



2025/11/13

實驗九

姓名：林承羿

學號：01257027

班級：資工 3A

E-mail：IanLin6225@gmail.com

※注意

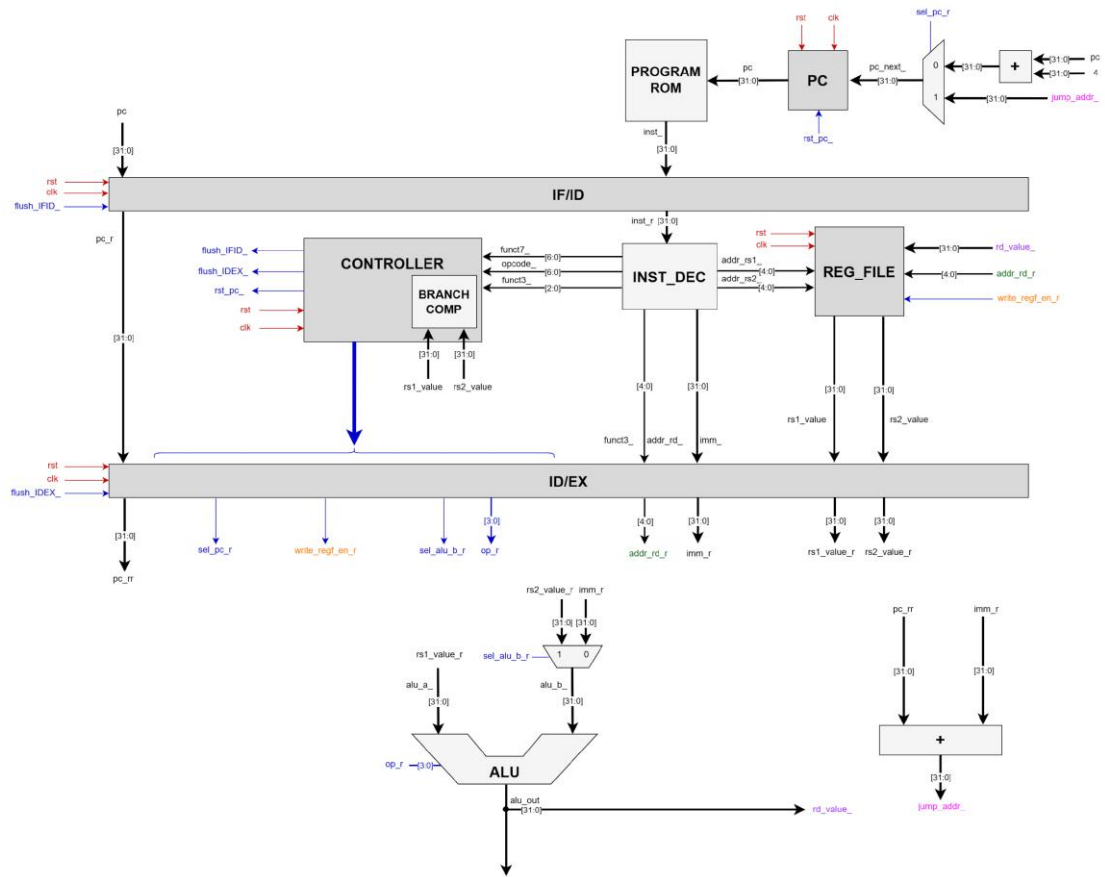
1. 繳交時一律轉 **PDF** 檔
2. 繳交期限為下周上課前
3. 一人繳交一份
4. 檔名請按照作業檔名格式進行填寫，未依照格式不予批改
5. 檔名範例：學號_姓名_HW9

1、組語撰寫練習一累加

●實驗說明：

1. 透過當前課堂所學的所有組合語言撰寫一個 1 累加到 10 的程式，須加註解。
2. 所有暫存器皆可使用，但請將結果存儲至暫存器 **x31**。
3. 程式撰寫完成後透過 **Tool Chain** 將其轉換成 **Program_ROM.sv** 檔，並使用 **ModelSim** 軟體觀察波形是否執行正確。
4. 請將所有用到的暫存器都加入至波形中以方便觀察。
5. 請注意資料危障的問題。

●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**

```
Clock_divider.sv
clock_divider.sv > ...
1  `timescale 1ns/100ps
2  module clock_divider(
3      input logic clk,
4      input logic rst,
5      input logic [31:0] DIVISOR,
6      //
7      output logic clk_out
8  );
9
10     logic [32:0] counter;
11
12     always_ff @(posedge clk or posedge rst) begin
13         if (rst) begin
14             counter <= 0;
15             clk_out <= 0;
16         end else begin
17             if (counter == (DIVISOR - 1)) begin
18                 counter <= 0;
19                 clk_out <= ~clk_out;
20             end else begin
21                 counter <= counter + 1;
22             end
23         end
24     end
25
26 endmodule

Controller.sv
```

```
controllers.v > controller
1  `include "mydefine.sv"
2  `timescale 1ns/100ps
3  module controller (
4      input logic [6:0] funct7_, opcode_,
5      input logic rst, clk,
6      input logic [2:0] funct3_,
7      input logic [31:0] rs1_value, rs2_value,
8      output logic flush_IFID_, flush_IDEX_, rst_pc_, sel_pc_, write_regf_en_, sel_alu_b_,
9      output logic [3:0] op_
10 );
11     logic BEQ_FLAG;
12     logic BNE_FLAG;
13     logic BLT_FLAG;
14     logic BGE_FLAG;
15     logic BLTU_FLAG;
16     logic BGEU_FLAG;
17
18     assign BEQ_FLAG = (rs1_value == rs2_value);
19     assign BNE_FLAG = (rs1_value != rs2_value);
20     assign BLT_FLAG = ($signed(rs1_value) < $signed(rs2_value));
21     assign BGE_FLAG = ($signed(rs1_value) >= $signed(rs2_value));
22     assign BLTU_FLAG = (rs1_value < rs2_value);
23     assign BGEU_FLAG = (rs1_value >= rs2_value);
24
25     typedef enum {s0, s1, s2} fsm_state;
26     fsm_state ps, ns;
27
28     always_ff @(posedge clk) begin
29         if (rst) begin
30             ps <= #1 s0;
31         end
32         else begin
33             ps <= #1 ns;
34         end
35     end
```

```
37     always_comb begin
38         rst_pc_ = 0;
39         sel_pc_ = 0;
40         flush_IFID_ = 0;
41         flush_IDEX_ = 0;
42         write_regf_en_ = 0;
43         sel_alu_b_ = 0;
44         ns = ps;
45         op_ = `ALUOP_ADD;
46         unique case (ps)
47             s0: begin
48                 flush_IFID_ = 1;
49                 flush_IDEX_ = 1;
50                 rst_pc_ = 1;
51                 ns = s1;
52             end
53             s1: begin
54                 flush_IFID_ = 1;
55                 flush_IDEX_ = 1;
56                 rst_pc_ = 1;
57                 ns = s2;
58             end
59         end case;
60     end
```

```

59     s2: begin
60         case (opcode_)
61             `Opcode_I: begin
62                 unique case (funct3_)
63                     `F_ADDI: begin
64                         op_ = `ALUOP_ADD;
65                         write_regf_en_ = 1;
66                     end
67                     `F_SLTI: begin
68                         op_ = `ALUOP_LT;
69                         write_regf_en_ = 1;
70                     end
71                     `F_SLTIU: begin
72                         op_ = `ALUOP_LTU;
73                         write_regf_en_ = 1;
74                     end
75                     `F_ANDI: begin
76                         op_ = `ALUOP_AND;
77                         write_regf_en_ = 1;
78                     end
79                     `F_ORI: begin
80                         op_ = `ALUOP_OR;
81                         write_regf_en_ = 1;
82                     end
83                     `F_XORI: begin
84                         op_ = `ALUOP_XOR;
85                         write_regf_en_ = 1;
86                     end
87                     `F_SLLI: begin
88                         op_ = `ALUOP_SLL;
89                         write_regf_en_ = 1;
90                     end

```

```

91         `F_SRLI_SRAI: begin
92             unique case (funct7_)
93                 `F7_SRLI: begin
94                     op_ = `ALUOP_SRL;
95                     write_regf_en_ = 1;
96                 end
97                 `F7_SRAI: begin
98                     op_ = `ALUOP_SRA;
99                     write_regf_en_ = 1;
100                 end
101             endcase
102         end
103     endcase
104 end
105 `Opcode_R_M: begin
106     unique case (funct3_)
107         `F_AND: begin
108             if (funct7_ == 7'b000_0000 || funct7_ == `F7_OPCODE_R) begin
109                 op_ = `ALUOP_AND;
110                 write_regf_en_ = 1;
111                 sel_alu_b_ = 1;
112             end
113         end
114         `F_OR: begin
115             if (funct7_ == `F7_OPCODE_R) begin
116                 op_ = `ALUOP_OR;
117                 write_regf_en_ = 1;
118                 sel_alu_b_ = 1;
119             end
120         end
121         `F_XOR: begin
122             if (funct7_ == `F7_OPCODE_R) begin
123                 op_ = `ALUOP_XOR;
124                 write_regf_en_ = 1;
125                 sel_alu_b_ = 1;
126             end
127         end
128         `F_ADD_SUB: begin
129             unique case (funct7_)
130                 `F7_ADD: begin
131                     op_ = `ALUOP_ADD;
132                     write_regf_en_ = 1;
133                     sel_alu_b_ = 1;
134                 end
135                 `F7_SUB: begin
136                     op_ = `ALUOP_SUB;
137                     write_regf_en_ = 1;
138                     sel_alu_b_ = 1;
139                 end
140             endcase
141         end
142         `F_SLT: begin
143             if (funct7_ == `F7_OPCODE_R) begin
144                 op_ = `ALUOP_LT;
145                 write_regf_en_ = 1;
146                 sel_alu_b_ = 1;
147             end
148         end
149         `F_SLTU: begin
150             if (funct7_ == `F7_OPCODE_R) begin
151                 op_ = `ALUOP_LTU;
152                 write_regf_en_ = 1;
153                 sel_alu_b_ = 1;
154             end
155         end

```



```

156     `F_SLL: begin
157         if (funct7_ == `F7_OPCODE_R) begin
158             op_ = `ALUOP_SLL;
159             write_regf_en_ = 1;
160             sel_alu_b_ = 1;
161         end
162     end
163     `F_SRL_SRA: begin
164         unique case (funct7_)
165             `F7_OPCODE_R: begin
166                 op_ = `ALUOP_SRL; // SRL
167                 write_regf_en_ = 1;
168                 sel_alu_b_ = 1;
169             end
170             `F7_SRA: begin
171                 op_ = `ALUOP_SRA; // SRA (if different funct7)
172                 write_regf_en_ = 1;
173                 sel_alu_b_ = 1;
174             end
175         endcase
176     end
177 endcase
178 end

179     `Opcode_B: begin
180         unique case (funct3_)
181             `F_BEQ: begin *****
182                 if (BEQ_FLAG) begin
183                     sel_pc_ = 1;
184                     flush_IDEX_ = 1;
185                     flush_IFID_ = 1;
186                 end
187             end
188             `F_BNE: begin
189                 if (BNE_FLAG) begin
190                     sel_pc_ = 1;
191                     flush_IDEX_ = 1;
192                     flush_IFID_ = 1;
193                 end
194             end
195             `F_BLT: begin
196                 if (BLT_FLAG) begin
197                     sel_pc_ = 1;
198                     flush_IFID_ = 1;
199                     flush_IDEX_ = 1;
200                 end
201             end
202             `F_BGE: begin
203                 if (BGE_FLAG) begin
204                     sel_pc_ = 1;
205                     flush_IDEX_ = 1;
206                     flush_IFID_ = 1;
207                 end
208             end

```



```

209         `F_BLTU: begin
210             if (BLTU_FLAG) begin
211                 sel_pc_ = 1;
212                 flush_IDEX_ = 1;
213                 flush_IFID_ = 1;
214             end
215         end
216         `F_BGEU: begin
217             if (BGEU_FLAG) begin
218                 sel_pc_ = 1;
219                 flush_IDEX_ = 1;
220                 flush_IFID_ = 1;
221             end
222         end
223     endcase
224 end
225 endcase
226 end
227 endcase
228 end
229 endmodule

```

INST_DEC.sv

```

INST_DEC.sv > INST_DEC
1  `include "mydefine.sv"
2  module INST_DEC (
3      input logic [31:0] inst_r,
4      output logic [6:0] funct7_, opcode_,
5      output logic [4:0] addr_rs1_, addr_rs2_, addr_rd_,
6      output logic [2:0] funct3_,
7      output logic [31:0] imm_
8  );
9      assign opcode_ = inst_r[6:0];
10     assign funct3_ = inst_r[14:12];
11     assign addr_rd_ = inst_r[11:7];
12     assign addr_rs1_ = inst_r[19:15];
13     assign addr_rs2_ = inst_r[24:20];
14     assign funct7_ = inst_r[31:25];
15
16     logic [31:0] IMM_I;
17     logic [31:0] IMM_B;
18     assign IMM_I = {{20{inst_r[31]}}, inst_r[31:20]};
19     assign IMM_B = {{20{inst_r[31]}}, inst_r[7], inst_r[30:25], inst_r[11:8], 1'd0};
20     always_comb begin
21         unique case (opcode_)
22             `Opcode_I: imm_ = IMM_I;
23             `Opcode_B: imm_ = IMM_B;
24         endcase
25     end
26
27 endmodule

```

Myalu.sv

```

1  `include "mydefine.sv"
2  module myalu (
3      input logic [3:0] op,
4      input logic [31:0] alu_a,
5      input logic [31:0] alu_b,
6      output logic [31:0] alu_out
7  );
8      always_comb begin
9          unique case (op)
10             `ALUOP_ADD : alu_out = alu_a + alu_b;
11             `ALUOP_SUB : alu_out = $signed(alu_a) - $signed(alu_b);
12             `ALUOP_AND : alu_out = alu_a & alu_b;
13             `ALUOP_OR : alu_out = alu_a | alu_b;
14             `ALUOP_XOR : alu_out = alu_a ^ alu_b;
15             `ALUOP_A : alu_out = alu_a;
16             `ALUOP_A_ADD_4 : alu_out = alu_a + 4;
17             `ALUOP_LTU : alu_out = alu_a < alu_b;
18             `ALUOP_LT : alu_out = $signed(alu_a) < $signed(alu_b);
19             `ALUOP_SLL : alu_out = alu_a << alu_b[4:0];
20             `ALUOP_SRL : alu_out = alu_a >> alu_b[4:0];
21             `ALUOP_SRA : alu_out = $signed(alu_a) >>> alu_b[4:0];
22             `ALUOP_B : alu_out = alu_b;
23             default : alu_out = alu_a;
24          endcase
25      end
26  endmodule

```

Mydefine.sv

mydefine.sv > ...

```
1  `define I_NOP 32'h13
2
3  `define Opcode_I 7'b0010011
4  `define Opcode_R_M 7'b0110011
5  `define Opcode_B 7'b1100011
6
7  // alu operation
8  `define ALUOP_ADD 4'h0
9  `define ALUOP_SUB 4'h1
10 `define ALUOP_AND 4'h2
11 `define ALUOP_OR 4'h3
12 `define ALUOP_XOR 4'h4
13 `define ALUOP_A 4'h5
14 `define ALUOP_A_ADD_4 4'h6
15 `define ALUOP_LTU 4'h7
16 `define ALUOP_LT 4'h8
17 `define ALUOP_SLL 4'h9
18 `define ALUOP_SRL 4'hA
19 `define ALUOP_SRA 4'hB
20 `define ALUOP_B 4'hC
21
22 // function 3
23 `define F_ADDI 3'b000
24 `define F_SLTI 3'b010
25 `define F_SLTIU 3'b011
26 `define F_XORI 3'b100
27 `define F_ORI 3'b110
28 `define F_ANDI 3'b111
29 `define F_SLLI 3'b001
30 `define F_SRLI_SRAI 3'b101
```

```
32 // function 7
33 `define F7_ADD 7'b0000000
34 `define F7_SUB 7'b0100000
35 `define F7_SRLI 7'b0000000
36 `define F7_SRAI 7'b0100000
37 `define F7_OPCODE_R 7'b0000000
38 `define F7_SRL 7'b0000000
39 `define F7_SRA 7'b0100000
40
41 // alu
42 `define F_ADD_SUB 3'b000
43 `define F_SLL 3'b001
44 `define F_SLT 3'b010
45 `define F_SLTU 3'b011
46 `define F_XOR 3'b100
47 `define F_SRL_SRA 3'b101
48 `define F_OR 3'b110
49 `define F_AND 3'b111
50
51 // branch
52 `define F_BEQ 3'b000
53 `define F_BNE 3'b001
54 `define F_BLT 3'b100
55 `define F_BGE 3'b101
56 `define F_BLTU 3'b110
57 `define F_BGEU 3'b111
```

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];
26
27      always_ff@(posedge clk)
28      begin
29          if(rst) begin
30              for(i = 0; i < 32; i = i+1) begin:rst_keywords
31                  regs[i] <= 0;
32              end
33          end
34          else begin
35              // Write
36              if (write_regf_en && addr_rd_not_0)
37                  regs[addr_rd] <= #1 rd_value;
38          end
39      end
40
41  endmodule

```

Seven_segment_display.sv

```

seven_segment_display.sv > ...
1  `timescale 1ns/100ps
2  module seven_segment_display(
3      input logic [3:0] digit,    // 4位元二進制數字輸入
4      output logic [6:0] seg      // 7段顯示器輸出 (g, f, e, d, c, b, a)
5  );
6      //共陽
7      always_comb begin
8          case (digit)
9              4'h0: seg = 7'b1000000;    // 顯示 "0"
10             4'h1: seg = 7'b1111001;    // 顯示 "1"
11             4'h2: seg = 7'b0100100;    // 顯示 "2"
12             4'h3: seg = 7'b0110000;    // 顯示 "3"
13             4'h4: seg = 7'b0011001;    // 顯示 "4"
14             4'h5: seg = 7'b0010010;    // 顯示 "5"
15             4'h6: seg = 7'b0000010;    // 顯示 "6"
16             4'h7: seg = 7'b1111000;    // 顯示 "7"
17             4'h8: seg = 7'b0000000;    // 顯示 "8"
18             4'h9: seg = 7'b0010000;    // 顯示 "9"
19             4'ha: seg = 7'b0001000;    // 顯示 "A"
20             4'hb: seg = 7'b0000011;    // 顯示 "B"
21             4'hc: seg = 7'b1000110;    // 顯示 "C"
22             4'hd: seg = 7'b0100001;    // 顯示 "D"
23             4'he: seg = 7'b0000110;    // 顯示 "E"
24             4'hf: seg = 7'b0001110;    // 顯示 "F"
25             default: seg = 7'b1111111; // 不顯示
26         endcase
27     end
28
29 endmodule

```

Mycpu.sv


```
mycpu.v > mycpu > [ ] alu_b_
1  `include "mydefine.sv"
2  module mycpu(
3      input logic clk, rst,
4      output logic [31:0] regs_31
5  );
6      // program-counter *****
7      logic rst_pc_, sel_pc_r;
8      logic [31:0] pc, pc_next, pc_r, pc_rr, jump_addr_;
9      always_comb begin
10         case (sel_pc_r)
11             1'd0: begin
12                 pc_next = pc + 4;
13             end
14             1'd1: begin
15                 pc_next = jump_addr_;
16             end
17         endcase
18     end
19     always_ff @(posedge clk) begin
20         if (rst|rst_pc_) begin
21             pc <= 0;
22         end
23         else begin
24             pc <= pc_next;
25         end
26     end
27
28     logic [31:0] inst_;
29     // program rom
30     Program_Rom myprogram_rom (
31         // in
32         .Rom_addr (pc),
33         // out
34         .Rom_data (inst_)
35     );
```

```
37 // pipe 1 IF_ID
38 logic flush_IFID_;
39 logic [31:0] inst_r;
40 assign rst_or_flush_IFID_ = rst | flush_IFID_;
41 always_ff @(posedge clk) begin
42     if (rst_or_flush_IFID_) begin
43         inst_r <= `I_NOP;
44         pc_r <= 0;
45     end
46     else begin
47         inst_r <= inst_;
48         pc_r <= pc;
49     end
50 end
51
52 // INST_DEC
53 logic [6:0] funct7_, opcode_;
54 logic [4:0] addr_rs1_, addr_rs2_, addr_rd_;
55 logic [2:0] funct3_;
56 logic [31:0] imm_;
57 INST_DEC myinst_dec (
58     .inst_r (inst_r),
59     .funct7_ (funct7_),
60     .opcode_ (opcode_),
61     .addr_rs1_ (addr_rs1_),
62     .addr_rs2_ (addr_rs2_),
63     .addr_rd_ (addr_rd_),
64     .funct3_ (funct3_),
65     .imm_ (imm_)
66 );
```

```

68 // Reg file
69 logic write_regf_en_r;
70 logic [4:0] addr_rd_r;
71 logic [31:0] rd_value_, rs1_value, rs2_value;
72 Reg_file myreg (
73     .clk (clk),
74     .rst (rst),
75     .write_regf_en (write_regf_en_r),
76     .addr_rd (addr_rd_r),
77     .addr_rs1 (addr_rs1_),
78     .addr_rs2 (addr_rs2_),
79     .rd_value (rd_value_),
80     .rs1_value (rs1_value),
81     .rs2_value (rs2_value),
82     .regs_31 (regs_31)
83 );
84 logic flush_IDEX_;
85 // controller
86 logic sel_pc_, sel_alu_b_, sel_alu_b_r;
87 logic [3:0] op_, op_r;
88 logic write_regf_en_;
89 controller mycontroller(
90     // in
91     .funct7_ (funct7_),
92     .funct3_ (funct3_),
93     .opcode_ (opcode_),
94     .rst (rst),
95     .clk (clk),
96     .rs1_value (rs1_value),
97     .rs2_value (rs2_value),
98     // out
99     .sel_alu_b_ (sel_alu_b_),
100    .write_regf_en_ (write_regf_en_),
101    .flush_IFID_ (flush_IFID_),
102    .flush_IDEX_ (flush_IDEX_),
103    .rst_pc_ (rst_pc_),
104    .sel_pc_ (sel_pc_),
105    .op_ (op_)
106 );

```

```

108 // pipe 2 ID_EX
109 logic [31:0] imm_r, rs1_value_r, rs2_value_r, alu_b_, flush_IFID_r, flush_IDEX_r;
110 assign rst_or_flush_IDEX_ = rst | flush_IDEX_r;
111 always_ff @(posedge clk) begin
112     if (rst_or_flush_IDEX_) begin
113         write_regf_en_r <= 0;
114         addr_rd_r <= 0;
115         imm_r <= 0;
116         rs1_value_r <= 0;
117         rs2_value_r <= 0;
118         op_r <= 0;
119         sel_alu_b_r <= 0;
120         pc_rr <= 0;
121         flush_IDEX_r <= 0;
122         flush_IFID_r <= 0;
123         sel_pc_r <= 0;
124     end
125     else begin
126         write_regf_en_r <= write_regf_en_;
127         addr_rd_r <= addr_rd_;
128         imm_r <= imm_;
129         rs1_value_r <= rs1_value;
130         rs2_value_r <= rs2_value;
131         op_r <= op_;
132         sel_alu_b_r <= sel_alu_b_;
133         pc_rr <= pc_r;
134         flush_IDEX_r <= flush_IDEX_;
135         flush_IFID_r <= flush_IFID_;
136         sel_pc_r <= sel_pc_;
137     end
138 end

140 // multi 2 to 1
141 always_comb begin
142     case (sel_alu_b_r)
143         1'd0: alu_b_ = imm_r;
144         1'd1: alu_b_ = rs2_value_r;
145     endcase
146 end

147
148 // alu
149 myalu alu_1 (
150     .op (op_r),
151     .alu_a (rs1_value_r),
152     .alu_b (alu_b_),
153     .alu_out (rd_value_)
154 );

155
156 // jump_addr_ (adder)
157 assign jump_addr_ = pc_rr + imm_r;
158 endmodule

```

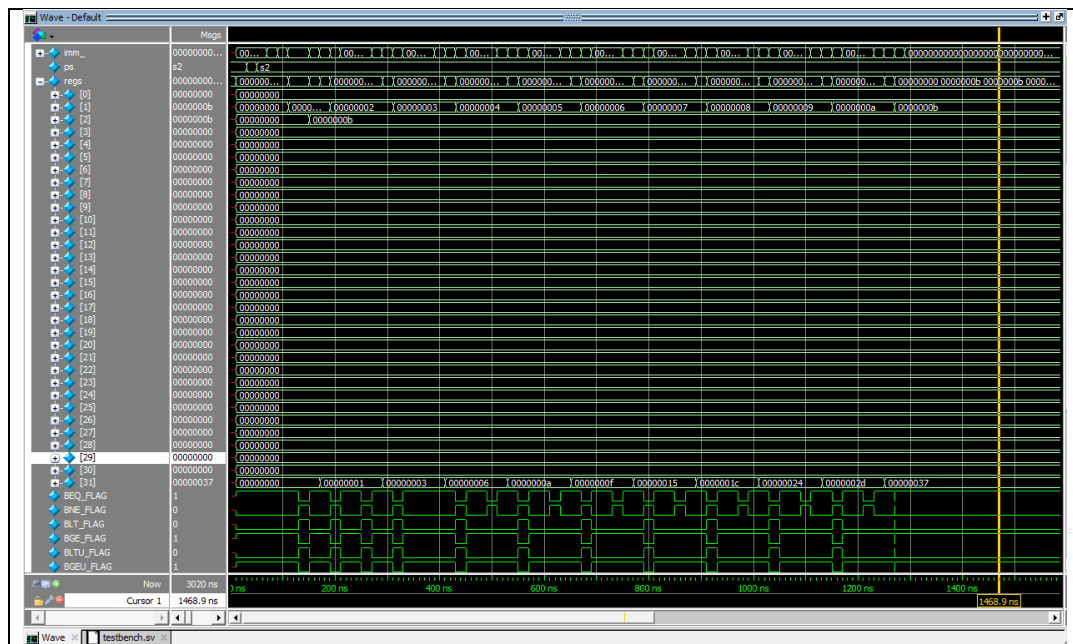
```

1  module Program_Rom(
2      output logic [31:0] Rom_data,
3      input [31:0] Rom_addr
4  );
5
6      always_comb begin
7          case (Rom_addr)
8              32'h00000000 : Rom_data = 32'h00100093; // addi x1 x0 1
9              32'h00000004 : Rom_data = 32'h00000F93; // addi x31 x0 0
10             32'h00000008 : Rom_data = 32'h00B00113; // addi x2 x0 11
11             32'h0000000C : Rom_data = 32'h001F8FB3; // add x31 x31 x1
12             32'h00000010 : Rom_data = 32'h00108093; // addi x1 x1 1
13             32'h00000014 : Rom_data = 32'h00000013; // nop
14             32'h00000018 : Rom_data = 32'hFE20CAE3; // blt x1 x2 -12
15             32'h0000001C : Rom_data = 32'h00000013; // addi x0 x0 0
16             default : Rom_data = 32'h00000013; // NOP
17         endcase
18     end
19
20 endmodule

```

本人定 X1 存此次要加總的數字，X2 存放加總的盡頭(11)，X31 由題目指定存放解答。其中可以看到 addi 與 blt 中有著 Hazard，加入 Nop 使 addi 的 EX 週期卡著 blt 的 IF 週期，當 blt 進入 ID 時 X1 值已經過 addi 的 EX 週期。

●模擬結果與結果說明：

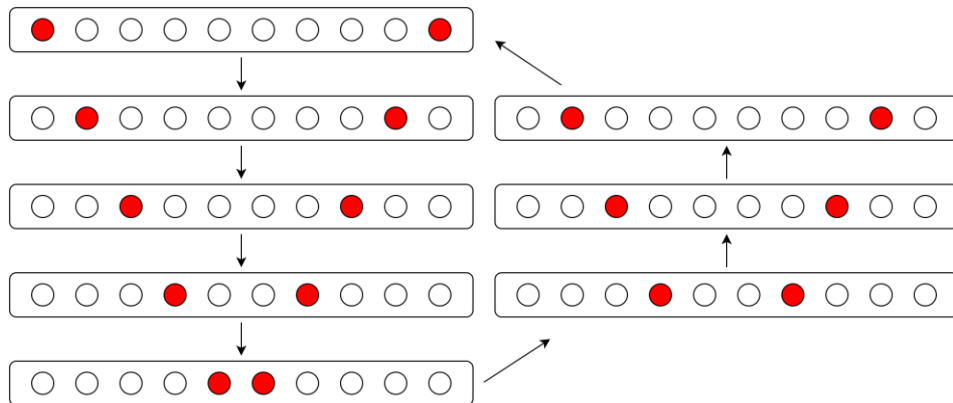


由 reg[31]可看出確實從 1 加至 55(Hex: 37)

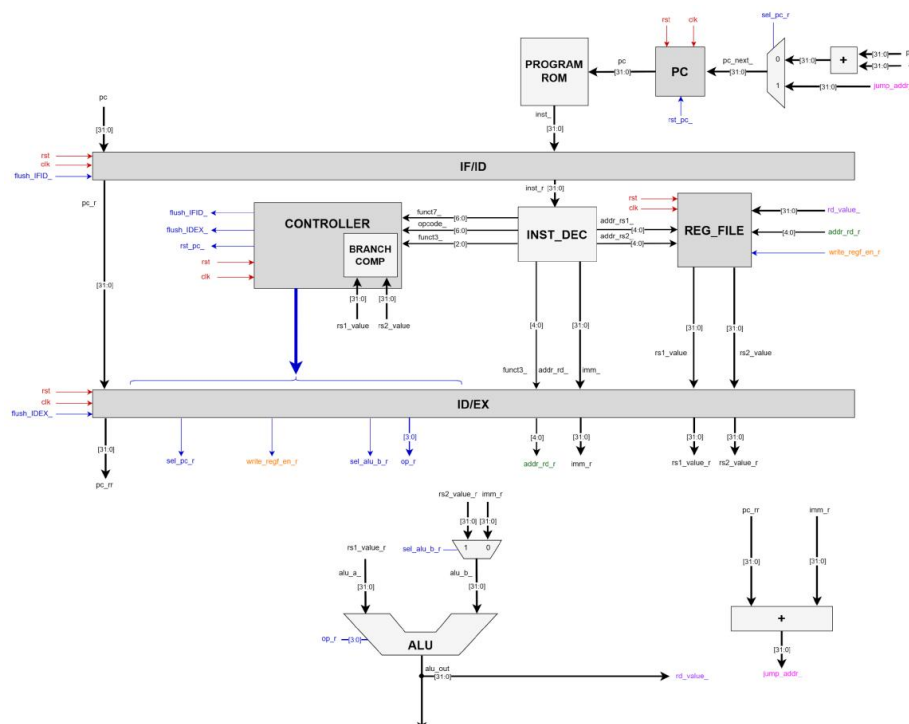
2、組語撰寫練習—LED

●實驗說明：

1. 透過當前課堂所學的所有組合語言撰寫一個 10 個 LED 交錯閃爍的程式，如下圖所示，程式碼須加註解。
2. 所有暫存器皆可使用，但請將結果存儲至暫存器 x31 較低的 10 個位元。
3. 程式撰寫完成後透過 Tool Chain 將其轉換成 Program_ROM.sv 檔，並使用 ModelSim 軟體觀察波形是否執行正確。
4. 請將所有用到的暫存器都加入至波形中以方便觀察。
5. 請注意資料危障的問題。



●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**

Clock_divider.sv

```
clock_divider.v > ...
1  `timescale 1ns/100ps
2  module clock_divider(
3      input logic clk,
4      input logic rst,
5      input logic [31:0] DIVISOR,
6      //
7      output logic clk_out
8  );
9
10     logic [32:0] counter;
11
12     always_ff @(posedge clk or posedge rst) begin
13         if (rst) begin
14             counter <= 0;
15             clk_out <= 0;
16         end else begin
17             if (counter == (DIVISOR - 1)) begin
18                 counter <= 0;
19                 clk_out <= ~clk_out;
20             end else begin
21                 counter <= counter + 1;
22             end
23         end
24     end
25
26 endmodule
```

Controller.v

```

1  `include "mydefine.sv"
2  `timescale 1ns/100ps
3  module controller (
4      input logic [6:0] funct7_, opcode_,
5      input logic rst, clk,
6      input logic [2:0] funct3_,
7      input logic [31:0] rs1_value, rs2_value,
8      output logic flush_IFID_, flush_IDEX_, rst_pc_, sel_pc_, write_regf_en_, sel_alu_b_,
9      output logic [3:0] op_
10 );
11     logic BEQ_FLAG;
12     logic BNE_FLAG;
13     logic BLT_FLAG;
14     logic BGE_FLAG;
15     logic BLTU_FLAG;
16     logic BGEU_FLAG;
17
18     assign BEQ_FLAG = (rs1_value == rs2_value);
19     assign BNE_FLAG = (rs1_value != rs2_value);
20     assign BLT_FLAG = ($signed(rs1_value) < $signed(rs2_value));
21     assign BGE_FLAG = ($signed(rs1_value) >= $signed(rs2_value));
22     assign BLTU_FLAG = (rs1_value < rs2_value);
23     assign BGEU_FLAG = (rs1_value >= rs2_value);
24
25     typedef enum {s0, s1, s2} fsm_state;
26     fsm_state ps, ns;
27
28     always_ff @(posedge clk) begin
29         if (rst) begin
30             ps <= #1 s0;
31         end
32         else begin
33             ps <= #1 ns;
34         end
35     end

```

```
37     always_comb begin
38         rst_pc_ = 0;
39         sel_pc_ = 0;
40         flush_IFID_ = 0;
41         flush_IDEX_ = 0;
42         write_regf_en_ = 0;
43         sel_alu_b_ = 0;
44         ns = ps;
45         op_ = `ALUOP_ADD;
46         unique case (ps)
47             s0: begin
48                 flush_IFID_ = 1;
49                 flush_IDEX_ = 1;
50                 rst_pc_ = 1;
51                 ns = s1;
52             end
53             s1: begin
54                 flush_IFID_ = 1;
55                 flush_IDEX_ = 1;
56                 rst_pc_ = 1;
57                 ns = s2;
58             end
59         end
```

```

59     s2: begin
60         case (opcode_)
61             `Opcode_I: begin
62                 unique case (funct3_)
63                     `F_ADDI: begin
64                         op_ = `ALUOP_ADD;
65                         write_regf_en_ = 1;
66                     end
67                     `F_SLTI: begin
68                         op_ = `ALUOP_LT;
69                         write_regf_en_ = 1;
70                     end
71                     `F_SLTIU: begin
72                         op_ = `ALUOP_LTU;
73                         write_regf_en_ = 1;
74                     end
75                     `F_ANDI: begin
76                         op_ = `ALUOP_AND;
77                         write_regf_en_ = 1;
78                     end
79                     `F_ORI: begin
80                         op_ = `ALUOP_OR;
81                         write_regf_en_ = 1;
82                     end
83                     `F_XORI: begin
84                         op_ = `ALUOP_XOR;
85                         write_regf_en_ = 1;
86                     end
87                     `F_SLLI: begin
88                         op_ = `ALUOP_SLL;
89                         write_regf_en_ = 1;
90                     end

```

```

91         `F_SRLI_SRAI: begin
92             unique case (funct7_)
93                 `F7_SRLI: begin
94                     op_ = `ALUOP_SRL;
95                     write_regf_en_ = 1;
96                 end
97                 `F7_SRAI: begin
98                     op_ = `ALUOP_SRA;
99                     write_regf_en_ = 1;
100                 end
101             endcase
102         end
103     endcase
104 end
105 `Opcode_R_M: begin
106     unique case (funct3_)
107         `F_AND: begin
108             if (funct7_ == 7'b000_0000 || funct7_ == `F7_OPCODE_R) begin
109                 op_ = `ALUOP_AND;
110                 write_regf_en_ = 1;
111                 sel_alu_b_ = 1;
112             end
113         end
114         `F_OR: begin
115             if (funct7_ == `F7_OPCODE_R) begin
116                 op_ = `ALUOP_OR;
117                 write_regf_en_ = 1;
118                 sel_alu_b_ = 1;
119             end
120         end
121         `F_XOR: begin
122             if (funct7_ == `F7_OPCODE_R) begin
123                 op_ = `ALUOP_XOR;
124                 write_regf_en_ = 1;
125                 sel_alu_b_ = 1;
126             end
127         end
128         `F_ADD_SUB: begin
129             unique case (funct7_)
130                 `F7_ADD: begin
131                     op_ = `ALUOP_ADD;
132                     write_regf_en_ = 1;
133                     sel_alu_b_ = 1;
134                 end
135                 `F7_SUB: begin
136                     op_ = `ALUOP_SUB;
137                     write_regf_en_ = 1;
138                     sel_alu_b_ = 1;
139                 end
140             endcase
141         end
142         `F_SLT: begin
143             if (funct7_ == `F7_OPCODE_R) begin
144                 op_ = `ALUOP_LT;
145                 write_regf_en_ = 1;
146                 sel_alu_b_ = 1;
147             end
148         end
149         `F_SLTU: begin
150             if (funct7_ == `F7_OPCODE_R) begin
151                 op_ = `ALUOP_LTU;
152                 write_regf_en_ = 1;
153                 sel_alu_b_ = 1;
154             end
155         end

```



```

156     `F_SLL: begin
157         if (funct7_ == `F7_OPCODE_R) begin
158             op_ = `ALUOP_SLL;
159             write_regf_en_ = 1;
160             sel_alu_b_ = 1;
161         end
162     end
163     `F_SRL_SRA: begin
164         unique case (funct7_)
165             `F7_OPCODE_R: begin
166                 op_ = `ALUOP_SRL; // SRL
167                 write_regf_en_ = 1;
168                 sel_alu_b_ = 1;
169             end
170             `F7_SRA: begin
171                 op_ = `ALUOP_SRA; // SRA (if different funct7)
172                 write_regf_en_ = 1;
173                 sel_alu_b_ = 1;
174             end
175         endcase
176     end
177 endcase
178 end

179     `Opcode_B: begin
180         unique case (funct3_)
181             `F_BEQ: begin *****
182                 if (BEQ_FLAG) begin
183                     sel_pc_ = 1;
184                     flush_IDEX_ = 1;
185                     flush_IFID_ = 1;
186                 end
187             end
188             `F_BNE: begin
189                 if (BNE_FLAG) begin
190                     sel_pc_ = 1;
191                     flush_IDEX_ = 1;
192                     flush_IFID_ = 1;
193                 end
194             end
195             `F_BLT: begin
196                 if (BLT_FLAG) begin
197                     sel_pc_ = 1;
198                     flush_IFID_ = 1;
199                     flush_IDEX_ = 1;
200                 end
201             end
202             `F_BGE: begin
203                 if (BGE_FLAG) begin
204                     sel_pc_ = 1;
205                     flush_IDEX_ = 1;
206                     flush_IFID_ = 1;
207                 end
208             end

```

```

209         `F_BLTU: begin
210             if (BLTU_FLAG) begin
211                 sel_pc_ = 1;
212                 flush_IDEX_ = 1;
213                 flush_IFID_ = 1;
214             end
215         end
216         `F_BGEU: begin
217             if (BGEU_FLAG) begin
218                 sel_pc_ = 1;
219                 flush_IDEX_ = 1;
220                 flush_IFID_ = 1;
221             end
222         end
223     endcase
224 end
225 endcase
226 end
227 endcase
228 end
229 endmodule

```

INST_DEC.sv

```

INST_DEC.sv > INST_DEC
1  `include "mydefine.sv"
2  module INST_DEC (
3      input logic [31:0] inst_r,
4      output logic [6:0] funct7_, opcode_,
5      output logic [4:0] addr_rs1_, addr_rs2_, addr_rd_,
6      output logic [2:0] funct3_,
7      output logic [31:0] imm_
8  );
9      assign opcode_ = inst_r[6:0];
10     assign funct3_ = inst_r[14:12];
11     assign addr_rd_ = inst_r[11:7];
12     assign addr_rs1_ = inst_r[19:15];
13     assign addr_rs2_ = inst_r[24:20];
14     assign funct7_ = inst_r[31:25];
15
16     logic [31:0] IMM_I;
17     logic [31:0] IMM_B;
18     assign IMM_I = {{20{inst_r[31]}}, inst_r[31:20]};
19     assign IMM_B = {{20{inst_r[31]}}, inst_r[7], inst_r[30:25], inst_r[11:8], 1'd0};
20     always_comb begin
21         unique case (opcode_)
22             `Opcode_I: imm_ = IMM_I;
23             `Opcode_B: imm_ = IMM_B;
24         endcase
25     end
26
27 endmodule

```

Myalu.sv

```

1  `include "mydefine.sv"
2  module myalu (
3      input logic [3:0] op,
4      input logic [31:0] alu_a,
5      input logic [31:0] alu_b,
6      output logic [31:0] alu_out
7  );
8      always_comb begin
9          unique case (op)
10             `ALUOP_ADD : alu_out = alu_a + alu_b;
11             `ALUOP_SUB : alu_out = $signed(alu_a) - $signed(alu_b);
12             `ALUOP_AND : alu_out = alu_a & alu_b;
13             `ALUOP_OR  : alu_out = alu_a | alu_b;
14             `ALUOP_XOR : alu_out = alu_a ^ alu_b;
15             `ALUOP_A   : alu_out = alu_a;
16             `ALUOP_A_ADD_4 : alu_out = alu_a + 4;
17             `ALUOP_LTU : alu_out = alu_a < alu_b;
18             `ALUOP_LT  : alu_out = $signed(alu_a) < $signed(alu_b);
19             `ALUOP_SLL : alu_out = alu_a << alu_b[4:0];
20             `ALUOP_SRL : alu_out = alu_a >> alu_b[4:0];
21             `ALUOP_SRA : alu_out = $signed(alu_a) >>> alu_b[4:0];
22             `ALUOP_B   : alu_out = alu_b;
23             default : alu_out = alu_a;
24          endcase
25      end
26  endmodule

```

Mydefine.sv

mydefine.sv > ...

```
1  `define I_NOP 32'h13
2
3  `define Opcode_I 7'b0010011
4  `define Opcode_R_M 7'b0110011
5  `define Opcode_B 7'b1100011
6
7  // alu operation
8  `define ALUOP_ADD 4'h0
9  `define ALUOP_SUB 4'h1
10 `define ALUOP_AND 4'h2
11 `define ALUOP_OR 4'h3
12 `define ALUOP_XOR 4'h4
13 `define ALUOP_A 4'h5
14 `define ALUOP_A_ADD_4 4'h6
15 `define ALUOP_LTU 4'h7
16 `define ALUOP_LT 4'h8
17 `define ALUOP_SLL 4'h9
18 `define ALUOP_SRL 4'hA
19 `define ALUOP_SRA 4'hB
20 `define ALUOP_B 4'hC
21
22 // function 3
23 `define F_ADDI 3'b000
24 `define F_SLTI 3'b010
25 `define F_SLTIU 3'b011
26 `define F_XORI 3'b100
27 `define F_ORI 3'b110
28 `define F_ANDI 3'b111
29 `define F_SLLI 3'b001
30 `define F_SRLI_SRAI 3'b101
```

```
32 // function 7
33 `define F7_ADD 7'b0000000
34 `define F7_SUB 7'b0100000
35 `define F7_SRLI 7'b0000000
36 `define F7_SRAI 7'b0100000
37 `define F7_OPCODE_R 7'b0000000
38 `define F7_SRL 7'b0000000
39 `define F7_SRA 7'b0100000
40
41 // alu
42 `define F_ADD_SUB 3'b000
43 `define F_SLL 3'b001
44 `define F_SLT 3'b010
45 `define F_SLTU 3'b011
46 `define F_XOR 3'b100
47 `define F_SRL_SRA 3'b101
48 `define F_OR 3'b110
49 `define F_AND 3'b111
50
51 // branch
52 `define F_BEQ 3'b000
53 `define F_BNE 3'b001
54 `define F_BLT 3'b100
55 `define F_BGE 3'b101
56 `define F_BLTU 3'b110
57 `define F_BGEU 3'b111
```

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];
26
27     always_ff@(posedge clk)
28     begin
29         if(rst) begin
30             for(i = 0; i < 32; i = i+1) begin:rst_keywords
31                 regs[i] <= 0;
32             end
33         end
34         else begin
35             // Write
36             if (write_regf_en && addr_rd_not_0)
37                 regs[addr_rd] <= #1 rd_value;
38         end
39     end
40
41 endmodule

```

Seven_segment_display.sv


```

seven_segment_display.sv > ...
1  `timescale 1ns/100ps
2  module seven_segment_display(
3      input logic [3:0] digit,    // 4位元二進制數字輸入
4      output logic [6:0] seg      // 7段顯示器輸出 (g, f, e, d, c, b, a)
5  );
6      //共陽
7      always_comb begin
8          case (digit)
9              4'h0: seg = 7'b1000000;    // 顯示 "0"
10             4'h1: seg = 7'b1111001;    // 顯示 "1"
11             4'h2: seg = 7'b0100100;    // 顯示 "2"
12             4'h3: seg = 7'b0110000;    // 顯示 "3"
13             4'h4: seg = 7'b0011001;    // 顯示 "4"
14             4'h5: seg = 7'b0010010;    // 顯示 "5"
15             4'h6: seg = 7'b0000010;    // 顯示 "6"
16             4'h7: seg = 7'b1111000;    // 顯示 "7"
17             4'h8: seg = 7'b0000000;    // 顯示 "8"
18             4'h9: seg = 7'b0010000;    // 顯示 "9"
19             4'ha: seg = 7'b0001000;    // 顯示 "A"
20             4'hb: seg = 7'b0000011;    // 顯示 "B"
21             4'hc: seg = 7'b1000110;    // 顯示 "C"
22             4'hd: seg = 7'b0100001;    // 顯示 "D"
23             4'he: seg = 7'b0000110;    // 顯示 "E"
24             4'hf: seg = 7'b0001110;    // 顯示 "F"
25             default: seg = 7'b1111111; // 不顯示
26         endcase
27     end
28
29 endmodule

```

Mycpu.sv

```
mycpu.v > mycpu > [ ] alu_b_
1  `include "mydefine.sv"
2  module mycpu(
3      input logic clk, rst,
4      output logic [31:0] regs_31
5  );
6      // program-counter *****
7      logic rst_pc_, sel_pc_r;
8      logic [31:0] pc, pc_next, pc_r, pc_rr, jump_addr_;
9      always_comb begin
10         case (sel_pc_r)
11             1'd0: begin
12                 pc_next = pc + 4;
13             end
14             1'd1: begin
15                 pc_next = jump_addr_;
16             end
17         endcase
18     end
19     always_ff @(posedge clk) begin
20         if (rst|rst_pc_) begin
21             pc <= 0;
22         end
23         else begin
24             pc <= pc_next;
25         end
26     end
27
28     logic [31:0] inst_;
29     // program rom
30     Program_Rom myprogram_rom (
31         // in
32         .Rom_addr (pc),
33         // out
34         .Rom_data (inst_)
35     );
```

```
37 // pipe 1 IF_ID
38 logic flush_IFID_;
39 logic [31:0] inst_r;
40 assign rst_or_flush_IFID_ = rst | flush_IFID_;
41 always_ff @(posedge clk) begin
42     if (rst_or_flush_IFID_) begin
43         inst_r <= `I_NOP;
44         pc_r <= 0;
45     end
46     else begin
47         inst_r <= inst_;
48         pc_r <= pc;
49     end
50 end
51
52 // INST_DEC
53 logic [6:0] funct7_, opcode_;
54 logic [4:0] addr_rs1_, addr_rs2_, addr_rd_;
55 logic [2:0] funct3_;
56 logic [31:0] imm_;
57 INST_DEC myinst_dec (
58     .inst_r (inst_r),
59     .funct7_ (funct7_),
60     .opcode_ (opcode_),
61     .addr_rs1_ (addr_rs1_),
62     .addr_rs2_ (addr_rs2_),
63     .addr_rd_ (addr_rd_),
64     .funct3_ (funct3_),
65     .imm_ (imm_)
66 );
```

```

68 // Reg file
69 logic write_regf_en_r;
70 logic [4:0] addr_rd_r;
71 logic [31:0] rd_value_, rs1_value, rs2_value;
72 Reg_file myreg (
73     .clk (clk),
74     .rst (rst),
75     .write_regf_en (write_regf_en_r),
76     .addr_rd (addr_rd_r),
77     .addr_rs1 (addr_rs1_),
78     .addr_rs2 (addr_rs2_),
79     .rd_value (rd_value_),
80     .rs1_value (rs1_value),
81     .rs2_value (rs2_value),
82     .regs_31 (regs_31)
83 );
84 logic flush_IDEX_;
85 // controller
86 logic sel_pc_, sel_alu_b_, sel_alu_b_r;
87 logic [3:0] op_, op_r;
88 logic write_regf_en_;
89 controller mycontroller(
90     // in
91     .funct7_ (funct7_),
92     .funct3_ (funct3_),
93     .opcode_ (opcode_),
94     .rst (rst),
95     .clk (clk),
96     .rs1_value (rs1_value),
97     .rs2_value (rs2_value),
98     // out
99     .sel_alu_b_ (sel_alu_b_),
100     .write_regf_en_ (write_regf_en_),
101     .flush_IFID_ (flush_IFID_),
102     .flush_IDEX_ (flush_IDEX_),
103     .rst_pc_ (rst_pc_),
104     .sel_pc_ (sel_pc_),
105     .op_ (op_)
106 );

```

```

108 // pipe 2 ID_EX
109 logic [31:0] imm_r, rs1_value_r, rs2_value_r, alu_b_, flush_IFID_r, flush_IDEX_r;
110 assign rst_or_flush_IDEX_ = rst | flush_IDEX_r;
111 always_ff @(posedge clk) begin
112     if (rst_or_flush_IDEX_) begin
113         write_regf_en_r <= 0;
114         addr_rd_r <= 0;
115         imm_r <= 0;
116         rs1_value_r <= 0;
117         rs2_value_r <= 0;
118         op_r <= 0;
119         sel_alu_b_r <= 0;
120         pc_rr <= 0;
121         flush_IDEX_r <= 0;
122         flush_IFID_r <= 0;
123         sel_pc_r <= 0;
124     end
125     else begin
126         write_regf_en_r <= write_regf_en_;
127         addr_rd_r <= addr_rd_;
128         imm_r <= imm_;
129         rs1_value_r <= rs1_value;
130         rs2_value_r <= rs2_value;
131         op_r <= op_;
132         sel_alu_b_r <= sel_alu_b_;
133         pc_rr <= pc_r;
134         flush_IDEX_r <= flush_IDEX_;
135         flush_IFID_r <= flush_IFID_;
136         sel_pc_r <= sel_pc_;
137     end
138 end

140 // multi 2 to 1
141 always_comb begin
142     case (sel_alu_b_r)
143         1'd0: alu_b_ = imm_r;
144         1'd1: alu_b_ = rs2_value_r;
145     endcase
146 end

147
148 // alu
149 myalu alu_1 (
150     .op (op_r),
151     .alu_a (rs1_value_r),
152     .alu_b (alu_b_),
153     .alu_out (rd_value_)
154 );

155
156 // jump_addr_ (adder)
157 assign jump_addr_ = pc_rr + imm_r;
158 endmodule

```



```

init: addi X1 X0 512 // left light
      addi X2 X0 1 // right light
      addi X3 X0 6 // First loop 5 time
      addi X4 X0 9 // Second loop 3 time
      addi X5 X0 1 // cnt loop time
      addi X6 X0 1 // shift left, right 1 bit
firstLoop: add X31 X1 X2 // show light
            add X5 X5 X6 // cnt time plus one
            srl X1 X1 X6 // left light shift right
            sll X2 X2 X6 // right light shift left
            blt X5 X3 firstLoop // if X5 < 6, loop
            nop
            nop
secLoop: srl X1 X1 X6 // left light shift right
          sll X2 X2 X6 // right light shift left
          add X5 X5 X6 // cnt plus one
          add X31 X1 X2 // show light
          blt X5 X4 secLoop // if X5 < 9, secLoop
          nop
          nop
          beq X0 X0 init // loop forever
          nop

```

no hazard

hazard of blt

no hazard

no hazard

hazard of blt

201 → 102 → 84 → 48 → 30 → 48 → 84 → 102

↑

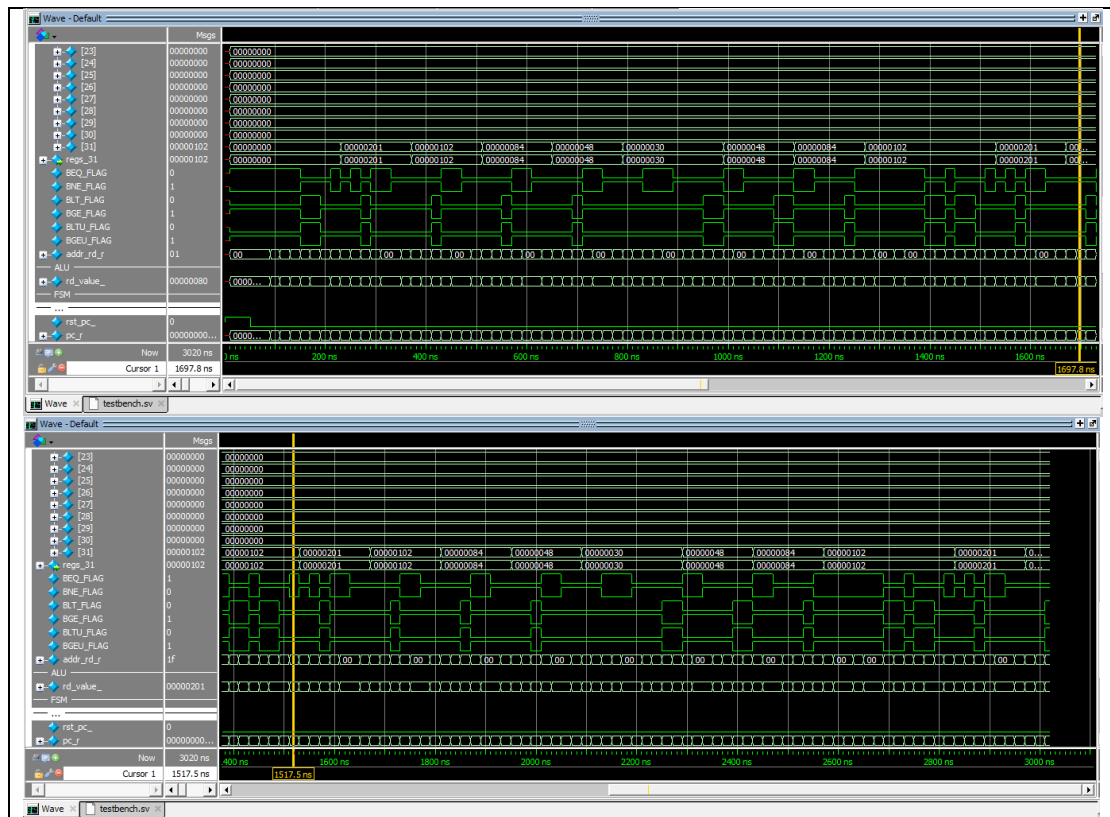

```

Program_Rom.sv > Program_Rom
1  module Program_Rom(
2      output logic [31:0] Rom_data,
3      input [31:0] Rom_addr
4  );
5
6      always_comb begin
7          case (Rom_addr)
8              32'h00000000 : Rom_data = 32'h20000093; // addi x1 x0 512
9              32'h00000004 : Rom_data = 32'h00100113; // addi x2 x0 1
10             32'h00000008 : Rom_data = 32'h00600193; // addi x3 x0 6
11             32'h0000000C : Rom_data = 32'h00900213; // addi x4 x0 9
12             32'h00000010 : Rom_data = 32'h00100293; // addi x5 x0 1
13             32'h00000014 : Rom_data = 32'h00100313; // addi x6 x0 1
14             32'h00000018 : Rom_data = 32'h00208FB3; // add x31 x1 x2
15             32'h0000001C : Rom_data = 32'h006282B3; // add x5 x5 x6
16             32'h00000020 : Rom_data = 32'h0060D0B3; // srl x1 x1 x6
17             32'h00000024 : Rom_data = 32'h00611133; // sll x2 x2 x6
18             32'h00000028 : Rom_data = 32'hFE32C8E3; // blt x5 x3 -16
19             32'h0000002C : Rom_data = 32'h00000013; // addi x0 x0 0
20             32'h00000030 : Rom_data = 32'h00000013; // addi x0 x0 0
21             32'h00000034 : Rom_data = 32'h0060D0B3; // srl x1 x1 x6
22             32'h00000038 : Rom_data = 32'h00611133; // sll x2 x2 x6
23             32'h0000003C : Rom_data = 32'h006282B3; // add x5 x5 x6
24             32'h00000040 : Rom_data = 32'h00208FB3; // add x31 x1 x2
25             32'h00000044 : Rom_data = 32'hFE42C8E3; // blt x5 x4 -16
26             32'h00000048 : Rom_data = 32'h00000013; // addi x0 x0 0
27             32'h0000004C : Rom_data = 32'h00000013; // addi x0 x0 0
28             32'h00000050 : Rom_data = 32'hFA0008E3; // beq x0 x0 -80
29             32'h00000054 : Rom_data = 32'h00000013; // addi x0 x0 0
30             default : Rom_data = 32'h00000013; // NOP
31         endcase
32     end
33
34 endmodule

```

在想法(第一張圖)階段即有考慮 **hazard** 問題，具體如 **cnt+1** 至少在 **blt** 前兩個，保證 **blt** 的 **ID** 階段有值; **blt** 後保留兩個 **Nop** 確保不會有指令進到 **IF/ID** 階段。

●模擬結果與結果說明：



由上圖可以看到 reg[31]中內容與題目要求的燈號相同，經 Binary 轉換成 HEX 後確實也是如此，且由於 $x0=x0$ ，此程式會永遠 Loop

● 結論與心得：

感謝助教幫我 Debug，讓我卡兩天的第一題獲得了解答，終於開始進入迴圈了：)