

# Computer Organization, Spring 2018

## Lab 3: Single Cycle CPU –Simple Edition

**Due : 2018/04/30**

### 1. Goal

Utilizing the ALU & Shifter in Lab2 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** as your HDL simulator.
- B. **One person forms a group.** Please use your student ID as your file name. (Ex. Lab3\_0416001.zip) The type of compressed file must be “**zip**”.
- Other form of file will get -10%.**

The assignment you upload on E3 must have the form of "**Lab3\_student ID.zip**".

- C. “Simple\_Single\_CPU.v”, “Adder.v”, “ALU.v”, “ALU\_Ctrl.v”, “Decoder.v”, “Instr\_Memory.v”, “Mux2to1.v”, “Mux3to1.v”, “Program\_Counter.v”, “Reg\_File.v”, “Shifter.v”, “Sign\_Extend.v”, “Zero\_Filled.v”, and “TestBench.v” are supplied. Please use these modules to accomplish the design of your CPU. You may create additional module(.v file) for your design, if necessary.

In Lab3, the complete modules of “Instr\_Memory.v”, “Program\_Counter.v”, “Reg\_File.v” are provided. You should accomplish other modules to make this CPU work.

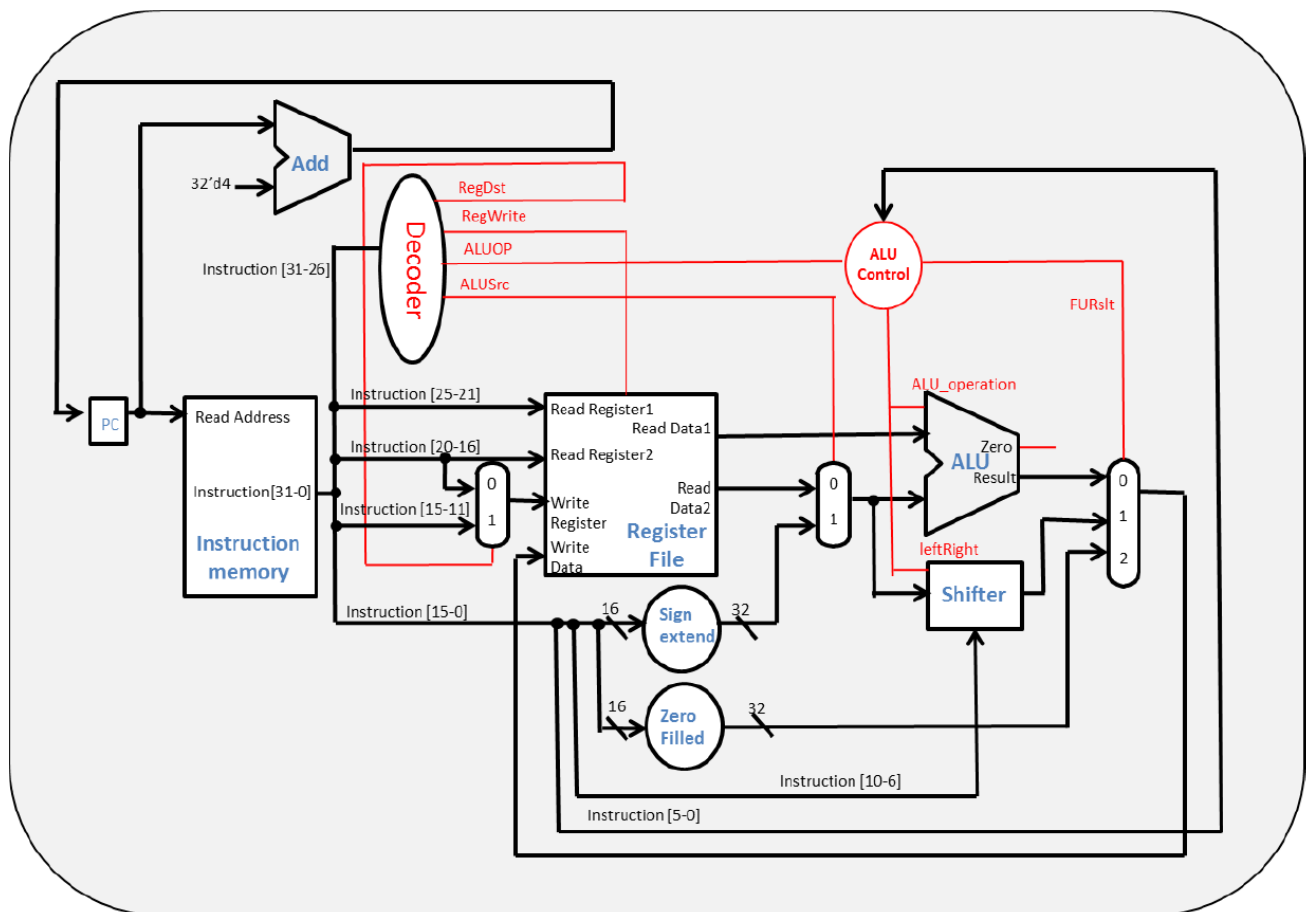
- D. Instruction set: the following instructions have to running in your designed CPU(**80pts.**).

Instruction	Example	Meaning	Op field	shamt	Function field
ADD	add r1,r2,r3	$r1 = r2 + r3$	6'b111111	X	18(6'b010010)
SUB	sub r1,r2,r3	$r1 = r2 - r3$	6'b111111	X	16(6'b010000)
AND	and r1,r2,r3	$r1 = r2 \& r3$	6'b111111	X	20(6'b010100)
OR	or r1,r2,r3	$r1 = r2   r3$	6'b111111	X	22(6'b010110)
NOR	nor r1,r2,r3	$r1 = \sim(r2   r3)$	6'b111111	X	21(6'b010101)

SLT	slt r1,r2,r3	if(r2<r3) r1=1 else r1=0	6'b111111	X	32(6'b100000)
SLL	sll rd,rt,5	rd=rt<<5	6'b111111	5'd5	0(6'b000000)
SRL	srl rd,rt,5	rd=rt>>5	6'b111111	5'd5	2(6'b000010)
ADDI	addi r1,r2,10	r1=r2+10	6'b110111	X	X
LUI	lui r1,100	r1=100*2^16	6'b110000	X	X

Table 1: Instruction definition and example

### 3. Architecture Diagram



### C. Bonus

Additional functions for ALU and shifter are described in the following subsection with bonus.

Implement sllv and srlv instructions.

You can add new module or control signal in this advanced design.

Instruction	Example	Meaning	Op field	shamt	Function field
SLLV	sllv rd,rt,rs	rd = rt<<rs	6'b111111	X	6(6'b000110)
SRLV	srlv rd,rt,rs	rd = rt>>rs	6'b111111	X	4(6'b000100)

#### 4. Test Bench

After you hand in your code, TA will use the same TestBench module and different test data to verify the correctness of your design of Simple\_Single\_CPU.

If you have attached additional modules for your design, do ensure that those modules would not affect our testing.

In Lab3, three test data (binary code), stored in “CO\_P3\_test\_data1.txt”~ “CO\_P3\_test\_data3.txt”, are provided. The default test data is the first one. If you would like to use second test data, modify the module “TestBench.v” in line 75 as follows:

```
$readmemb("CO_P3_test_data2.txt", cpu.IM.Instr_Mem);
```

Use the similar way to check the third test data “CO\_P3\_test\_data3.txt”. You can design other test data by yourself to verify the correctness of your design.

The Assembly codes of the test data are given as follows:

CO_P3_test_data1.txt	CO_P3_test_data2.txt	CO_P3_test_data3.txt
addi r1 r0 10    # r1 = 10	addi r6 r0 3     # r6 = 3	addi r1 r0 -5    # r1 = -5
addi r2 r0 4     # r2 = 4	addi r7 r0 14    # r7 = 14	addi r2 r0 5     # r2 = 5
slt r3 r1 r2     # r3 = 0	and r8 r6 r7     # r8 = 2	slt r3 r1 r2     # r3 = 1
add r4 r1 r2     # r4 = 15	or r9 r6 r7      # r9 = 15	slt r4 r2 r1     # r4 = 0
sub r5 r1 r2     # r5 = 6	sll r10 r9 3     # r10 = 120	add r5 r1 r2     # r5 = 0
nor r5 r5 r0     # r5 = -7	lui r11 10       # r11 = 65536	sub r6 r1 r2     # r6 = -10
	srl r12 r11 5    # r12 = 2048	lui r7 -5        # r7 = -327680

After the simulation of TestBench, you will get the file "CO\_P3\_result.txt".

You can verify the result.

If your design pass the test data, the following words would show in the Transcript windows.

```

Transcript
# ** warning: (vsim-3015) C:/Modeltech_pe_edu_10.3/examples/CO_Lab3/Simple_Single_CPU.v(126): [PCDFC] -
definition is at: C:/Modeltech_pe_edu_10.3/examples/CO_Lab3/Mux2to1.v(1).
#
#       Region: /TestBench/cpu/shiftershamt
add wave -position insertpoint sim:/TestBench/cpu/*
VSIM3> run -all
# =====
#   Congratulation. You pass  TA's pattern
# =====
# ** Note: $stop      : C:/Modeltech_pe_edu_10.3/examples/CO_Lab3/TestBench.v(263)
#   Time: 130 ns  Iteration: 0  Instance: /TestBench
# Break in Module TestBench at C:/Modeltech_pe_edu_10.3/examples/CO_Lab3/TestBench.v line 263
VSIM4>
Ln: 263 Col: 0  READ  Project : CO_Lab3  Now: 130 ns  Delta: 0  /TestBench/cpu/Shifter_result

```

## 5. Grade

- Total score: 110pts. **COPY WILL GET A 0 POINT!**
- Instruction score: 80 pts.
- Report: 20pts –format is in StudentID\_report
- Bonus: 10pts

## 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 **Lab3\_student ID.zip** to be the name of your compressed file)

## 7. Q&A

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

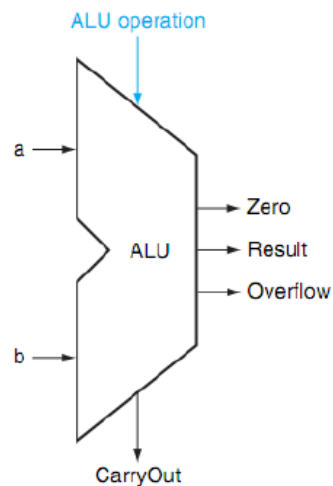
## 8. Notice

- Use the modules provided to implement your Single Cycle CPU
- Do not** modify any existing code except the **Input filename** in TestBench.v
- Write your code in "/\*your code here\*/"
- In Lab3, the modules to be designed by students should be implemented as **combination circuits**. (Do not design as sequential circuits or Zero score will be given!)

## 9. Appendix

In lab3, 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.