Computer Organization, Spring 2019

Lab 2：Single Cycle CPU– Simple Edition

Due : 2019/05/09

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 ModelSim as your HDL simulator.

B. Put all of *.v source files and report (*.pdf) into the same compressed file. Please use your student ID as your file name (e.g., 0616001_0616002.zip). The type of compressed file must be ‘zip’.

C. For the ease of grading, a team's submissions should be uploaded by only one person.

D. “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 cannot create additional source files (.v file) for your design.

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

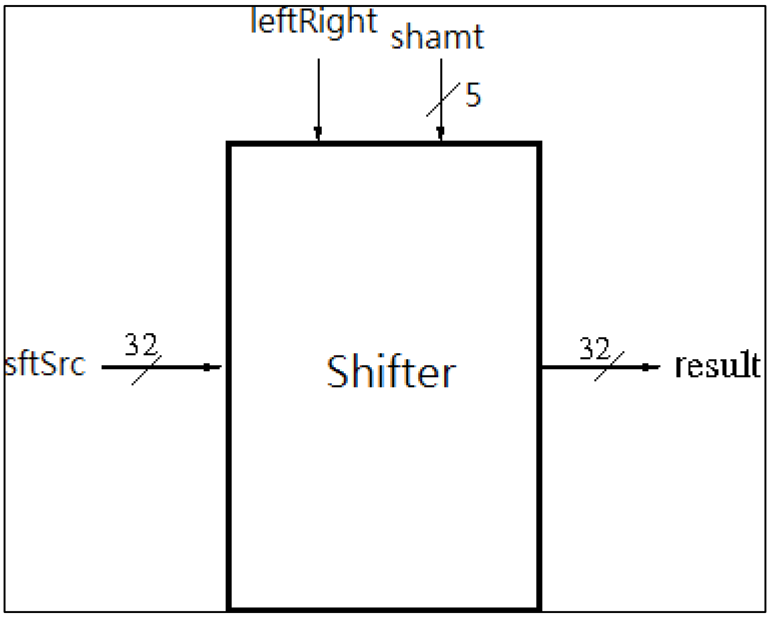
Instruction	Example	Meaning	Op field	shamt	Function field
ADD	add r1,r2,r3	r1 = r2+r3	6’b111111	X	18(6’b010010)
SUB	sub r1,r2,r3	r1 = r2-r3	6’b111111	X	16(6’b010000)
AND	and r1,r2,r3	r1=r2&r3	6’b111111	X	20(6’b010100)
OR	or r1,r2,r3	r1=r2|r3	6’b111111	X	22(6’b010110)
NOR	nor r1,r2,r3	r1=~(r2|r3)	6’b111111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

Shifter

The block diagram of the shifter to be implemented is shown:



Shift.v contains the following inputs and outputs:

- sftSrc: A 32-bit input data, is the source data of the shifter.
- leftRight: A 1-bit input control signal. When it is set to 1, the shifter perform logical left shift; else, does logical right shift.
- shamt: A 5-bit input data, represents the number of bit positions to be shifted.
- result: A 32-bit output data, which represents the shifting result of the shifter.

SRL Rd, Rt, shamt (Rs is ignored for SRL)

Shift register Rt right by the distance indicated by immediate shamt

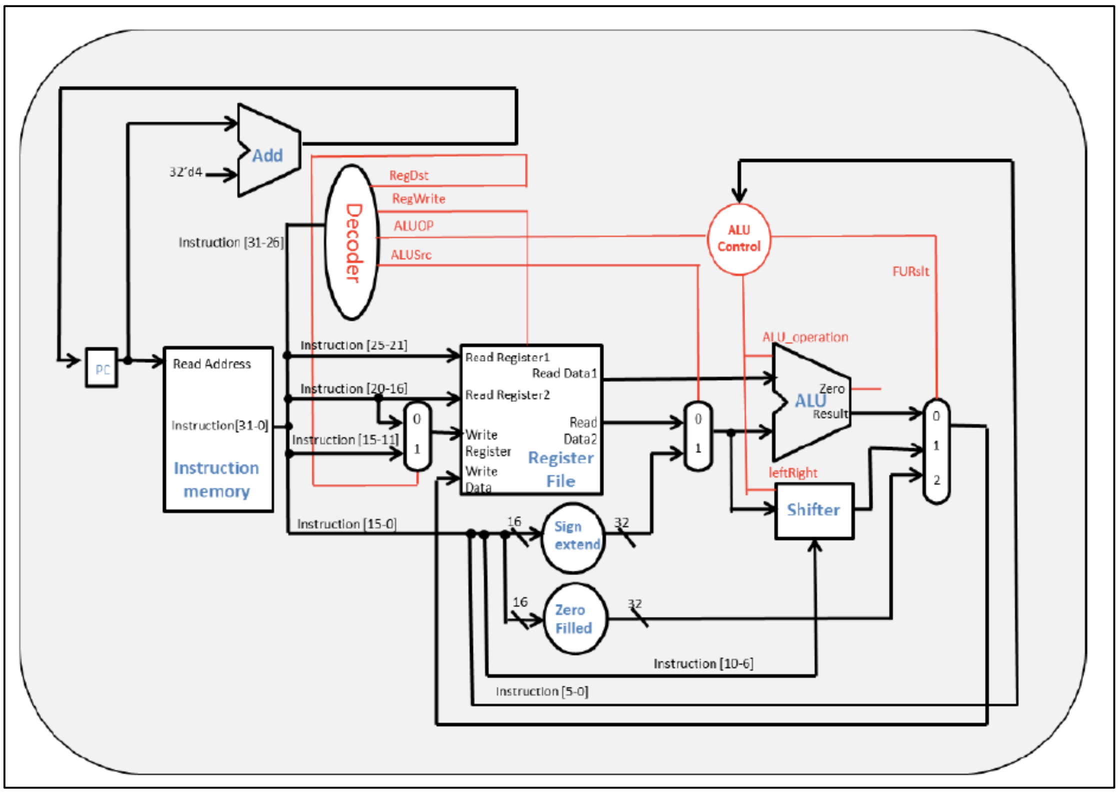
0	Rs	Rt	Rd	shamt	2

LUI Rt, Imm

The immediate value is shifted left 16 bits and stored in the register Rt. The lower 16 bits are zeroes.

0xf	0	Rt	Imm

3. Architecture Diagram



4. Bonus(10pts.)

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)

SRLV Rd, Rt, Rs
Shift register Rt right by the distance indicated by the register Rs

0	Rs	Rt	Rd	shamt	6

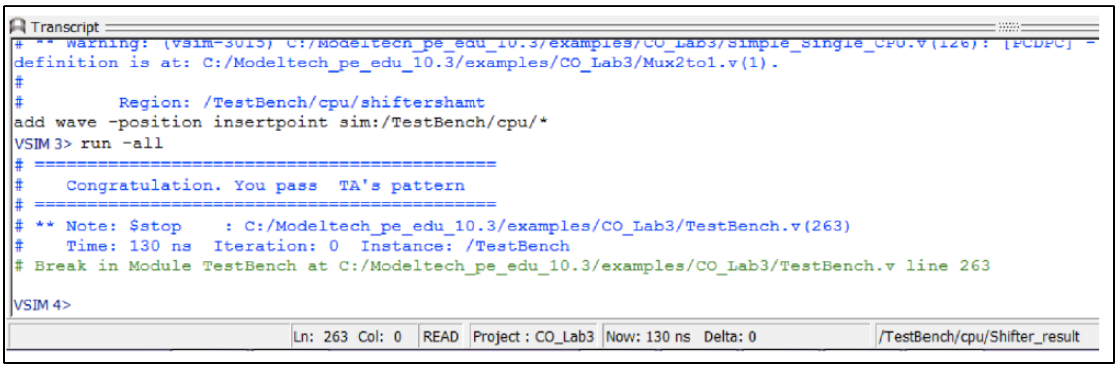
5. Test Bench

In Lab2, three test data (binary code), stored in "CO_P2_test_data1.txt" ~ "CO_P2_test_data3.txt", are provided. The default test data is the first one. If you would like to use second test data, modify line 75 in the file "TestBench.v" as follows:
$readmemb("CO_P2_test_data2.txt", cpu.IM.Instr_Mem);

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

CO_P2_test_data1.txt	CO_P2_test_data2.txt	CO_P2_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_P2_result.txt". You can verify the result. If your design passes the test data, the following words would show in the Transcript windows.



```
# ** Warning: (vsim-3015) C:/Modeltech_pe_edu_10.3/examples/CO_Lab3/Simple_Single_CPU.v(126): [PCDPC] -
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/*
VSIM 3> 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

VSIM 4>
```

```
Ln: 263  Col: 0 | READ | Project : CO_Lab3 | Now: 130 ns  Delta: 0 | /TestBench/cpu/Shifter_result
```

6. Grade

A. Total score: 110pts. COPY WILL GET A 0 POINT!
B. Instruction score: 80 pts.
C. Report: 20pts – format is in StudentID_report
D. Bonus: 10pts
E. Late Submission: 10 points off per day

7. Q&A

If you have any question, just send email to all TAs via new E3 platform.

8. Notice

A. Use the modules provided to implement your Single Cycle CPU
B. Do not modify any existing code except the input filename in TestBench.v
C. Write your code in "/*your code here*/"
D. In Lab2, 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 Lab2, you can use 32bits ALU.
Here is the example of 32bits ALU from textbook

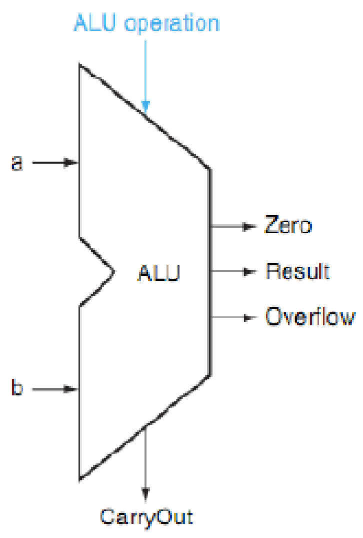


FIGURE C.5.14 The symbol commonly used to represent an ALU, as shown in Figure C.5.12. This symbol is also used to represent an adder, so it is normally labeled either with ALU or Adder.

```
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.