Lab #4: RISC-V Single-Cycle Processor

Introduction

In this lab you will build a simplified RISC-V single-cycle processor using Verilog. The hardware implementation for a single cycle processor can be found in chapter 7.3 of the textbook, while the guidelines for coding can be found in chapter 7.6. You will add a new instruction, load a test program and confirm that the system works. By the end of this lab, you should thoroughly understand the internal operation of the RISC-V single-cycle processor.

Before starting this lab, you should be very familiar with the single-cycle implementation of the RISC-V processor described in Section 7.3 of your textbook, *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 – see Figure 2. Please make sure to follow this design. This version of the RISC-V single-cycle processor can execute the following instructions: `add`, `addi`, `sub`, `and`, `andi`, `or`, `ori`, `slt`, `slti`, `lw`, `sw`, and `beq`.

Our model of the single-cycle RISC-V 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 1, the register file, the sign extension logic, and several multiplexers.

1. RISC-V Single-Cycle Processor

The skeleton codes for the RISC-V processor are provided. The only two modules that you need to implement are the controller ("ucsbece154a_controller.v") and the datapath ("ucsbece154a_datapath.v"). Specifically, the single-cycle RISC-V top module is following the one given in Section 7.6 of the textbook. Such a module, `top`, instantiates a data memory, instruction memory, and the RISC-V processor. Each of the memories is a 64-word × 32-bit array. The codes for the instruction and data memories are provided in the starter code. The former initializes the memory with a test program in "memfile.dat", which is the machine language code for the slightly modified program (Figure 1) of the coded shown in the Figure 7.64 of the textbook. Study the program until you understand what it does.

For the datapath module, use register file module from the starter code. Use ALU that you implemented in Lab 1. Use flat organization and integrate other needed functionalities (such as "extend" module) into the datapath file, similar to how "maindec" and "aludec" functionalities are integrated into controller module in the starter code. Be sure to use the parameters defined in "ucsbece154a_defines.vh" wherever possible to avoid magic numbers. Correlate signal names in the Verilog code with the wires on the schematic.

2. Modifying the RISC-V single-cycle processor

You need to modify the RISC-V single-cycle processor by adding the `jal` and `lui` instructions. First, modify the RISC-V processor schematic at the end of this lab to show what changes are necessary. You can draw your changes directly onto the schematic. Finally, modify the Verilog code as needed to include your modifications.

You will run the program below to check that your new instructions work properly and that the old ones didn't break.

```
             #Assembly Code
main:     addi        x2, x0, 5
               addi       x3,
x0, 12
               addi       x7,
x3, -9
               or           x4,
x7, x2
               and        x5,
x3, x4
               add        x5,
x5, x4
               beq        x5,
x7, end
               slt          x4,
x3, x4
               beq       x4,
x0, around
               addi      x5,
x0, 0
around: slt           x4, x7,
x2
               add       x7,
x4, x5
               sub       x7,
x7, x2
               sw        x7, 84
(x3)
               lw         x2,
96 (x0)
               add      x9, x2,
x5
                jal        x3,
end
              addi      x2, x0,
1
   end:   add        x2, x2, x9
                sw       x2,
0x20 (x3)
                lui        x2,
0x0BEEF
                sw        x2,
0x24(x3)
```

```
done: beq       x2, x2, done
```

Figure 1. Modified test code of Fig. 7.64 in the textbook

The program is already converted into machine language (in hex) and put it in a file named memfile.dat provided in the starter code package.

3. Testing the single-cycle RISC-V processor

You will use a test-bench module provided in the starter code that instantiates the "top" module to test your design. 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 addresses will the final two `sw` instructions write to and what values will they write?

Simulate your processor with ModelSim. Add all 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. Look at the waveforms and check that they match your predictions in Table 1. 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 <u>must be readable</u> to get full credit.** After you have fixed any bugs, print out your final waveform containing these signals.

What to Turn In

Lab Report

a. 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.

b. A completed version of Table 1.

c. An image of the simulation waveforms showing correct operation of the processor. Does it write the correct value to addresses 100 and 104?

   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 in enough that the grader can read your bus values. **Unreadable waveforms will receive no credit.** Use several pages and multiple images if necessary.

d. Marked up versions of the datapath schematic (Figure 1) and decoder table (Table 2) that add the `jal` and `lui` instructions.

e. Your Verilog code for your modified RISC-V processor (including `jal` and `lui` functionality) with the **changes highlighted and commented**.

RTL Autograder

After you have finished your design (including `jal` and `lui` instructions) you will need to submit the following files to the Gradescope assignment, “lab4 RTL”:

- “ucsbece154a_controller.v”
- “ucsbece154a_datapath.v”

When you make a submission, an autograder will run several tests to ensure your design is correct. You must pass all the tests to get a full score on the lab. You can resubmit as many times as you like.
Note that your design is linted with Verilator and synthesized with Yosys to ensure your code follows all best practices. This is a significant part of your grade because you should be comfortable with writing code that can actually be used in deployable designs. You can see all possible Verilator errors here: https://verilator.org/guide/latest/warnings.html.

What to Turn In

You will need to make two separate Gradescope submissions: “lab4 RTL” and “lab4 Report”.

- For “lab4 RTL”, please submit all your RTL files.
- For “lab4 Report”, please submit your lab report PDF file.

Only one submission per group is needed; be sure to add all your group members.

Cycle	reset	PC	Instr	Src A	Src B	ALUResult	Zero	PCSrc	WriteData	MemWrite	ReadData	RegWrite	ImmSrc	ImmExt	PCTarget
1	1	00	`addi x2,x0,5`	0	5	5	0	0	0	0	x	0	000	5	x
2	0	04	`addi x3,x0,12`	0	c	c	0	0	0	0	x	0	000	c	x
3	0	08	`addi x7,x3,-9`	c	-9	3	0	0	0	0	x	0	000	-9	x
4	0	0C													
5															
6															
7															
8															
9															
10															
11															
12															
13															
14															
15															
16															
17															
18															
19															
20															
21															

Table 1. Show all cycles (up to the last instruction marked by "done" label" for executing the modified test code of Fig. 7.64 in the textbook. Use "x" when we don't care about the value.

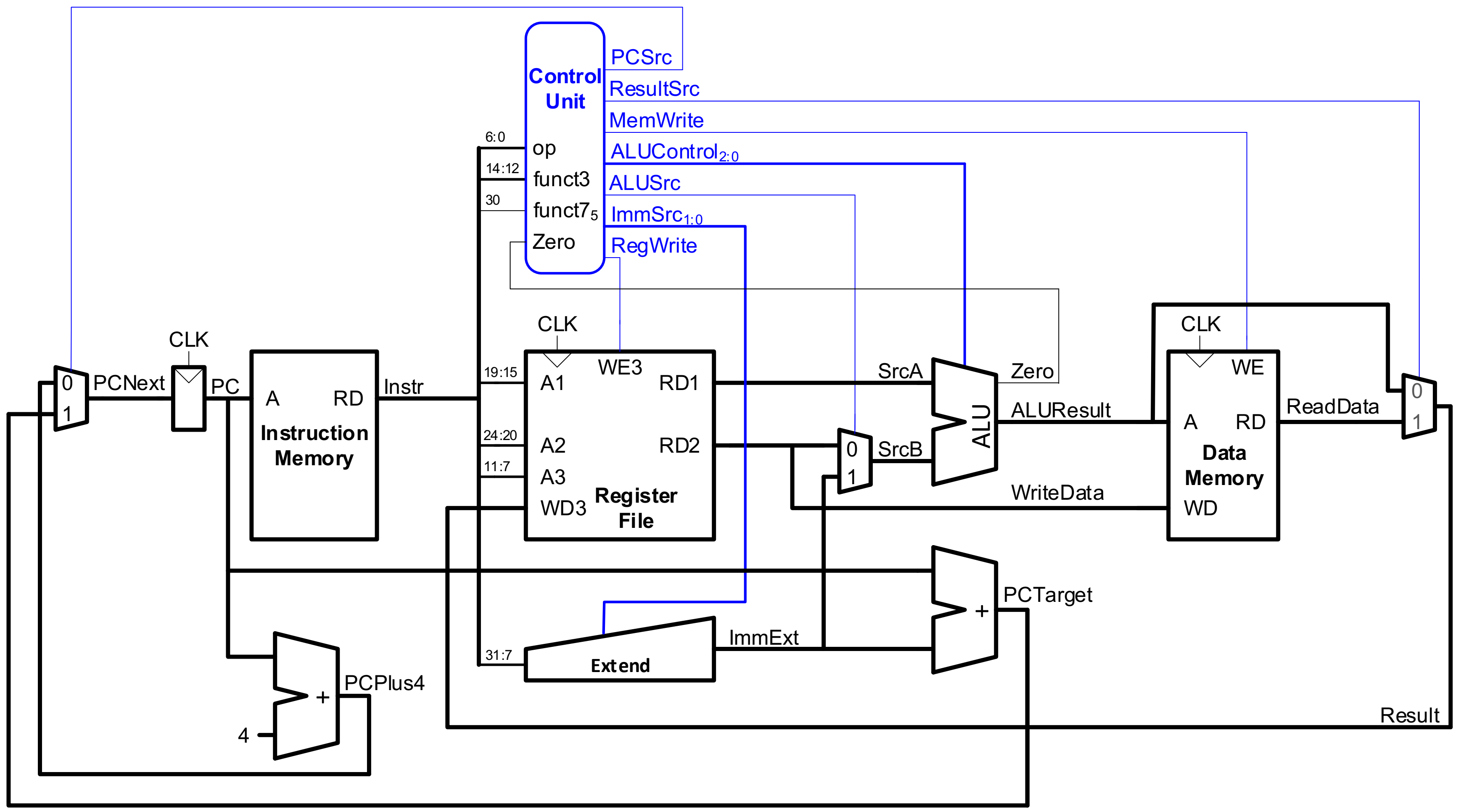

Figure. 2. Single-cycle RISC-V processor for implementing `add`, `addi`, `sub`, `and`, `andi`, `or`, `ori`, `slt`, `slti`, `lw`, `sw`, and `beq`. Show changes required for implementing `jal` and `lui`.

Table 2. Extended functionality for the main decoder. Fill in missing values in the table

op	Instruct.	RegWrite	ImmSrc	ALUSrc	MemWrite	ResultSrc	Branch	ALUOp	Jump
3	**`lw`**	1	00	1	0	01	0	00	
35	**`sw`**	0	01	1	1	XX	0	00	
51	R-type	1	XX	0	0	00	0	10	
99	**`beq`**	0	10	0	0	XX	1	01	
19	**I-type**	1	00	1	0	00	0	10	
	`jal`								
	`lui`								