Computer Organization, Spring 2019

Lab 2: Single-Cycle CPU (Simple Version)

Due: 2019/5/2

1. Goal

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

2. Requirement

(1) Please use Icarus Verilog and GTKWave as your HDL simulator.

(2) Please attach your names and student IDs as comment at the top of each file.

(3) Please use the Program Counter, Instruction Memory, Register File and Test Bench we provide you.

(4) Instruction set: the following instructions are to run on your CPU (60 pts.).

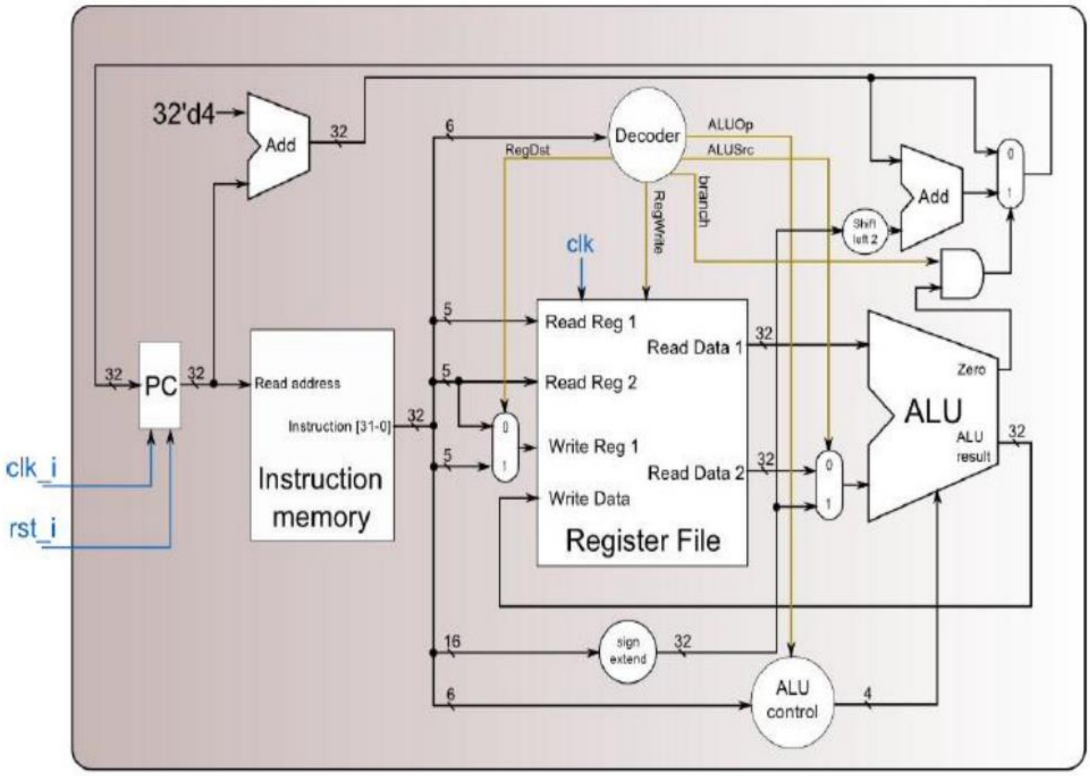
Instruction	Example	Meaning	Opcode	Function
Add unsigned	addu r1, r2, r3	r1 = r2 + r3	000 000	100 001
Add immediate	addi r1, r2, 100	r1 = r2 +100	001 000	-
Subtract unsigned	subu r1, r2, r3	r1 = r2 - r3	000 000	100 011
Bitwise and	and r1, r2, r3	r1 = r2 & r3	000 000	100 100
Bitwise or	or r1, r2, r3	r1 = r2 | r3	000 000	100 101

Set on less than	slt r1, r2, r3	if(r2 < r3)
		r1 = 1
		else
		r1 = 0
			000 000	101 010

Set on less than immediate unsigned	sltiu r1, r2, 10	if(r2 < 10)
		r1 = 1
		else
		r1 = 0
			001 011	-

Branch on equal	beq r1, r2, 25	if(r1 == r2)
		PC += (25 << 2)
			000 100	-

3. Architecture Diagram



Top module: Simple_Single_CPU

4. Advance Instructions (20 pts.)

Modify the architecture of the basic design above.

(1) ALUOp should be extended to 3bits to implement I-type instructions. Original 2bits ALUOp from textbook : 00 -> 000, 01 -> 001, 10 -> 010.

(2) Encode shift right and LUI instruction by using unused ALU_ctrl. Ex. ALU_ctrl = 0 is AND, 1 is OR..., 0 1 2 6 7 &12 are used by basic instructions.

Instruction	Example	Meaning	Opcode	Function
Shift right arithmetic	sra r1, r2, 10	r1 = r2 >> 10	000 000	000 011
Shift right arithmetic variable	srav r1, r2, r3	r1 = r2 >> r3	000 000	000 111
Load upper immediate	lui r1, 10	r1 = 10 << 16	001 111	-
Or immediate	ori r1, r2, 100	r1 = r2 | 100	001 101	-

Branch on not equal	bne r1, r2, 30	if(r1 != r2)
		PC += (30 << 2)
			000 101	-

To implement those advanced instructions, please note about the following formats.

SRA Rd, Rt, shamt

0	-	Rt	Rd	shamt	3
6	5	5	5	5	6

Shift register Rt right arithmetically by the distance indicated by immediate shamt. Rs is ignored for sra.

SRAV Rd, Rt, Rs

0	Rs	Rt	Rd	0	7
6	5	5	5	5	6

Shift register Rt right arithmetically by the distance indicated by the register Rs. Hint: Be careful of using Verilog operator >>> directly in your code. To use this operator, you have to declare the variable as signed.

LUI Rt, Imm

0xf	0	Rt	Imm
6	5	5	16

Load the lower halfword of the immediate imm into the upper halfword of register Rt. The lower bits of the register are set to 0.

ORI Rt, Rs, Imm

0xd	Rs	Rt	Imm
6	5	5	16

Put the logical OR of register Rs and the zero-extended immediate into register Rt.

5. Test

There are 3 test patterns, CO_P2_test_data1.txt ~ CO_P2_test_data3.txt. The default pattern is the first one. Please change the column 39 in the file "Instr_Memory.v" if you want to test the other cases.

column 39 : $readmemb("CO_P2_test_data1.txt", Instr_Mem)

The following are the assembly code for the test patterns.

1	2	3
addi r1,r0,13 addi r2,r0,7 sltiu r3,r1,0xFFFF beq r3,r0,1 slt r4,r2,r1 and r5,r1,r4 subu r4,r1,r5	addi r6,r0,-2 addi r7,r0,5 or r8,r6,r7 addi r9,r0,-1 addi r6,r6,2 addu r9,r9,r6 beq r6,r0,-3	ori r10,r0,3 lui r11,-10 sra r11,r11,8 srav r11,r11,r10 addi r10,r10,-1 bne r10,r0,-3
final result	**final result**	**final result**

r1 = 13, r2 = 7, r3 = 1,
r4 = 12, r5 = 1
	r6 = 2, r7 = 5,
	r8 = -1, r9 = 1
		r10 = 0, r11 = -40

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

6. Grade

(1) Total score: 100 pts. COPY WILL GET A 0 POINT!

(2) Basic score: 60 pts. Advance instructions: 20 pts.

(3) Report: 20 pts – format is in CO_Report.docx.

(4) Delay: 10 pts off per day

7. Hand in your assignment

(1) Zip your folder and name it as "ID1_ID2.zip" (e.g., 0616001_0616002.zip) before uploading to New e3. Other filenames and formats such as *.rar and *.7z are NOT accepted! Multiple submissions are accepted, and the version with the latest time stamp will be graded.

(2) Please include ONLY Verilog source codes (*.v) and your report (0616xxx.pdf) in the zipped folder.

8. Q&A

For any questions regarding Lab 2, please contact 曾威凱 (k50402k@gmail.com) and 周煥然 (kulugu2@gmail.com).

9. Appendix

You can use 32bits ALU to do this lab.

Here is the example of 32bits ALU from textbook.

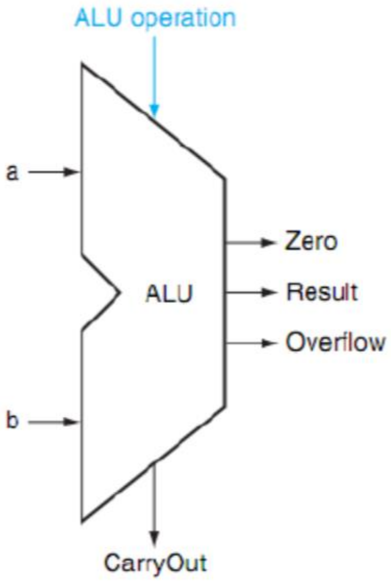
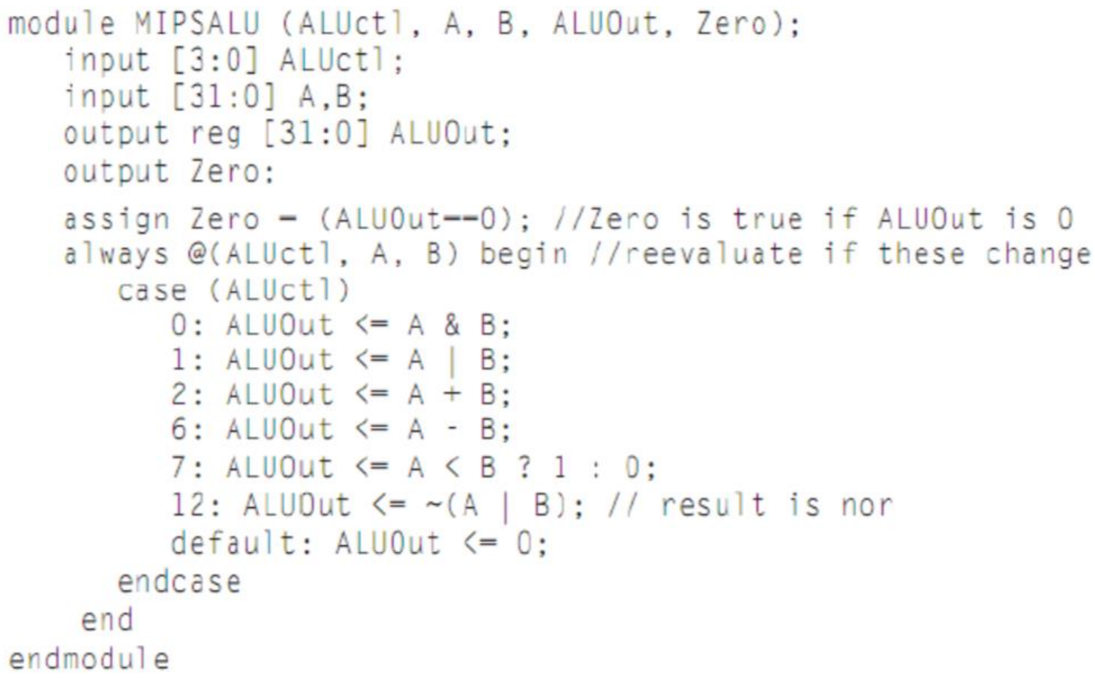


FIGURE C.5.14 The symbol commonly used to represent an ALU, as shown in Figure C.5.12. This symbol is also used to represent an adder, so it is normally labeled either with ALU or Adder.



```
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.