Digital Design and Computer Architecture

J. Spjut

Lab 9: MIPS Single-Cycle Processor

Introduction

In this lab you will build a simplified MIPS single-cycle processor using SystemVerilog. You will combine your ALU from Lab 5 with the code for the rest of the processor taken from the textbook. Then you will load a test program and confirm that the system works. Next, you will implement two new instructions, and then write a new test program that confirms the new instructions work as well. By the end of this lab, you should thoroughly understand the internal operation of the MIPS single-cycle processor.

Please read and follow the instructions in this lab carefully. In the past, many students have lost points for silly errors like not printing all the signals requested.

Before starting this lab, you should be very familiar with the single-cycle implementation of the MIPS processor described in Section 7.3 of your text, *Digital Design and Computer Architecture*. The single-cycle processor schematic from the text is repeated at the end of this lab assignment for your convenience. This version of the MIPS single-cycle processor can execute the following instructions: `add`, `sub`, `and`, `or`, `slt`, `lw`, `sw`, `beq`, `addi`, and `j`.

Our model of the single-cycle MIPS processor divides the machine into two major units: the control and the datapath. Each unit is constructed from various functional blocks. For example, as shown in the figure on the last page of this lab, the datapath contains the 32-bit ALU that you designed in Lab 5, the register file, the sign extension logic, and five multiplexers to choose appropriate operands.

1. MIPS Single-Cycle Processor

The SystemVerilog single-cycle MIPS module is given in Section 7.6 of the text. Use the electronic versions of all these files are in the class directory. Copy them to your own lab9_xx folder.

Study the files until you are familiar with their contents. Look in mips.sv at the `mips` module, which instantiates two sub-modules, `controller` and `datapath`. Then take a look at the controller module and its submodules. It contains two sub-modules: `maindec` and `aludec`. The `maindec` module produces all control signals except those for the ALU. The `aludec` module produces the control signal, `alucontrol[2:0]`, for the ALU. Make sure you thoroughly understand the controller module. Correlate signal names in the SystemVerilog code with the wires on the schematic.

After you thoroughly understand the controller module, take a look at the datapath SystemVerilog module. The datapath has quite a few submodules. Make sure you understand why each submodule is there and where each is located on the MIPS single-cycle processor schematic. You'll notice that the `alu` module is not defined. Copy your ALU from Lab 5 into your lab9_xx directory. Be sure the module name matches the instance module name (`alu`), and make sure the inputs and outputs are in the same order as in they are expected in the `datapath` module.

The highest-level module, `top`, includes the instruction and data memories as well as the processors. Each of the memories is a 64-word × 32-bit array. The instruction memory needs to contain some initial values representing the program. The test program is given in Figure 7.60 of the textbook. Study the program until you understand what it does. The machine language code for the program is stored in memfile.dat.

2. Testing the single-cycle MIPS processor

In this section, you will test the processor with your ALU.

In a complex system, if you don't know what to expect the answer should be, you are unlikely to get the right answer. Begin by predicting what should happen on each cycle when running the program. Complete the chart in Table 1 at the end of the lab with your predictions. What address will the final `sw` instruction write to and what value will it write?

Simulate your processor with ModelSim. Refer to your earlier lab handouts if you need a refresher on how to use ModelSim. Be sure to add all of the .sv files, including the one containing your ALU. Add all of the signals from Table 1 to your waves window. (Note that many are not at the top level; you'll have to drill down into the appropriate part of the hierarchy to find them.)

Run the simulation. If all goes well, the testbench will print "Simulation succeeded." Look at the waveforms and check that they match your predictions in Table 1. If they don't, the problem is likely in your ALU or because you didn't properly add all of the files.

If you need to debug, you'll likely want to view more internal signals. However, on the final waveform that you turn in, show ONLY the following signals in this order: `clk`, `reset`, `pc`, `instr`, `aluout`, `writedata`, `memwrite`, and `readdata`. **All the values need to be output in hexadecimal and must be readable to get full credit.**

After you have fixed any bugs, print out your final waveform.

3. Modifying the MIPS single-cycle processor

You now need to modify the MIPS single-cycle processor by adding the `ori` and `bne` instructions. First, modify the MIPS processor schematic at the end of this lab to show what changes are necessary. You can draw your changes directly onto the schematic. Then modify the main decoder and ALU decoder as required. Show your changes in the tables at the end of the lab. Finally, modify the SystemVerilog code as needed to include your modifications.

4. Testing your modified MIPS single-cycle processor

Next, you'll need a test program to verify that your modified processor work. The program should check that your new instructions work properly and that the old ones didn't break. Use test2.asm below.

```
# test2.asm
# 23 March 2006 S. Harris sharris@hmc.edu
# Test MIPS instructions.

#Assembly Code
main:       ori  $t0, $0,  0x8000
            addi $t1, $0,  -32768
            ori  $t2, $t0, 0x8001
            beq  $t0, $t1, there
            slt  $t3, $t1, $t0
            bne  $t3, $0,  here
            j    there
here:       sub  $t2, $t2, $t0
            ori  $t0, $t0, 0xFF
there:      add  $t3, $t3, $t2
            sub  $t0, $t2, $t0
            sw   $t0, 82($t3)
```

Figure 1. MIPS assembly program: test2.asm

Convert the program to machine language and put it in a file named memfile2.dat. You may choose to use the MPLAB assembler to check your work. Modify imem to load this file. Modify the testbench to check for the appropriate address and data value indicating that the simulation succeeded. Run the program and check your results. Debug if necessary. When you are done, print out the waveforms as before and indicate the address and data value written by the `sw` instruction.

What to Turn In

Please turn in each of the following items, clearly labeled and in the following order:

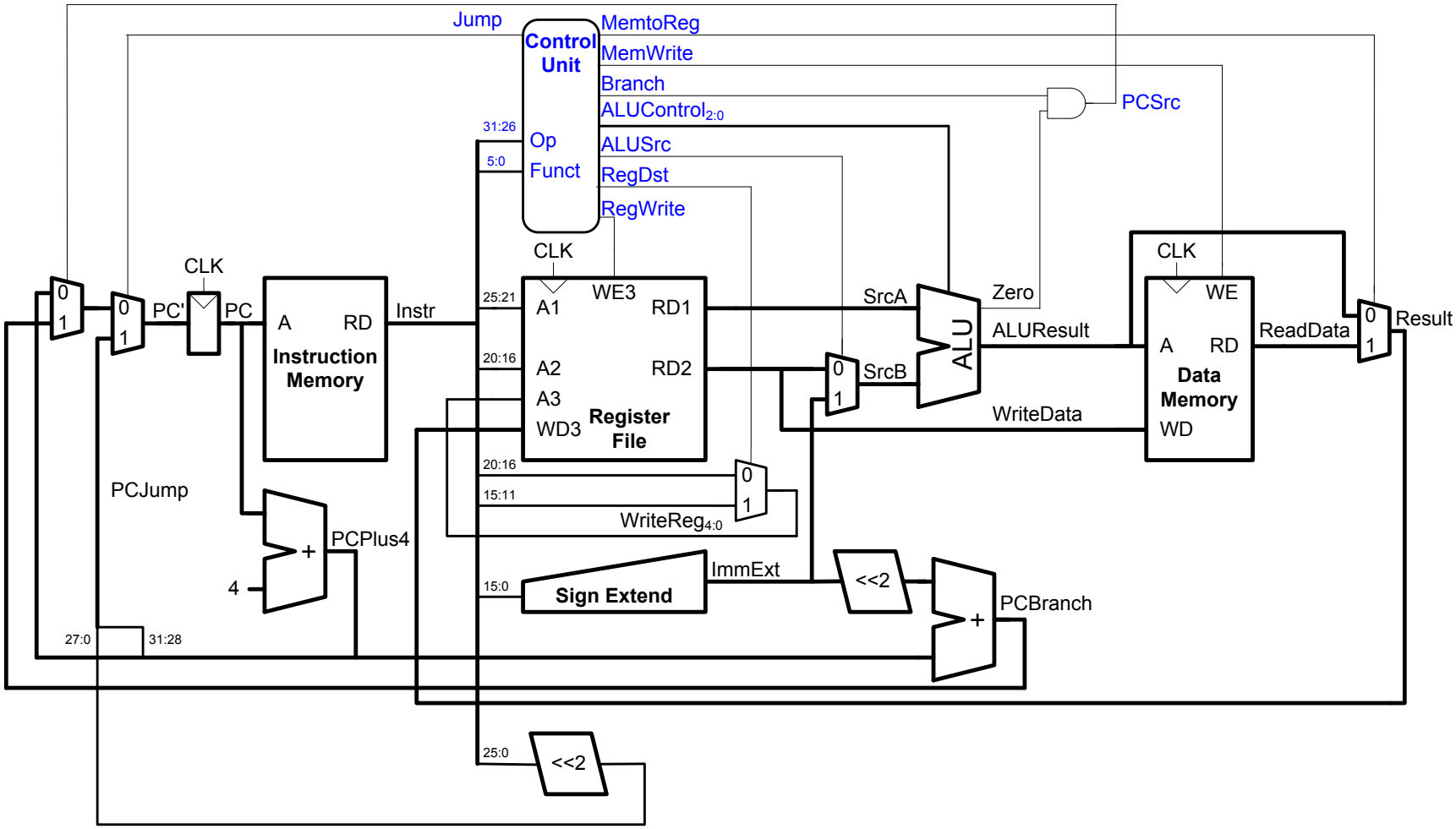
1. **Please indicate how many hours you spent on this lab.** This will not affect your grade (unless omitted), but will be helpful for calibrating the workload for next semester's labs.
2. A completed version of Table 1.
3. An image of the simulation waveforms showing correct operation of the processor. Does it write the correct value to address 84?

   The simulation waveforms should give the signal values in hexadecimal format and should be in the following order: `clk`, `reset`, `pc`, `instr`, `aluout`, `writedata`, `memwrite`, and `readdata`. Do not display any other signals in the waveform. Check that the waveforms are zoomed out enough that the grader can read your bus values. Unreadable waveforms will receive no credit. Use several pages and multiple images as necessary.
4. Marked up versions of the datapath schematic and decoder tables that adds the `ori` and `bne` instructions.

5. Your SystemVerilog code for your modified MIPS computer (including `ori` and `bne` functionality) with the changes highlighted and commented in the code.
6. The contents of your memfile2.dat containing your test2 machine language code.
7. An image of the simulation waveforms showing correct operation of your modified processor on the new program. What address and data value are written by the `sw` instruction?

Cycle	reset	pc	instr	branch	srca	srcb	aluout	zero	pcsrc	writedata	memwrite	read data
1	1	00	addi $2,$0,5 20020005	0	0	5	5	0	0	0	0	x
2	0	04	addi $3,$0,12 2003000c	0	0	c	c	0	0	0	0	x
3	0	08	addi $7,$3,-9 20067fff7	0	c	-9	3	0	0	0	0	x
4	0	0C										

Table 1. First sixteen cycles of executing mipstest.asm



Single-cycle MIPS processor

Extended functionality. Main Decoder:

Instruction	$Op_{5:0}$	RegWrite	RegDst	AluSrc	Branch	MemWrite	MemtoReg	$ALUOp_{1:0}$	Jump			
R-type	000000	1	1	0	0	0	0	10	0			
`lw`	100011	1	0	1	0	0	1	00	0			
`sw`	101011	0	X	1	0	1	X	00	0			
`beq`	000100	0	X	0	1	0	X	01	0			
`addi`	001000	1	0	1	0	0	0	00	0			
`j`	000010	0	X	X	X	0	X	XX	1			
`ori`	001101											
`bne`	000101											

Extended functionality. ALU Decoder:

$ALUOp_{1:0}$	Meaning
00	Add
01	Subtract
10	Look at `funct` field
11	