Computer Organization Lab 2

Simplified Single-cycle CPU

Deadline: 5/14 (Thu.) 23:55

I. Introduction

You have learned how to implement a 32-bit ALU with some basic arithmetic operations in lab1. As well as how to compile Verilog using Icarus Verilog and waveform visualization using GTKWave. In this lab, you are required to implement a simplified 32-bit single-cycle CPU building on top of the ALU created in the previous lab.

II. Requirements

(a) Environments & Prerequisites

- You and your teammate cooperatively complete this lab. But both of you need to know every details, including the part handled by your partner.
- Source code should be able to be compiled by Icarus Verilog and generate waveform using GTKWave.
- Module templates are included inside the "sample_code" directory. You need to fill in the module's behavior. Also, you can create new modules at your will.
- ProgramCounter.v, Instr_Memory.v, Reg_File.v, Test_Bench.v provided are fully functional modules. Feel free to modify them, but do not include these files when submitting.
- All your designs should be implemented as combinational circuits.
- Prepend your student ID and name at the top of each source code file. The format should follow the guideline provided below.

```
// Author: 0123456 連花鴿, 0654321 郃基殿

module CPU(
// The rest of the file is omitted
```

(b) R-type Instruction Set

R-type instructions are identified by an opcode of 0. They can be differentiated by their funct values.

31:26	25:21	20:16	15:11	10:6	5:0
opcode (0)	rs	rt	rd	shamt	funct

The following instructions are required in this lab.

Instruction Name	Example	Pseudo Code	funct
Unsigned addition	addu r1, r2, r3	r1 = r2 + r3	100 001
Unsigned substraction	subu r1, r2, r3	r1 = r2 - r3	100 011
Bitwise and	and r1, r2, r3	r1 = r2 & r3	100 100
Bitwise or	or r1, r2, r3	r1 = r2 | r3	100 101

Set on less than	slt r1, r2, r3	if (r2 < r3)
		  r1 = 1
		else
		  r1 = 0
			101 010

Arithmetic shift right	sra r1, r2, 10	r1 = r2 >> 10	000 011
Arithmetic shift right variable	srav r1, r2, r3	r1 = r2 >> r3	000 111

(c) I-type Instruction Set

I-type instructions can be identified by opcode with any number greater than 3.

31:26	25:21	20:16	15:0
opcode	rs	rt	imm

The following instructions are required in this lab.

Instruction Name	Example	Pseudo Code	opcode
Add immediate	addi r1, r2, 64	r1 = r2 + 64(sign ext.)	001 000

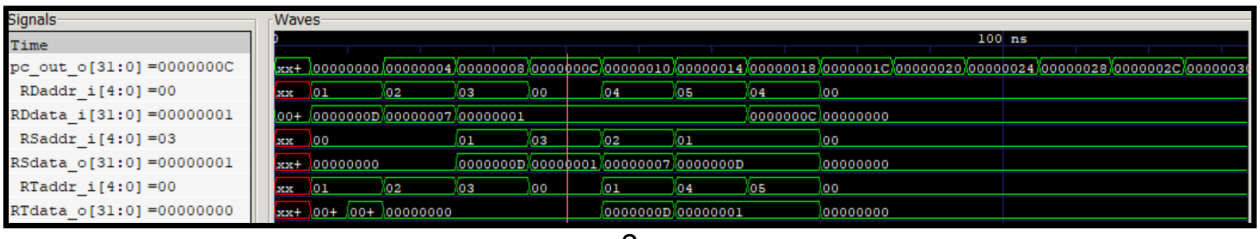
Set on less than immediate unsigned	sltiu r1, r2, 87	if (r2 < 87)
		  r1 = 1
		else
		  r1 = 0
			001 011

Branch on equal	beq r1, r2, 6	if (r1 == r2) PC += 4 + (6 << 2)	000 100
Load upper immediate	lui r1, 1922	r1 = 1922 << 16	001 111
Or immediate	ori r1, r2, 1337	r1 = r2 | 1337	001 101
Branch on not equal	bne r1, r2, 6	if (r1 != r2) PC += 4 + (6 << 2)	000 101

(d) Report

Each person needs to submit a report in Chinese or English with the following contents:

- Architecture diagram: Draw your own copy of the diagram, do not copy any existing diagram (including the ones you find on the internet or from your teammate). Make sure to preserve a modifiable version of the diagram, so you do not need to start from scratch in the upcoming labs.
- Module description: What is the purpose of each module.
- Waveform: Please take a screenshot of waveform shown in GTKWave when running any of the testing data with the following wire:
  - The output of program counter module
  - Reg_File module
    - rs and rt register selection input and data output
    - rd register selection input and data input



- Questions: Please answer the following questions in your report
  - What is the difference between "input [15:0] input_0" and "input [0:15] input_0" inside the module?
  - What is the meaning of "always" block in Verilog?
  - What are the advantages and disadvantages of port connection by order and port connection by name in Verilog?

```
// port connection by order example
foo bar(clock, rst, data, output);
// port connection by name example
foo bar(.d(data), .out(output), .rst(reset), .clk(clock));
```

- Contribution
  - If you finish the lab on your own. State "I finish all requirements on my own" in this section.
  - Otherwise, state the individual contribution in this lab. E.g. ALU module, test case generation, bug fixing, etc.
  - You can make a complaint if your teammate barely does anything.
- Discussions, problem encountered and miscellaneous
  - Anything you want to share, suggestions, etc.

III. Testing

- Compile your code with:

```
$ vvp simple_cpu.vvp
```

- Run test bench with:

```
$ iverilog -o simple_cpu.vvp *.v
```

  - If you get "WARNING: Test_Bench.v:22: $readmemb(<File>): Not enough words in the file for the requested range [0:31]". Don't worry, this is totally fine.
- Open waveform with

```
$gtkwave simple_cpu.vcd
```

- You can change the test data you want to load in Test_Bench.v:22

```
22: $readmemb("<test_data_name>", cpu.IM.Instr_Mem);
```

- The test data provided perform basic unit test, which checks one type of instruction only. You should create your own test data to test your design thoroughly. There will be hidden test data when TAs evaluate your design.
- Because the simulation will not stop until specified cycles (defined in Test_Bench.v END_COUNT). If you want to create your own test data, you should set END_COUNT to "<number_of_instructions_in_file> * 2 + 1" to prevent simulation continues to run after defined instructions ran out.
- MIPS defined 32 registers (r0 ~ r31), we only use 12 (r0 ~ r11) of them in this lab. In order to test each instruction, r0 ~ r11 has been initialized to 0 ~ 9, -1 and -2, and the rest of registers to 0. You can verify it by inspecting "Reg_File.v".
- Make sure your module can be tested with Test_Bench.v provided by us, or we cannot test your code.

IV. Grading

- Plagiarism is strictly prohibited. Including the source code you find on the Internet or inherited from others. Same applies to your report. You will get 0 points in this lab if you choose to do so. Despite that, we still encourage you to discuss with your classmates.

- Testing will be done on CentOS 7 with Icarus Verilog 10.2
- You will get points after you complete the following required items
  - CPU instructions (75 %)
    - R-type (40 %)
      - addu (4 %)
      - subu (4 %)
      - and (4 %)
      - or (4 %)
      - slt (8 %)
      - sra (8 %)
      - srav (8 %)
    - I-type (30 %)
      - addi (4 %)
      - sltiu (6 %)
      - beq (6 %)
      - lui (4 %)
      - ori (4 %)
      - bne (6 %)
    - Hidden test cases (5 %)
  - Report (25 %)
    - Architecture diagram (5 %)
    - Module description (5 %)
    - Waveform (5 %)
    - Questions (5 %)
    - Contribution (0 %, but we will evaluate your overall performance based on this)
    - Discussions (5 %)

V. Submission

- You need to compress the following files into a zip file and name it "<YourID>_<TeammateID>.zip", "<YourID>.zip" if you finish the lab on your own. Other archive formats are not acceptable.
  - e.g. "0123456_0654321.zip", for you and "0654321_0123456.zip" for your teammate. "0123456.zip" if you go solo.
  - All source code (*.v) excluding ProgramCounter.v, Instr_Memory.v, Reg_File.v, Test_Bench.v.
  - Do not include .vcd, .vvp files.
  - A report named "<YourID>.pdf" e.g. "0123456.pdf".
    - Reminder: You need to submit a report written by yourself. Even you finish this lab as a team.
- If you do not follow the policy above, you will get 10 points penalty.
- Upload the zip file to New E3 before 5/14 (Thu.) 23:55.
- You will get 15 points penalty per day for late submission. Any submission will be rejected after 5/18 (Mon.) 23:55.

VI. Contacts & Discussions

Please contact 許賀傑 (hchsu0426@cs.nctu.edu.tw) and 周煥然 (kulugu2@gmail.com) via New E3 if you have any problems.

Head to KCW's COVID (Computer Organization Via Interactive Distance-learning) Microsoft Teams, channel #lab2_discussions for open discussions.

VII. Hints & Suggestions

- Although not required, it is suggested to place one module per file.
- Use port connection by name to prevent wiring errors.
- Read the warning messages produced by the compiler, chances that solving the warning eliminate the bug.

- Inspecting waveform helps you debug. $display function can be helpful too.
- Use the website https://www.eg.bucknell.edu/~csci320/mips_web/ to convert MIPS assembly to machine code.
- Detailed MIPS reference sheet on https://inst.eecs.berkeley.edu/~cs61c/resources/MIPS_help.html
- You can change the data type in GTKWave, so you don't need to convert it yourself.
- You can use the free online tool https://draw.io to design your architecture diagram.
- The following architecture diagram shows how the modules interconnect. You should modify your architecture as this is not the final version of architecture of this lab.