EECE2323 **Digital Logic Design Lab** **Lab 5**

Adding Data Memory to the Datapath

1 Objective

In this lab you will add a new type of storage, *Data Memory*, to your datapath. Data memory is typically slower but larger than registers. You access it with load and store instructions, which we will cover in future labs. Values from the register file can be stored in memory, and data from memory can be loaded into registers.

You will add *Data Memory* to the datapath that you designed in Lab 4. You must complete Lab 4 before you start this experiment.

2 Overview

In this lab you will add *Data Memory* to the datapath you started in Lab 4. Data memory is typically slower but larger than registers. Your register file contains just a few locations, while your data memory will be much larger.

The processor you are designing is a RISC (Reduced Instruction Set Computer) that has a load-store architecture. This means that values are stored to memory from a register and values are loaded from memory to a register. 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, but not the data. We will cover load and store instructions and memory addresses in future labs.

3 Data Memory and Connection to Datapath

In this lab, you will add data memory to the datapath you completed in Lab 4. You should implement a memory with 8 bits of address, for a total of 256 data words. Each word should contain 9 bits, just like in the register file. In order to have reading of data memory (DM) be combinational (no clock signal required) we will use distributed RAM. Instructions for implementing the memory block are below.

For the rest of the labs till lab 7, testing in hardware will use VIO.

Figure 1 shows the datapath including the memory and the VIO. The probes to the left of the VIO block are input probes and those on its right are output probes. Another important note about Figure 1 is that the Data Memory only has a write enable signal. It does not have read enable.

You will also see a clock input(*clk*). The clock signal is essential to sequential logic, and provides a sense of time to the register file and determines when the data input will be looked at.

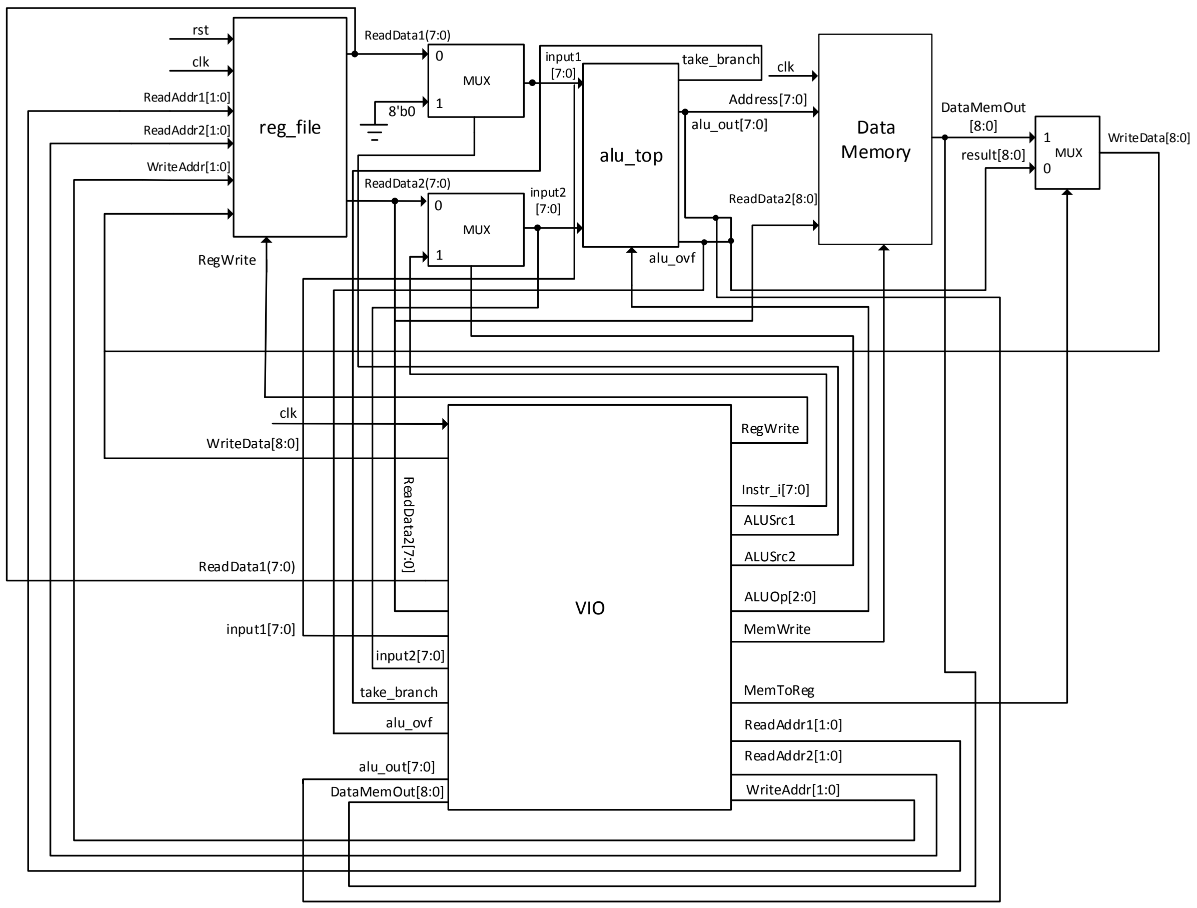

Figure 1: Datapath with Data Memory

3.1 Prelab

Begin by reading Sections 1, 2, and 3 above. Then answer the following questions.

3.1.1 Prelab Part 1: Load & Store Operations

Data memory has two corresponding operations, Load and Store. In the load operation, an element from the data memory will be read and loaded to the register file, while in the store operation an element from the register file will be written to and stored into an address in the data memory.

We have listed a sequence of tasks below which will implement an example of store and load operations. Go through them and answer the questions that follow:

```
1.  reset
2.  Store values 2 and 4 to memory addresses 0 and 1.
3.  Load the value of memory address 0 to reg 0.
4.  Load the value of memory address 1 and put the bitwise invert of this value in reg 1.
5.  Write 8'h15 to reg 2.
6.  Add reg 0 + reg 2 and store in reg 3.
7.  Store the contents of registers 2 and 3 to memory addresses 2 and 3.
8.  Clear the regfile.
```

- What would be the contents of the registers in the register file after the steps described in the file?
- Explain what these tests do functionally, (e.g. load Reg[1] from Mem[1], store value x in Mem[4] memory location, etc.)

3.1.2 Prelab Part 2: Create your Own Sequence!

- Write your own sequence of steps that tests the datapath using different values and different ALU instructions. What are the expected values in the registers after your test is complete? What ALU operations have you NOT tested?
- Explain what these tests do functionally, (e.g. load Reg[1] from Mem[1], store value x in Mem[4] memory location, etc.)

Submit your Prelab responses with your report submission.

==

3.2 Entering Your Design

Get the top level Verilog file, `pdatapath_top.v` from the course webpage. This Verilog code shows how all of your components are connected. You should use your `regfile` and `alu` from Lab 4. You will need to generate data memory and VIO to add to this experiment. Note that this VIO module has several new inputs and outputs.

3.2.1 Data Memory

For implementing Data Memory, add a Distributed Memory IP Core to the design and configure it. You can find this IP core in the IP catalog. To add Data Memory to your design in Vivado, follow these steps:

Open the IP catalog by clicking on "IP Catalog" in the Flow Navigator Pane under the Project Manager section. You should see the IP catalog windows opened. There you can find *Distributed Memory Generator* in the Memories & Storage Elements category under the RAMs & ROMs section, or you can look it up by typing Distributed Memory Generator in the search bar. After that, double click on the core name (Distributed Memory Generator) to open the configuration window. In the memory config tab of the configuration window, change the Depth to 256 and Data Width to 9 (8-bit data and 1-bit overflow) and set Memory Type to "Single Port RAM". Change the name of the component to `data_memory` by writing the name in the component name field. Leave other all options as they are, then press OK and then Generate to add the core to your design. Find the instantiation template in the IP Sources tab under:
`data_memory->Instantiation Template-> data_memory.veo`. Check that this template matches the interface for the memory in the `pdatapath_top.v` module. Your interface signals should be:

clk clock signal for the memory core

we Write enable signal. If 1, write is enabled

a The target address in the memory for writing dina to or reading douta from that

d Input data (write data)

spo Output data (read data)

3.2.2 Adding the VIO

We will use the *Virtual Input/Output (VIO)* IP core from Xilinx. This is the same core as in labs 3 and 4, but you will need to configure it differently for this lab. The VIO probes you will use in this lab are shown in Fig. 1.

To add this core to your design, open the IP core catalog and type "VIO" in its search bar. Double click on *VIO (Virtual Input/Output)* to open the configuration window. Configure your VIO as shown in Fig. 1 by setting up the right number of input and output probes, and the correct width for each probe.

Add the constraint file for your design. The constraint file for this design is named: `pdatapath.xdc`

3.3 Testing in Hardware

3.3.1 Implementing the Design with Vivado Synthesizer

Click on `Generate Bitstream` **under the Program and Debug section in the Flow Navigator pane to run the synthesis process. Vivado IDE will run the Synthesis and Implementation processes automatically.**

If the synthesis process runs correctly, continue by programming the FPGA board. Follow the same steps you have for previous labs.

3.3.2 Programming the FPGA with Hardware Manager

Program your hardware the way you have in previous labs. Open the hardware manager, open the target device, and program the TUL PYNQ board.

Now you should see the `hw_vio_1` window open. Click on the + sign and add all of the probes by selecting and adding them. Now you are ready to test your design in hardware.

3.3.3 Testing Your Design

For this design, we are using one reset, named `rst_general` in Verilog, to reset the memory elements (memory and register file). On the board, it is connected to the right push button (`BTN0`).

Use VIO to test your design. A sequence of inputs and outputs needed to test your design, including loading and storing memory, is provided on the Prelab section above. Test this sequence. Keep screen shots to show that you have done this.

Come up with your own sequence to test the datapath. It should use different values and different ALU operations in addition to those used in the sequence provided. Keep track of your sequence of tests and use screen shots to capture your results. The screen shots should be included with your report submission. You should explain functionally what your tests do (load register 1, store value x in memory location y, etc.).

Use VIO to test your design. Write down a sequence of inputs and outputs you will need to test your design. Remember that you need to write values to the register file, and then read them out and set the muxes appropriately to get the outputs from the ALU. You want to use a representative set of tests from your testbench.

Once your hardware works, capture screen shots of your test sequence and include them in your submission. Each screen shot should have a description of what the test do.

4 What to submit on Canvas:

1. A demonstration of your working design as determined by the TA.
2. **A single PDF report containing:**
   a) Pre-lab responses.
   b) Summary of hardware/design implementation report as outlined in the rubric.
   c) All your output screen shots demonstrating the correct operation of your design.