Computer Organization
Lab 2: Single Cycle CPU - Simple Edition

Due: 2023/4/23

1. Goal

Utilizing the ALU in Lab1 to implement a simple single cycle CPU (Not required. See Appendix.). CPU is the most important unit in computer system. Read the document carefully and do the lab, and you will have elementary knowledge of a MIPS CPU.

2. Homework Requirement

a. Please use Vivado as your HDL simulator(if the execution result in your environment is different from ours, you need to bring your laptop to the lab and demo it to us).
b. Please **attach student IDs as comments** at the top of each file.
c. The file type of your report should be **PDF.**
d. Please add the files listed below into one directory named **"your_student_id"**, and **zip it as " your_student_id.zip"**.
   The file structure in this lab should be (for example, id=310551072):

```
310551072/
├── Adder.v
├── ALU_Ctrl.v
├── ALU.v
├── Decoder.v
├── MUX_2to1.v
├──Lab2_310551072.pdf
├── Shift_Left_Two_32.v
├── Sign_Extend.v
└── Simple_Single_CPU.v
```

e. Please **do not add unnecessary or given files**(like .txt, testbench.v, Instr_Memory.v, ProgramCounter.v, Reg_File.v) and folders (like .DS_Store, __MACOSX).
f. Program Counter, Instruction Memory, Register File and Testbench are given (don't need to modify these files).

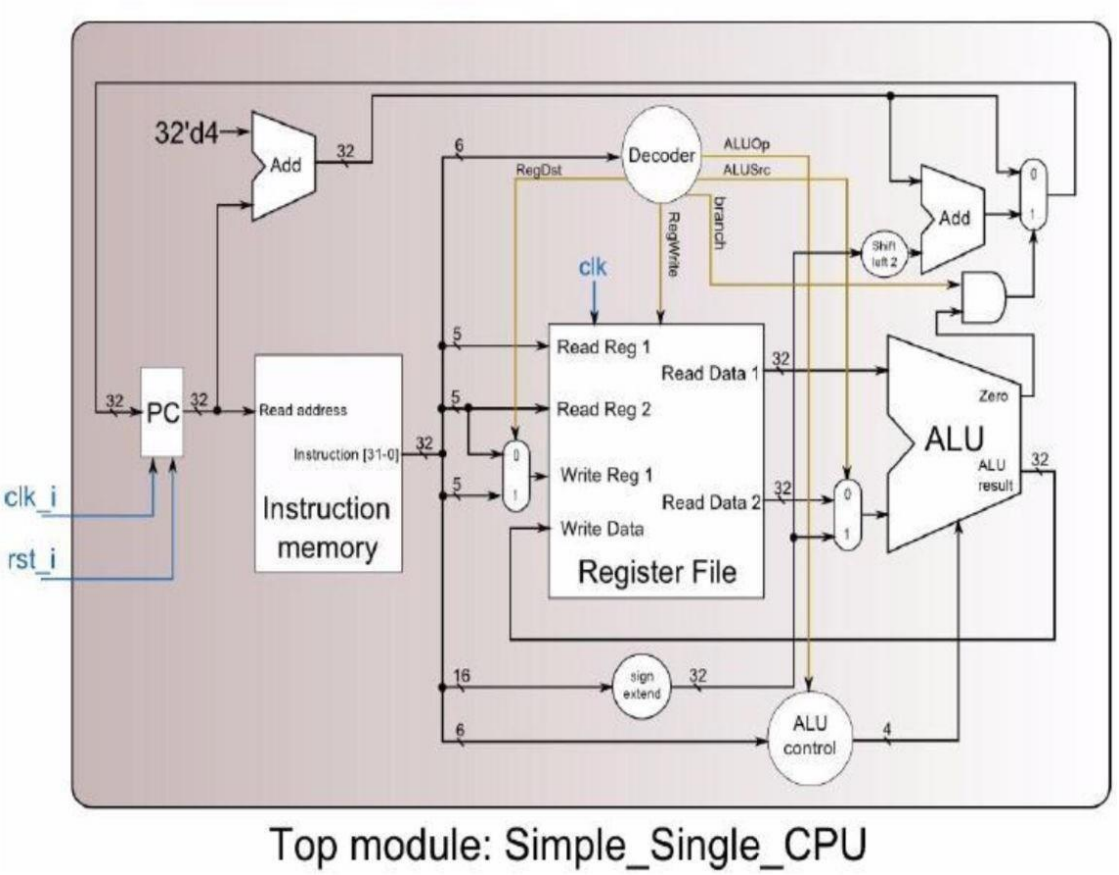
g. **Instruction set: the following instructions have to be executable in your CPU design, and we will use some hidden cases to further evaluate your design.**

Instruction	Name	Example	Meaning	Op Field	Function Field
add	Addition	add r1,r2,r3	r1=r2+r3	0	32 (0x20)
addi	Add Immediate	addi r1,r2,100	r1=r2+100	8	0
sub	Subtraction	sub r1,r2,r3	r1=r2-r3	0	34 (0x22)
and	Logic AND	and r1,r2,r3	r1=r2&r3	0	36 (0x24)
or	Logic OR	or r1,r2,r3	r1=r2|r3	0	37 (0x25)
slt	Set on Less Than	slt r1,r2,r3	if(r2<r3) r1=1 else r1=0	0	42 (0x2a)
slti	Set on Less Than Immediate	slti r1,r2,10	if(r2<10) r1=1 else r1=0	10 (0xa)	0
beq	Branch on Equal	beq r1,r2,25	if(r1==r2) goto PC+4+100	4	0

h. **Any work by fraud will absolutely get 0 point.**

i. **If you don’t follow the architecture diagram, you’ll get 0 point.**

3. Architecture Diagram



Top module: Simple_Single_CPU

Notice that every component in the diagram has a corresponding module(.v).

4. Test

There are 2 test patterns, CO_P2_test_data1.txt, CO_P2_test_data2.txt. You need to add these two files as simulation sources.

The default pattern is the first one. Please change the column 39 in the file “Instr_Memory.v” if you want to test another case:

```
$readmemb("CO_P2_test_data1.txt", Instr_Mem)
```

The following are the assembly code for the test patterns:

Case 1	Case 2
addi r1,r0,10	addi r6,r0,2
addi r2,r0,4	addi r7,r0,14
slt r3,r1,r2	and r8,r6,r7
beq r3,r0,1	or r9,r6,r7
add r4,r1,r2	addi r6,r6,-1
sub r5,r1,r2	slti r1,r6,1
	beq r1,r0,-5
Result	**Result**
r1=10 r2=4 r3=0 r4=0 r5=6	r6=0 r7=14 r8=0 r9=15 r1=1

We will show the execution results on the terminal.

Besides, the file ”CO_P2_Result.txt” will be generated after executing the testbench. You can also check your answer with it. The default path of ”CO_P2_Result.txt” will be “Your_project_directory/Your_project_name.sim/sim_1/behave/xsim/CO_P2_Result.txt”, like following

D:/Verilog/Lab2/Lab2.sim/sim_1/behave/xsim/CO_P2_Result.txt

(For the ones who can't read testcase txt files or dump result txt file, please change the relative path in testbench and instr_memory file to absolute path.) like following “C:/Users/Joinet/Downloads/CO_Lab2/CO_Lab2/testcase/CO_P2_test_data1.txt”

Remember to use forward slash.

5. Grade

a. **Total:** 100 points (plagiarism will get 0 point)
   - **Report:** 20 points
   - **Hardware design:** 80 points

b. **Late submission:** Score * 0.8 before 4/30. After 4/30, you will get 0.

c. **Wrong format:** 10 points punishment

6. Q&A

If you have any question, it is recommended to ask in the Facebook discussion forum.

7. Appendix

In this lab, you can use a behavioral 32-bit ALU. Here is the example of an 32-bit from the textbook:

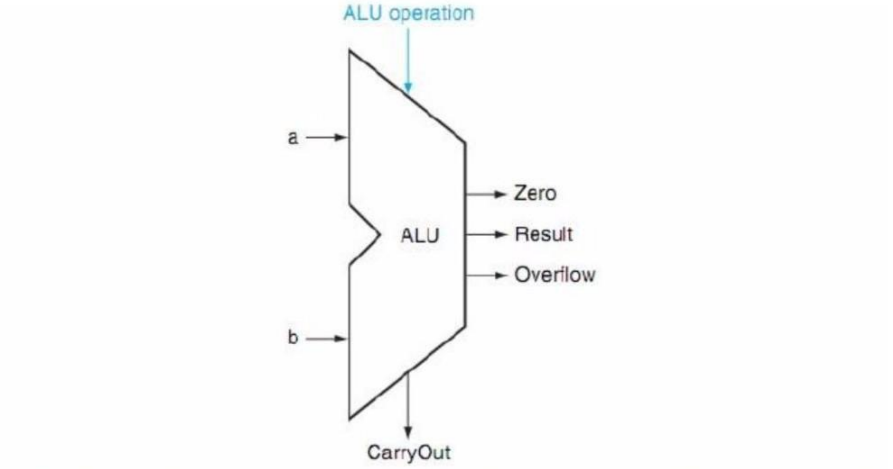


FIGURE C.5.14 The symbol commonly used to represent an ALU, as shown in Figure C.5.12. This symbol is also used to represent an adder, so it is normally labeled either with ALU or Adder.

```
module MIPSALU (ALUctl, A, B, ALUOut, Zero);
   input [3:0] ALUctl;
   input [31:0] A,B;
   output reg [31:0] ALUOut;
   output Zero;
   assign Zero = (ALUOut==0); //Zero is true if ALUOut is 0
   always @(ALUctl, A, B) begin //reevaluate if these change
      case (ALUctl)
         0: ALUOut <= A & B;
         1: ALUOut <= A | B;
         2: ALUOut <= A + B;
         6: ALUOut <= A - B;
         7: ALUOut <= A < B ? 1 : 0;
         12: ALUOut <= ~(A | B); // result is nor
         default: ALUOut <= 0;
      endcase
    end
endmodule
```

FIGURE C.5.15 A Verilog behavioral definition of a MIPS ALU.