Computer Organization

Lab 2: Single Cycle CPU – Simple Edition

Due : 2018/04/28

1. Goal

Utilizing the ALU in Lab1 to implement a simple single cycle CPU. CPU is the most important unit in computer system. Reading the document carefully and do the Lab, you will have elementary knowledge of CPU.

2. Demands

a. Please use ModleSim or Xilinx as your HDL simulator.

b. **Group member same as Lab1**. Just hand in one assignment for one group. Please attach your names and student IDs as comments in each of .v source code, unless you'll get 0 point.

c. Put .v source files and report into a compressed file. The compressed file you upload on E3 must have the form of "studentID.zip". (Ex. 0216310_0216077.zip or 0216310.zip)

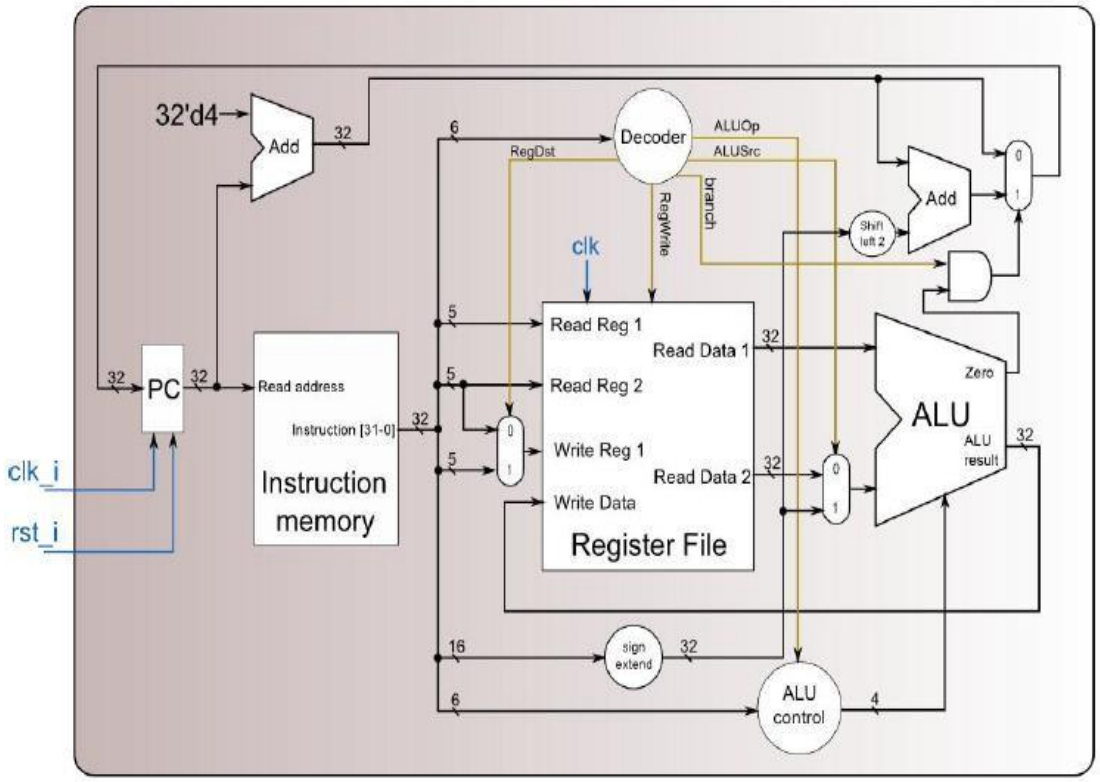
d. Program Counter, Instruction Memory, Register File and Test Bench are supplied.

e. Instruction set: the following instructions have to running in your designed CPU (80pts.).

f. 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.

Instruction	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) go to PC+4+100	4	0

3. Architecture Diagram



Top module: Simple_Single_CPU

4. Test

There are 2 test patterns, CO_P2_test_data1.txt, CO_P2_test_data2.txt.
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.

1	2

addi r1, r0, 10
addi r2, r0, 4
slt r3, r1, r2
beq r3, r0, 1
add r4, r1, r2.
sub r5, r1, r2
	addi r6, r0, 2
	addi r7, r0, 14
	and r8, r6, r7
	or r9, r6, r7
	addi r6, r6, -1
	slti r1, r6, 1
	beq r1, r0, -5

final result	final result
r1 = 10, r2 = 4, r3 = 0 r4 = 0, r5 = 6.	r6 = 0, r7 = 14, r8 = 0 r9 = 15 , r1 = 1

The file "CO_P2_Result.txt" will be generated after executing the Testbench.
Check your answer with it.

5. Grade

a. Total score: 100 pts. COPY WILL GET A 0 POINT!

b. Instruction score: 80 pts.

c. Report: 20pts – format is in CO_document.

d. TA will use Modelsim and the testbench that give it to you to test your code, please make sure your code can run on Modelsim and don't modify testbench. If your code cannot run on Modelsim with the testbench we give it to you, you'll get 0 point.

e. Put .v source files and report into a compressed file.The compressed file must be zip and named by your student ID (ex.0216310.zip for one person and 0216310_0216077.zip for two people).Don't put all project file into the compressed file,or you'll get 0 point. If you don't follow the rule to name the compressed file, also get 0 point. If your upload file isn't .zip, 0 point.

f. If you don't follow the architecture diagram, you'll get 0 point.

g. Late submission: Score*0.8 before 5/5. After 5/5, you will get 0 point.

6. Hand in your assignment

Please upload the assignment to the E3.

Put all of .v source files and report into same compressed file. (Use your student ID to be the name of your compressed file)

7. Q&A

If you have any question, just send email to TAs.

If you have code problem, only TA time permitted.

8. Appendix

In lab2, you can use 32bits ALU .

Here is the example of 32bits ALU from textbook

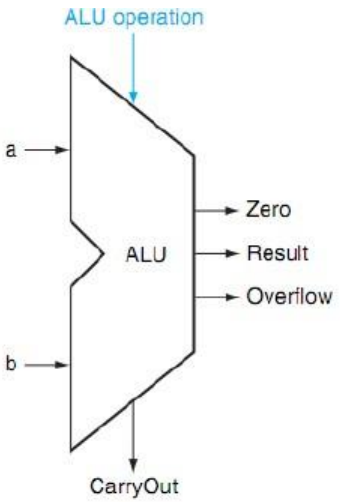


FIGURE C.5.14 The symbol commonly used to represent an ALU, as shown in Figure C.5.12. This symbol is also used to represent an adder, so it is normally labeled either with ALU or Adder.

```
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.