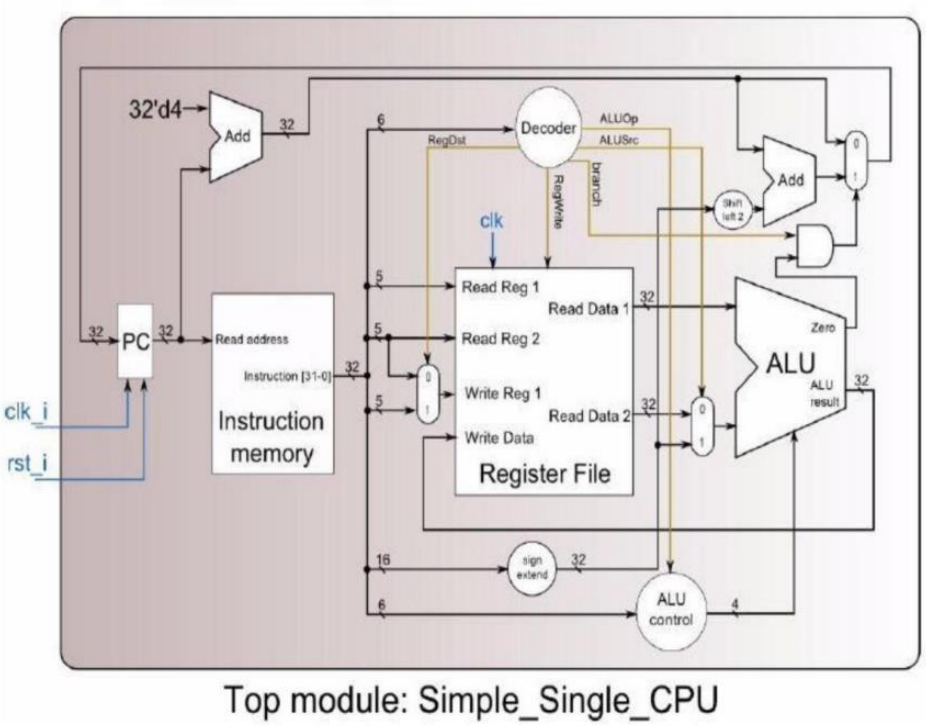
110550130 劉秉驊

Computer Organization

Architecture diagrams:



Top module: Simple_Single_CPU

Hardware module analysis:

I followed the architecture diagram and designed the simple cpu. As we can see, according to the R or I format that instruction memory found out, register file would decide to use read register 2 or write register 1.

If there was an R format, cpu also ran through ALU control to output the 4 bit signal controlling ALU. As for I format, cpu would use sign extend to handle the address as constant, and if cpu needed to branch, then it shifted left twice and added with the output of the first adder as the input of PC in next round. With no branch, ALU result would feed back to register file. If cpu needn’t to branch, PC would get the next memory address for the next round.

Instruction	Opcode	Sh_B	S_zext	Regwrite	Regdst	Alusrc	Branch	Brchne	Blez	Bltz	Bgtz	Memwrite	Memtoreg	Jump	Jal	ALUop	hlt
R_type	000000	xx	x	1	01	00	0	0	0	0	0	0	00	0	0	110	0
lw	100011	xx	x	1	00	01	0	0	0	0	0	0	01	0	0	000	0
sw	101011	11	x	0	xx	01	0	0	0	0	0	1	xx	0	0	000	0
beq	000100	xx	x	0	xx	00	1	0	0	0	0	0	xx	0	0	001	0
bne	000101	xx	x	0	xx	00	0	1	0	0	0	0	xx	0	0	001	0
blez	000111	xx	x	0	xx	00	0	0	1	0	0	0	xx	0	0	001	0
bltz	000001	xx	x	0	xx	00	0	0	0	1	0	0	xx	0	0	001	0
bgtz	000110	xx	x	0	xx	00	0	0	0	0	1	0	xx	0	0	001	0
addi	001000	xx	x	1	00	01	0	0	0	0	0	0	00	0	0	000	0
addiu	001001	xx	x	1	00	01	0	0	0	0	0	0	00	0	0	000	0
j	000010	xx	x	0	xx	xx	x	x	x	x	x	0	xx	1	0	xxx	0
jal	000011	xx	x	1	10	xx	x	x	x	x	x	0	xx	1	1	xxx	0
andi	001100	xx	x	1	00	10	0	0	0	0	0	0	00	0	0	010	0
ori	001101	xx	x	1	00	10	0	0	0	0	0	0	00	0	0	011	0
xori	001110	xx	x	1	00	10	0	0	0	0	0	0	00	0	0	100	0
slti	001010	xx	x	1	00	01	0	0	0	0	0	0	00	0	0	101	0
sltiu	001011	xx	x	1	00	01	0	0	0	0	0	0	00	0	0	101	0
lui	001111	xx	x	1	00	11	0	0	0	0	0	0	00	0	0	000	0
lb	100000	xx	0	1	00	01	0	0	0	0	0	0	11	0	0	000	0
lbu	100100	xx	1	1	00	01	0	0	0	0	0	0	11	0	0	000	0
lh	100001	xx	0	1	00	01	0	0	0	0	0	0	10	0	0	000	0
lhu	100101	xx	1	1	00	01	0	0	0	0	0	0	10	0	0	000	0
sb	101000	00	x	0	xx	01	0	0	0	0	0	1	xx	0	0	000	0
sh	101001	01	x	0	xx	01	0	0	0	0	0	1	xx	0	0	000	0
hlt	111100	xx	x	0	xx	xx	x	x	x	X	x	0	xx	x	x	xxx	1

The picture above was used to design my decoder, which was found from https://www.researchgate.net/figure/main-control-truth-table_tbl2_317490993 .

Finished part:

Testcase 1:

```
    );

//I/O ports
input  [32-1:0]  pc_addr_i;
output [32-1:0]  instr_o;

//Internal Signals
reg    [32-1:0]  instr_o;
integer          i;

//32 words Memory
reg    [32-1:0]  Instr_Mem [0:32-1];

//Parameter

//Main function
always @(pc_addr_i) begin
    instr_o = Instr_Mem[pc_addr_i/4];
end

//Initial Memory Contents
initial begin
    for ( i=0; i<32; i=i+1 )
        Instr_Mem[i] = 32'b0;
    $readmemb("C:/Users/user/Desktop/Verilog/Lab2_110550130/Lab2_110550130.sim/sim_1/behav/xsim/CO_P2_test_data1.txt", Instr_Mem);  //Read instruction f

end
```

```
CO_P2_Re
檔案  編輯  檢視

r0=          0
r1=         10
r2=          4
r3=          0
r4=          0
r5=          6
r6=          0
r7=          0
r8=          0
r9=          0
r10=         0
r11=         0
r12=         0

第 1 行，第 1  100%  Windows (CRLF)  UTF-8
```

Testcase 2:

```
    );

//I/O ports
input  [32-1:0]  pc_addr_i;
output [32-1:0]  instr_o;

//Internal Signals
reg    [32-1:0]  instr_o;
integer          i;

//32 words Memory
reg    [32-1:0]  Instr_Mem [0:32-1];

//Parameter

//Main function
always @(pc_addr_i) begin
    instr_o = Instr_Mem[pc_addr_i/4];
end

//Initial Memory Contents
initial begin
    for ( i=0; i<32; i=i+1 )
        Instr_Mem[i] = 32'b0;
    $readmemb("C:/Users/user/Desktop/Verilog/Lab2_110550130/Lab2_110550130.sim/sim_1/behav/xsim/CO_P2_test_data2.txt", Instr_Mem);  //Read instruction fro

end
endmodule
```

```
CO_P2
檔案  編輯  檢視

r0=          0
r1=          1
r2=          0
r3=          0
r4=          0
r5=          0
r6=          0
r7=         14
r8=          0
r9=         15
r10=         0
r11=         0
r12=         0

第 1 行，第 1  100%  Windows (CRLF)  UTF-8
```

Problems you met and solutions:

Besides plenty of syntax problems in Verilog, most difficult problem was to figure out the relationship between opcode, function code, and ALU opcode. I spent lots of time

searching how to construct RegDst, RegWrite, and other inputs, outputs with the relationship mentioned above. And when I searching the net, I found out a wonderful table for that function code, explaining everything.

Summary:

For me, this lab was also an obstacle that actually trained my Verilog ability. I felt my Verilog code was getting better and better. And meanwhile, I found out making a cpu was interesting as well just like that constructing a big deal with Lego needed to connect electric wires. Hope to try these similar problems in the future.