



2025/11/28

# 實驗十一

姓名：林承羿

學號：01257027

班級：資工 3A

E-mail：[IanLin6225@gmail.com](mailto: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\_HW11

## 1、組語撰寫練習—排序

### ● 實驗說明：

1. 如下圖 1，在記憶體位址 10~17 中共有 8 筆亂序的 8 位元資料，請透過當前課堂所學的所有組合語言撰寫一個排序演算法將 8 筆資料由小排到大，並且放入相同的記憶體位址中。程式碼須加註解。
2. 如下圖 2，已提供部分組合語言程式碼，請從中間的地方開始撰寫您的排序演算法。
3. 所有暫存器皆可使用，但請將結果透過 LW 指令載入到 x30 和 x31 暫存器。
4. 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。
5. 請將所有用到的暫存器都加入至波形中以方便觀察。
6. 請注意資料危障的問題。

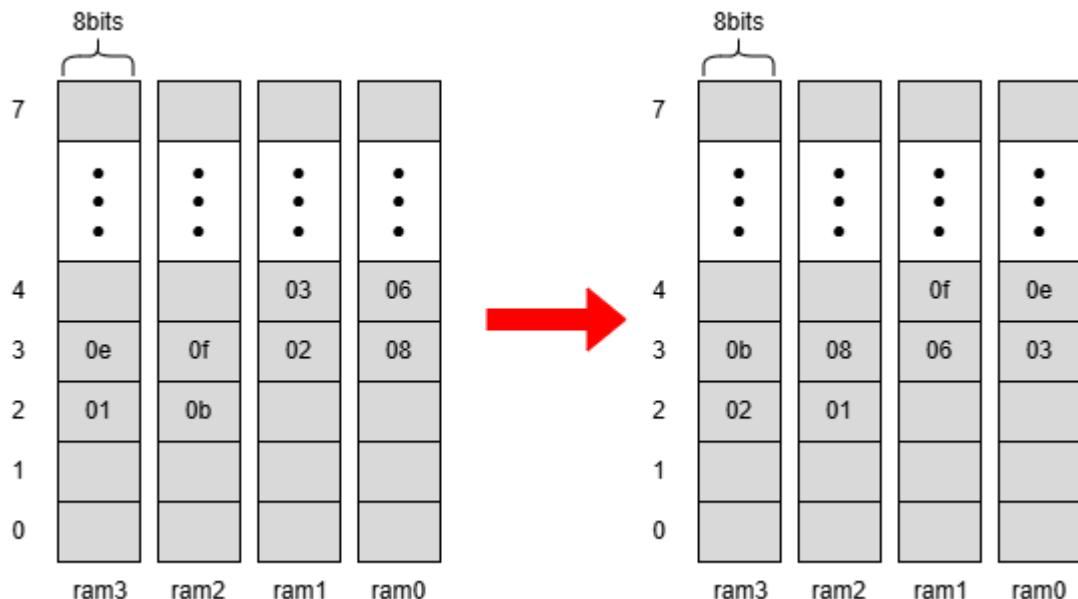


圖 1

```

init:
    li x2, 10
    li x3, 0x0208010b
    li x4, 0x03060e0f
    nop
    sw x3, 0(x2)
    sw x4, 4(x2)

```

#-----開始撰寫您的組合語言-----

#-----您的組合語言-----

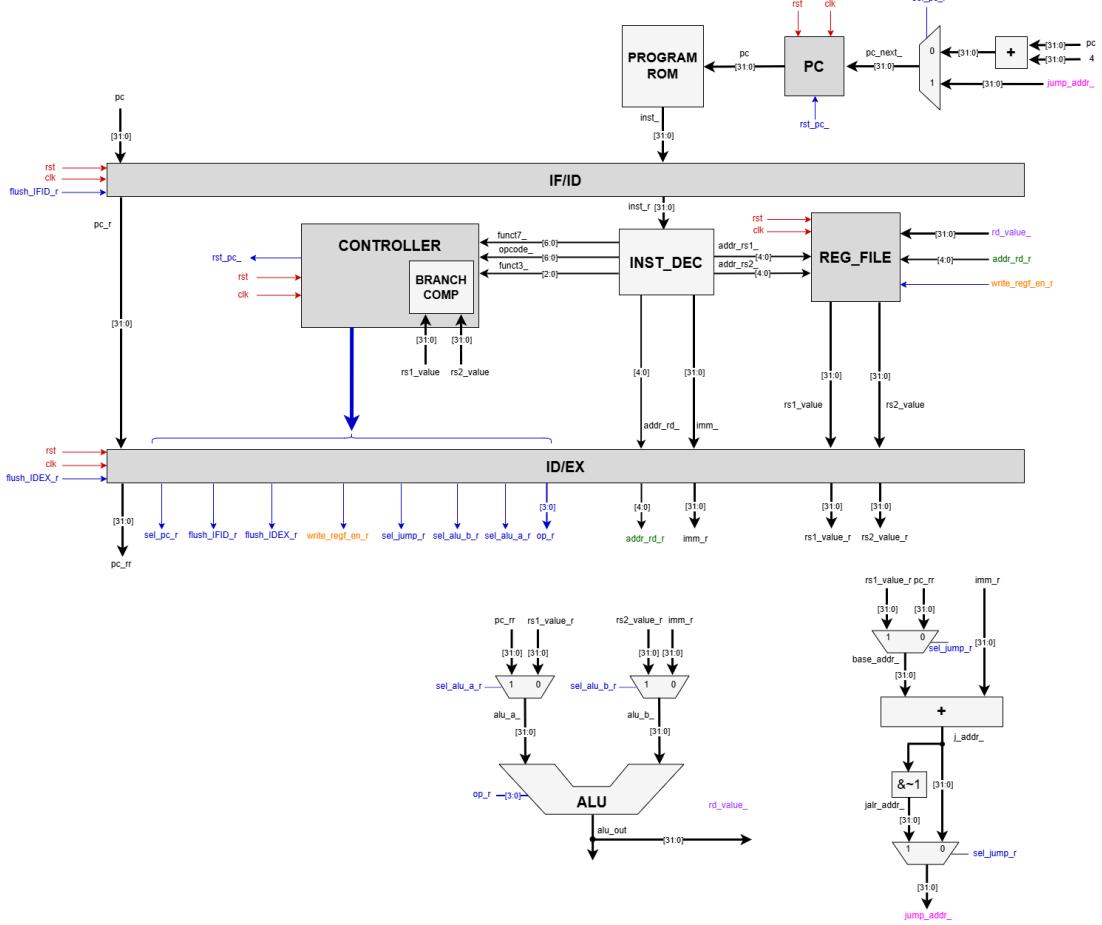
```

done:
    lw x30, 0(x2)
    lw x31, 4(x2)

```

圖 2

## ●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**

以下為此次作業 RISC-V 架構

**Mycpu.sv**

```
mycpu.sv > ...
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     );
36
```

```
37 // pipe 1 IF_ID
38 logic flush_IFID_r, flush_IFID_;
39 logic [31:0] inst_r;
40 assign rst_or_flush_IFID_r = rst | flush_IFID_r;
41 always_ff @(posedge clk) begin
42     if (rst_or_flush_IFID_r) 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_, funct3_r;
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 );
67
```

```
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, alu_out;
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_INDEX_;
85     // controller
86     logic sel_pc_, sel_alu_a_r, sel_jump_, sel_jump_r, sel_alu_a_, write_ram_, write_ram_r, sel_rd_value_, sel_rd_value_r;
87     logic [1:0] sel_alu_b_r, sel_alu_b_;
88     logic [3:0] op_, op_r;
89     logic write_regf_en_;
90     controller mycontroller(
91         // in
92         .funct7_ (funct7_),
93         .funct3_ (funct3_),
94         .opcode_ (opcode_),
95         .rst (rst),
96         .clk (clk),
97         .rs1_value (rs1_value),
98         .rs2_value (rs2_value),
99         // out
100        .sel_alu_b_ (sel_alu_b_),
101        .write_regf_en_ (write_regf_en_),
102        .flush_IFID_ (flush_IFID_),
103        .flush INDEX_ (flush INDEX_),
104        .rst_pc_ (rst_pc_),
105        .sel_pc_ (sel_pc_),
106        .op_ (op_),
107        .sel_jump_ (sel_jump_),
108        .sel_alu_a_ (sel_alu_a_),
109        .write_ram_ (write_ram_),
110        .sel_rd_value_ (sel_rd_value_)
111    );
```

```
113 // pipe 2 ID_EX
114 logic [31:0] imm_r, rs1_value_r, rs2_value_r, alu_b_, alu_a_, flush_IDEX_r, base_addr_, j_addr_;
115 assign rst_or_flush_IDEX_r = rst | flush_IDEX_r;
116 always_ff @(posedge clk) begin
117     if (rst_or_flush_IDEX_r) begin
118         write_regf_en_r <= 0;
119         addr_rd_r <= 0;
120         imm_r <= 0;
121         rs1_value_r <= 0;
122         rs2_value_r <= 0;
123         op_r <= 0;
124         sel_alu_b_r <= 0;
125         pc_rr <= 0;
126         flush_IDEX_r <= 0;
127         flush_IFID_r <= 0;
128         sel_pc_r <= 0;
129         sel_jump_r <= 0;
130         sel_alu_a_r <= 0;
131         funct3_r <= 0;
132         write_ram_r <= 0;
133         sel_rd_value_r <= 0;
134     end
135     else begin
136         write_regf_en_r <= write_regf_en_;
137         addr_rd_r <= addr_rd_;
138         imm_r <= imm_;
139         rs1_value_r <= rs1_value;
140         rs2_value_r <= rs2_value;
141         op_r <= op_;
142         sel_alu_b_r <= sel_alu_b_;
143         pc_rr <= pc_r;
144         flush_IDEX_r <= flush_IDEX_;
145         flush_IFID_r <= flush_IFID_;
146         sel_pc_r <= sel_pc_;
147         sel_jump_r <= sel_jump_;
148         sel_alu_a_r <= sel_alu_a_;
149         funct3_r <= funct3_;
150         write_ram_r <= write_ram_;
151         sel_rd_value_r <= sel_rd_value_;
152     end
153 end
```

```

155     // multi 2 to 1 (sel_alu_a_r)
156     always_comb begin
157         case (sel_alu_a_r)
158             1'd0: alu_a_ = rs1_value_r;
159             1'd1: alu_a_ = pc_rr;
160         endcase
161     end
162
163     // multi 3 to 1 (sel_alu_b_r)
164     always_comb begin
165         unique case (sel_alu_b_r)
166             2'd0: alu_b_ = imm_r;
167             2'd1: alu_b_ = rs2_value_r;
168             2'd2: alu_b_ = 4;
169         endcase
170     end
171
172     // alu
173     myalu alu_1 (
174         .op (op_r),
175         .alu_a (alu_a_),
176         .alu_b (alu_b_),
177         .alu_out (alu_out)
178     );
179
180     // LSU
181     logic [31:0] read_data;
182     mylsu lsu1 (
183         .clk (clk),
184         .write_ram (write_ram_r),
185         .funct3 (funct3_r),
186         .write_data (rs2_value_r),
187         .ram_addr (alu_out),
188         .read_data (read_data)
189     );
190
191     assign rd_value_ = sel_rd_value_r ? read_data : alu_out;
192
193     // jump_addr_ (adder)
194     // 2 to 1 multi (sel_jump_r)
195     assign base_addr_ = sel_jump_r ? pc_rr : rs1_value_r;
196     assign j_addr_ = base_addr_ + imm_r;
197     assign jump_addr_ = {j_addr_[31:1], (j_addr_[0]&sel_jump_r)};
198 endmodule

```

## Controller.sv

```
controller.sv > 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_INDEX_, rst_pc_, write_regf_en_, sel_jump_, sel_alu_a_, write_ram_, sel_rd_value_,
9     output logic [1:0] sel_alu_b_,
10    output logic [3:0] op_
11 );
12    logic BEQ_FLAG;
13    logic BNE_FLAG;
14    logic BLT_FLAG;
15    logic BGE_FLAG;
16    logic BLTU_FLAG;
17    logic BGEU_FLAG;
18
19    assign BEQ_FLAG = (rs1_value == rs2_value);
20    assign BNE_FLAG = (rs1_value != rs2_value);
21    assign BLT_FLAG = ($signed(rs1_value) < $signed(rs2_value));
22    assign BGE_FLAG = ($signed(rs1_value) >= $signed(rs2_value));
23    assign BLTU_FLAG = (rs1_value < rs2_value);
24    assign BGEU_FLAG = (rs1_value >= rs2_value);
25
26    typedef enum {s0, s1, s2} fsm_state;
27    fsm_state ps, ns;
28
29    always_ff @(posedge clk) begin
30        if (rst) begin
31            ps <= #1 s0;
32        end
33        else begin
34            ps <= #1 ns;
35        end
36    end
```

```

37
38     always_comb begin
39         rst_pc_ = 0;
40         sel_pc_ = 0;
41         flush_IFID_ = 0;
42         flush_INDEX_ = 0;
43         write_regf_en_ = 0;
44         sel_alu_b_ = 0;
45         ns = ps;
46         op_ = `ALUOP_ADD;
47         sel_alu_a_ = 0;
48         sel_jump_ = 0;
49         sel_rd_value_ = 0;
50         write_ram_ = 0;
51         unique case (ps)
52             s0: begin
53                 flush_IFID_ = 1;
54                 flush_INDEX_ = 1;
55                 rst_pc_ = 1;
56                 ns = s1;
57             end
58             s1: begin
59                 flush_IFID_ = 1;
60                 flush_INDEX_ = 1;
61                 rst_pc_ = 1;
62                 ns = s2;
63             end
64             s2: begin
65                 case (opcode_)
66                     `Opcode_I: begin
67                         unique case (funct3_)
68                             `F_ADDI: begin
69                                 op_ = `ALUOP_ADD;
70                                 write_regf_en_ = 1;
71                             end
72                             `F_SLTI: begin
73                                 op_ = `ALUOP_LT;
74                                 write_regf_en_ = 1;
75                             end
76                             `F_SLTIU: begin
77                                 op_ = `ALUOP_LTU;
78                                 write_regf_en_ = 1;
79                             end
80                             `F_ANDI: begin
81                                 op_ = `ALUOP_AND;
82                                 write_regf_en_ = 1;
83                             end
84                             `F_ORI: begin
85                                 op_ = `ALUOP_OR;
86                                 write_regf_en_ = 1;
87                             end
88                             `F_XORI: begin
89                                 op_ = `ALUOP_XOR;
90                                 write_regf_en_ = 1;
91                             end

```

```

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

```

```

146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
end
`F_SLT: begin
    if (funct7_ == `F7_OPCODE_R) begin
        op_ = `ALUOP_LT;
        write_regf_en_ = 1;
        sel_alu_b_ = 1;
    end
end
`F_SLTU: begin
    if (funct7_ == `F7_OPCODE_R) begin
        op_ = `ALUOP_LTU;
        write_regf_en_ = 1;
        sel_alu_b_ = 1;
    end
end
`F_SLL: begin
    if (funct7_ == `F7_OPCODE_R) begin
        op_ = `ALUOP_SLL;
        write_regf_en_ = 1;
        sel_alu_b_ = 1;
    end
end
`F_SRL_SRA: begin
    unique case (funct7_)
        `F7_OPCODE_R: begin
            op_ = `ALUOP_SRL; // SRL
            write_regf_en_ = 1;
            sel_alu_b_ = 1;
        end
        `F7_SRA: begin
            op_ = `ALUOP_SRA; // SRA (if different funct7)
            write_regf_en_ = 1;
            sel_alu_b_ = 1;
        end
    endcase
end
`Opcode_B: begin
    sel_jump_ = 1;
    unique case (funct3_)
        `F_BEQ: begin
            if (BEQ_FLAG) begin
                sel_pc_ = 1;
                flush_INDEX_ = 1;
                flush_IFID_ = 1;
            end
        end
        `F_BNE: begin
            if (BNE_FLAG) begin
                sel_pc_ = 1;
                flush_INDEX_ = 1;
                flush_IFID_ = 1;
            end
        end
    endcase
end

```

```

200
201      end
202      `F_BLT: begin
203          if (BLT_FLAG) begin
204              sel_pc_ = 1;
205              flush_IFID_ = 1;
206              flush_INDEX_ = 1;
207          end
208      end
209      `F_BGE: begin
210          if (BGE_FLAG) begin
211              sel_pc_ = 1;
212              flush_INDEX_ = 1;
213              flush_IFID_ = 1;
214          end
215      end
216      `F_BLTU: begin
217          if (BLTU_FLAG) begin
218              sel_pc_ = 1;
219              flush_INDEX_ = 1;
220              flush_IFID_ = 1;
221          end
222      end
223      `F_BGEU: begin
224          if (BGEU_FLAG) begin
225              sel_pc_ = 1;
226              flush_INDEX_ = 1;
227              flush_IFID_ = 1;
228          end
229      endcase
230  end
231  `Opcode_JAL: begin
232      sel_pc_ = 1;
233      flush_IFID_ = 1;
234      flush_INDEX_ = 1;
235      sel_alu_a_ = 1;
236      sel_alu_b_ = 2;
237      sel_jump_ = 1;
238      op_ = `ALUOP_ADD;
239      write_regf_en_ = 1;
240  end
241  `Opcode_JALR: begin
242      sel_pc_ = 1;
243      flush_IFID_ = 1;
244      flush_INDEX_ = 1;
245      sel_alu_a_ = 1;
246      sel_alu_b_ = 2;
247      sel_jump_ = 0;
248      op_ = `ALUOP_ADD;
249      write_regf_en_ = 1;
250  end

```

```

251     `Opcode_LUI: begin
252         sel_alu_b_ = 0;
253         op_ = `ALUOP_B;
254         write_regf_en_ = 1;
255     end
256     `Opcode_AUIPC: begin
257         sel_alu_a_ = 1;
258         sel_alu_b_ = 0;
259         op_ = `ALUOP_ADD;
260         write_regf_en_ = 1;
261     end
262     `Opcode_L: begin
263         unique case (funct3_)
264             `F_LB: begin
265                 op_ = `ALUOP_ADD;
266                 sel_rd_value_ = 1;
267                 write_regf_en_ = 1;
268             end
269             `F_LH: begin
270                 op_ = `ALUOP_ADD;
271                 sel_rd_value_ = 1;
272                 write_regf_en_ = 1;
273             end
274             `F_LW: begin
275                 op_ = `ALUOP_ADD;
276                 sel_rd_value_ = 1;
277                 write_regf_en_ = 1;
278             end
279             `F_LBU: begin
280                 op_ = `ALUOP_ADD;
281                 sel_rd_value_ = 1;
282                 write_regf_en_ = 1;
283             end
284             `F_LHU: begin
285                 op_ = `ALUOP_ADD;
286                 sel_rd_value_ = 1;
287                 write_regf_en_ = 1;
288             end
289         endcase
290     end
291     `Opcode_S: begin
292         unique case (funct3_)
293             `F_SB: begin
294                 op_ = `ALUOP_ADD;
295                 write_ram_ = 1;
296             end
297             `F_SH: begin
298                 op_ = `ALUOP_ADD;
299                 write_ram_ = 1;
300             end
301             `F_SW: begin
302                 op_ = `ALUOP_ADD;
303                 write_ram_ = 1;
304             end

```

```

305           endcase
306       end
307   endcase
308 end
309 endcase
310 end
311 endmodule

```

## Inst\_DEC.sv

```

@ INST_DECsv > ...
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    logic [31:0] IMM_JAL;
19    logic [31:0] IMM_LUI_AUIPC;
20    logic [31:0] IMM_S;
21    assign IMM_I = {{20{inst_r[31]}}, inst_r[31:20]};
22    assign IMM_B = {{20{inst_r[31]}}, inst_r[7], inst_r[30:25], inst_r[11:8], 1'b0};
23    assign IMM_JAL = {{12{inst_r[31]}}, inst_r[19:12], inst_r[20], inst_r[30:21], 1'b0};
24    assign IMM_LUI_AUIPC = {inst_r[31:12], 12'b0};
25    assign IMM_S = {{20{inst_r[31]}}, inst_r[31:25], inst_r[11:7]};
26
27    always_comb begin
28        unique case (opcode_)
29            `Opcode_I: imm_ = IMM_I;
30            `Opcode_B: imm_ = IMM_B;
31            `Opcode_JAL: imm_ = IMM_JAL;
32            `Opcode_JALR: imm_ = IMM_I;
33            `Opcode_LUI: imm_ = IMM_LUI_AUIPC;
34            `Opcode_AUIPC: imm_ = IMM_LUI_AUIPC;
35            `Opcode_L: imm_ = IMM_I;
36            `Opcode_S: imm_ = IMM_S;
37        endcase
38    end
39
40 endmodule

```

## Myalu.sv

```
④ 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

```
mydefines.sv > LNOP
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 `define Opcode_L 7'b0000011
7 `define Opcode_S 7'b0100011
8
9 `define Opcode_JAL 7'b1101111
10 `define Opcode_JALR 7'b1100111
11 `define Opcode_LUI 7'b0110111
12 `define Opcode_AUIPC 7'b0010111
13
14 // alu operation
15 `define ALUOP_ADD 4'h0
16 `define ALUOP_SUB 4'h1
17 `define ALUOP_AND 4'h2
18 `define ALUOP_OR 4'h3
19 `define ALUOP_XOR 4'h4
20 `define ALUOP_A 4'h5
21 `define ALUOP_A_ADD_4 4'h6
22 `define ALUOP_LTU 4'h7
23 `define ALUOP_LT 4'h8
24 `define ALUOP_SLL 4'h9
25 `define ALUOP_SRL 4'hA
26 `define ALUOP_SRA 4'hB
27 `define ALUOP_B 4'hC
28
29 // function 3
30 `define F_ADDI 3'b000
31 `define F_SLTI 3'b010
32 `define F_SLTIU 3'b011
33 `define F_XORI 3'b100
34 `define F_ORI 3'b110
35 `define F_ANDI 3'b111
36 `define F_SLLI 3'b001
37 `define F_SRRI_SRAI 3'b101
38
39 // function 7
40 `define F7_ADD 7'b0000000
41 `define F7_SUB 7'b0100000
42 `define F7_SRRI 7'b0000000
43 `define F7_SRAI 7'b0100000
44 `define F7_OPCODE_R 7'b0000000
45 `define F7_SRL 7'b0000000
46 `define F7_SRA 7'b0100000
```

```
47 // alu
48 `define F_ADD_SUB 3'b000
49 `define F_SLL 3'b001
50 `define F_SLT 3'b010
51 `define F_SLTU 3'b011
52 `define F_XOR 3'b100
53 `define F_SRL_SRA 3'b101
54 `define F_OR 3'b110
55 `define F_AND 3'b111
56
57 // branch
58 `define F_BEQ 3'b000
59 `define F_BNE 3'b001
60 `define F_BLT 3'b100
61 `define F_BGE 3'b101
62 `define F_BLTU 3'b110
63 `define F_BGEU 3'b111
64
65 // LSU load
66 `define F_LB 3'b000
67 `define F_LH 3'b001
68 `define F_LW 3'b010
69 `define F_LBU 3'b100
70 `define F_LHU 3'b101
71 // store
72 `define F_SB 3'b000
73 `define F_SH 3'b001
74 `define F_SW 3'b010
```

Mylsu.sv

```
@: mylsu.sv > ...
1 `include "mydefine.sv"
2 `timescale 1ns/100ps
3 module mylsu(
4     input    logic clk,
5     input    logic write_ram,
6     input    logic [2:0] funct3,
7     input    logic [31:0] write_data,
8     input    logic [31:0] ram_addr,
9
10    output   logic [31:0] read_data
11 );
12
13    logic [29:0] ram_addr_0;
14    logic [29:0] ram_addr_1;
15    logic [29:0] ram_addr_2;
16    logic [29:0] ram_addr_3;
17
18    logic [29:0] ram_addr_p4;
19
20    logic [7:0] read_data_0;
21    logic [7:0] read_data_1;
22    logic [7:0] read_data_2;
23    logic [7:0] read_data_3;
24
25    logic [7:0] write_data_0;
26    logic [7:0] write_data_1;
27    logic [7:0] write_data_2;
28    logic [7:0] write_data_3;
29
30    logic en_bank_0;
31    logic en_bank_1;
32    logic en_bank_2;
33    logic en_bank_3;
34
35    logic write_ram_0;
36    logic write_ram_1;
37    logic write_ram_2;
38    logic write_ram_3;
39
40    assign ram_addr_p4 = ram_addr[31:2] + 1;
```

```

41 //分配四塊 RAM 之位址和寫入資料
42
43 always_comb begin
44     unique case (ram_addr[1:0])
45         2'b00: begin
46             ram_addr_0 = ram_addr[31:2];
47             ram_addr_1 = ram_addr[31:2];
48             ram_addr_2 = ram_addr[31:2];
49             ram_addr_3 = ram_addr[31:2];
50             write_data_0 = write_data[7:0];
51             write_data_1 = write_data[15:8];
52             write_data_2 = write_data[23:16];
53             write_data_3 = write_data[31:24];
54         end
55         2'b01: begin
56             ram_addr_0 = ram_addr_p4;
57             ram_addr_1 = ram_addr[31:2];
58             ram_addr_2 = ram_addr[31:2];
59             ram_addr_3 = ram_addr[31:2];
60             write_data_0 = write_data[31:24];
61             write_data_1 = write_data[7:0];
62             write_data_2 = write_data[15:8];
63             write_data_3 = write_data[23:16];
64         end
65         2'b10: begin
66             ram_addr_0 = ram_addr_p4;
67             ram_addr_1 = ram_addr_p4;
68             ram_addr_2 = ram_addr[31:2];
69             ram_addr_3 = ram_addr[31:2];
70             write_data_0 = write_data[23:16];
71             write_data_1 = write_data[31:24];
72             write_data_2 = write_data[7:0];
73             write_data_3 = write_data[15:8];
74         end
75         2'b11: begin
76             ram_addr_0 = ram_addr_p4;
77             ram_addr_1 = ram_addr_p4;
78             ram_addr_2 = ram_addr_p4;
79             ram_addr_3 = ram_addr[31:2];
80             write_data_0 = write_data[15:8];
81             write_data_1 = write_data[23:16];
82             write_data_2 = write_data[31:24];
83             write_data_3 = write_data[7:0];
84         end
85     endcase
86
87
88     assign write_ram_0 = en_bank_0 & write_ram;
89     assign write_ram_1 = en_bank_1 & write_ram;
90     assign write_ram_2 = en_bank_2 & write_ram;
91     assign write_ram_3 = en_bank_3 & write_ram;

```

```
93 //決定哪些 RAM 可以寫入
94 always_comb begin
95     en_bank_0 = 0;
96     en_bank_1 = 0;
97     en_bank_2 = 0;
98     en_bank_3 = 0;
99
100    unique case (funct3)
101        `F_SB: begin
102            unique case (ram_addr[1:0])
103                2'b00: en_bank_0 = 1;
104                2'b01: en_bank_1 = 1;
105                2'b10: en_bank_2 = 1;
106                2'b11: en_bank_3 = 1;
107            endcase
108        end
109        `F_SH: begin
110            unique case (ram_addr[1:0])
111                2'b00: begin
112                    en_bank_0 = 1;
113                    en_bank_1 = 1;
114                end
115                2'b01: begin
116                    en_bank_1 = 1;
117                    en_bank_2 = 1;
118                end
119                2'b10: begin
120                    en_bank_2 = 1;
121                    en_bank_3 = 1;
122                end
123                2'b11: begin
124                    en_bank_3 = 1;
125                    en_bank_0 = 1;
126                end
127            endcase
128        end
129        `F_SW: begin
130            en_bank_0 = 1;
131            en_bank_1 = 1;
132            en_bank_2 = 1;
133            en_bank_3 = 1;
134        end
135    endcase
136 end
137
138    RAM ram_0  (
139        .clk          (clk      ),
140        .write        (write_ram_0   ),
141        .write_data   (write_data_0   ),
142        .ram_addr     (ram_addr_0    ),
143        .read_data    (read_data_0   )
144    );
145
```

```
146     RAM ram_1  (
147         .clk          (clk          ),
148         .write        (write_ram_1  ),
149         .write_data   (write_data_1 ),
150         .ram_addr    (ram_addr_1  ),
151
152         .read_data   (read_data_1  )
153     );
154
155     RAM ram_2  (
156         .clk          (clk          ),
157         .write        (write_ram_2  ),
158         .write_data   (write_data_2 ),
159         .ram_addr    (ram_addr_2  ),
160
161         .read_data   (read_data_2  )
162     );
163
164     RAM ram_3  (
165         .clk          (clk          ),
166         .write        (write_ram_3  ),
167         .write_data   (write_data_3 ),
168         .ram_addr    (ram_addr_3  ),
169
170         .read_data   (read_data_3  )
171     );
172
```

```

173 //組合四塊 RAM 的 read_data
174
175 always_comb begin
176     unique case (funct3)
177         `F_LB: begin
178             unique case (ram_addr[1:0])
179                 2'b00: read_data = {{24{read_data_0[7]}}, read_data_0};
180                 2'b01: read_data = {{24{read_data_1[7]}}, read_data_1};
181                 2'b10: read_data = {{24{read_data_2[7]}}, read_data_2};
182                 2'b11: read_data = {{24{read_data_3[7]}}, read_data_3};
183             endcase
184         end
185         `F_LH: begin
186             unique case (ram_addr[1:0])
187                 2'b00: read_data = {{16{read_data_1[7]}}, read_data_1, read_data_0};
188                 2'b01: read_data = {{16{read_data_2[7]}}, read_data_2, read_data_1};
189                 2'b10: read_data = {{16{read_data_3[7]}}, read_data_3, read_data_2};
190                 2'b11: read_data = {{16{read_data_0[7]}}, read_data_0, read_data_3};
191             endcase
192         end
193         `F_LW: begin
194             unique case (ram_addr[1:0])
195                 2'b00: read_data = {read_data_3, read_data_2, read_data_1, read_data_0};
196                 2'b01: read_data = {read_data_0, read_data_3, read_data_2, read_data_1};
197                 2'b10: read_data = {read_data_1, read_data_0, read_data_3, read_data_2};
198                 2'b11: read_data = {read_data_2, read_data_1, read_data_0, read_data_3};
199             endcase
200         end
201         `F_LBU: begin
202             unique case (ram_addr[1:0])
203                 2'b00: read_data = {24'h0, read_data_0};
204                 2'b01: read_data = {24'h0, read_data_1};
205                 2'b10: read_data = {24'h0, read_data_2};
206                 2'b11: read_data = {24'h0, read_data_3};
207             endcase
208         end
209         `F_LHU: begin
210             unique case (ram_addr[1:0])
211                 2'b00: read_data = {16'h0, read_data_1, read_data_0};
212                 2'b01: read_data = {16'h0, read_data_2, read_data_1};
213                 2'b10: read_data = {16'h0, read_data_3, read_data_2};
214                 2'b11: read_data = {16'h0, read_data_0, read_data_3};
215             endcase
216         end
217     endcase
218 end
219 endmodule

```

## Ram.sv

```

@ RAM.sv > ...
1 `timescale 1ns/100ps
2 module RAM (
3     input logic clk,
4     input logic write,
5     input logic [7:0] write_data,
6     input logic [29:0] ram_addr,
7     output logic [7:0] read_data
8 );
9     logic [7:0] ram[0:127];
10    assign read_data = ram[ram_addr];
11    always_ff @(posedge clk) begin
12        if (write) begin
13            ram[ram_addr] <= #1 write_data;
14        end
15    end
16 endmodule

```

## Reg\_file.sv

```
④ 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
```

## (第一題) Program\_rom

```
asm > [10] code.s
1  init:
2      li x2, 10
3
4      # li x3, 0x0208010b # 02, 08, 01, 0b
5      lui x3, 0x02080
6      nop
7      addi x3, x3, 0x10b
8
9      # li x4, 0x03060e0f # 03, 06, 0e, 0f
10     lui x4, 0x03061
11     nop
12     addi x4, x4, -497
13
14     sw x3, 0(x2)
15     sw x4, 4(x2)
16     call bubbleSort
17     addi x2, x0, 10      # reset array index
18     j done
19
20     bubbleSort:
21     addi x4, x0, 1      # if (1 < n) then loopOne, x4 = 1
22     addi x5, x0, 8      # 8 HEX in reg, x5 = n
23     j cmpLoopOne
24
25     endBubbleSort:
26     ret
27
28     loopOne:
29     addi x2, x0, 10      # reset array index
30     addi x5, x5, -1      # n -= 1
31     addi x6, x0, 0      # cnt for loopTwo, x6 = i
32     j cmpLoopTwo
33
34     cmpLoopOne:
35     blt x4, x5, loopOne # if (n>1) then loopOne
36     j endBubbleSort
```

```
37
38    loopTwo:
39        lb x7, 0(x2)      # load arr[i]
40        lb x8, 1(x2)      # load arr[i+1]
41        nop             # data hazard for x8
42        blt x8, x7, swap # if (arr[i] > arr[i+1]) then swap
43        j afterLoopTwo
44
45    swap:
46        sb x8, 0(x2)      # arr[i] = arr[i+1]
47        sb x7, 1(x2)      # arr[i+1] = arr[i]
48        j afterLoopTwo
49
50    afterLoopTwo:
51        addi x2, x2, 1     # shift x2 (address of array)
52        addi x6, x6, 1     # i++
53        j cmpLoopTwo
54
55    cmpLoopTwo:
56        blt x6, x5, loopTwo # i<n
57        j cmpLoopOne
58
59    done:
60        lw x30, 0(x2)
61        lw x31, 4(x2)
62        nop
```

可以看到最開始 li 不能用，因為兩個步驟中出現  
data hazard，我真的會謝：）

```

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'h00A00113; // addi x2 x0 10
9             32'h00000004 : Rom_data = 32'h020801B7; // lui x3 8320
10            32'h00000008 : Rom_data = 32'h00000013; // addi x0 x0 0
11            32'h0000000C : Rom_data = 32'h10B18193; // addi x3 x3 267
12            32'h00000010 : Rom_data = 32'h03061237; // lui x4 12385
13            32'h00000014 : Rom_data = 32'h00000013; // addi x0 x0 0
14            32'h00000018 : Rom_data = 32'hE0F20213; // addi x4 x4 -497
15            32'h0000001C : Rom_data = 32'h00312023; // sw x3 0(x2)
16            32'h00000020 : Rom_data = 32'h00412223; // sw x4 4(x2)
17            32'h00000024 : Rom_data = 32'h00000317; // auipc x6 0
18            32'h00000028 : Rom_data = 32'h010300E7; // jalr x1 x6 16
19            32'h0000002C : Rom_data = 32'h00A00113; // addi x2 x0 10
20            32'h00000030 : Rom_data = 32'h0600006F; // jal x0 96
21            32'h00000034 : Rom_data = 32'h00100213; // addi x4 x0 1
22            32'h00000038 : Rom_data = 32'h00800293; // addi x5 x0 8
23            32'h0000003C : Rom_data = 32'h0180006F; // jal x0 24
24            32'h00000040 : Rom_data = 32'h00008067; // jalr x0 x1 0
25            32'h00000044 : Rom_data = 32'h00A00113; // addi x2 x0 10
26            32'h00000048 : Rom_data = 32'hFFF28293; // addi x5 x5 -1
27            32'h0000004C : Rom_data = 32'h00000313; // addi x6 x0 0
28            32'h00000050 : Rom_data = 32'h0380006F; // jal x0 56
29            32'h00000054 : Rom_data = 32'hFE5248E3; // blt x4 x5 -16
30            32'h00000058 : Rom_data = 32'hFE9FF06F; // jal x0 -24
31            32'h0000005C : Rom_data = 32'h00010383; // lb x7 0(x2)
32            32'h00000060 : Rom_data = 32'h00110403; // lb x8 1(x2)
33            32'h00000064 : Rom_data = 32'h00000013; // addi x0 x0 0
34            32'h00000068 : Rom_data = 32'h00744463; // blt x8 x7 8
35            32'h0000006C : Rom_data = 32'h0100006F; // jal x0 16
36            32'h00000070 : Rom_data = 32'h00810023; // sb x8 0(x2)
37            32'h00000074 : Rom_data = 32'h007100A3; // sb x7 1(x2)
38            32'h00000078 : Rom_data = 32'h0040006F; // jal x0 4
39            32'h0000007C : Rom_data = 32'h00110113; // addi x2 x2 1
40            32'h00000080 : Rom_data = 32'h00130313; // addi x6 x6 1
41            32'h00000084 : Rom_data = 32'h0040006F; // jal x0 4
42            32'h00000088 : Rom_data = 32'hFC534AE3; // blt x6 x5 -44
43            32'h0000008C : Rom_data = 32'hFC9FF06F; // jal x0 -56
44            32'h00000090 : Rom_data = 32'h00012F03; // lw x30 0(x2)
45            32'h00000094 : Rom_data = 32'h00412F83; // lw x31 4(x2)
46            32'h00000098 : Rom_data = 32'h00000013; // addi x0 x0 0
47            default : Rom_data = 32'h00000013;      // NOP
48        endcase
49    end
50
51 endmodule

```

## ●模擬結果與結果說明：

+ ram	xx xx xx 0...	xx xx xx 03 0e xx >
+ ram	xx xx xx 0...	xx xx xx 06 0f xx >
+ ram	xx xx 01 0...	xx xx 01 08 xx xx >
+ ram	xx xx 02 0...	xx xx 02 0b xx xx >

由上至下分別是 ram\_0 至 ram\_3

[30]	06030201	00000000	06030201	
[31]	0f0e0b08	00000000	0f0e0b08	

依照題目要求，將結果顯示在 x30, x31

## 2、組語撰寫練習一找最大

### ●實驗說明：

1. 如下圖 3，在記憶體位址 10~17 中共有 4 筆 16 位元資料，請透過當前課堂所學的所有組合語言撰寫一個程式，該程式可以找出這 4 筆資料的最大值，程式碼須加註解。
2. 所有暫存器皆可使用，但請將結果（最大值 0x0000e0f）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。

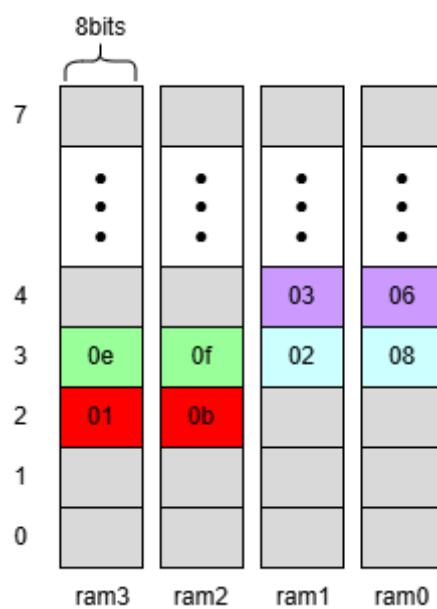


圖 3

```

init:
    li x2, 10
    li x3, 0x0208010b
    li x4, 0x03060e0f
    nop
    sw x3, 0(x2)
    sw x4, 4(x2)

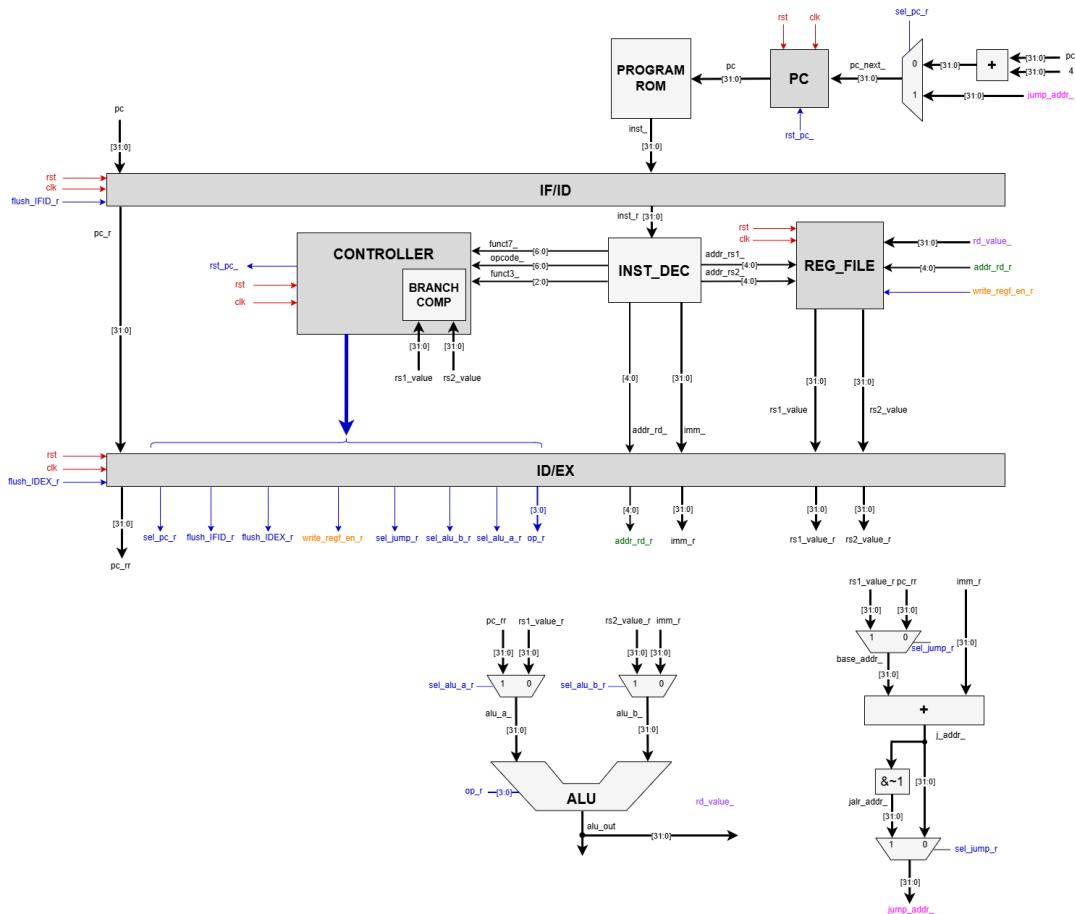
```

#-----開始撰寫您的組合語言-----

#-----您的組合語言-----

圖 4

## ●系統硬體架構方塊圖（接線圖）：



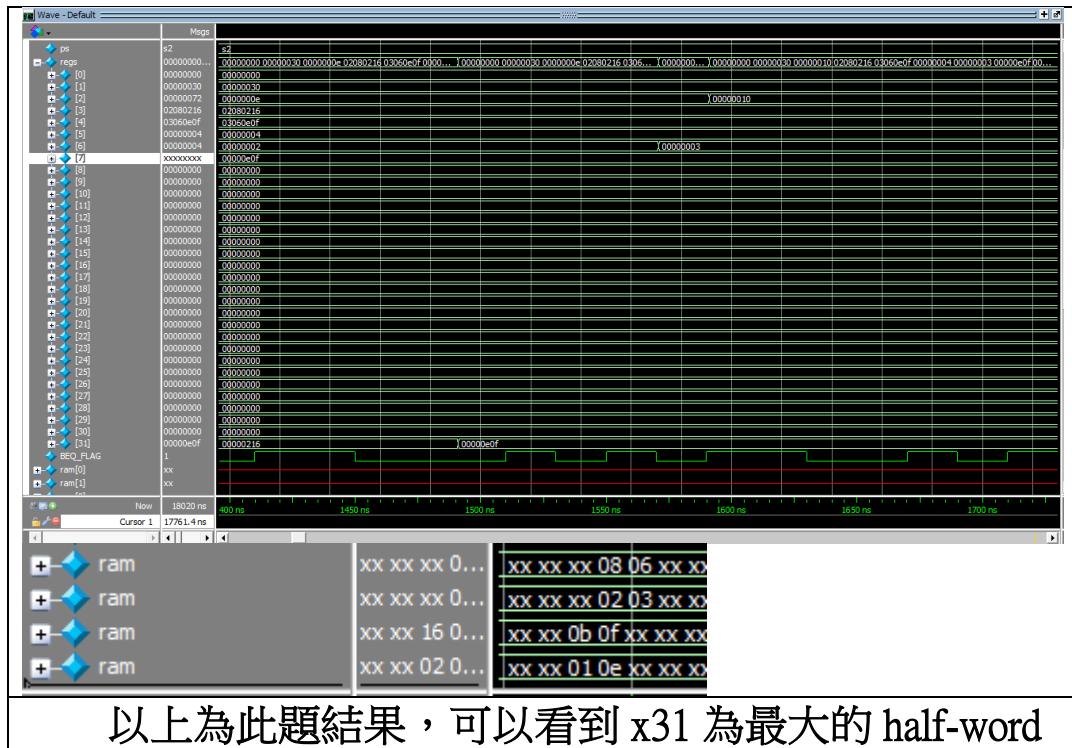
## ●系統架構程式碼、測試資料程式碼與程式碼說明 截圖請善用 win+shift+S

### Program\_rom.sv

```
asm > [x] code.s
 1  init:
 2    li x2, 10
 3
 4    # li x3, 0x0208010b # 02, 08, 01, 0b
 5    lui x3, 0x02080
 6    nop
 7    addi x3, x3, 0x10b
 8
 9    # li x4, 0x03060e0f # 03, 06, 0e, 0f
10   lui x4, 0x03061
11   nop
12   addi x4, x4, -497
13
14   sw x3, 0(x2)
15   sw x4, 4(x2)
16
17   addi x31, x0, 0      # init max-value 0 to x31
18   call getMax
19   nop
20
21 getMax:
22   addi x5, x0, 4      # i<n, x5 = n
23   add x6, x0, x0       # cnt i, x6 = i
24   j cmpLoop
25
26 cmpLoop:
27   blt x6, x5, loop    # if (i<n) then loop
28   ret
29
30 loop:
31   lh x7, 0(x2)        # load half-word to x7, arr[i]
32   nop
33   blt x31, x7, cgMax  # if (max < arr[i]) then max=arr[i]
34   j endLoop
35
36 cgMax:
37   lh x31, 0(x2)        # change max-value, store arr[i]
38   j endLoop
39
40 endLoop:
41   addi x6, x6, 1        # i++
42   addi x2, x2, 2        # shift arr index
43   j cmpLoop
```

```
④ Program_Rom.sv > ...
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'h00A00113; // addi x2 x0 10
9          32'h00000004 : Rom_data = 32'h020801B7; // lui x3 8320
10         32'h00000008 : Rom_data = 32'h00000013; // addi x0 x0 0
11         32'h0000000C : Rom_data = 32'h10B18193; // addi x3 x3 267
12         32'h00000010 : Rom_data = 32'h03061237; // lui x4 12385
13         32'h00000014 : Rom_data = 32'h00000013; // addi x0 x0 0
14         32'h00000018 : Rom_data = 32'hE0F20213; // addi x4 x4 -497
15         32'h0000001C : Rom_data = 32'h00312023; // sw x3 0(x2)
16         32'h00000020 : Rom_data = 32'h00412223; // sw x4 4(x2)
17         32'h00000024 : Rom_data = 32'h00000F93; // addi x31 x0 0
18         32'h00000028 : Rom_data = 32'h00000317; // auipc x6 0
19         32'h0000002C : Rom_data = 32'h00C300E7; // jalr x1 x6 12
20         32'h00000030 : Rom_data = 32'h00000013; // addi x0 x0 0
21         32'h00000034 : Rom_data = 32'h00400293; // addi x5 x0 4
22         32'h00000038 : Rom_data = 32'h00000333; // add x6 x0 x0
23         32'h0000003C : Rom_data = 32'h0040006F; // jal x0 4
24         32'h00000040 : Rom_data = 32'h00534463; // blt x6 x5 8
25         32'h00000044 : Rom_data = 32'h00008067; // jalr x0 x1 0
26         32'h00000048 : Rom_data = 32'h00011383; // lh x7 0(x2)
27         32'h0000004C : Rom_data = 32'h00000013; // addi x0 x0 0
28         32'h00000050 : Rom_data = 32'h007FC463; // blt x31 x7 8
29         32'h00000054 : Rom_data = 32'h00C0006F; // jal x0 12
30         32'h00000058 : Rom_data = 32'h00011F83; // lh x31 0(x2)
31         32'h0000005C : Rom_data = 32'h0040006F; // jal x0 4
32         32'h00000060 : Rom_data = 32'h00130313; // addi x6 x6 1
33         32'h00000064 : Rom_data = 32'h00210113; // addi x2 x2 2
34         32'h00000068 : Rom_data = 32'hFD9FF06F; // jal x0 -40
35         default : Rom_data = 32'h00000013; // NOP
36     endcase
37 end
38
39 endmodule
40
```

## ●模擬結果與結果說明：



以上為此題結果，可以看到 x31 為最大的 half-word

## ●結論與心得：

這是一場博奕我有沒有聽課的作業，為什麼要騙我有 li 指令可以用，沒有為什麼不早說，為什麼不早說呢？