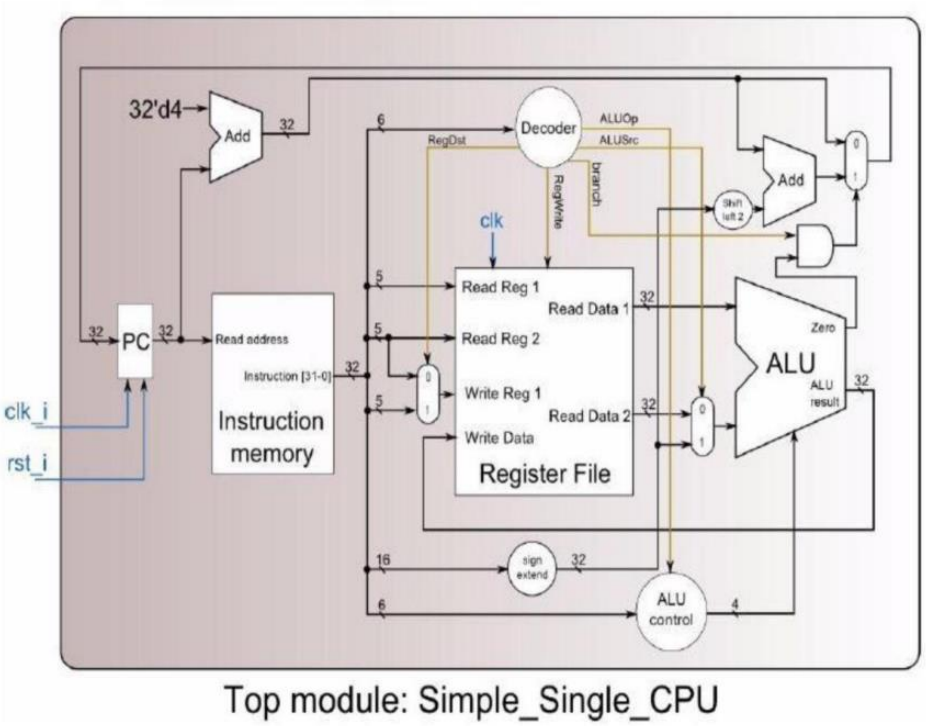
# 110550130 劉秉驊

# 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.

# 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
```


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
```

## 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.