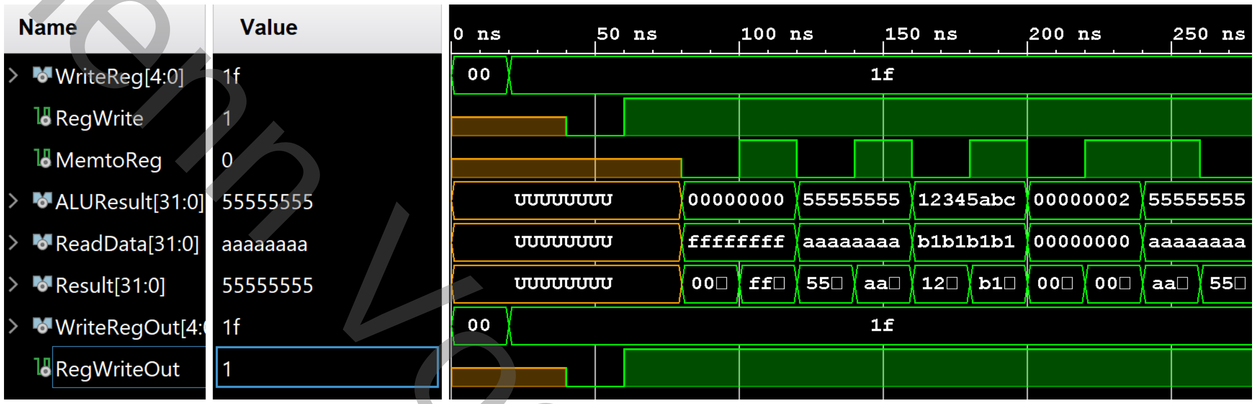

Figure 3: *Write-Back Behavioral Simulation*

This test bench performs five tests. For each test, two pieces of data are assigned; one to the ALU result and one to the memory-read data. In each of these tests the select line MemtoReg is toggled, picking one of the two inputs. For the final test, the ALU result is selected before the memory output, while in the others the opposite is the case.

Conclusion

This exercise was successful because two stages of the MIPS processor design were tested using various simulations and test cases. These stages are important as they handle a lot of data movement in the processor design, whether it be out of or into memory, or deciding where and what to write to registers. Instructions that move data between different parts of a system are very important for implementing higher-level programming concepts such as variables and data structures. Not only is the specific data important but where it goes and how it is accessed are just as important. This exercise was a success and will be useful for the implementation of a final processor design.