EECE2323 **Digital Logic Design Lab** **Lab 8**

Adding Branch Logic to the Datapath to Complete Your Computer

1 Objective and Overview

In this lab you will complete your computer by making two changes to lab 7. You must complete Lab 7 before you start this experiment.

1. Rewrite the Program Counter (PC) logic to support branching.
2. Set the seven segment display (SSD) to your datapath to display the output of your ALU instructions. *(This was already set by default).*

Figure 1 shows the top level design for your computer.

When you have completed these changes, you should run the assembly code you wrote for the prelab to demonstrate that your computer works correctly and can run code. **You should now have a working computer!**

==========

1.1 Prelab

Begin by reading Section 1 above. Then ***use the Instruction Set document on Canvas*** as a road-map for answering the following questions.

1.1.1 Design a PC logic with branching capability

Write a Verilog module which accepts the clock, a reset, the immediate value from the instruction word (least significant byte), and the `take_branch` output from the ALU as inputs and generates an 8-bit Program Counter (PC) for the output.

Note that in this architecture when we have a branch, the next PC value should be the current PC value plus the offset which is extracted from the branch instruction. The offset is represented in two's complement, so the range of branch target is from PC - 128 to PC + 127. Note that the value of PC should not exceed 0xFF as we have a 256-deep instruction memory. You do not need to check for this condition in your hardware.

1.1.2 A Signed Multiplier in Assembly

Based on the instruction set document, write an assembly program which multiplies two 8-bit signed numbers (in two's complement system).

The multiplier and multiplicand could be with any sign (positive or negative). However, assume the multiplication result fits in one register only.

Note that in this lab you have branching hardware so you can insert conditions and labels in your program.

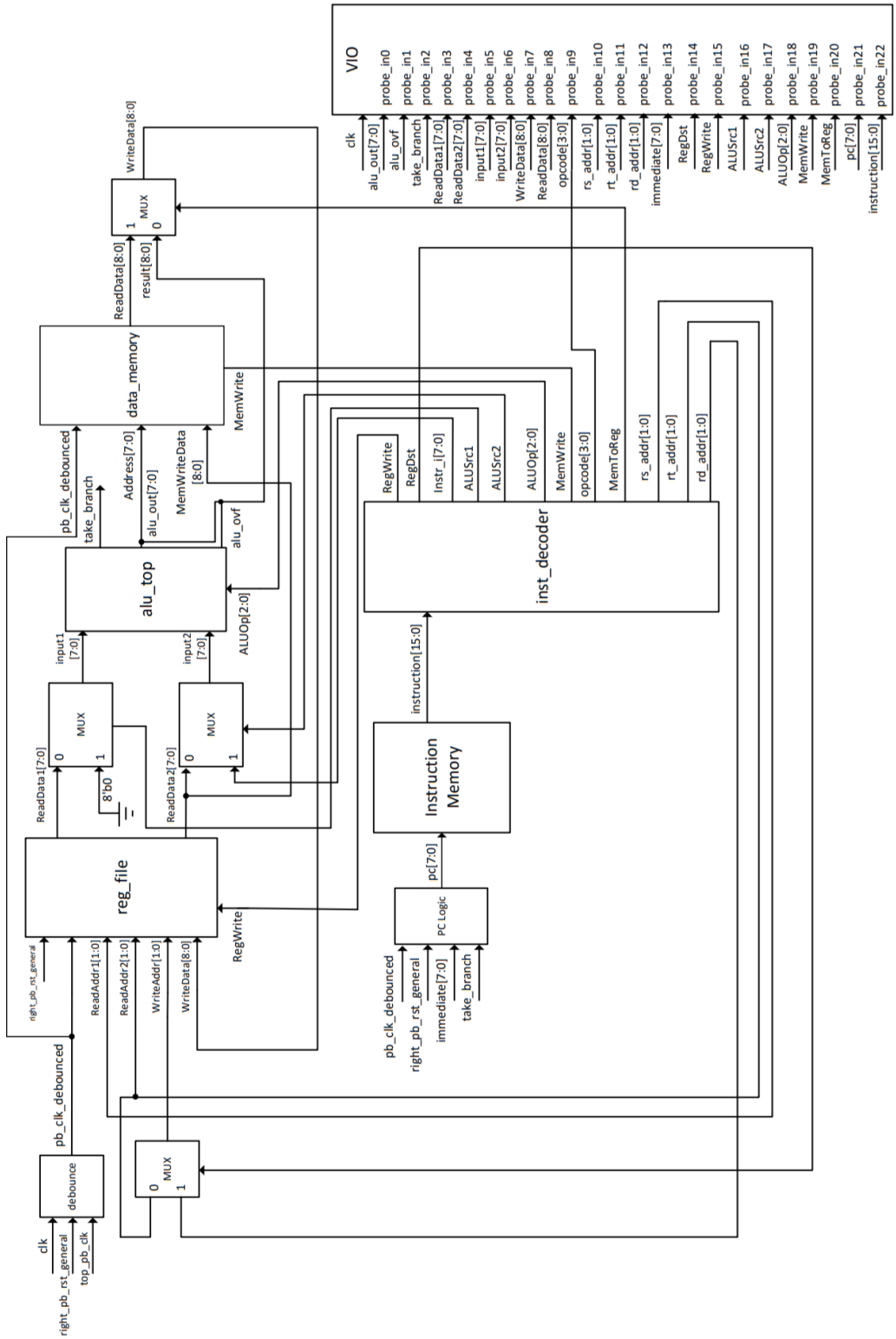

Figure 1: Final Computer

1.1.3 Generate the Machine Codes using the Assembler

Use the assembler available on Canvas to generate the machine codes out of your assembly program. Use different sign combinations for multiplier and multiplicand, such as both being positive or negative, or having opposite sign for them. Use immediate values to initialize your data inputs.

Verify the output of the assembler by comparing the output machine codes with the instruction structure described in the instruction set document.

Turn in the output file of the assembler, together with the MIPS assembly program.

Note: You can use an online C IDE from `http://www.tutorialspoint.com/codingground.htm` for your code development, compilation and running the assembler.

========== END Prelab ==========

2 Entering Your Design

2.1 Designing the top level

Note: The files `datapath_top_lab8.v` and `pdatapath_lab8.xdc` are the exact same files used in lab 7 and you can use those instead.

Get the top level Verilog file, `datapath_top_lab8.v` from the course webpage. This Verilog code shows how all of your components are connected. You should have regfile, alu and instruction decode components already designed. Get the debounce circuit, `debounce.v` from the course webpage.

You will need to generate instruction memory, data memory, SSD, and VIO to add to this experiment. You also will need to add the Verilog code that describes the program counter as described below. Note that the datapath uses two clocks: the debounced output of the pushbutton (`pb_clk_debounced`), and the free running clock, (`clk`). The push button clock is used by elements in your datapath. The free running clock is used by the SSD and the VIO.

2.2 Program Counter

Enhance your program counter (PC) from lab 7 to handle branches. Inputs and outputs to the PC logic are shown in Fig. 1. Note that the signal `take_branch` is high when a branch, should be taken and low otherwise. In this architecture when we have a branch the next PC value should be the current PC value plus the offset which is extracted from the branch instruction. The offset is represented in two's complement, so the range of branch target is from PC - 128 to PC + 127. Note that the value of PC should not exceed 0xFF as we have a 256-deep instruction memory. You do not need to check for this condition in your hardware.

2.3 Adding VIO and SSD

In this experiment we will be using both VIO and the SSD to display the output. The VIO module has the inputs shown in Fig. 1 and no outputs. You are using VIO to monitor what is happening in your hardware. All the control logic is being generated by you in hardware.

The SSD is implemented in the Verilog file, `adaptor_display.v` provided on Canvas earlier. This circuit has been instantiated for you in the `datapath_top_lab8.v` file.

2.4 Completing the top level

Add the constraint file for your design. The constraint file for this design is named: `pdatapath_lab8.xdc`

2.5 Testing in Hardware

2.5.1 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.

2.5.2 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 Zynq.

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.

2.5.3 Testing Your Design

This experiment uses the same buttons as in experiment 6. There is one reset, named `rst general` in Verilog, to reset the PC, register file, and instruction memory output port. On the board, it is connected to the right push button (`BTN0`). The upper push button (`BTN1`) allows you to single step your design. You need to press `BTN1` whenever you want to store data or update the PC.

Use VIO and the SSD to test your design. Run the code you generated in prelab on your computer. Keep screen shots to show that you have done this.

To change the inputs you will need to change the immediate values in your code, recompile and reload the file into instruction memory. To load a new file follow these steps: Double click on the instruction memory in the design hierarchy. This opens the configuration panel. Under the RST & Initialization tab browse for the new .coe file. Press ok then regenerate the memory IP. You will have to rerun synthesis and implementation and generate a new bitstream.

Once your hardware works, take screen shots of your VIO and SSD showing the output of your various tests. Include these in you submission.

Congratulations, you now have a working computer!.

3 What to submit on Canvas:

A single PDF report containing:

a) All your Pre-lab responses and code.
b) Summary of hardware/design implementation report as outlined in the rubric.
c) Your top module showing the connection of all the other modules, and your output screen shots demonstrating the correct operation of your design.
Note: You do not need to include a video demo.