Lab #4 & 5: Adding Data Memory to the Datapath

EECE 2323 – Prof. Xiaolin Xu

Date: 02/15/2022

Name: Fengyi Quan && Yichen Sun

Northeastern University

1. **Background & Purpose**

In this experiment, the objective of this lab is to build the processor datapath using an ALU and add a new type of storage, Data Memory. The sequential circuit that holds a state and is timed with a clock signal has been introduced. To complete this lab, a register file and data memory should be created first. During this lab, a new type of storage, Data Memory, is to be added to the datapath. Data memory is typically slower but larger than registers. It is accessed with load and store instructions. Values from the register file can be stored in memory, and data from memory can be loaded into registers. The goal of this experiment is to properly design and connect data memory into the partially assembled central processing unit. To support imperative computation, it is necessary that a computer system provide methods such as storing, accessing, and modifying persistent state. Modern CPU architectures typically a finite set of registers which may be used to store machine words across the execution of multiple instructions. The processor to be designed is a RISC (Reduced Instruction Set Computer) that has a load-store architecture. The results of ALU operations are always stored in a register before they can be stored in memory. The ALU is used in load and store instructions to calculate the memory address.

1. **Prelab**

3.1.1

The final contents of all registers would be zero after ‘clear the regfile’. However, the contents before that are following:

Reg1: 9’d2 = 9’b0\_0000\_0010

Reg2: not 4 = 9’b1\_1111\_1011

Reg3: 8’h15 = 9’ b0\_0001\_0101

Reg4: 1+3 = 9’b0\_0001\_0111

1. Reset all register and memory to all zeros.
2. Load the value 2 to memory location 1 and the value 1 to memory location 2.
3. The value in memory location 1 is loaded to register 1 which would have value of 2.
4. Inverse the value stored in memory location 2 by applying ~ operation. Result in 9’b1\_1111\_1011
5. 8’h15 is loaded in register 3 as b0\_0001\_0101.
6. Add numbers in register 1 and 3 together by applying + operation and get 9’b0\_0001\_0111, store it in register 4.
7. Store value of 9’ b0\_0001\_0101 to memory location 3 and 9’b0\_0001\_0111 to memory location 4
8. Reset and clear everything in register and memory to all zeros.

3.1.2

Reg1: 9’d3 = 9’b0\_0000\_0011

Reg2: 9’d4 = 9’b0\_0000\_0100

Reg3: 4-3 = 9’ b0\_0000\_0001

1. Reset all register and memory to all zeros.

2. Load the value 3 to memory location 1 and the value 4 to memory location 2.

3. The value in memory location 1 is loaded to register 1 and memory location 2 is loads to register 2.

4. Subtract integers in two registers, then store output value in register 3.

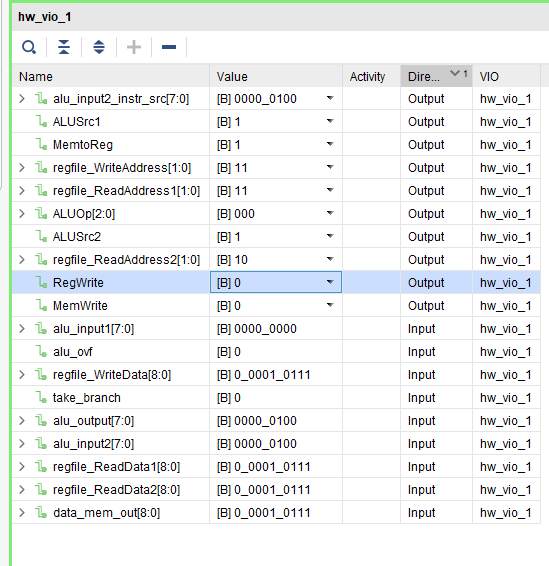
5. Reset and clear

1. **Results and Analysis**

Implementation of a sequential logic requires to design and build the processor datapath using an ALU, a zero register and a register file. In the lab, the ALU performs major action in the processor, for instance, choosing operations on the operands and generates new results. A register file controls reset, clock, read and write address and data as an array of registers. And the zero register helps the processor to call “clr” or “clear”. In the lab, the functionality of previous ALU is extended which now supports more operations. Also, the output of a register file is determined by the wr\_addr input, rd0\_data and rd1\_data outputs. If wr\_en is activated, the regfile will pass the values stored in wr\_data to each bit in the register, then work flow will make new data to register file otherwise the content of register file should not to modified as the output should maintain the values of which rd0\_addr and rd1\_addr point. Since the sequential logic is not as simple as the combinational logic, we will have to test simulation on sequences of instructions to ensure that operands are read from, and the results written to, corresponding to the sequences of instructions. In the simulation,

In the second lab, a new type of storage, Data Memory, was implemented in the datapath. Data memory allows the user to access and store instructions similar to registers. It typically stores values from the register file and load memory data into registers.

Appendix I: VIO Testing result



In VIO simulation, the data memory stores value 2 to memory location, and value 4 to memory location 2, where locations are represented by binary bits. Next, the value of memory location 1 and 2 can be loaded into reg 1 and reg 2 via regfile\_ReadData1 and regfile\_ReadData2. Also, the user can write new hexadecimal data to new register as binary bits string. Then, ALUOp computes the operands from multiple registers and stored into a new register.

1. **Conclusion & Recommendations**

In the first part of this lab, sequential logic was explored by a way of designing and building the preprocessor path connecting an ALU, a zero register, and a register file. This assignment demonstrated some of the functions of one of the central parts of a MIPS processor. It was designed with a clock cycle. And all components were connected through initializing each of the component’s files and linking the outputs one with the inputs of another to model the wires between them.

In the second part of this lab, the concept of data memory was successfully implemented by a memory bank and used into the data path. Memory banks act by design similarly to the register file, but with significantly more size. The data memory was designed in Verilog using the IP source single-port RAM option by Xilinx.

1. **Appendices**

Appendix A: ALU Regfile Module

Text

Description automatically generated with medium confidence

Appendix B: Reg File Module

Timeline

Description automatically generated

Appendix C: ALU Module

Graphical user interface, text

Description automatically generated

Appendix D: ALU Regfile testbench

Text

Description automatically generated

Text

Description automatically generated

A picture containing text

Description automatically generated

Text

Description automatically generated with low confidence

Graphical user interface, text, application, email

Description automatically generated

Appendix E: ALU Regfile testbench simulation waveform

Graphical user interface

Description automatically generated

Appendix F: ALU Regfile testbench TCL console output

A picture containing text, window

Description automatically generated

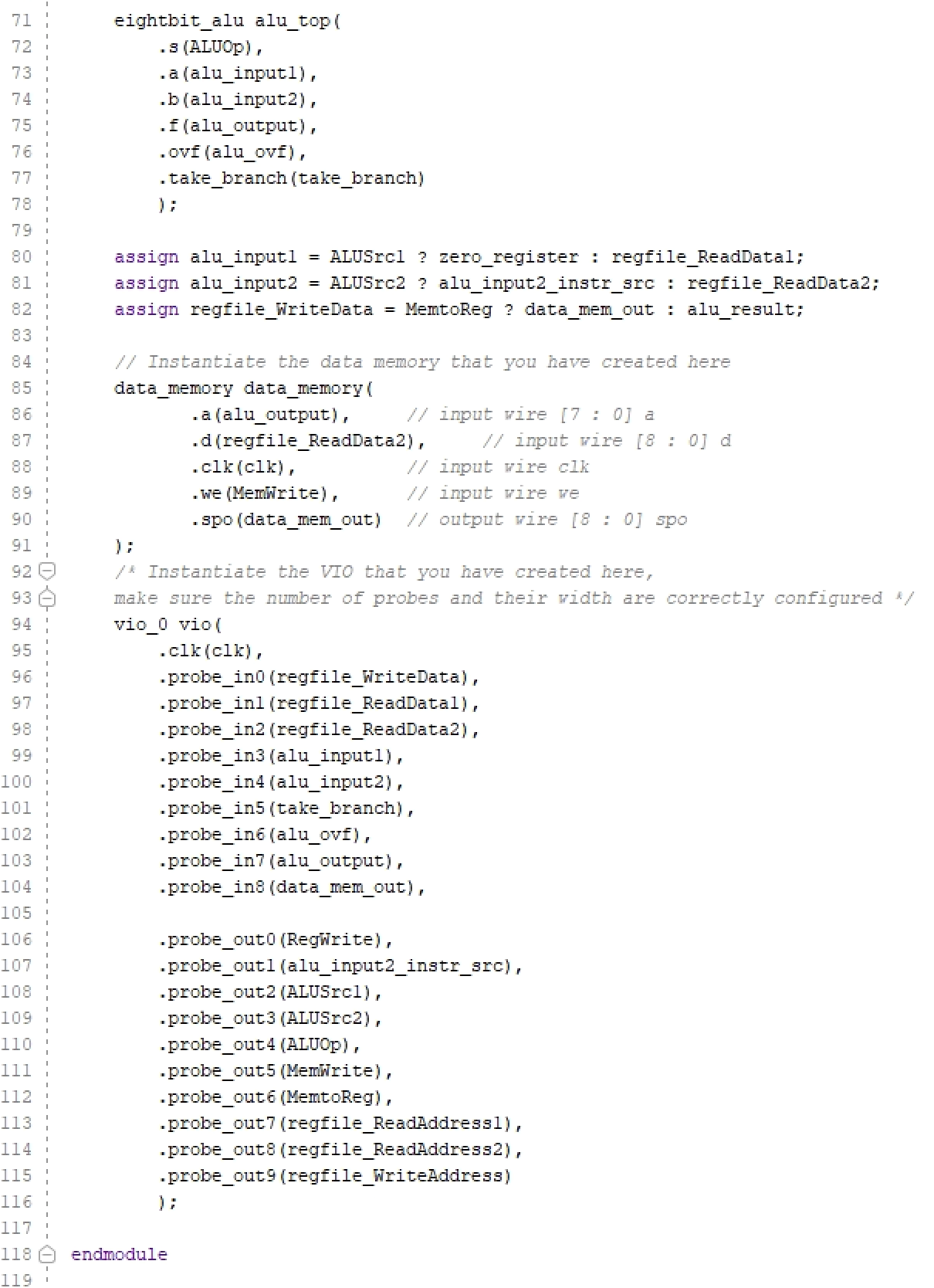
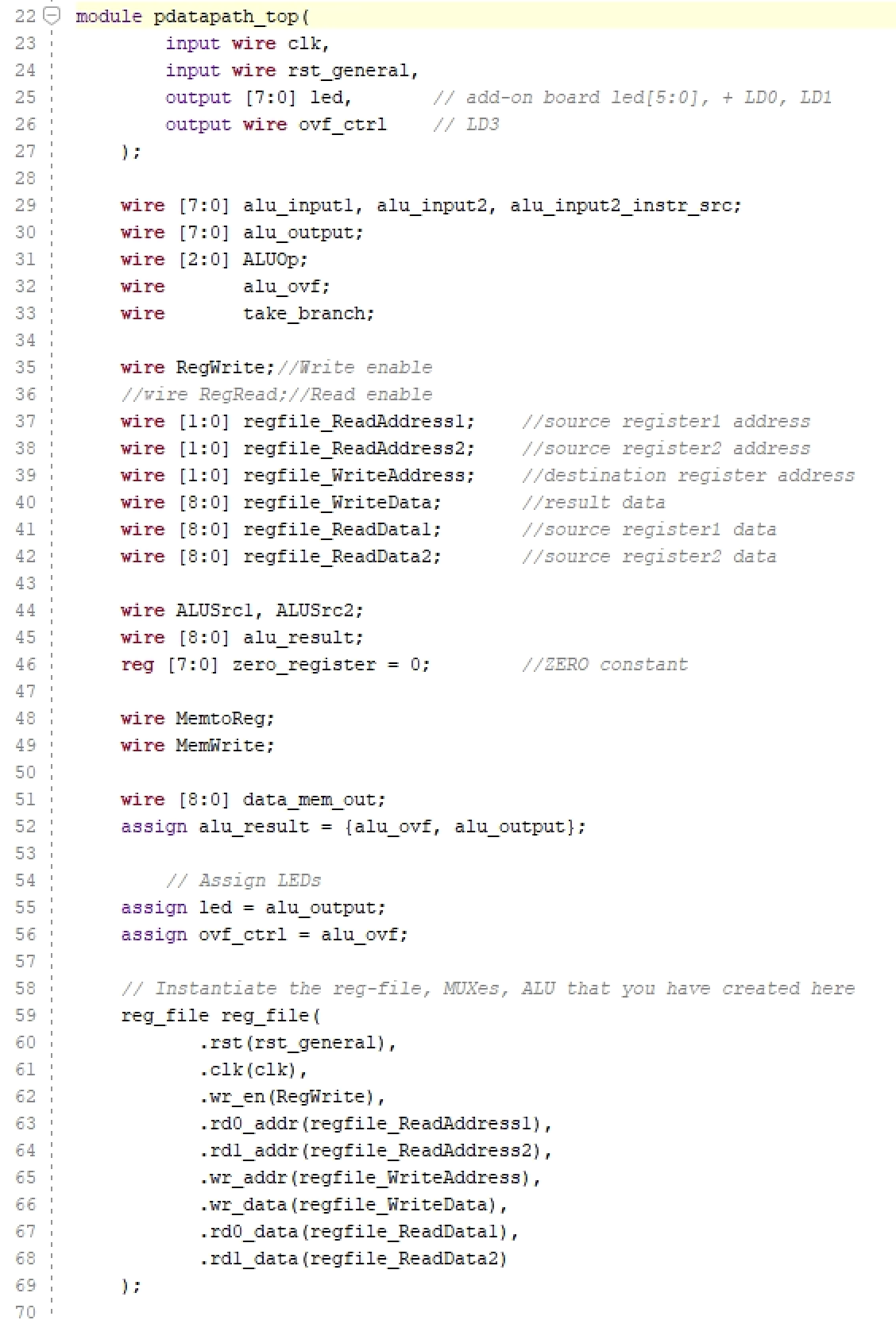
Appendix G: ALU Regfile VIO Top Module Lab 4

Table

Description automatically generated with low confidenceText

Description automatically generated with medium confidence

Appendix H: datapath VIO Top Module Lab 5



Appendix I: VIO Testing result

