2025/12/11

實驗十三

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_HW13

一、 組語撰寫練習—矩陣乘法

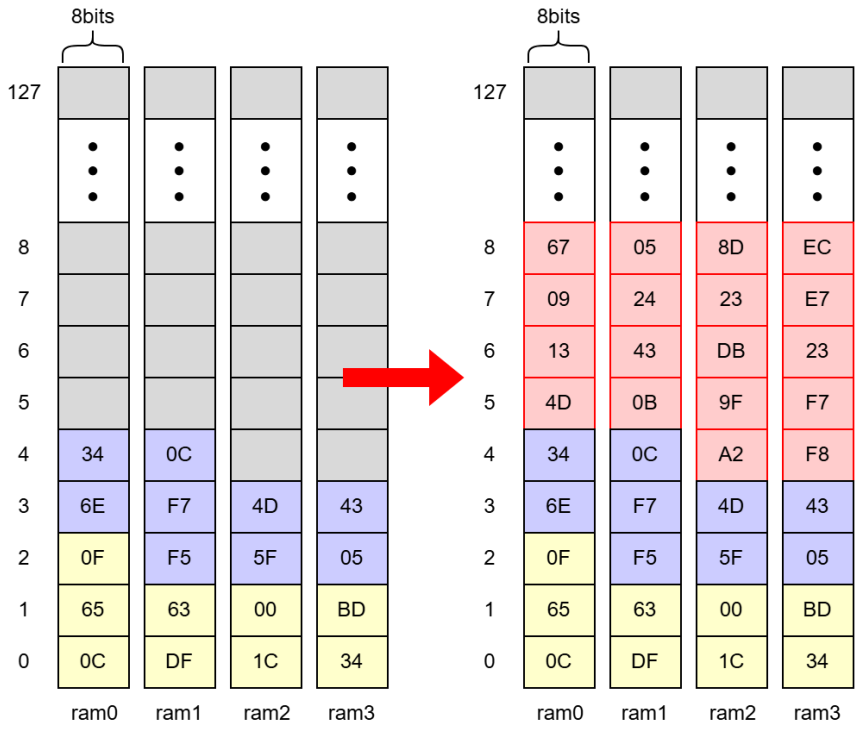
● **實驗說明：**

1. 請透過當前課堂所學的所有組合語言撰寫一個程式計算兩個矩陣 A 和 B 的乘積，並將結果矩陣 C 儲存於記憶體位址 0x12 起的位置。矩陣 A 和 B 的元素皆為有號數 1 Byte，而矩陣 C 的元素為有號數 2 Bytes。程式需依據指定的矩陣儲存格式進行運算，如圖一所示。程式碼須加註解。

矩陣 A　　矩陣 B　　矩陣 C

$$\begin{bmatrix} 0C & DF & 1C \\ 34 & 65 & 63 \\ 00 & BD & 0F \end{bmatrix} \times \begin{bmatrix} F5 & 5F & 05 \\ 6E & F7 & 4D \\ 43 & 34 & 0C \end{bmatrix} = \begin{bmatrix} F8A2 & 0B4D & F79F \\ 4313 & 23DB & 2409 \\ E723 & 0567 & EC8D \end{bmatrix}$$

2. 所有暫存器皆可使用，但請將結果依序(0xF8A2→0x0B4D →... →0x0567 →0xEC8D)載入到 x31 暫存器中 。
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。

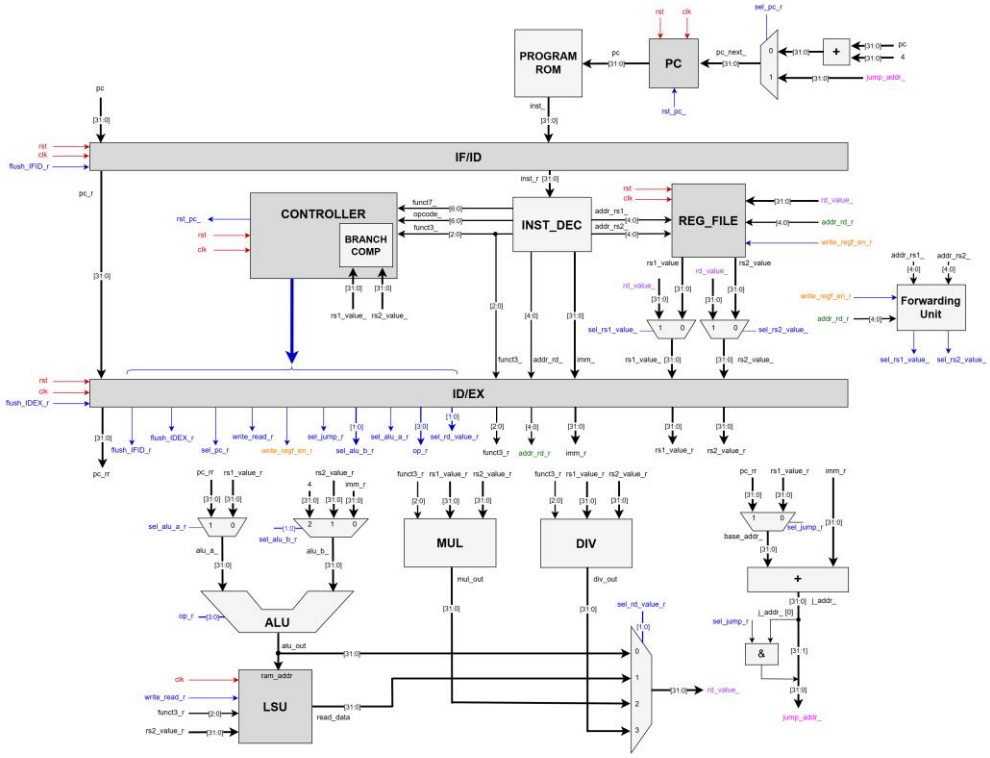


圖一

```
init:
    li x2, 0x341CDF0C
    sw x2, 0(x0)
    li x2, 0xBD006365
    sw x2, 4(x0)
    li x2, 0xF
    sb x2, 8(x0)
    li x3, 0x9
    li x2, 0x6E055FF5
    sw x2, 0(x3)
    li x2, 0x34434DF7
    sw x2, 4(x3)
    li x2, 0xC
    sb x2, 8(x3)

#-----開始撰寫您的組合語言-----


#--------您的組合語言---------

    li x3, 0x12
    li x4, 9
done:
    lh x31, 0(x3)
    addi x3, x3, 2
    addi x4, x4, -1
    bne x4, x0, done
```

圖二

- 系統硬體架構方塊圖（接線圖）：



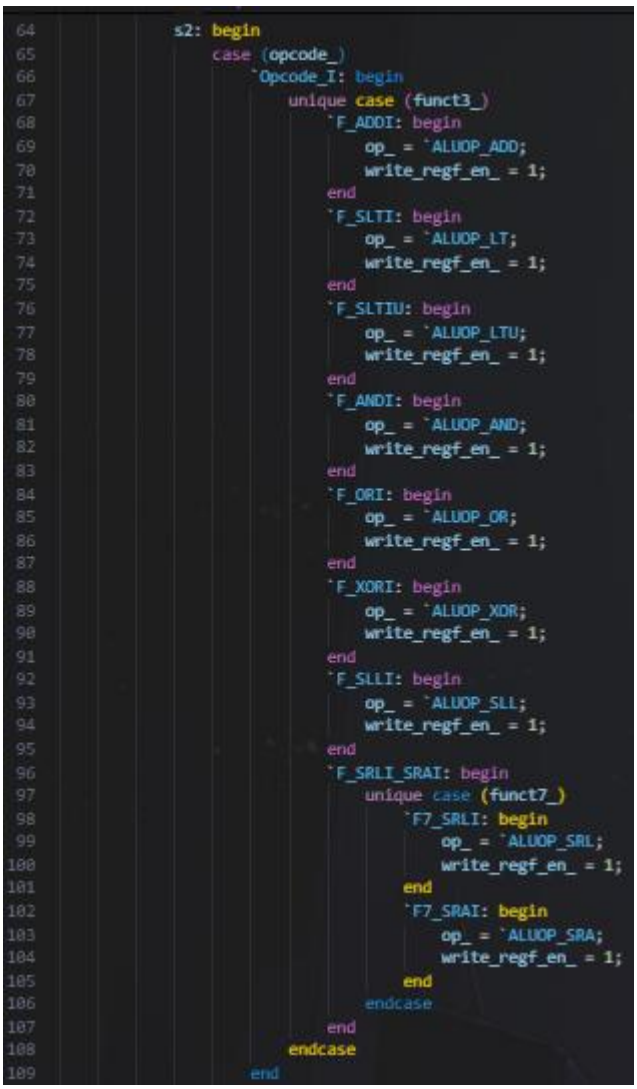
- 系統架構程式碼、測試資料程式碼與程式碼說明

  截圖請善用 win+shift+S

Controller.sv

controller.sv > controller

```sv
`include "mydefine.sv"
`timescale 1ns/100ps
module controller (
    input logic [6:0] funct7_, opcode_,
    input logic rst, clk,
    input logic [2:0] funct3_,
    input logic [31:0] rs1_value, rs2_value,
    output logic flush_IFID_, flush_IDEX_, rst_pc_, sel_pc_, write_regf_en_, sel_jump_, sel_alu_a_, write_ram_,
    output logic [1:0] sel_alu_b_, sel_rd_value_,
    output logic [3:0] op_
);
    logic BEQ_FLAG;
    logic BNE_FLAG;
    logic BLT_FLAG;
    logic BGE_FLAG;
    logic BLTU_FLAG;
    logic BGEU_FLAG;

    assign BEQ_FLAG = (rs1_value == rs2_value);
    assign BNE_FLAG = (rs1_value != rs2_value);
    assign BLT_FLAG = ($signed(rs1_value) < $signed(rs2_value));
    assign BGE_FLAG = ($signed(rs1_value) >= $signed(rs2_value));
    assign BLTU_FLAG = (rs1_value < rs2_value);
    assign BGEU_FLAG = (rs1_value >= rs2_value);

    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;
        sel_alu_b_ = 0;
        ns = ps;
        op_ = `ALUOP_ADD;
        sel_alu_a_ = 0;
        sel_jump_ = 0;
        sel_rd_value_ = 0;
        write_ram_ = 0;
        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
```



```
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
```

```
109                 end
110                 `Opcode_R_M: begin
111                     unique case (funct3_)
112                         `F_AND: begin
113                             if (funct7_ == `F7_OPCODE_R) begin
114                                 op_ = `ALUOP_AND;
115                                 write_regf_en_ = 1;
116                                 sel_alu_b_ = 1;
117                             end
118                             // `F_REMU
119                             else if (funct7_ == `F7_R_M) begin
120                                 write_regf_en_ = 1;
121                                 sel_rd_value_ = 3;
122                             end
123                         end
124                         `F_OR: begin
125                             if (funct7_ == `F7_OPCODE_R) begin
126                                 op_ = `ALUOP_OR;
127                                 write_regf_en_ = 1;
128                                 sel_alu_b_ = 1;
129                             end
130                             // F_REM
131                             else if (funct7_ == `F7_R_M) begin
132                                 write_regf_en_ = 1;
133                                 sel_rd_value_ = 3;
134                             end
135                         end
136                         `F_XOR: begin
137                             if (funct7_ == `F7_OPCODE_R) begin
138                                 op_ = `ALUOP_XOR;
139                                 write_regf_en_ = 1;
140                                 sel_alu_b_ = 1;
141                             end
142                             // F_DIV
143                             else if (funct7_ == `F7_R_M) begin
144                                 write_regf_en_ = 1;
145                                 sel_rd_value_ = 3;
146                             end
147                         end
148                         `F_ADD_SUB: begin
149                             unique case (funct7_)
150                                 `F7_ADD: begin
151                                     op_ = `ALUOP_ADD;
152                                     write_regf_en_ = 1;
153                                     sel_alu_b_ = 1;
154                                 end
155                                 `F7_SUB: begin
156                                     op_ = `ALUOP_SUB;
157                                     write_regf_en_ = 1;
158                                     sel_alu_b_ = 1;
159                                 end
160                                 // MUL
161                                 `F7_R_M: begin
162                                     write_regf_en_ = 1;
163                                     sel_rd_value_ = 2;
164                                 end
165                             endcase
166                         end
167                         `F_SLT: begin
168                             if (funct7_ == `F7_OPCODE_R) begin
169                                 op_ = `ALUOP_LT;
170                                 write_regf_en_ = 1;
171                                 sel_alu_b_ = 1;
172                             end
173                             // F_MULHSU
174                             else if (funct7_ == `F7_R_M) begin
175                                 write_regf_en_ = 1;
176                                 sel_rd_value_ = 2;
177                             end
178                         end
```

```
                    `F_SLTU: begin
                        if (funct7_ == `F7_OPCODE_R) begin
                            op_ = `ALUOP_LTU;
                            write_regf_en_ = 1;
                            sel_alu_b_ = 1;
                        end
                        // F_MULHU
                        else if (funct7_ == `F7_R_M) begin
                            write_regf_en_ = 1;
                            sel_rd_value_ = 2;
                        end
                    end
                    `F_SLL: begin
                        if (funct7_ == `F7_OPCODE_R) begin
                            op_ = `ALUOP_SLL;
                            write_regf_en_ = 1;
                            sel_alu_b_ = 1;
                        end
                        // `F_MULH
                        else if (funct7_ == `F7_R_M) begin
                            write_regf_en_ = 1;
                            sel_rd_value_ = 2;
                        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
                            // F_DIVU
                            `F7_R_M: begin
                                write_regf_en_ = 1;
                                sel_rd_value_ = 3;
                            end
                        endcase
                    end
                endcase
            end
            `Opcode_B: begin
                sel_jump_ = 1;
                unique case (funct3_)
                    `F_BEQ: begin
                        if (BEQ_FLAG) begin
                            sel_pc_ = 1;
                            flush_IDEX_ = 1;
                            flush_IFID_ = 1;
                        end
                    end
                    `F_BNE: begin
                        if (BNE_FLAG) begin
                            sel_pc_ = 1;
                            flush_IDEX_ = 1;
                            flush_IFID_ = 1;
                        end
                    end
                    `F_BLT: begin
                        if (BLT_FLAG) begin
                            sel_pc_ = 1;
                            flush_IFID_ = 1;
                            flush_IDEX_ = 1;
                        end
                    end
                    `F_BGE: begin
                        if (BGE_FLAG) begin
                            sel_pc_ = 1;
                            flush_IDEX_ = 1;
                            flush_IFID_ = 1;
                        end
                    end
```

```
                    `OPCODE_S: begin
                        unique case (funct3_)
                            `F_SB: begin
                                op_ = `ALUOP_ADD;
                                write_ram_ = 1;
                            end
                            `F_SH: begin
                                op_ = `ALUOP_ADD;
                                write_ram_ = 1;
                            end
                            `F_SW: begin
                                op_ = `ALUOP_ADD;
                                write_ram_ = 1;
                            end
                        endcase
                    end
                endcase
            end
        endcase
    end
endmodule
```

div.sv

```
`include "mydefine.sv"
module DIV (
    input logic [2:0] funct3,
    input logic [31:0] rs1_value,
    input logic [31:0] rs2_value,

    output logic [31:0] div_out
);

    always_comb begin
        unique case (funct3)
            `F_DIV: div_out = $signed(rs1_value) / $signed(rs2_value);
            `F_DIVU: div_out = rs1_value / rs2_value;
            `F_REM: div_out = $signed(rs1_value) % $signed(rs2_value);
            `F_REMU: div_out = rs1_value % rs2_value;
        endcase
    end

endmodule
```

Inst_dec.sv

INST_DEC.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 funct3_ = inst_r[14:12];
    assign addr_rd_ = inst_r[11:7];
    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;
    logic [31:0] IMM_B;
    logic [31:0] IMM_JAL;
    logic [31:0] IMM_LUI_AUIPC;
    logic [31:0] IMM_S;
    assign IMM_I = {{20{inst_r[31]}}, inst_r[31:20]};
    assign IMM_B = {{20{inst_r[31]}}, inst_r[7], inst_r[30:25], inst_r[11:8], 1'b0};
    assign IMM_JAL = {{12{inst_r[31]}}, inst_r[19:12], inst_r[20], inst_r[30:21], 1'b0};
    assign IMM_LUI_AUIPC = {inst_r[31:12], 12'b0};
    assign IMM_S = {{20{inst_r[31]}}, inst_r[31:25], inst_r[11:7]};

    always_comb begin
        unique case (opcode_)
            `Opcode_I: imm_ = IMM_I;
            `Opcode_B: imm_ = IMM_B;
            `Opcode_JAL: imm_ = IMM_JAL;
            `Opcode_JALR: imm_ = IMM_I;
            `Opcode_LUI: imm_ = IMM_LUI_AUIPC;
            `Opcode_AUIPC: imm_ = IMM_LUI_AUIPC;
            `Opcode_L: imm_ = IMM_I;
            `Opcode_S: imm_ = IMM_S;
        endcase
    end

endmodule
```

Mul.sv

MUL.sv

```
`include "mydefine.sv"
module MUL (
    input logic [2:0] funct3,
    input logic [31:0] rs1_value,
    input logic [31:0] rs2_value,

    output logic [31:0] mul_out
);
    logic [63:0] product;

    always_comb begin
        unique case (funct3)
            `F_MUL: product = $signed(rs1_value) * $signed(rs2_value);
            `F_MULH: product = $signed(rs1_value) * $signed(rs2_value);
            `F_MULHSU: product = $signed($signed(rs1_value) * rs2_value);
            `F_MULHU: product = rs1_value * rs2_value;
        endcase
    end

    assign mul_out = (funct3 == `F_MUL) ? product[31:0] : product[63:32];

endmodule
```

Myalu.sv

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

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      // progrom-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     // promgram 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      );
```

```
68      // Reg file
69      logic write_regf_en_r, sel_rs1_value_, sel_rs2_value_;
70      logic [4:0] addr_rd_r;
71      logic [31:0] rd_value_, rs1_value, rs2_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_IDEX_;
```

```
91      controller mycontroller(
92          // in
93          .funct7_ (funct7_),
94          .funct3_ (funct3_),
95          .opcode_ (opcode_),
96          .rst (rst),
97          .clk (clk),
98          .rs1_value (rs1_value_),
99          .rs2_value (rs2_value_),
100         // out
101         .sel_alu_b_ (sel_alu_b_),
102         .write_regf_en_ (write_regf_en_),
103         .flush_IFID_ (flush_IFID_),
104         .flush_IDEX_ (flush_IDEX_),
105         .rst_pc_ (rst_pc_),
106         .sel_pc_ (sel_pc_),
107         .op_ (op_),
108         .sel_jump_ (sel_jump_),
109         .sel_alu_a_ (sel_alu_a_),
110         .write_ram_ (write_ram_),
111         .sel_rd_value_ (sel_rd_value_)
112     );
113
114     // forwarding
115     assign sel_rs1_value_ = write_regf_en_r & (addr_rd_r == addr_rs1_);
116     assign sel_rs2_value_ = write_regf_en_r & (addr_rd_r == addr_rs2_);
117     assign rs1_value_ = sel_rs1_value_ ? rd_value_ : rs1_value;
118     assign rs2_value_ = sel_rs2_value_ ? rd_value_ : rs2_value;
```

```
120 // pipe 2 ID_EX
121 logic [31:0] imm_r, rs1_value_r, rs2_value_r, alu_b_, alu_a_, flush_IDEX_r, base_addr_, j_addr_, mul_out, div_out;
122 assign rst_or_flush_IDEX_r = rst | flush_IDEX_r;
123 always_ff @(posedge clk) begin
124     if (rst_or_flush_IDEX_r) begin
125         write_regf_en_r <= 0;
126         addr_rd_r <= 0;
127         imm_r <= 0;
128         rs1_value_r <= 0;
129         rs2_value_r <= 0;
130         op_r <= 0;
131         sel_alu_b_r <= 0;
132         pc_rr <= 0;
133         flush_IDEX_r <= 0;
134         flush_IFID_r <= 0;
135         sel_pc_r <= 0;
136         sel_jump_r <= 0;
137         sel_alu_a_r <= 0;
138         funct3_r <= 0;
139         write_ram_r <= 0;
140         sel_rd_value_r <= 0;
141     end
142     else begin
143         write_regf_en_r <= write_regf_en_;
144         addr_rd_r <= addr_rd_;
145         imm_r <= imm_;
146         rs1_value_r <= rs1_value_;
147         rs2_value_r <= rs2_value_;
148         op_r <= op_;
149         sel_alu_b_r <= sel_alu_b_;
150         pc_rr <= pc_r;
151         flush_IDEX_r <= flush_IDEX_;
152         flush_IFID_r <= flush_IFID_;
153         sel_pc_r <= sel_pc_;
154         sel_jump_r <= sel_jump_;
155         sel_alu_a_r <= sel_alu_a_;
156         funct3_r <= funct3_;
157         write_ram_r <= write_ram_;
158         sel_rd_value_r <= sel_rd_value_;
159     end
160 end
```

```
162     // multi 2 to 1 (sel_alu_a_r)
163     always_comb begin
164         case (sel_alu_a_r)
165             1'd0: alu_a_ = rs1_value_r;
166             1'd1: alu_a_ = pc_rr;
167         endcase
168     end
169
170     // multi 3 to 1 (sel_alu_b_r)
171     always_comb begin
172         unique case (sel_alu_b_r)
173             2'd0: alu_b_ = imm_r;
174             2'd1: alu_b_ = rs2_value_r;
175             2'd2: alu_b_ = 4;
176         endcase
177     end
178
179     // alu
180     myalu alu_1 (
181         .op (op_r),
182         .alu_a (alu_a_),
183         .alu_b (alu_b_),
184         .alu_out (alu_out)
185     );
186
187     // m MUL myMul (
188     MUL myMul (
189         .funct3 (funct3_r),
190         .rs1_value (rs1_value_r),
191         .rs2_value (rs2_value_r),
192         .mul_out (mul_out)
193     );
194
195     // div
196     DIV myDiv (
197         .funct3 (funct3_r),
198         .rs1_value (rs1_value_r),
199         .rs2_value (rs2_value_r),
200         .div_out (div_out)
201     );
```

```
202
203     // LSU
204     logic [31:0] read_data;
205     mylsu lsu1 (
206         .clk (clk),
207         .write_ram (write_ram_r),
208         .funct3 (funct3_r),
209         .write_data (rs2_value_r),
210         .ram_addr (alu_out),
211         .read_data (read_data)
212     );
213
214     // multi 4 to 1
215     always_comb begin
216         case (sel_rd_value_r)
217             2'd0: rd_value_ = alu_out;
218             2'd1: rd_value_ = read_data;
219             2'd2: rd_value_ = mul_out;
220             2'd3: rd_value_ = div_out;
221         endcase
222     end
223
224     // jump_addr_ (adder)
225     // 2 to 1 multi (sel_jump_r)
226     assign base_addr_ = sel_jump_r ? pc_rr : rs1_value_r;
227     assign j_addr_ = base_addr_ + imm_r;
228     assign jump_addr_ = {j_addr_[31:1], (j_addr_[0]&sel_jump_r)};
229 endmodule
```

myDefine.sv

```
`define I_NOP 32'h13

`define Opcode_I 7'b0010011
`define Opcode_R_M 7'b0110011
`define Opcode_B 7'b1100011
`define Opcode_L 7'b0000011
`define Opcode_S 7'b0100011

`define Opcode_JAL 7'b1101111
`define Opcode_JALR 7'b1100111
`define Opcode_LUI 7'b0110111
`define Opcode_AUIPC 7'b0010111

// alu operation
`define ALUOP_ADD 4'h0
`define ALUOP_SUB 4'h1
`define ALUOP_AND 4'h2
`define ALUOP_OR 4'h3
`define ALUOP_XOR 4'h4
`define ALUOP_A 4'h5
`define ALUOP_A_ADD_4 4'h6
`define ALUOP_LTU 4'h7
`define ALUOP_LT 4'h8
`define ALUOP_SLL 4'h9
`define ALUOP_SRL 4'hA
`define ALUOP_SRA 4'hB
`define ALUOP_B 4'hC

// function 3
`define F_ADDI 3'b000
`define F_SLTI 3'b010
`define F_SLTIU 3'b011
`define F_XORI 3'b100
`define F_ORI 3'b110
`define F_ANDI 3'b111
`define F_SLLI 3'b001
`define F_SRLI_SRAI 3'b101

// function 7
`define F7_ADD 7'b0000000
`define F7_SUB 7'b0100000
`define F7_SRLI 7'b0000000
`define F7_SRAI 7'b0100000
`define F7_OPCODE_R 7'b0000000
`define F7_SRL 7'b0000000
`define F7_SRA 7'b0100000
`define F7_R_M 7'b0000001

// alu
`define F_ADD_SUB 3'b000
`define F_SLL 3'b001
`define F_SLT 3'b010
`define F_SLTU 3'b011
`define F_XOR 3'b100
`define F_SRL_SRA 3'b101
`define F_OR 3'b110
`define F_AND 3'b111

// branch
`define F_BEQ 3'b000
`define F_BNE 3'b001
`define F_BLT 3'b100
`define F_BGE 3'b101
`define F_BLTU 3'b110
`define F_BGEU 3'b111

// LSU load
`define F_LB 3'b000
`define F_LH 3'b001
`define F_LW 3'b010
`define F_LBU 3'b100
`define F_LHU 3'b101
// store
`define F_SB 3'b000
`define F_SH 3'b001
`define F_SW 3'b010
```

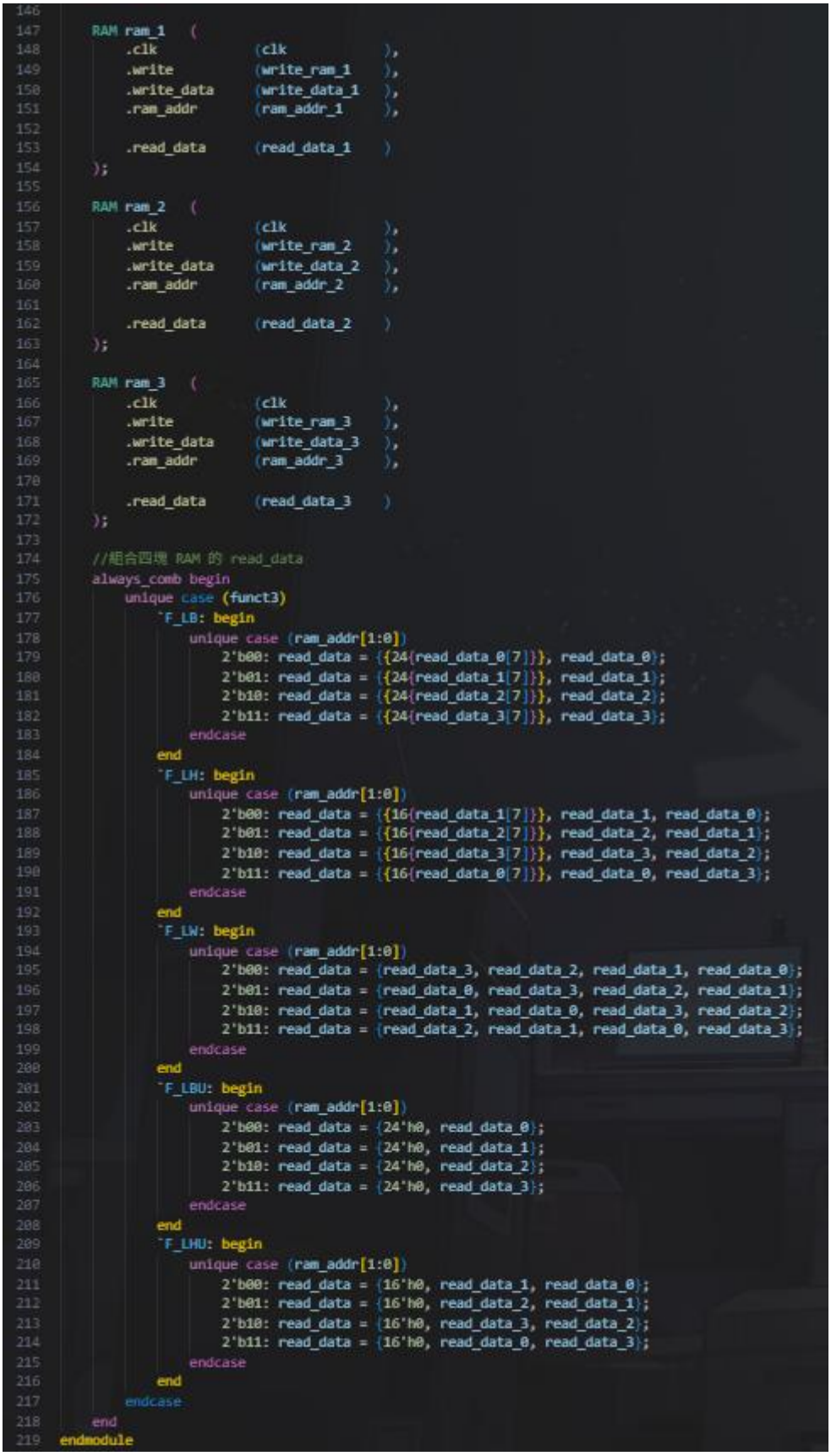
```
78  // div, mux
79  `define F_MUL 3'b000
80  `define F_MULH 3'b001
81  `define F_MULHSU 3'b010
82  `define F_MULHU 3'b011
83  `define F_DIV 3'b100
84  `define F_DIVU 3'b101
85  `define F_REM 3'b110
86  `define F_REMU 3'b111
```

Mylsu.sv

mylsu.sv

```sv
`include "mydefine.sv"
`timescale 1ns/100ps
module mylsu(
    input   logic clk,
    input   logic write_ram,
    input   logic [2:0] funct3,
    input   logic [31:0] write_data,
    input   logic [31:0] ram_addr,

    output  logic [31:0] read_data
);

    logic [29:0] ram_addr_0;
    logic [29:0] ram_addr_1;
    logic [29:0] ram_addr_2;
    logic [29:0] ram_addr_3;

    logic [29:0] ram_addr_p4;

    logic [7:0] read_data_0;
    logic [7:0] read_data_1;
    logic [7:0] read_data_2;
    logic [7:0] read_data_3;

    logic [7:0] write_data_0;
    logic [7:0] write_data_1;
    logic [7:0] write_data_2;
    logic [7:0] write_data_3;

    logic en_bank_0;
    logic en_bank_1;
    logic en_bank_2;
    logic en_bank_3;

    logic write_ram_0;
    logic write_ram_1;
    logic write_ram_2;
    logic write_ram_3;

    assign ram_addr_p4 = ram_addr[31:2] + 1;

    //分配四塊 RAM 之位址和寫入資料
    always_comb begin
        unique case (ram_addr[1:0])
            2'b00: begin
                ram_addr_0 = ram_addr[31:2];
                ram_addr_1 = ram_addr[31:2];
                ram_addr_2 = ram_addr[31:2];
                ram_addr_3 = ram_addr[31:2];
                write_data_0 = write_data[7:0];
                write_data_1 = write_data[15:8];
                write_data_2 = write_data[23:16];
                write_data_3 = write_data[31:24];
            end
            2'b01: begin
                ram_addr_0 = ram_addr_p4;
                ram_addr_1 = ram_addr[31:2];
                ram_addr_2 = ram_addr[31:2];
                ram_addr_3 = ram_addr[31:2];
                write_data_0 = write_data[31:24];
                write_data_1 = write_data[7:0];
                write_data_2 = write_data[15:8];
                write_data_3 = write_data[23:16];
            end
            2'b10: begin
                ram_addr_0 = ram_addr_p4;
                ram_addr_1 = ram_addr_p4;
                ram_addr_2 = ram_addr[31:2];
                ram_addr_3 = ram_addr[31:2];
                write_data_0 = write_data[23:16];
                write_data_1 = write_data[31:24];
                write_data_2 = write_data[7:0];
                write_data_3 = write_data[15:8];
            end
```

```
            2'b11: begin
                ram_addr_0 = ram_addr_p4;
                ram_addr_1 = ram_addr_p4;
                ram_addr_2 = ram_addr_p4;
                ram_addr_3 = ram_addr[31:2];
                write_data_0 = write_data[15:8];
                write_data_1 = write_data[23:16];
                write_data_2 = write_data[31:24];
                write_data_3 = write_data[7:0];
            end
        endcase
    end

    assign write_ram_0 = en_bank_0 & write_ram;
    assign write_ram_1 = en_bank_1 & write_ram;
    assign write_ram_2 = en_bank_2 & write_ram;
    assign write_ram_3 = en_bank_3 & write_ram;


    //决定哪些 RAM 可以写入
    always_comb begin
        en_bank_0 = 0;
        en_bank_1 = 0;
        en_bank_2 = 0;
        en_bank_3 = 0;
        unique case (funct3)
            `F_SB: begin
                unique case (ram_addr[1:0])
                    2'b00: en_bank_0 = 1;
                    2'b01: en_bank_1 = 1;
                    2'b10: en_bank_2 = 1;
                    2'b11: en_bank_3 = 1;
                endcase
            end
            `F_SH: begin
                unique case (ram_addr[1:0])
                    2'b00: begin
                        en_bank_0 = 1;
                        en_bank_1 = 1;
                    end
                    2'b01: begin
                        en_bank_1 = 1;
                        en_bank_2 = 1;
                    end
                    2'b10: begin
                        en_bank_2 = 1;
                        en_bank_3 = 1;
                    end
                    2'b11: begin
                        en_bank_3 = 1;
                        en_bank_0 = 1;
                    end
                endcase
            end
            `F_SW: begin
                en_bank_0 = 1;
                en_bank_1 = 1;
                en_bank_2 = 1;
                en_bank_3 = 1;
            end
        endcase
    end

    RAM ram_0   (
        .clk            (clk            ),
        .write          (write_ram_0    ),
        .write_data     (write_data_0   ),
        .ram_addr       (ram_addr_0     ),

        .read_data      (read_data_0    )
    );
```



```
146
147     RAM ram_1   (
148         .clk            (clk            ),
149         .write          (write_ram_1    ),
150         .write_data     (write_data_1   ),
151         .ram_addr       (ram_addr_1     ),
152
153         .read_data      (read_data_1    )
154     );
155
156     RAM ram_2   (
157         .clk            (clk            ),
158         .write          (write_ram_2    ),
159         .write_data     (write_data_2   ),
160         .ram_addr       (ram_addr_2     ),
161
162         .read_data      (read_data_2    )
163     );
164
165     RAM ram_3   (
166         .clk            (clk            ),
167         .write          (write_ram_3    ),
168         .write_data     (write_data_3   ),
169         .ram_addr       (ram_addr_3     ),
170
171         .read_data      (read_data_3    )
172     );
173
174     //組合四塊 RAM 的 read_data
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

```systemverilog
`timescale 1ns/100ps
module RAM (
    input logic clk,
    input logic write,
    input logic [7:0] write_data,
    input logic [29:0] ram_addr,
    output logic [7:0] read_data
);
    logic [7:0] ram[0:127];
    assign read_data = ram[ram_addr];
    always_ff @(posedge clk ) begin
        if (write) begin
            ram[ram_addr] <= #1 write_data;
        end
    end
endmodule
```

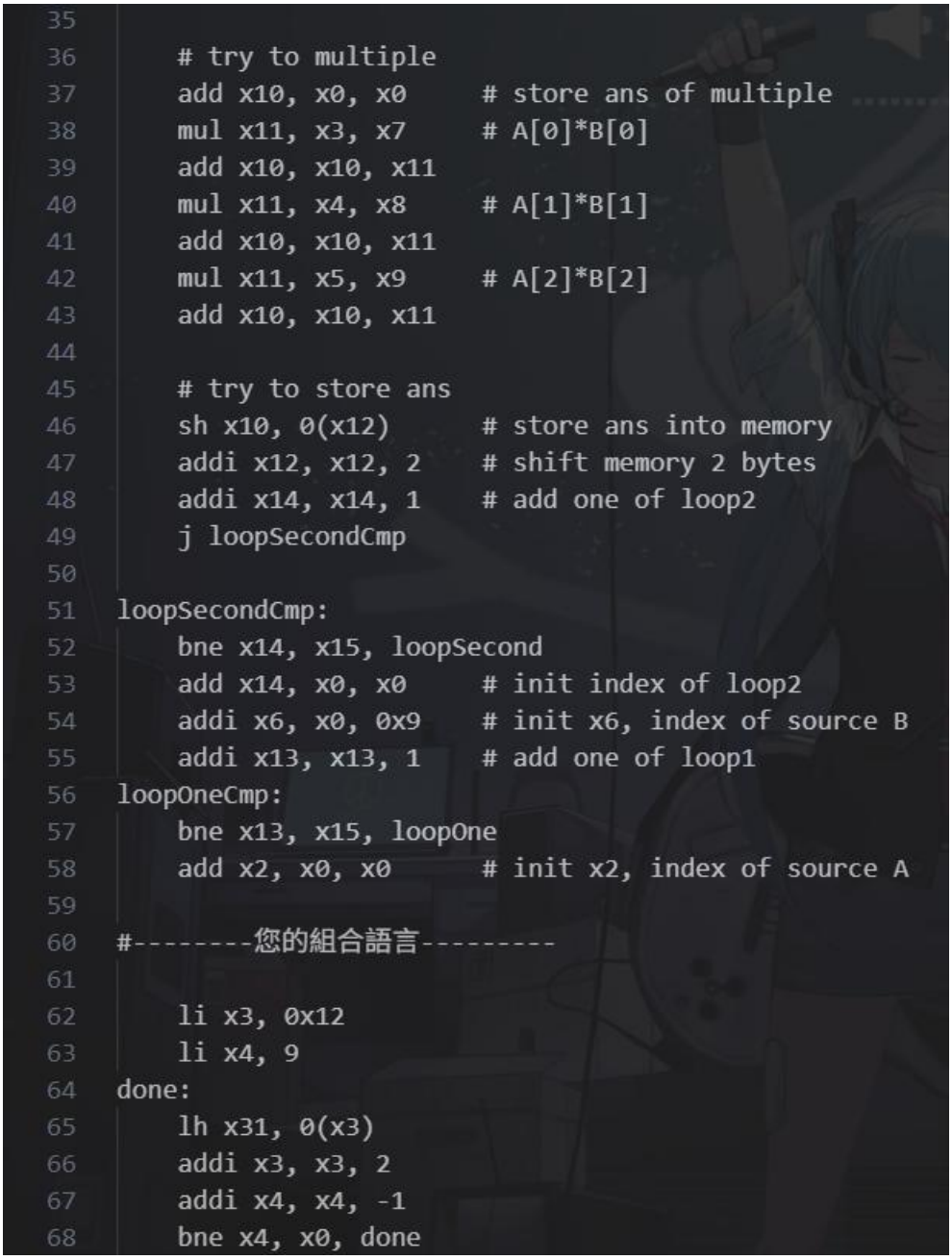
Reg_file.sv

```systemverilog
`timescale 1ns/100ps
module Reg_file(
    input   logic clk,
    input   logic rst,
    input   logic write_regf_en,
    input   logic [4:0] addr_rd,
    input   logic [4:0] addr_rs1,
    input   logic [4:0] addr_rs2,
    input   logic [31:0] rd_value,

    output  logic [31:0] rs1_value,
    output  logic [31:0] rs2_value,
    output logic [31:0] regs_31
);

    logic [31:0] regs[0:31];
    logic addr_rd_not_0;
    integer i;

    assign regs_31 = regs[31];

    assign addr_rd_not_0 = |addr_rd;

    assign rs1_value = regs[addr_rs1];
    assign rs2_value = regs[addr_rs2];

    always_ff@(posedge clk)
    begin
        if(rst) begin
            for(i = 0; i < 32; i = i+1) begin:rst_keywords
                regs[i] <= 0;
            end
        end
        else begin
            // Write
            if (write_regf_en && addr_rd_not_0)
                regs[addr_rd] <= #1 rd_value;
        end
    end

endmodule
```

assembly

asm > code.s

```
init:
    li x2, 0x341CDF0C
    sw x2, 0(x0)
    li x2, 0xBD006365
    sw x2, 4(x0)
    li x2, 0xF
    sb x2, 8(x0)
    li x3, 0x9
    li x2, 0x6E055FF5
    sw x2, 0(x3)
    li x2, 0x34434DF7
    sw x2, 4(x3)
    li x2, 0xC
    sb x2, 8(x3)

#-----開始撰寫您的組合語言-----
    addi x2, x0, 0      # init x2 = 0, load matrix A index
    addi x6, x0, 0x9    # init x6 = 0x9, load matrix B index
    addi x12, x0, 0x12  # init memory start multiple ans address
    add x13, x0, x0     # index of loopOne
    add x14, x0, x0     # index of loopSecond
    addi x15, x0, 3     # bound of loop end
loopOne:
    lb x3, 0(x2)        # source A index[0]
    lb x4, 1(x2)        # source A index[1]
    lb x5, 2(x2)        # source A index[2]
    addi x2, x2, 3      # shift A index + 3
    j loopSecond

loopSecond:
    lb x7, 0(x6)        # source B index[0]
    lb x8, 3(x6)        # source B index[1]
    lb x9, 6(x6)        # source B index[2]
    addi x6, x6, 1      # shift B index + 1
```



```
35
36      # try to multiple
37      add x10, x0, x0     # store ans of multiple
38      mul x11, x3, x7     # A[0]*B[0]
39      add x10, x10, x11
40      mul x11, x4, x8     # A[1]*B[1]
41      add x10, x10, x11
42      mul x11, x5, x9     # A[2]*B[2]
43      add x10, x10, x11
44
45      # try to store ans
46      sh x10, 0(x12)      # store ans into memory
47      addi x12, x12, 2    # shift memory 2 bytes
48      addi x14, x14, 1    # add one of loop2
49      j loopSecondCmp
50
51  loopSecondCmp:
52      bne x14, x15, loopSecond
53      add x14, x0, x0     # init index of loop2
54      addi x6, x0, 0x9    # init x6, index of source B
55      addi x13, x13, 1    # add one of loop1
56  loopOneCmp:
57      bne x13, x15, loopOne
58      add x2, x0, x0      # init x2, index of source A
59
60  #--------您的組合語言---------
61
62      li x3, 0x12
63      li x4, 9
64  done:
65      lh x31, 0(x3)
66      addi x3, x3, 2
67      addi x4, x4, -1
68      bne x4, x0, done
```

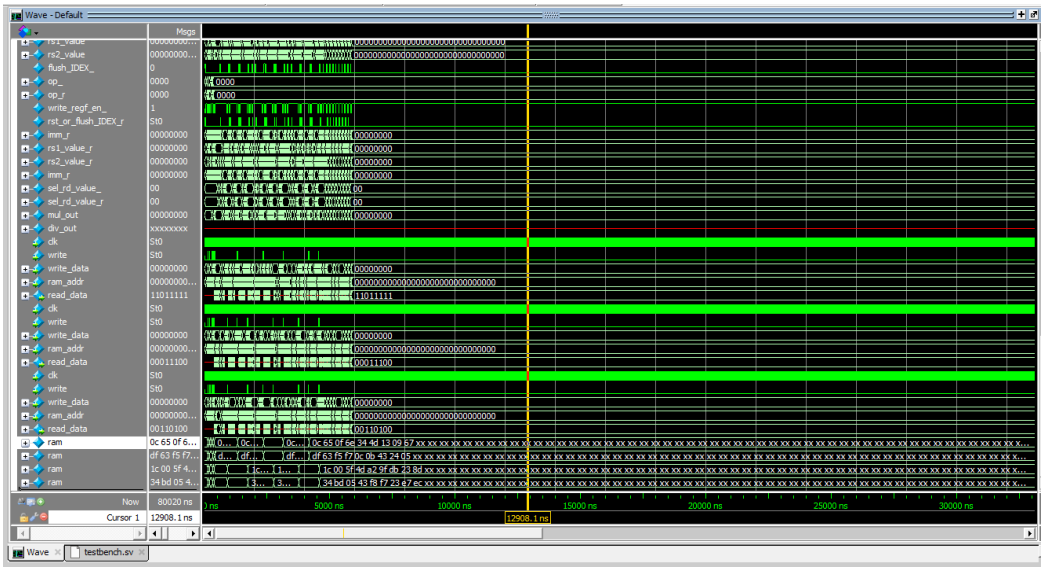
Program_rom.sv

Program_Rom.sv > Program_Rom

```
module Program_Rom(
    output logic [31:0] Rom_data,
    input [31:0] Rom_addr
);

    always_comb begin
        case (Rom_addr)
            32'h00000000 : Rom_data = 32'h341CE137; // lui x2 213454
            32'h00000004 : Rom_data = 32'hF0C10113; // addi x2 x2 -244
            32'h00000008 : Rom_data = 32'h00202023; // sw x2 0(x0)
            32'h0000000C : Rom_data = 32'hBD006137; // lui x2 774150
            32'h00000010 : Rom_data = 32'h36510113; // addi x2 x2 869
            32'h00000014 : Rom_data = 32'h00202223; // sw x2 4(x0)
            32'h00000018 : Rom_data = 32'h00F00113; // addi x2 x0 15
            32'h0000001C : Rom_data = 32'h00200423; // sb x2 8(x0)
            32'h00000020 : Rom_data = 32'h00900193; // addi x3 x0 9
            32'h00000024 : Rom_data = 32'h6E056137; // lui x2 450646
            32'h00000028 : Rom_data = 32'hFF510113; // addi x2 x2 -11
            32'h0000002C : Rom_data = 32'h0021A023; // sw x2 0(x3)
            32'h00000030 : Rom_data = 32'h34435137; // lui x2 214069
            32'h00000034 : Rom_data = 32'hDF710113; // addi x2 x2 -521
            32'h00000038 : Rom_data = 32'h0021A223; // sw x2 4(x3)
            32'h0000003C : Rom_data = 32'h00C00113; // addi x2 x0 12
            32'h00000040 : Rom_data = 32'h00218423; // sb x2 8(x3)
            32'h00000044 : Rom_data = 32'h00000113; // addi x2 x0 0
            32'h00000048 : Rom_data = 32'h00900313; // addi x6 x0 9
            32'h0000004C : Rom_data = 32'h01200613; // addi x12 x0 18
            32'h00000050 : Rom_data = 32'h000006B3; // add x13 x0 x0
            32'h00000054 : Rom_data = 32'h00000733; // add x14 x0 x0
            32'h00000058 : Rom_data = 32'h00300793; // addi x15 x0 3
            32'h0000005C : Rom_data = 32'h00010183; // lb x3 0(x2)
            32'h00000060 : Rom_data = 32'h00110203; // lb x4 1(x2)
            32'h00000064 : Rom_data = 32'h00210283; // lb x5 2(x2)
            32'h00000068 : Rom_data = 32'h00310113; // addi x2 x2 3
            32'h0000006C : Rom_data = 32'h0040006F; // jal x0 4
            32'h00000070 : Rom_data = 32'h00030383; // lb x7 0(x6)
            32'h00000074 : Rom_data = 32'h00330403; // lb x8 3(x6)
            32'h00000078 : Rom_data = 32'h00630483; // lb x9 6(x6)
            32'h0000007C : Rom_data = 32'h00130313; // addi x6 x6 1
            32'h00000080 : Rom_data = 32'h00000533; // add x10 x0 x0
            32'h00000084 : Rom_data = 32'h027185B3; // mul x11 x3 x7
            32'h00000088 : Rom_data = 32'h00B50533; // add x10 x10 x11
            32'h0000008C : Rom_data = 32'h028205B3; // mul x11 x4 x8
            32'h00000090 : Rom_data = 32'h00B50533; // add x10 x10 x11
            32'h00000094 : Rom_data = 32'h029285B3; // mul x11 x5 x9
            32'h00000098 : Rom_data = 32'h00B50533; // add x10 x10 x11
```

```
            32'h0000009C : Rom_data = 32'h00A61023; // sh x10 0(x12)
            32'h000000A0 : Rom_data = 32'h00260613; // addi x12 x12 2
            32'h000000A4 : Rom_data = 32'h00170713; // addi x14 x14 1
            32'h000000A8 : Rom_data = 32'h0040006F; // jal x0 4
            32'h000000AC : Rom_data = 32'hFCF712E3; // bne x14 x15 -60
            32'h000000B0 : Rom_data = 32'h00000733; // add x14 x0 x0
            32'h000000B4 : Rom_data = 32'h00900313; // addi x6 x0 9
            32'h000000B8 : Rom_data = 32'h00168693; // addi x13 x13 1
            32'h000000BC : Rom_data = 32'hFAF690E3; // bne x13 x15 -96
            32'h000000C0 : Rom_data = 32'h00000133; // add x2 x0 x0
            32'h000000C4 : Rom_data = 32'h01200193; // addi x3 x0 18
            32'h000000C8 : Rom_data = 32'h00900213; // addi x4 x0 9
            32'h000000CC : Rom_data = 32'h00019F83; // lh x31 0(x3)
            32'h000000D0 : Rom_data = 32'h00218193; // addi x3 x3 2
            32'h000000D4 : Rom_data = 32'hFFF20213; // addi x4 x4 -1
            32'h000000D8 : Rom_data = 32'hFE021AE3; // bne x4 x0 -12
            default : Rom_data = 32'h00000013;      // NOP
        endcase
    end

endmodule
```

● 模擬結果與結果說明：



可以看到下方四個 ram，與提供的解答相同。

● 結論與心得：

終於有前饋了，寫 assembly 終於不用再考慮該死的 nop 了，真

是太棒了：）