Computer Organization, Spring 2020

Lab 3: Single-Cycle CPU (Simple Version)

Due: 2020/05/14

1. Goal

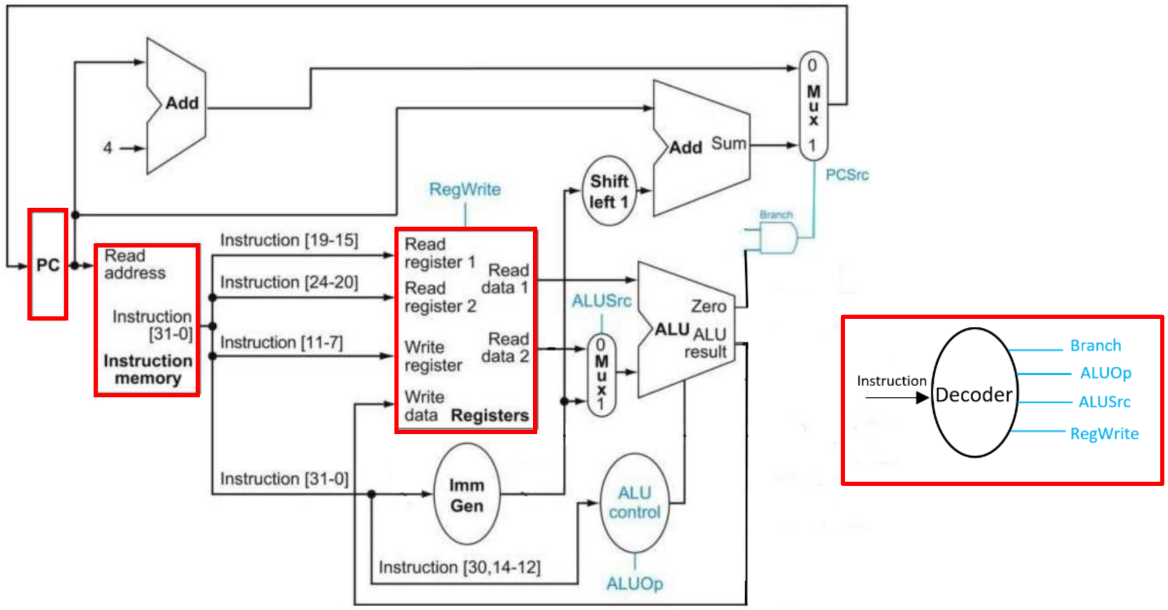
Utilizing the ALU in Lab2 to implement a Simple Single-Cycle CPU. CPU is the most important unit in computer system. Read the document carefully and do the Lab, and you will have the elementary knowledge of CPU.

2. HW Requirement

(1) Please use ModelSim/ISE as you 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 Testbench we provide you.
(4) Basic instruction set (60%)

Instruction	Example	Meaning	Opcode	Funct3	Funct7
Addition	add r1, r2, r3	r1 = r2 + r3	0110011	000	0000000
Add immediate	addi r1, r2, 100	r1 = r2 + 100	0010011	000	-
Subtraction	sub r1, r2, r3	r1 = r2 - r3	0110011	000	0100000
Bitwise and	and r1, r2, r3	r1 = r2 & r3	0110011	111	0000000
Bitwise or	or r1, r2, r3	r1 = r2 | r3	0110011	110	0000000
Exclusive OR	xor r1, r2, r3	r1 = r2 ⊕ r3	0110011	100	0000000
Set on less than	slt r1, r2, r3	if(r2 < r3) r1 = 1 else r1 = 0	0110011	010	0000000
Set on less than immediate	slti r1, r2, 10	if(r2 < 10) r1 = 1 else r1 = 0	0010011	010	-
Branch on equal	beq r1, r2, 4	If(r1 == r2) PC += 4	1100011	000	-

3. Architecture Diagram



4. Advanced set (20%)

Instruction	Example	Meaning	Opcode	Funct3	Funct7
Shift left logical	sll r1, r2, r3	r1 = r2 << r3	0110011	001	0000000
Shift left logical immediate	slli r1, r2, 10	r1 = r2 << 10	0010011	001	0000000
Shift right arithmetic	sra r1, r2, r3	r1 = r2 >>> r3	0110011	101	0100000
Shift right logical immediate	srli r1, r2, 10	r1 = r2 >> 10	0010011	101	0000000
Bitwise and immediate	andi r1, r2, 10	r1 = r2 & 10	0010011	111	-
Bitwise or immediate	ori r1, r2, 10	r1 = r2 | 10	0010011	110	-
Branch on not equal	bne r1, r2, 4	if(r1 != r2) PC += 4	1100011	001	-

5. Testbench

There contain 3 test pattern, CO_test_data1.txt ~ CO_test_data3.txt.
The default pattern is the first one. Please edit the line 19 in the file “Instr_Memory.v” to test the other cases.
Line 19: $readmemb("CO_test_data1.txt", instruction_file);

```
16 initial begin
17     for ( i=0; i<32; i=i+1 )
18         instruction_file[i] = 32'b0;
19     $readmemb("CO_test_data1.txt", instruction_file);
20 end
```

The following are the assembly code for the test pattern:

Test data 1	Test data 2	Test data 3

addi r1, r0,21
addi r2, r0, 9
slti r3, r1, 0xFF
beq r3, r0, 12
slt r4, r2, r1
and r5, r1, r4
sub r4, r1, r5
	addi r6, r0, 2
	addi r7, r0, 5
	or r8, r6, r7
	xor r9, r0, r8
	beq r8, r9, 8
	addi r6, r6, 2
	add r9, r9, r6
		ori r10, r0, 4
		addi r11, r0, 63
		sra r11, r11, r10
		sll r11, r11, r11
		slli r10, r10, 2
		addi r11, r11, 8
		srli r10, r10, 3
		srli r11, r11, 2
		bne r11, r10, -4

Final result	**Final result**	**Final result**
r1 = 21, r2 = 9, r3 = 1, r4 = 20, r5 = 1	r6 = 2, r7 = 5, r8 = 7, r9 = 9	r10 = 2, r11 = 2

“CO_Result.txt” will be generated after Run -All the testbench. Check you answer with it.

6. Grade

(1) Basic instructions score: 60 points.
(2) Advance instructions score: 20 points.
(3) Report: 20 points – format is in CO_Report.docx.
(4) Late submission: 10 percent penalty per day
(5) No plagiarism, or you will get 0 point.

7. Hand in

(1) Zip your folder and name it as “GroupID_ID1_ID2.zip” (e.g. G1_0816001_0816002.zip) before uploading to newe3. 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 (*.docx or *.pdf) in the zipped folder.

8. Q&A

For any questions regarding Lab 3, please contact
張祐銘 yumingchang.cs03@g2.nctu.edu.tw
賴柏宏 bhbruce.cs07g@nctu.edu.tw
鄭俊賢 petertay1996.cs08g@nctu.edu.tw

9. References

<table>
<tr><td>31</td><td>30 … 25</td><td>24 … 21</td><td>20</td><td>19 … 15</td><td>14 … 12</td><td>11 … 8</td><td>7</td><td>6 … 0</td><td></td></tr>
<tr><td colspan="2">funct7</td><td colspan="2">rs2</td><td>rs1</td><td>funct3</td><td colspan="2">rd</td><td>opcode</td><td>R-type</td></tr>
<tr><td colspan="4">imm[11:0]</td><td>rs1</td><td>funct3</td><td colspan="2">rd</td><td>opcode</td><td>I-type</td></tr>
<tr><td colspan="2">imm[11:5]</td><td colspan="2">rs2</td><td>rs1</td><td>funct3</td><td colspan="2">imm[4:0]</td><td>opcode</td><td>S-type</td></tr>
<tr><td>imm[12]</td><td>imm[10:5]</td><td colspan="2">rs2</td><td>rs1</td><td>funct3</td><td>imm[4:1]</td><td>imm[11]</td><td>opcode</td><td>B-type</td></tr>
<tr><td colspan="6">imm[31:12]</td><td colspan="2">rd</td><td>opcode</td><td>U-type</td></tr>
<tr><td>imm[20]</td><td colspan="2">imm[10:1]</td><td>imm[11]</td><td colspan="2">imm[19:12]</td><td colspan="2">rd</td><td>opcode</td><td>J-type</td></tr>
</table>

ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract

XOR
ALU_ctrl: 0111
ALUop: 10

Instruction opcode	ALUOp	Operation	Funct7 field	Funct3 field	Desired ALU action	ALU control input
ld	00	load doubleword	XXXXXXX	XXX	add	0010
sd	00	store doubleword	XXXXXXX	XXX	add	0010
beq	01	branch if equal	XXXXXXX	XXX	subtract	0110
R-type	10	add	0000000	000	add	0010
R-type	10	sub	0100000	000	subtract	0110
R-type	10	and	0000000	111	AND	0000
R-type	10	or	0000000	110	OR	0001

ALUOp		Funct7 field							Funct3 field			
ALUOp1	ALUOp0	I[31]	I[30]	I[29]	I[28]	I[27]	I[26]	I[25]	I[14]	I[13]	I[12]	Operation
0	0	X	X	X	X	X	X	X	X	X	X	0010
X	1	X	X	X	X	X	X	X	X	X	X	0110
1	X	0	0	0	0	0	0	0	0	0	0	0010
1	X	0	1	0	0	0	0	0	0	0	0	0110
1	X	0	0	0	0	0	0	0	1	1	1	0000
1	X	0	0	0	0	0	0	0	1	1	0	0001

FIGURE 4.13 The truth table for the 4 ALU control bits (called Operation).

Instruction	ALUSrc	Memto-Reg	Reg-Write	Mem-Read	Mem-Write	Branch	ALUOp1	ALUOp0
R-format	0	0	1	0	0	0	1	0
ld	1	1	1	1	0	0	0	0
sd	1	X	0	0	1	0	0	0
beq	0	X	0	0	0	1	0	1

FIGURE 4.18 The setting of the control lines is completely determined by the opcode fields of the instruction.