2025/11/07
實驗八

姓名：林承羿
學號：01257027
班級：資工 3A
E-mail：IanLin6225@gmail.com

※注意

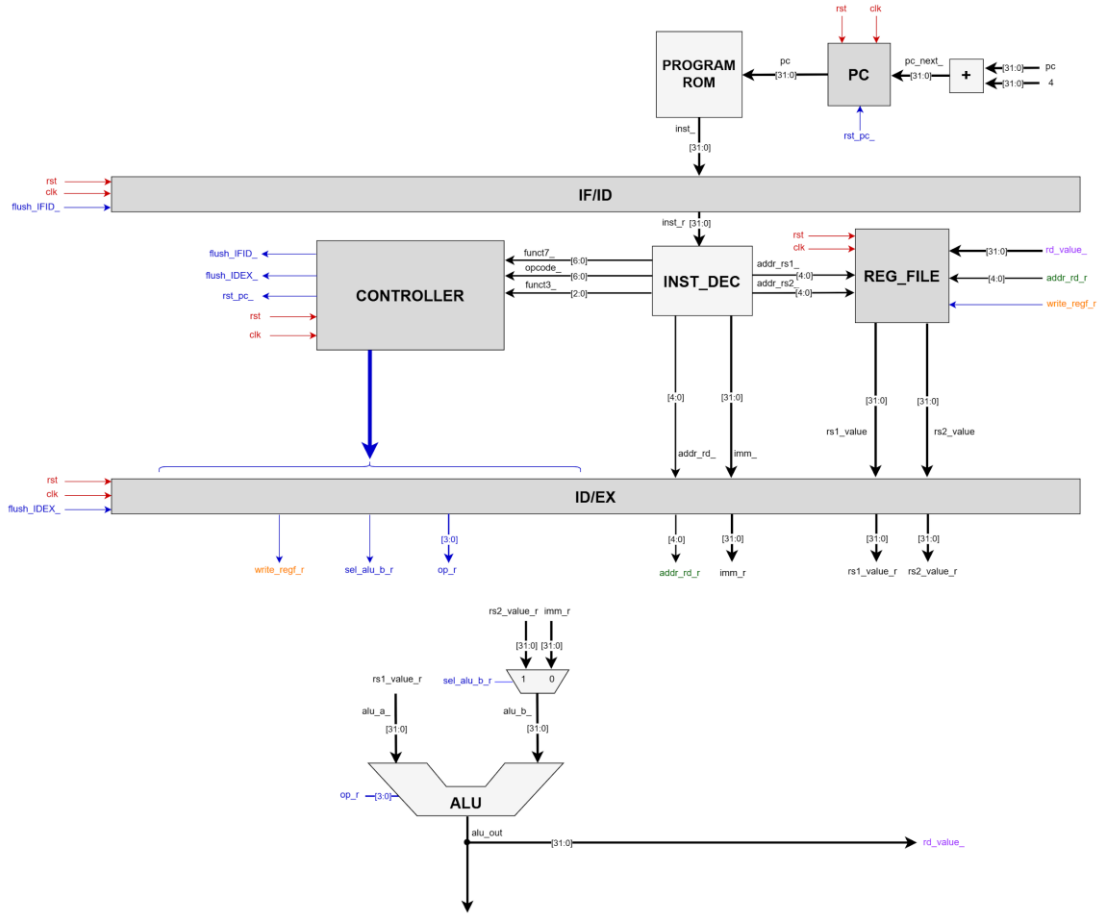
1. 繳交時一律轉 PDF 檔
2. 繳交期限為下周上課前
3. 一人繳交一份
4. 檔名請按照作業檔名格式進行填寫，未依照格式不予批改
5. 檔名範例：學號_姓名_HW8

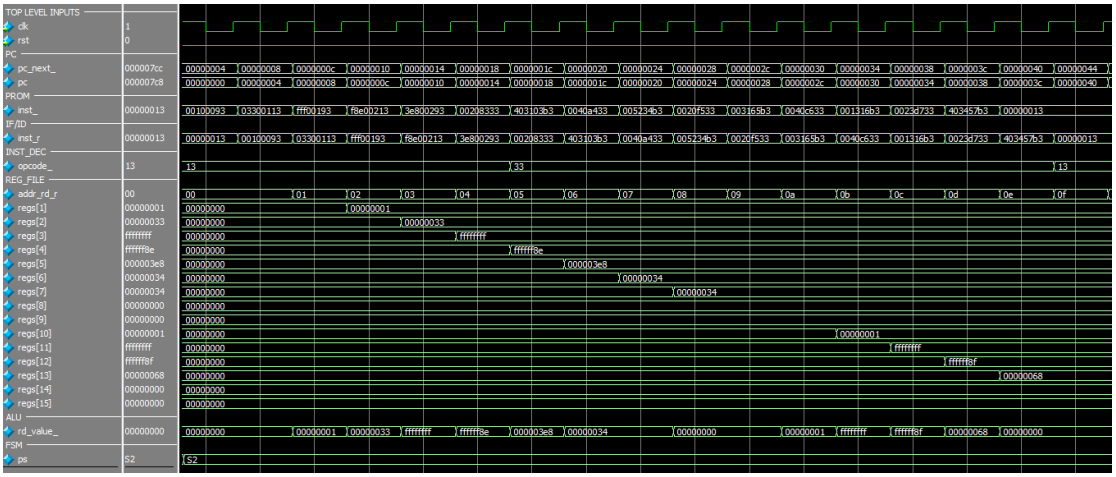
1、 暫存器定址

●實驗說明：

1. 完成下圖架構，Program_Rom 已提供

●系統硬體架構方塊圖（接線圖）：





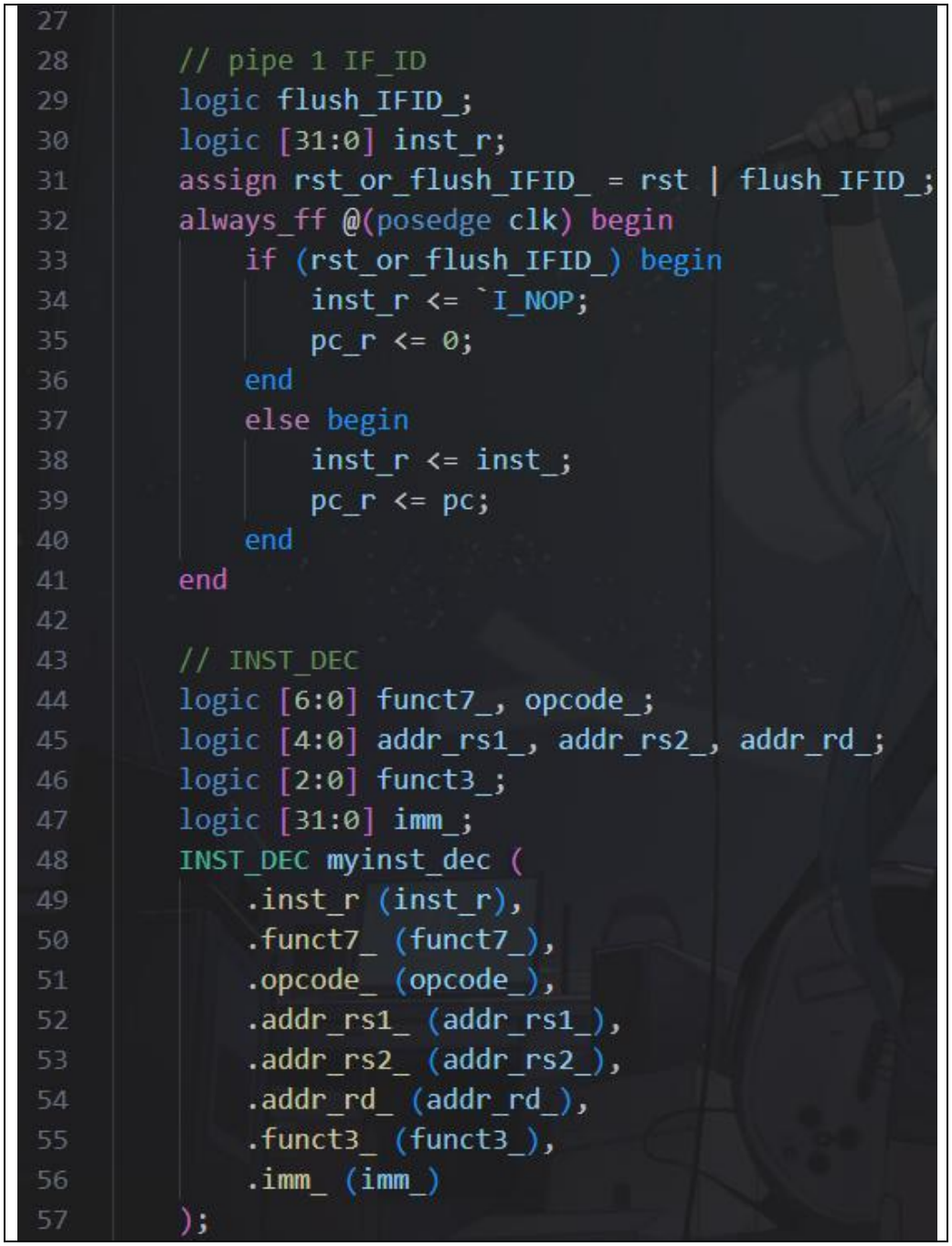
●系統架構程式碼、測試資料程式碼與程式碼說明

截圖請善用 **win+shift+S**

RISC-V

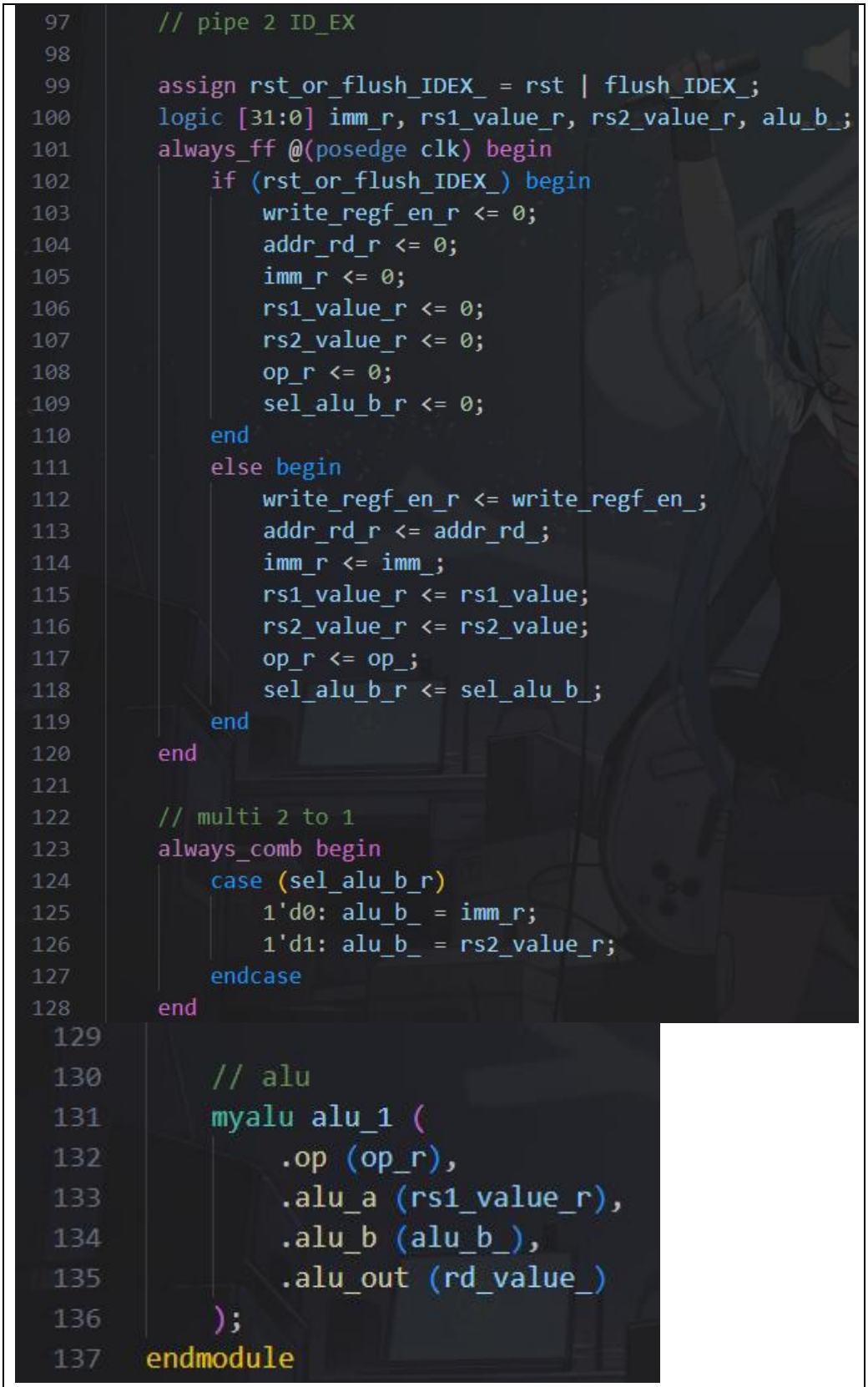
mycpu.sv > mycpu

```
1  `include "mydefine.sv"
2  module mycpu(
3      input logic clk, rst,
4      output logic [31:0] regs_31
5  );
6
7      logic rst_pc_;
8      logic [31:0] pc,pc_next, pc_r;
9      assign pc_next = pc+4;
10     always_ff @(posedge clk) begin
11         if (rst|rst_pc_) begin
12             pc <= 0;
13         end
14         else begin
15             pc <= pc_next;
16         end
17     end
18
19     logic [31:0] inst_;
20     // promgram rom
21     Program_Rom myprogram_rom (
22         // in
23         .Rom_addr (pc),
24         // out
25         .Rom_data (inst_)
26     );
```



```
27
28      // pipe 1 IF_ID
29      logic flush_IFID_;
30      logic [31:0] inst_r;
31      assign rst_or_flush_IFID_ = rst | flush_IFID_;
32      always_ff @(posedge clk) begin
33          if (rst_or_flush_IFID_) begin
34              inst_r <= `I_NOP;
35              pc_r <= 0;
36          end
37          else begin
38              inst_r <= inst_;
39              pc_r <= pc;
40          end
41      end
42
43      // INST_DEC
44      logic [6:0] funct7_, opcode_;
45      logic [4:0] addr_rs1_, addr_rs2_, addr_rd_;
46      logic [2:0] funct3_;
47      logic [31:0] imm_;
48      INST_DEC myinst_dec (
49          .inst_r (inst_r),
50          .funct7_ (funct7_),
51          .opcode_ (opcode_),
52          .addr_rs1_ (addr_rs1_),
53          .addr_rs2_ (addr_rs2_),
54          .addr_rd_ (addr_rd_),
55          .funct3_ (funct3_),
56          .imm_ (imm_)
57      );
```

```
59      // Reg file
60      logic write_regf_en_r;
61      logic [4:0] addr_rd_r;
62      logic [31:0] rd_value_, rs1_value, rs2_value;
63      Reg_file myreg (
64          .clk (clk),
65          .rst (rst),
66          .write_regf_en (write_regf_en_r),
67          .addr_rd (addr_rd_r),
68          .addr_rs1 (addr_rs1_),
69          .addr_rs2 (addr_rs2_),
70          .rd_value (rd_value_),
71          .rs1_value (rs1_value),
72          .rs2_value (rs2_value),
73          .regs_31 (regs_31)
74      );
75      logic flush_IDEX_;
76      // controller
77      logic sel_pc, sel_alu_b_, sel_alu_b_r;
78      logic [3:0] op_, op_r;
79      logic write_regf_en_;
80      controller mycontroller(
81          // in
82          .funct7_ (funct7_),
83          .opcode_ (opcode_),
84          .rst (rst),
85          .clk (clk),
86          .funct3_ (funct3_),
87          // out
88          .sel_alu_b_ (sel_alu_b_),
89          .write_regf_en_ (write_regf_en_),
90          .flush_IFID_ (flush_IFID_),
91          .flush_IDEX_ (flush_IDEX_),
92          .rst_pc_ (rst_pc_),
93          .sel_pc (sel_pc),
94          .op_ (op_)
95      );
```



```
97      // pipe 2 ID_EX
98
99      assign rst_or_flush_IDEX_ = rst | flush_IDEX_;
100     logic [31:0] imm_r, rs1_value_r, rs2_value_r, alu_b_;
101     always_ff @(posedge clk) begin
102         if (rst_or_flush_IDEX_) begin
103             write_regf_en_r <= 0;
104             addr_rd_r <= 0;
105             imm_r <= 0;
106             rs1_value_r <= 0;
107             rs2_value_r <= 0;
108             op_r <= 0;
109             sel_alu_b_r <= 0;
110         end
111         else begin
112             write_regf_en_r <= write_regf_en_;
113             addr_rd_r <= addr_rd_;
114             imm_r <= imm_;
115             rs1_value_r <= rs1_value;
116             rs2_value_r <= rs2_value;
117             op_r <= op_;
118             sel_alu_b_r <= sel_alu_b_;
119         end
120     end
121
122     // multi 2 to 1
123     always_comb begin
124         case (sel_alu_b_r)
125             1'd0: alu_b_ = imm_r;
126             1'd1: alu_b_ = rs2_value_r;
127         endcase
128     end
129
130         // alu
131         myalu alu_1 (
132             .op (op_r),
133             .alu_a (rs1_value_r),
134             .alu_b (alu_b_),
135             .alu_out (rd_value_)
136         );
137 endmodule
```

INST-DEC

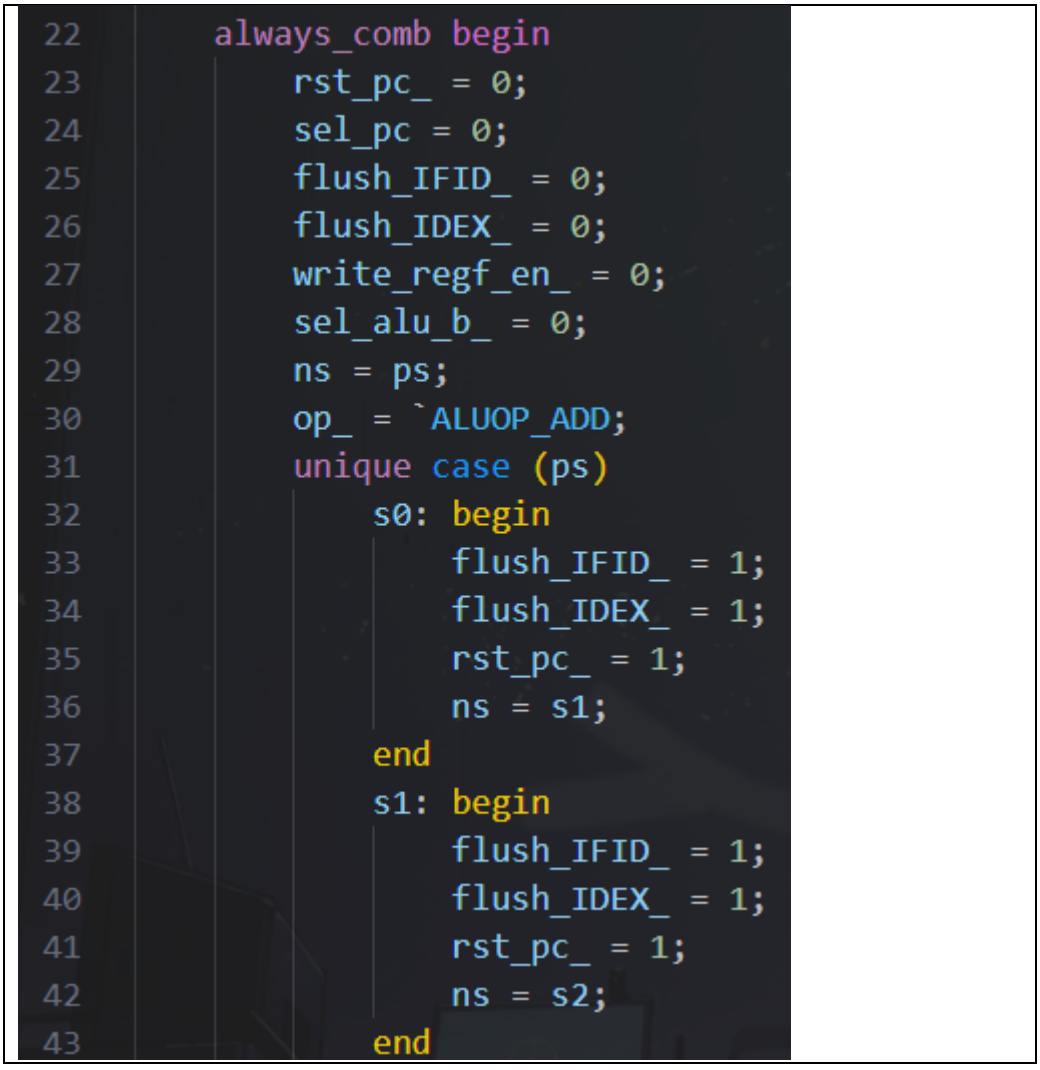
INST_DEC.sv > ...

```sv
`include "mydefine.sv"
module INST_DEC (
    input logic [31:0] inst_r,
    output logic [6:0] funct7_, opcode_,
    output logic [4:0] addr_rs1_, addr_rs2_, addr_rd_,
    output logic [2:0] funct3_,
    output logic [31:0] imm_
);
    assign opcode_ = inst_r[6:0];
    assign addr_rd_ = inst_r[11:7];
    assign funct3_ = inst_r[14:12];
    assign addr_rs1_ = inst_r[19:15];
    assign addr_rs2_ = inst_r[24:20];
    assign funct7_ = inst_r[31:25];

    logic [31:0] IMM_I;
    assign IMM_I = {{20{inst_r[31]}}, inst_r[31:20]};
    always_comb begin
        unique case (opcode_)
            `Opcode_I: imm_ = IMM_I;
        endcase
    end

endmodule
```

Controller

controller.sv > controller

```sv
`include "mydefine.sv"
`timescale 1ns/100ps
module controller (
    input logic [6:0] funct7_, opcode_,
    input logic rst, clk,
    input logic [2:0] funct3_,
    output logic flush_IFID_, flush_IDEX_, rst_pc_, sel_pc, write_regf_en_, sel_alu_b_,
    output logic [3:0] op_
);
    typedef enum {s0, s1, s2} fsm_state;
    fsm_state ps, ns;

    always_ff @(posedge clk) begin
        if (rst) begin
            ps <= #1 s0;
        end
        else begin
            ps <= #1 ns;
        end
    end
```



```
22      always_comb begin
23          rst_pc_ = 0;
24          sel_pc = 0;
25          flush_IFID_ = 0;
26          flush_IDEX_ = 0;
27          write_regf_en_ = 0;
28          sel_alu_b_ = 0;
29          ns = ps;
30          op_ = `ALUOP_ADD;
31          unique case (ps)
32              s0: begin
33                  flush_IFID_ = 1;
34                  flush_IDEX_ = 1;
35                  rst_pc_ = 1;
36                  ns = s1;
37              end
38              s1: begin
39                  flush_IFID_ = 1;
40                  flush_IDEX_ = 1;
41                  rst_pc_ = 1;
42                  ns = s2;
43              end
```

```
            // I-type
            if ((opcode_ == `Opcode_I) && (funct3_ == `F_ADDI)) begin
                op_ = `ALUOP_ADD;
                write_regf_en_ = 1;
            end
            if ((opcode_ == `Opcode_I) && (funct3_ == `F_SLTI)) begin
                op_ = `ALUOP_LT;
                write_regf_en_ = 1;
            end
            if ((opcode_ == `Opcode_I) && (funct3_ == `F_SLTIU)) begin
                op_ = `ALUOP_LTU;
                write_regf_en_ = 1;
            end
            if ((opcode_ == `Opcode_I) && (funct3_ == `F_ANDI)) begin
                op_ = `ALUOP_AND;
                write_regf_en_ = 1;
            end
            if ((opcode_ == `Opcode_I) && (funct3_ == `F_ORI)) begin
                op_ = `ALUOP_OR;
                write_regf_en_ = 1;
            end
            if ((opcode_ == `Opcode_I) && (funct3_ == `F_XORI)) begin
                op_ = `ALUOP_XOR;
                write_regf_en_ = 1;
            end
            if ((opcode_ == `Opcode_I) && (funct3_ == `F_SLLI)) begin
                op_ = `ALUOP_SLL;
                write_regf_en_ = 1;
            end
            if ((opcode_ == `Opcode_I) && (funct3_ == `F_SRLI_SRAI) && (funct7_ == `F7_SRLI)) begin
                op_ = `ALUOP_SRL;
                write_regf_en_ = 1;
            end
            if ((opcode_ == `Opcode_I) && (funct3_ == `F_SRLI_SRAI) && (funct7_ == `F7_SRAI)) begin
                op_ = `ALUOP_SRA;
                write_regf_en_ = 1;
            end
            // R-type
            if ((opcode_ == `Opcode_R_M) && (funct7_ == 7'b000_0000) && (funct3_ == `F_AND)) begin
                op_ = `ALUOP_AND;
                write_regf_en_ = 1;
                sel_alu_b_ = 1;
            end
            if ((opcode_ == `Opcode_R_M) && (funct7_ == `F7_ADD) && (funct3_ == `F_ADD_SUB)) begin
                op_ = `ALUOP_ADD;
                write_regf_en_ = 1;
                sel_alu_b_ = 1;
            end
            if ((opcode_ == `Opcode_R_M) && (funct7_ == `F7_SUB) && (funct3_ == `F_ADD_SUB)) begin
                op_ = `ALUOP_SUB;
                write_regf_en_ = 1;
                sel_alu_b_ = 1;
            end
            if ((opcode_ == `Opcode_R_M) && (funct7_ == `F7_OPCODE_R) && (funct3_ == `F_SLT)) begin
                op_ = `ALUOP_LT;
                write_regf_en_ = 1;
                sel_alu_b_ = 1;
            end
            if ((opcode_ == `Opcode_R_M) && (funct7_ == `F7_OPCODE_R) && (funct3_ == `F_SLTU)) begin
                op_ = `ALUOP_LTU;
                write_regf_en_ = 1;
                sel_alu_b_ = 1;
            end
            if ((opcode_ == `Opcode_R_M) && (funct7_ == `F7_OPCODE_R) && (funct3_ == `F_AND)) begin
                op_ = `ALUOP_AND;
                write_regf_en_ = 1;
                sel_alu_b_ = 1;
            end
            if ((opcode_ == `Opcode_R_M) && (funct7_ == `F7_OPCODE_R) && (funct3_ == `F_OR)) begin
                op_ = `ALUOP_OR;
                write_regf_en_ = 1;
                sel_alu_b_ = 1;
            end
```

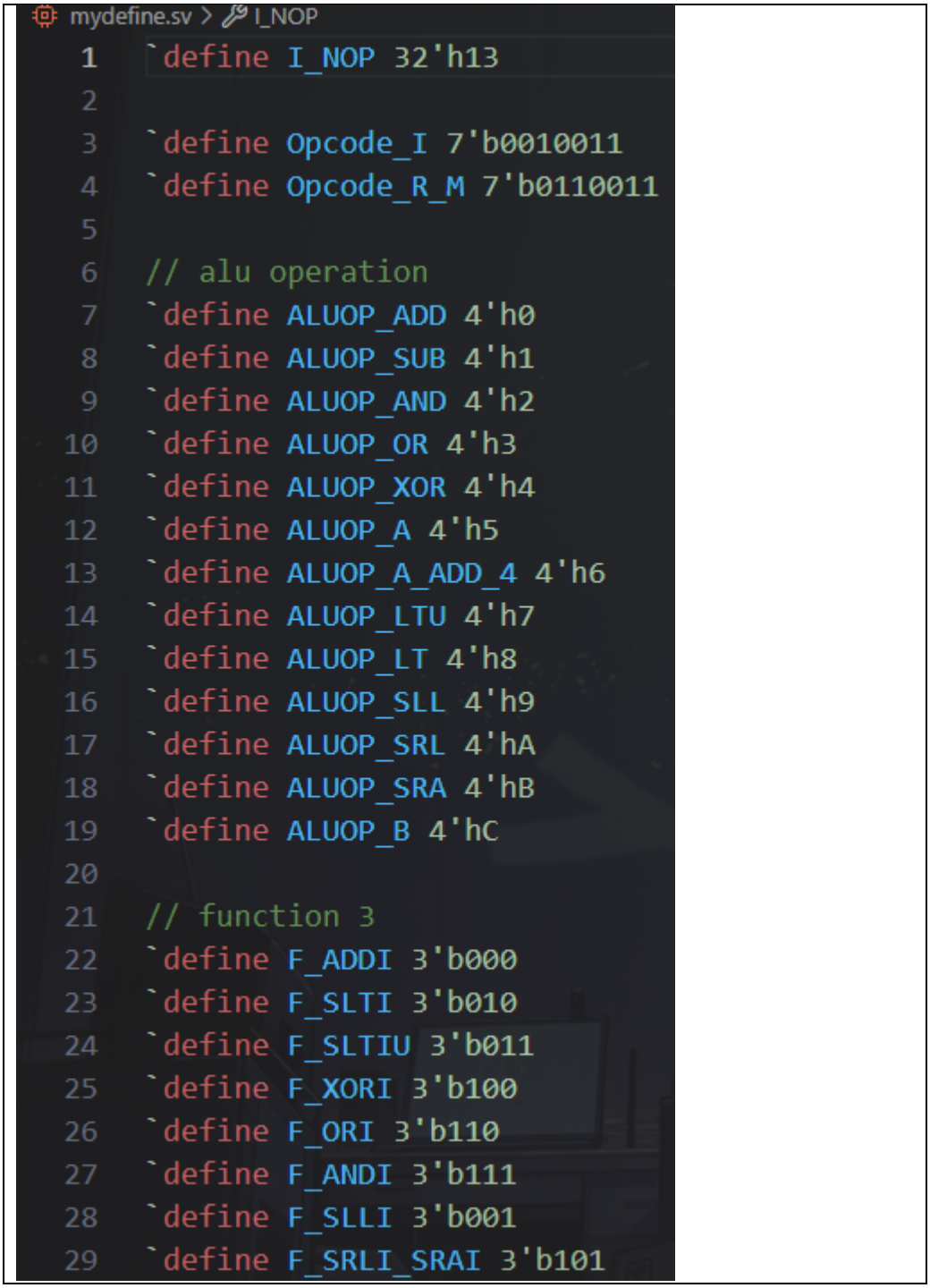
```
            // R-type
            if ((opcode_ == `Opcode_R_M) && (funct7_ == 7'b000_0000) && (funct3_ == `F_AND)) begin
                op_ = `ALUOP_AND;
                write_regf_en_ = 1;
                sel_alu_b_ = 1;
            end
            if ((opcode_ == `Opcode_R_M) && (funct7_ == `F7_ADD) && (funct3_ == `F_ADD_SUB)) begin
                op_ = `ALUOP_ADD;
                write_regf_en_ = 1;
                sel_alu_b_ = 1;
            end
            if ((opcode_ == `Opcode_R_M) && (funct7_ == `F7_SUB) && (funct3_ == `F_ADD_SUB)) begin
                op_ = `ALUOP_SUB;
                write_regf_en_ = 1;
                sel_alu_b_ = 1;
            end
            if ((opcode_ == `Opcode_R_M) && (funct7_ == `F7_OPCODE_R) && (funct3_ == `F_SLT)) begin
                op_ = `ALUOP_LT;
                write_regf_en_ = 1;
                sel_alu_b_ = 1;
            end
            if ((opcode_ == `Opcode_R_M) && (funct7_ == `F7_OPCODE_R) && (funct3_ == `F_SLTU)) begin
                op_ = `ALUOP_LTU;
                write_regf_en_ = 1;
                sel_alu_b_ = 1;
            end
            if ((opcode_ == `Opcode_R_M) && (funct7_ == `F7_OPCODE_R) && (funct3_ == `F_AND)) begin
                op_ = `ALUOP_AND;
                write_regf_en_ = 1;
                sel_alu_b_ = 1;
            end
            if ((opcode_ == `Opcode_R_M) && (funct7_ == `F7_OPCODE_R) && (funct3_ == `F_OR)) begin
                op_ = `ALUOP_OR;
                write_regf_en_ = 1;
                sel_alu_b_ = 1;
            end
```

Alu

myalu.sv > ...

```
`include "mydefine.sv"
module myalu (
    input logic [3:0] op,
    input logic [31:0] alu_a,
    input logic [31:0] alu_b,
    output logic [31:0] alu_out
);
    always_comb begin
        unique case (op)
            `ALUOP_ADD : alu_out = alu_a + alu_b;
            `ALUOP_SUB : alu_out = $signed(alu_a) - $signed(alu_b);
            `ALUOP_AND : alu_out = alu_a & alu_b;
            `ALUOP_OR : alu_out = alu_a | alu_b;
            `ALUOP_XOR : alu_out = alu_a ^ alu_b;
            `ALUOP_A : alu_out = alu_a;
            `ALUOP_A_ADD_4 : alu_out = alu_a + 4;
            `ALUOP_LTU : alu_out = alu_a < alu_b;
            `ALUOP_LT : alu_out = $signed(alu_a) < $signed(alu_b);
            `ALUOP_SLL : alu_out = alu_a << alu_b[4:0];
            `ALUOP_SRL : alu_out = alu_a >> alu_b[4:0];
            `ALUOP_SRA : alu_out = $signed(alu_a) >>> alu_b[4:0];
            `ALUOP_B : alu_out = alu_b;
            default : alu_out = alu_a;
        endcase
    end
endmodule
```

Define


mydefine.sv > I_NOP
1 `define I_NOP 32'h13
2
3 `define Opcode_I 7'b0010011
4 `define Opcode_R_M 7'b0110011
5
6 // alu operation
7 `define ALUOP_ADD 4'h0
8 `define ALUOP_SUB 4'h1
9 `define ALUOP_AND 4'h2
10 `define ALUOP_OR 4'h3
11 `define ALUOP_XOR 4'h4
12 `define ALUOP_A 4'h5
13 `define ALUOP_A_ADD_4 4'h6
14 `define ALUOP_LTU 4'h7
15 `define ALUOP_LT 4'h8
16 `define ALUOP_SLL 4'h9
17 `define ALUOP_SRL 4'hA
18 `define ALUOP_SRA 4'hB
19 `define ALUOP_B 4'hC
20
21 // function 3
22 `define F_ADDI 3'b000
23 `define F_SLTI 3'b010
24 `define F_SLTIU 3'b011
25 `define F_XORI 3'b100
26 `define F_ORI 3'b110
27 `define F_ANDI 3'b111
28 `define F_SLLI 3'b001
29 `define F_SRLI_SRAI 3'b101

mydefine.sv > I_NOP

```
1   `define I_NOP 32'h13
2
3   `define Opcode_I 7'b0010011
4   `define Opcode_R_M 7'b0110011
5
6   // alu operation
7   `define ALUOP_ADD 4'h0
8   `define ALUOP_SUB 4'h1
9   `define ALUOP_AND 4'h2
10  `define ALUOP_OR 4'h3
11  `define ALUOP_XOR 4'h4
12  `define ALUOP_A 4'h5
13  `define ALUOP_A_ADD_4 4'h6
14  `define ALUOP_LTU 4'h7
15  `define ALUOP_LT 4'h8
16  `define ALUOP_SLL 4'h9
17  `define ALUOP_SRL 4'hA
18  `define ALUOP_SRA 4'hB
19  `define ALUOP_B 4'hC
20
21  // function 3
22  `define F_ADDI 3'b000
23  `define F_SLTI 3'b010
24  `define F_SLTIU 3'b011
25  `define F_XORI 3'b100
26  `define F_ORI 3'b110
27  `define F_ANDI 3'b111
28  `define F_SLLI 3'b001
29  `define F_SRLI_SRAI 3'b101
```

Program_Rom

Program_Rom.sv > ...

```sv
`timescale 1ns/100ps
module Program_Rom(
    input   logic [31:0] Rom_addr,
    output  logic [31:0] Rom_data
);

    always_comb begin
        case (Rom_addr)
            32'h0  : Rom_data = 32'h00100093;    //addi  x1, x0, 1
            32'h4  : Rom_data = 32'h03300113;    //addi  x2, x0, 51
            32'h8  : Rom_data = 32'hfff00193;    //addi  x3, x0, -1
            32'hc  : Rom_data = 32'hf8e00213;    //addi  x4, x0, -114
            32'h10 : Rom_data = 32'h3e800293;    //addi  x5, x0, 1000
            32'h14 : Rom_data = 32'h00208333;    //add   x6, x1, x2
            32'h18 : Rom_data = 32'h403103b3;    //sub   x7, x2, x3
            32'h1c : Rom_data = 32'h0040a433;    //slt   x8, x1, x4
            32'h20 : Rom_data = 32'h005234b3;    //sltu  x9, x4, x5
            32'h24 : Rom_data = 32'h0020f533;    //and   x10, x1, x2
            32'h28 : Rom_data = 32'h003165b3;    //or    x11, x2, x3
            32'h2c : Rom_data = 32'h0040c633;    //xor   x12, x1, x4
            32'h30 : Rom_data = 32'h001316b3;    //sll   x13, x6, x1
            32'h34 : Rom_data = 32'h0023d733;    //srl   x14, x7, x2
            32'h38 : Rom_data = 32'h403457b3;    //sra   x15, x8, x3
            default: Rom_data = 32'h00000013;    //NOP
        endcase
    end
endmodule
```

Reg_file

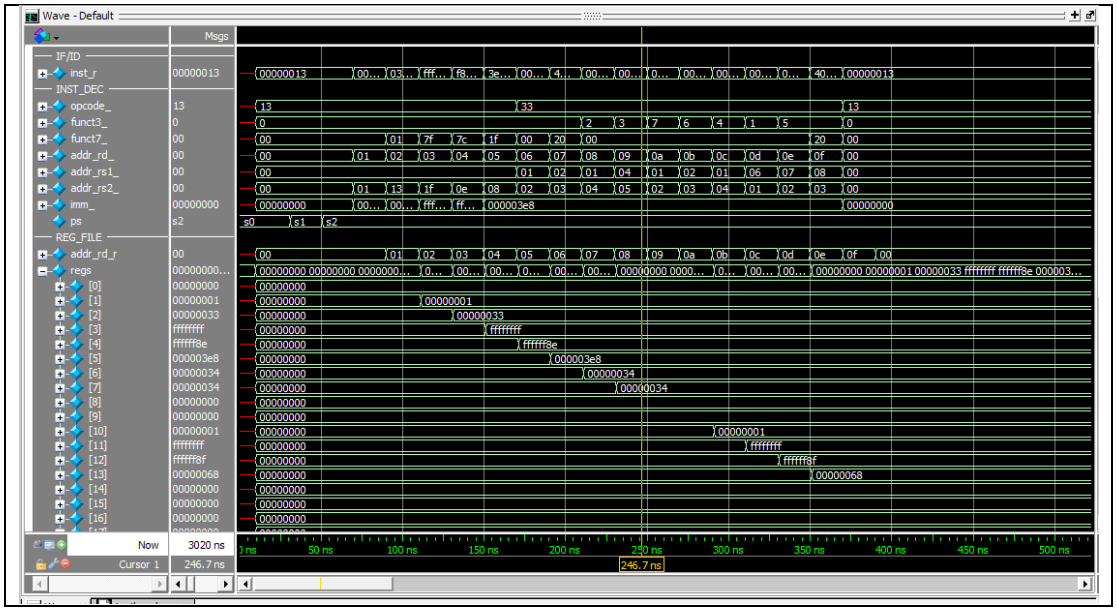
Reg_file.sv > ...

```
1   `timescale 1ns/100ps
2   module Reg_file(
3       input   logic clk,
4       input   logic rst,
5       input   logic write_regf_en,
6       input   logic [4:0] addr_rd,
7       input   logic [4:0] addr_rs1,
8       input   logic [4:0] addr_rs2,
9       input   logic [31:0] rd_value,
10
11      output  logic [31:0] rs1_value,
12      output  logic [31:0] rs2_value,
13      output logic [31:0] regs_31
14  );
15
16      logic [31:0] regs[0:31];
17      logic addr_rd_not_0;
18      integer i;
19
20      assign regs_31 = regs[31];
21
22      assign addr_rd_not_0 = |addr_rd;
23
24      assign rs1_value = regs[addr_rs1];
25      assign rs2_value = regs[addr_rs2];
```

Reg_file.sv > ...

```
1   `timescale 1ns/100ps
2   module Reg_file(
3       input   logic clk,
4       input   logic rst,
5       input   logic write_regf_en,
6       input   logic [4:0] addr_rd,
7       input   logic [4:0] addr_rs1,
8       input   logic [4:0] addr_rs2,
9       input   logic [31:0] rd_value,
10
11      output  logic [31:0] rs1_value,
12      output  logic [31:0] rs2_value,
13      output logic [31:0] regs_31
14  );
15
16      logic [31:0] regs[0:31];
17      logic addr_rd_not_0;
18      integer i;
19
20      assign regs_31 = regs[31];
21
22      assign addr_rd_not_0 = |addr_rd;
23
24      assign rs1_value = regs[addr_rs1];
25      assign rs2_value = regs[addr_rs2];
```

●模擬結果與結果說明：



●結論與心得：

經過今天的實驗，我學會如何透過最上層串接七段顯示器、LED 燈，驗證自己程式是否錯誤。