2025/10/23
實驗七

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

※注意

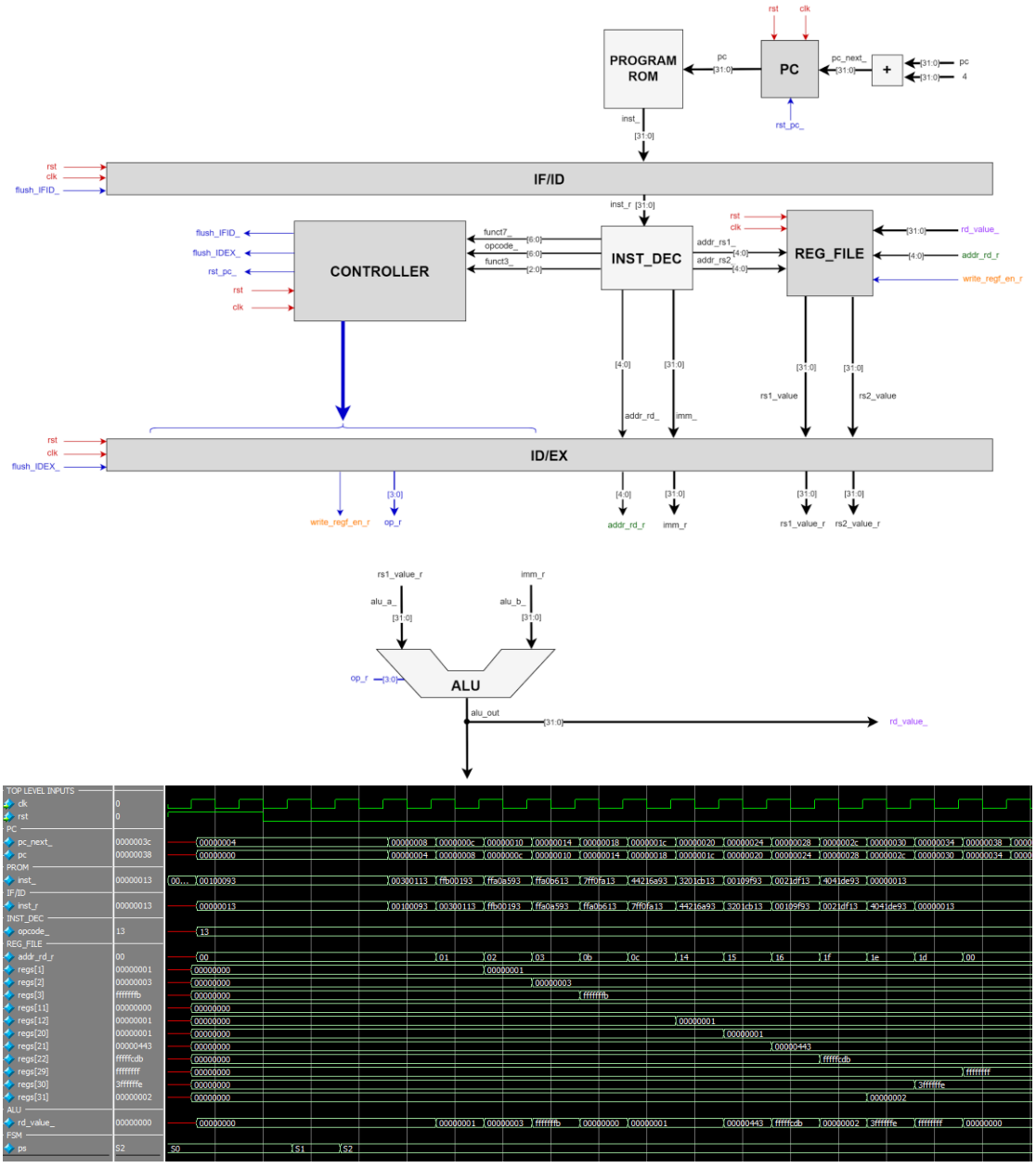
1. 繳交時一律轉 PDF 檔
2. 繳交期限為下周上課前
3. 一人繳交一份
4. 檔名請按照作業檔名格式進行填寫，未依照格式不予批改
5. 檔名範例：學號_姓名_HW7

1、 立即數定址

●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**

myalu.sv > myalu

```systemverilog
`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
```

這是今天上課刻的 **ALU**。

mydefine.sv > ...

```
1   `define I_NOP 32'h13
2
3   `define Opcode_I 7'b0010011
4
5   // alu operation
6   `define ALUOP_ADD 4'h0
7   `define ALUOP_SUB 4'h1
8   `define ALUOP_AND 4'h2
9   `define ALUOP_OR 4'h3
10  `define ALUOP_XOR 4'h4
11  `define ALUOP_A 4'h5
12  `define ALUOP_A_ADD_4 4'h6
13  `define ALUOP_LTU 4'h7
14  `define ALUOP_LT 4'h8
15  `define ALUOP_SLL 4'h9
16  `define ALUOP_SRL 4'hA
17  `define ALUOP_SRA 4'hB
18  `define ALUOP_B 4'hC
19
20  // function 3
21  `define F_ADDI 3'b000
22  `define F_SLTI 3'b010
23  `define F_SLTIU 3'b011
24  `define F_XORI 3'b100
25  `define F_ORI 3'b110
26  `define F_ANDI 3'b111
27  `define F_SLLI 3'b001
28  `define F_SRLI_SRAI 3'b101
29
30  // function 7
31  `define F7_SRLI 7'b0000000
32  `define F7_SRAI 7'b0100000
```

這是今天所用的所有 Define

Program_Rom.sv > Program_Rom

```
1  `timescale 1ns/100ps
2  module Program_Rom(
3      input   logic [31:0] Rom_addr,
4      output  logic [31:0] Rom_data
5  );
6      always_comb begin
7          case (Rom_addr)
8              32'h0  : Rom_data = 32'h00100093;   //addi  x1, x0, 1
9              32'h4  : Rom_data = 32'h00300113;   //addi  x2, x0, 3
10             32'h8  : Rom_data = 32'hffb00193;   //addi  x3, x0, -5
11             32'hc  : Rom_data = 32'hffa0a593;   //slti  x11, x1, -6
12             32'h10 : Rom_data = 32'hffa0b613;   //sltiu x12, x1, -6
13             32'h14 : Rom_data = 32'h7ff0fa13;   //andi  x20, x1, 2047
14             32'h18 : Rom_data = 32'h44216a93;   //ori   x21, x2, 1090
15             32'h1c : Rom_data = 32'h3201cb13;   //xori  x22, x3, 800
16             32'h20 : Rom_data = 32'h00109f93;   //slli  x31, x1, 1
17             32'h24 : Rom_data = 32'h0021df13;   //srli  x30, x3, 2
18             32'h28 : Rom_data = 32'h4041de93;   //srai  x29, x3, 4
19             default: Rom_data = 32'h00000013;   //NOP
20         endcase
21     end
22 endmodule
```

這是作業用到的 Program_Rom

Reg_file.sv > Reg_file

```
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 );
14
15     logic [31:0] regs[0:31];
16     logic addr_rd_not_0;
17     integer i;
18
19     assign addr_rd_not_0 = |addr_rd;
20
21     assign rs1_value = regs[addr_rs1];
22     assign rs2_value = regs[addr_rs2];
23
24     always_ff@(posedge clk)
25     begin
26         if(rst) begin
27             for(i = 0; i < 32; i = i+1) begin:rst_keywords
28                 regs[i] <= 0;
29             end
30         end
31         else begin
32             // Write
33             if (write_regf_en && addr_rd_not_0)
34                 regs[addr_rd] <= #1 rd_value;
35         end
36     end
37
38 endmodule
```

這是 **Reg_file**

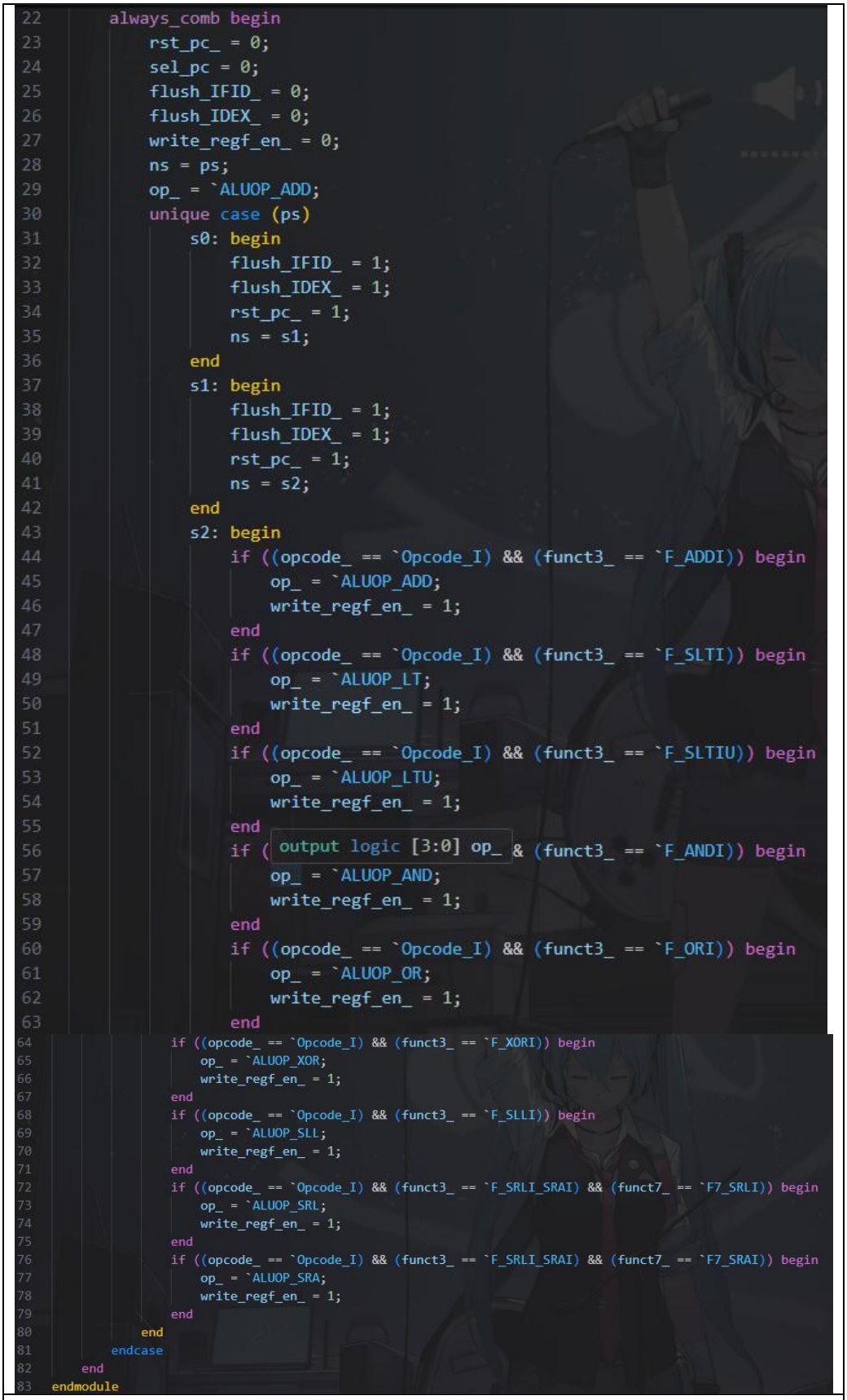
INST_DEC.sv > ...

```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
```

這是 **INST_DEC**

controller.sv > ...

```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_,
    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
```



```
    always_comb begin
        rst_pc_ = 0;
        sel_pc = 0;
        flush_IFID_ = 0;
        flush_IDEX_ = 0;
        write_regf_en_ = 0;
        ns = ps;
        op_ = `ALUOP_ADD;
        unique case (ps)
            s0: begin
                flush_IFID_ = 1;
                flush_IDEX_ = 1;
                rst_pc_ = 1;
                ns = s1;
            end
            s1: begin
                flush_IFID_ = 1;
                flush_IDEX_ = 1;
                rst_pc_ = 1;
                ns = s2;
            end
            s2: begin
                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 ( output logic [3:0] op_ & (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
            end
        endcase
    end
endmodule
```

這是 **controller**，以 **opcode, funct3_, funct7_**控制做出的動作。

mycpu.sv > mycpu

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

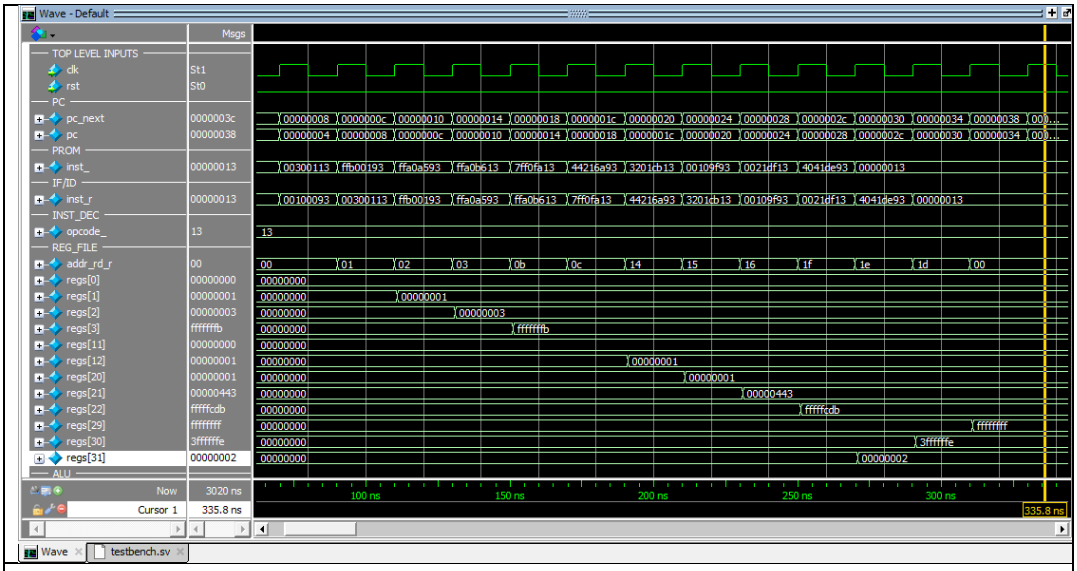
```
25
26      // pipe 1 IF_ID
27      logic flush_IFID_;
28      logic [31:0] inst_r;
29      assign rst_or_flush_IFID_ = rst | flush_IFID_;
30      always_ff @(posedge clk) begin
31          if (rst_or_flush_IFID_) begin
32              inst_r <= `I_NOP;
33              pc_r <= 0;
34          end
35          else begin
36              inst_r <= inst_;
37              pc_r <= pc;
38          end
39      end
40
41      // INST_DEC
42      logic [6:0] funct7_, opcode_;
43      logic [4:0] addr_rs1_, addr_rs2_, addr_rd_;
44      logic [2:0] funct3_;
45      logic [31:0] imm_;
46      INST_DEC myinst_dec (
47          .inst_r (inst_r),
48          .funct7_ (funct7_),
49          .opcode_ (opcode_),
50          .addr_rs1_ (addr_rs1_),
51          .addr_rs2_ (addr_rs2_),
52          .addr_rd_ (addr_rd_),
53          .funct3_ (funct3_),
54          .imm_ (imm_)
55      );
```

```
56
57      // Reg file
58      logic write_regf_en_r;
59      logic [4:0] addr_rd_r;
60      logic [31:0] rd_value_, rs1_value, rs2_value;
61      Reg_file myreg (
62          .clk (clk),
63          .rst (rst),
64          .write_regf_en (write_regf_en_r),
65          .addr_rd (addr_rd_r),
66          .addr_rs1 (addr_rs1_),
67          .addr_rs2 (addr_rs2_),
68          .rd_value (rd_value_),
69          .rs1_value (rs1_value),
70          .rs2_value (rs2_value)
71      );
72      logic flush_IDEX_;
73      // controller
74      logic sel_pc;
75      logic [3:0] op_, op_r;
76      logic write_regf_en_;
77      controller mycontroller(
78          // in
79          .funct7_ (funct7_),
80          .opcode_ (opcode_),
81          .rst (rst),
82          .clk (clk),
83          .funct3_ (funct3_),
84          // out
85          .write_regf_en_ (write_regf_en_),
86          .flush_IFID_ (flush_IFID_),
87          .flush_IDEX_ (flush_IDEX_),
88          .rst_pc_ (rst_pc_),
89          .sel_pc (sel_pc),
90          .op_ (op_)
91      );
```

```
93      // pipe 2 ID_EX
94
95      assign rst_or_flush_IDEX_ = rst | flush_IDEX_;
96      logic [31:0] imm_r, rs1_value_r, rs2_value_r;
97      always_ff @(posedge clk) begin
98          if (rst_or_flush_IDEX_) begin
99              write_regf_en_r <= 0;
100             addr_rd_r <= 0;
101             imm_r <= 0;
102             rs1_value_r <= 0;
103             rs2_value_r <= 0;
104             op_r <= 0;
105         end
106         else begin
107             write_regf_en_r <= write_regf_en_;
108             addr_rd_r <= addr_rd_;
109             imm_r <= imm_;
110             rs1_value_r <= rs1_value;
111             rs2_value_r <= rs2_value;
112             op_r <= op_;
113         end
114     end
115
116     // alu
117     myalu alu_1 (
118         .op (op_r),
119         .alu_a (rs1_value_r),
120         .alu_b (imm_r),
121         .alu_out (rd_value_)
122     );
123 endmodule
```

這是 **RISC-V** 架構

●模擬結果與結果說明：



這是跑出來的波形圖。

●結論與心得：

經過此次的實驗，我學到了如何指定運行動作，也更了解 RISC-V 的立即值運作。