# 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

※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。

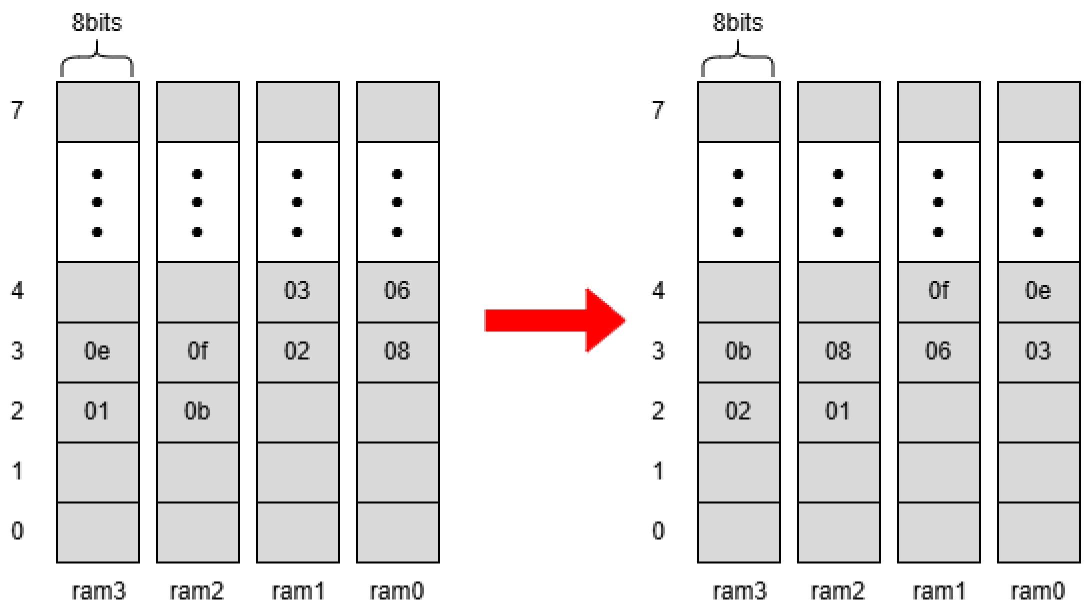
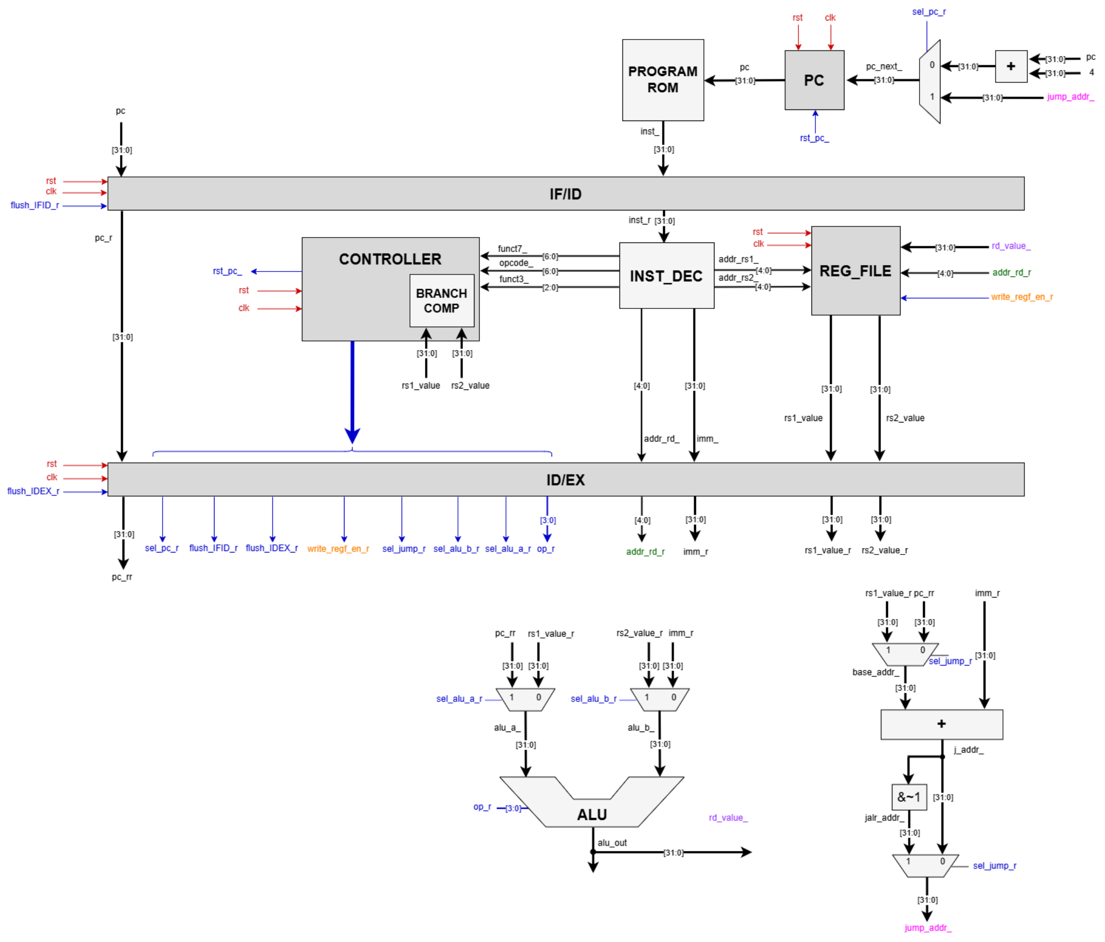


圖 1

```
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)
```

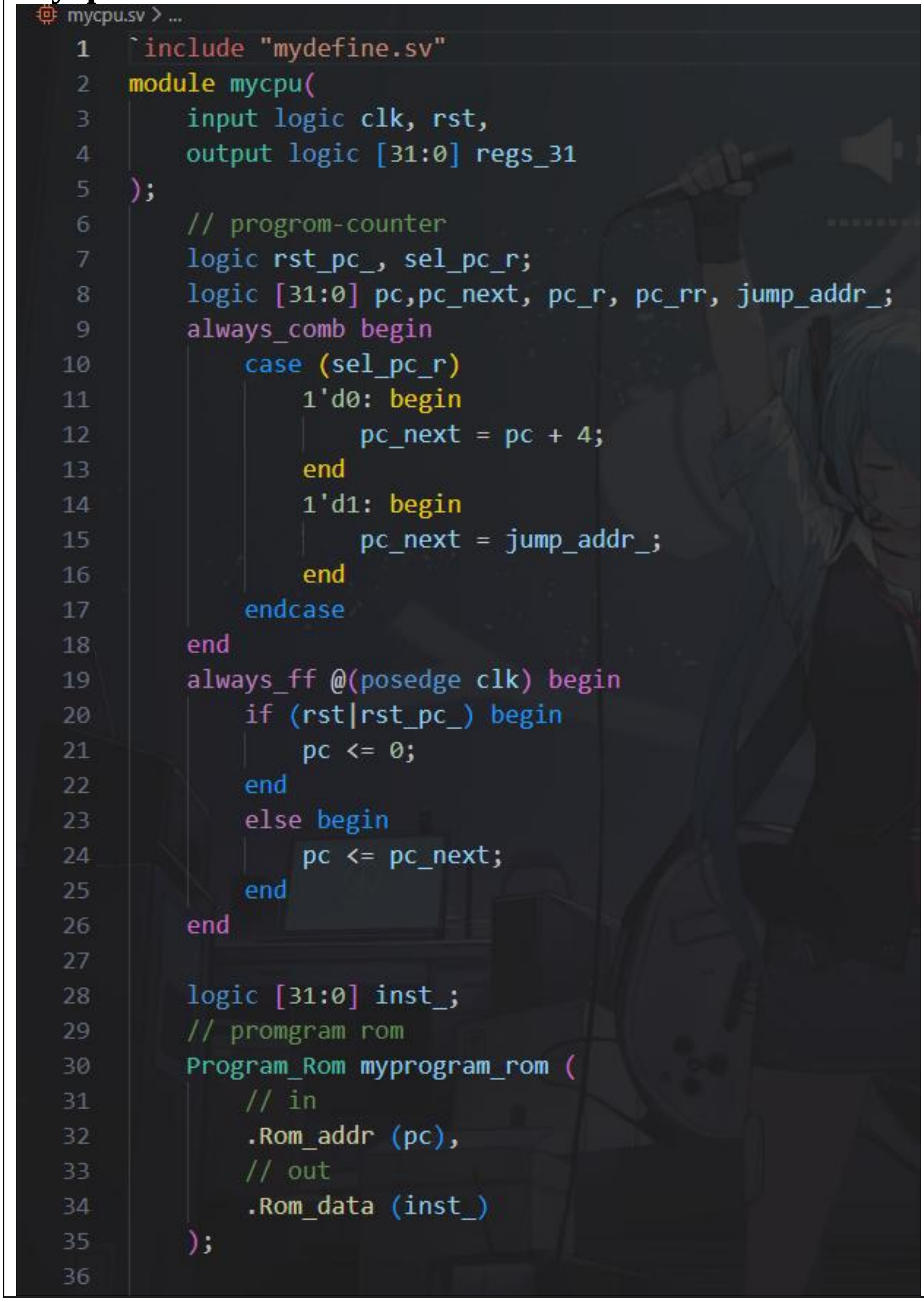
圖 2

## ●系統硬體架構方塊圖（接線圖）：



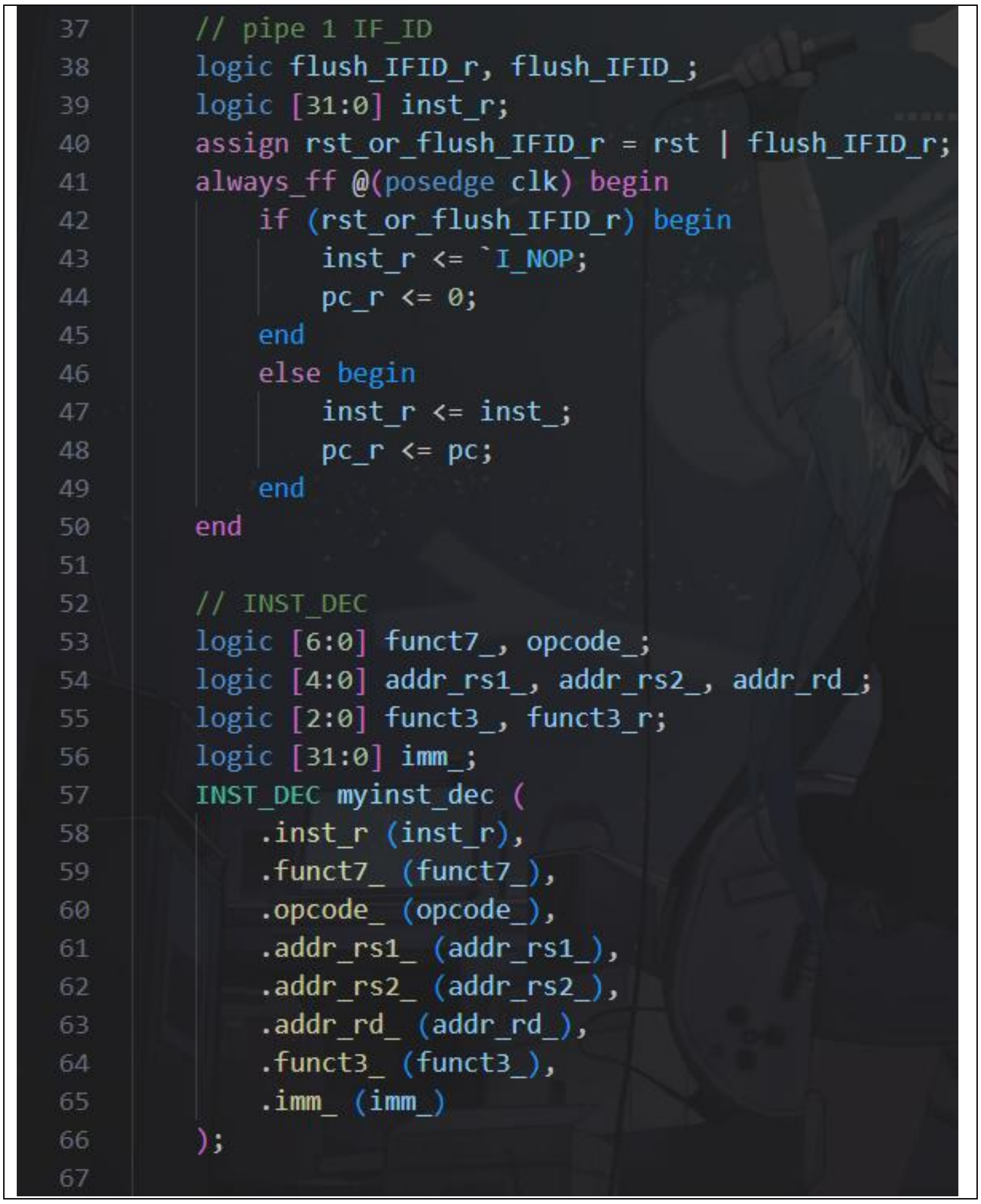
PROGRAM ROM
PC
rst
clk
sel_pc_r
pc_next_
pc
4
+
jump_addr_
rst_pc_
inst_
IF/ID
flush_IFID_r
pc_r
inst_r
CONTROLLER
BRANCH COMP
funct7_
opcode_
funct3_
INST_DEC
addr_rs1_
addr_rs2_
REG_FILE
rd_value_
addr_rd_r
write_regf_en_r
rs1_value
rs2_value
addr_rd_
imm_
ID/EX
flush_IDEX_r
sel_pc_r
flush_IFID_r
flush_IDEX_r
write_regf_en_r
sel_jump_r
sel_alu_b_r
sel_alu_a_r
op_r
addr_rd_r
imm_r
rs1_value_r
rs2_value_r
pc_rr
sel_alu_a_r
sel_alu_b_r
alu_a_
alu_b_
op_r
ALU
alu_out
rd_value_
base_addr_
sel_jump_r
j_addr_
&~1
jalr_addr_
jump_addr_
[31:0]
[6:0]
[2:0]
[4:0]
[3:0]

●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**

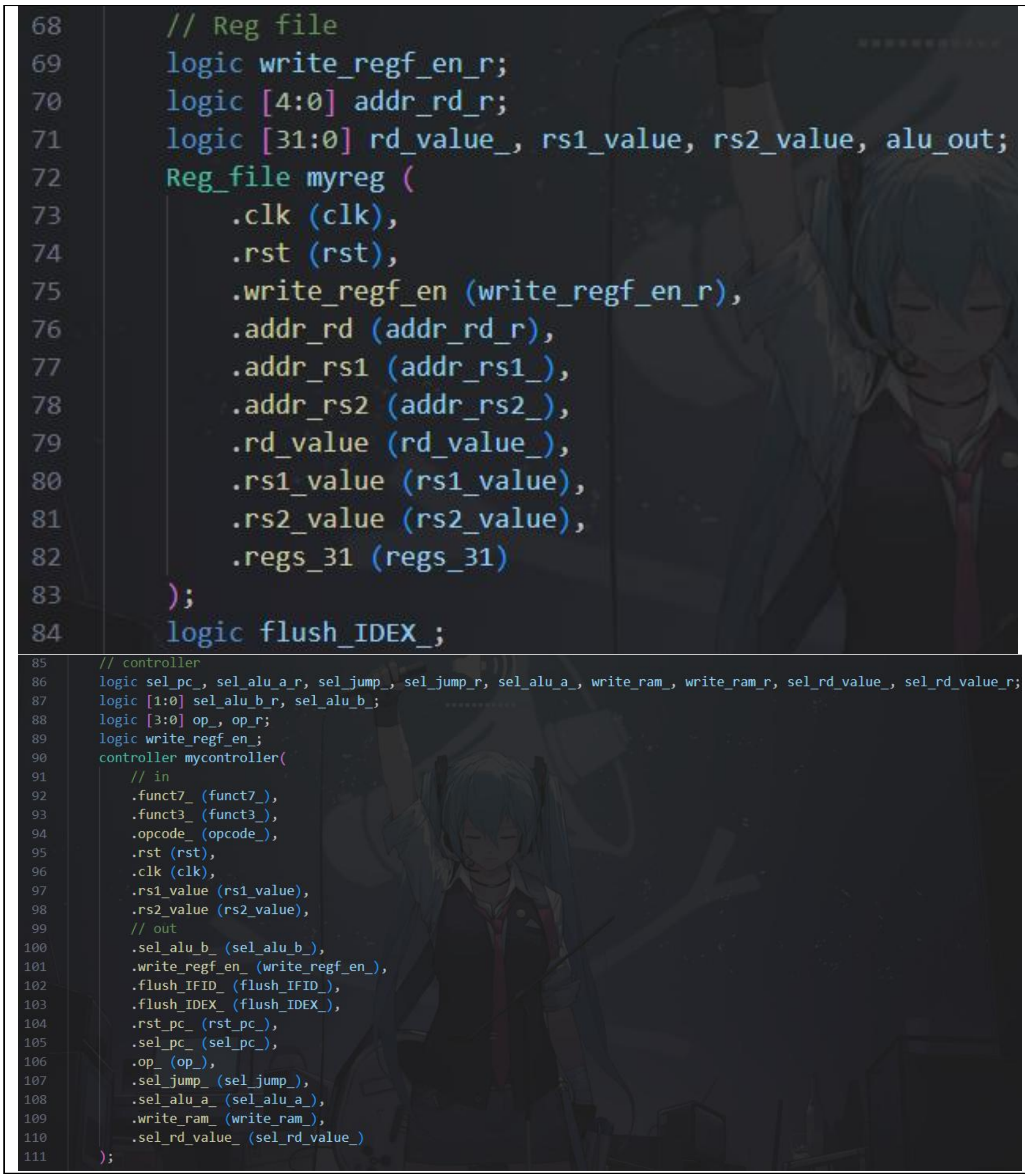
| 以下為此次作業 **RISC-V** 架構 |
|---|
| **Mycpu.sv**  |

mycpu.sv > ...

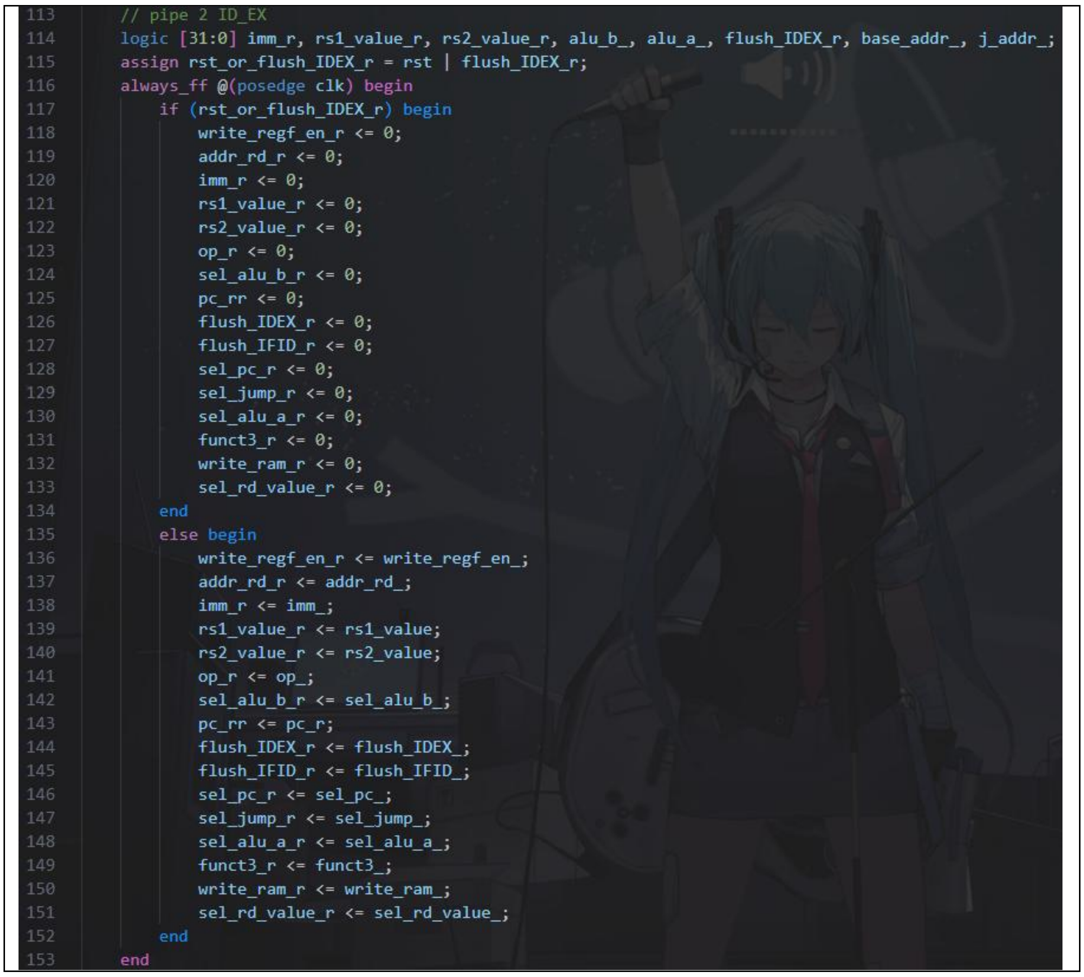
```
`include "mydefine.sv"
module mycpu(
    input logic clk, rst,
    output logic [31:0] regs_31
);
    // progrom-counter
    logic rst_pc_, sel_pc_r;
    logic [31:0] pc,pc_next, pc_r, pc_rr, jump_addr_;
    always_comb begin
        case (sel_pc_r)
            1'd0: begin
                pc_next = pc + 4;
            end
            1'd1: begin
                pc_next = jump_addr_;
            end
        endcase
    end
    always_ff @(posedge clk) begin
        if (rst|rst_pc_) begin
            pc <= 0;
        end
        else begin
            pc <= pc_next;
        end
    end

    logic [31:0] inst_;
    // promgram rom
    Program_Rom myprogram_rom (
        // in
        .Rom_addr (pc),
        // out
        .Rom_data (inst_)
    );

```



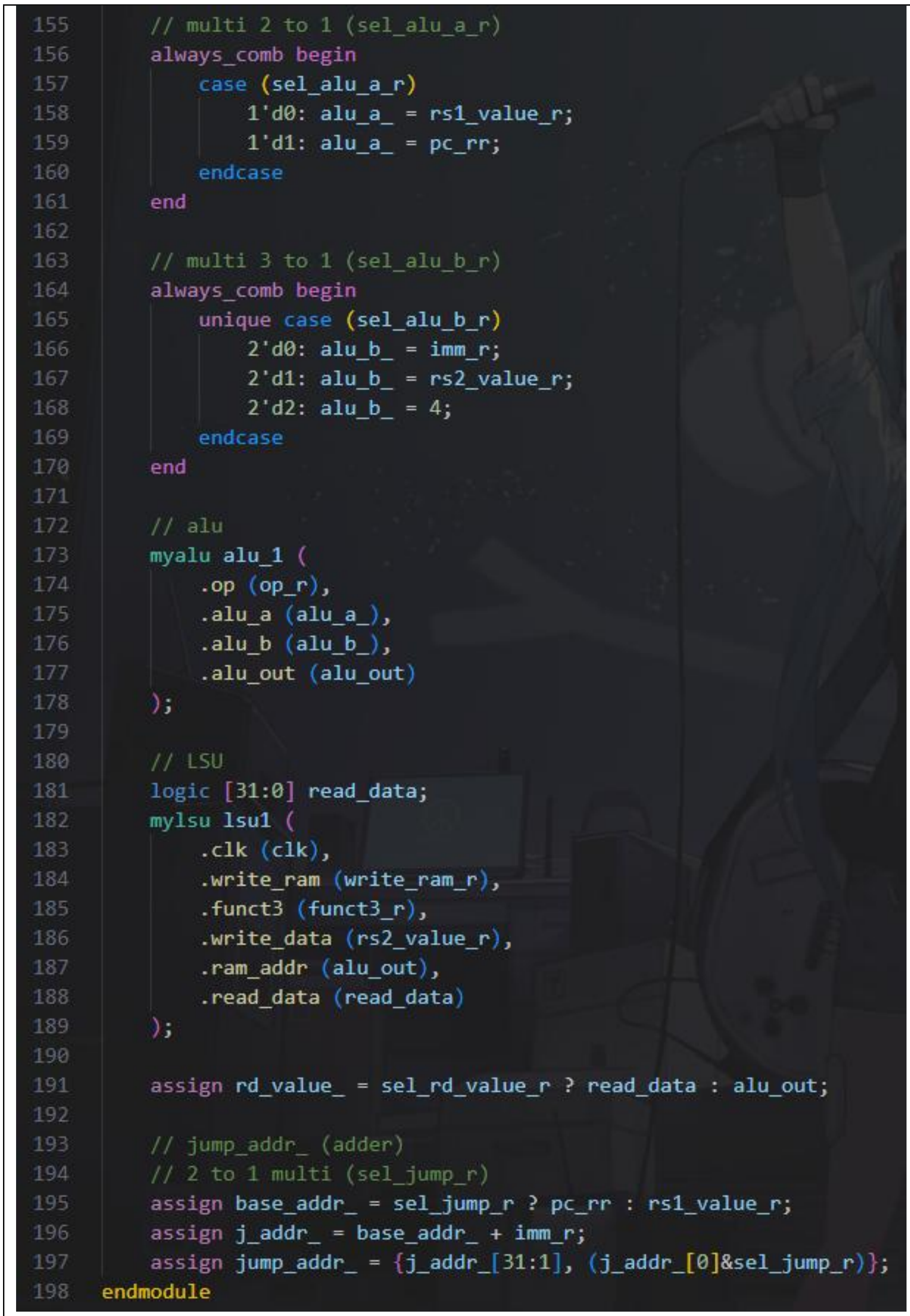
```
    // pipe 1 IF_ID
    logic flush_IFID_r, flush_IFID_;
    logic [31:0] inst_r;
    assign rst_or_flush_IFID_r = rst | flush_IFID_r;
    always_ff @(posedge clk) begin
        if (rst_or_flush_IFID_r) begin
            inst_r <= `I_NOP;
            pc_r <= 0;
        end
        else begin
            inst_r <= inst_;
            pc_r <= pc;
        end
    end

    // INST_DEC
    logic [6:0] funct7_, opcode_;
    logic [4:0] addr_rs1_, addr_rs2_, addr_rd_;
    logic [2:0] funct3_, funct3_r;
    logic [31:0] imm_;
    INST_DEC myinst_dec (
        .inst_r (inst_r),
        .funct7_ (funct7_),
        .opcode_ (opcode_),
        .addr_rs1_ (addr_rs1_),
        .addr_rs2_ (addr_rs2_),
        .addr_rd_ (addr_rd_),
        .funct3_ (funct3_),
        .imm_ (imm_)
    );

```



```
    // Reg file
    logic write_regf_en_r;
    logic [4:0] addr_rd_r;
    logic [31:0] rd_value_, rs1_value, rs2_value, alu_out;
    Reg_file myreg (
        .clk (clk),
        .rst (rst),
        .write_regf_en (write_regf_en_r),
        .addr_rd (addr_rd_r),
        .addr_rs1 (addr_rs1_),
        .addr_rs2 (addr_rs2_),
        .rd_value (rd_value_),
        .rs1_value (rs1_value),
        .rs2_value (rs2_value),
        .regs_31 (regs_31)
    );
    logic flush_IDEX_;
    // controller
    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;
    logic [1:0] sel_alu_b_r, sel_alu_b_;
    logic [3:0] op_, op_r;
    logic write_regf_en_;
    controller mycontroller(
        // in
        .funct7_ (funct7_),
        .funct3_ (funct3_),
        .opcode_ (opcode_),
        .rst (rst),
        .clk (clk),
        .rs1_value (rs1_value),
        .rs2_value (rs2_value),
        // out
        .sel_alu_b_ (sel_alu_b_),
        .write_regf_en_ (write_regf_en_),
        .flush_IFID_ (flush_IFID_),
        .flush_IDEX_ (flush_IDEX_),
        .rst_pc_ (rst_pc_),
        .sel_pc_ (sel_pc_),
        .op_ (op_),
        .sel_jump_ (sel_jump_),
        .sel_alu_a_ (sel_alu_a_),
        .write_ram_ (write_ram_),
        .sel_rd_value_ (sel_rd_value_)
    );
```

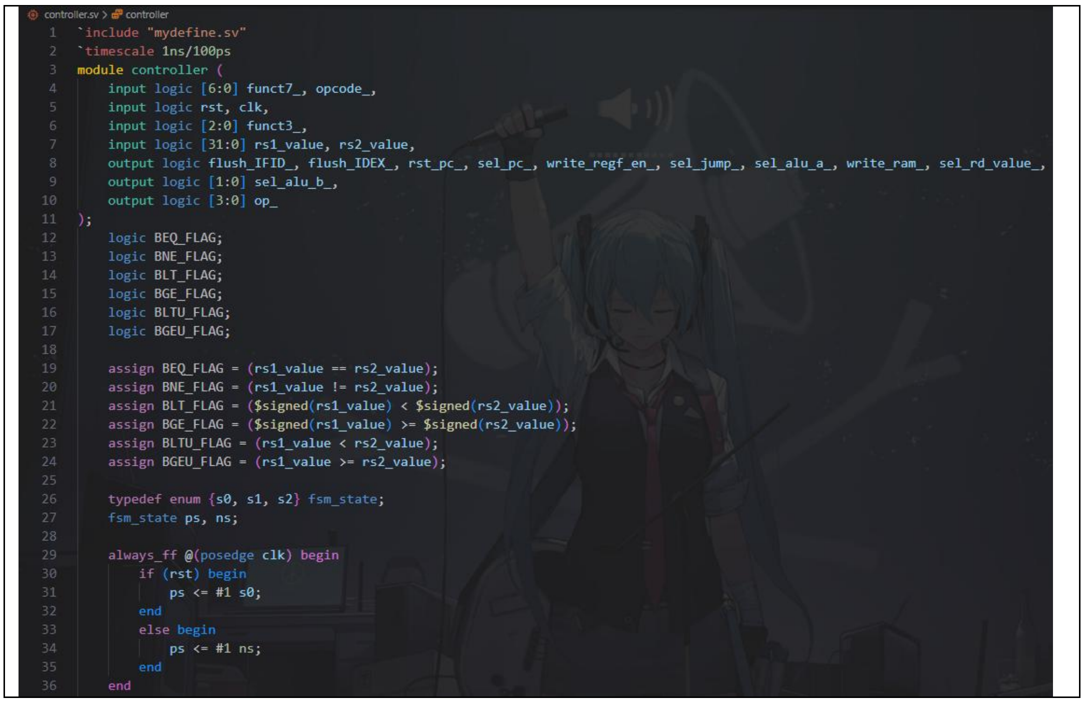


```
// pipe 2 ID_EX
logic [31:0] imm_r, rs1_value_r, rs2_value_r, alu_b_, alu_a_, flush_IDEX_r, base_addr_, j_addr_;
assign rst_or_flush_IDEX_r = rst | flush_IDEX_r;
always_ff @(posedge clk) begin
    if (rst_or_flush_IDEX_r) begin
        write_regf_en_r <= 0;
        addr_rd_r <= 0;
        imm_r <= 0;
        rs1_value_r <= 0;
        rs2_value_r <= 0;
        op_r <= 0;
        sel_alu_b_r <= 0;
        pc_rr <= 0;
        flush_IDEX_r <= 0;
        flush_IFID_r <= 0;
        sel_pc_r <= 0;
        sel_jump_r <= 0;
        sel_alu_a_r <= 0;
        funct3_r <= 0;
        write_ram_r <= 0;
        sel_rd_value_r <= 0;
    end
    else begin
        write_regf_en_r <= write_regf_en_;
        addr_rd_r <= addr_rd_;
        imm_r <= imm_;
        rs1_value_r <= rs1_value;
        rs2_value_r <= rs2_value;
        op_r <= op_;
        sel_alu_b_r <= sel_alu_b_;
        pc_rr <= pc_r;
        flush_IDEX_r <= flush_IDEX_;
        flush_IFID_r <= flush_IFID_;
        sel_pc_r <= sel_pc_;
        sel_jump_r <= sel_jump_;
        sel_alu_a_r <= sel_alu_a_;
        funct3_r <= funct3_;
        write_ram_r <= write_ram_;
        sel_rd_value_r <= sel_rd_value_;
    end
end
```

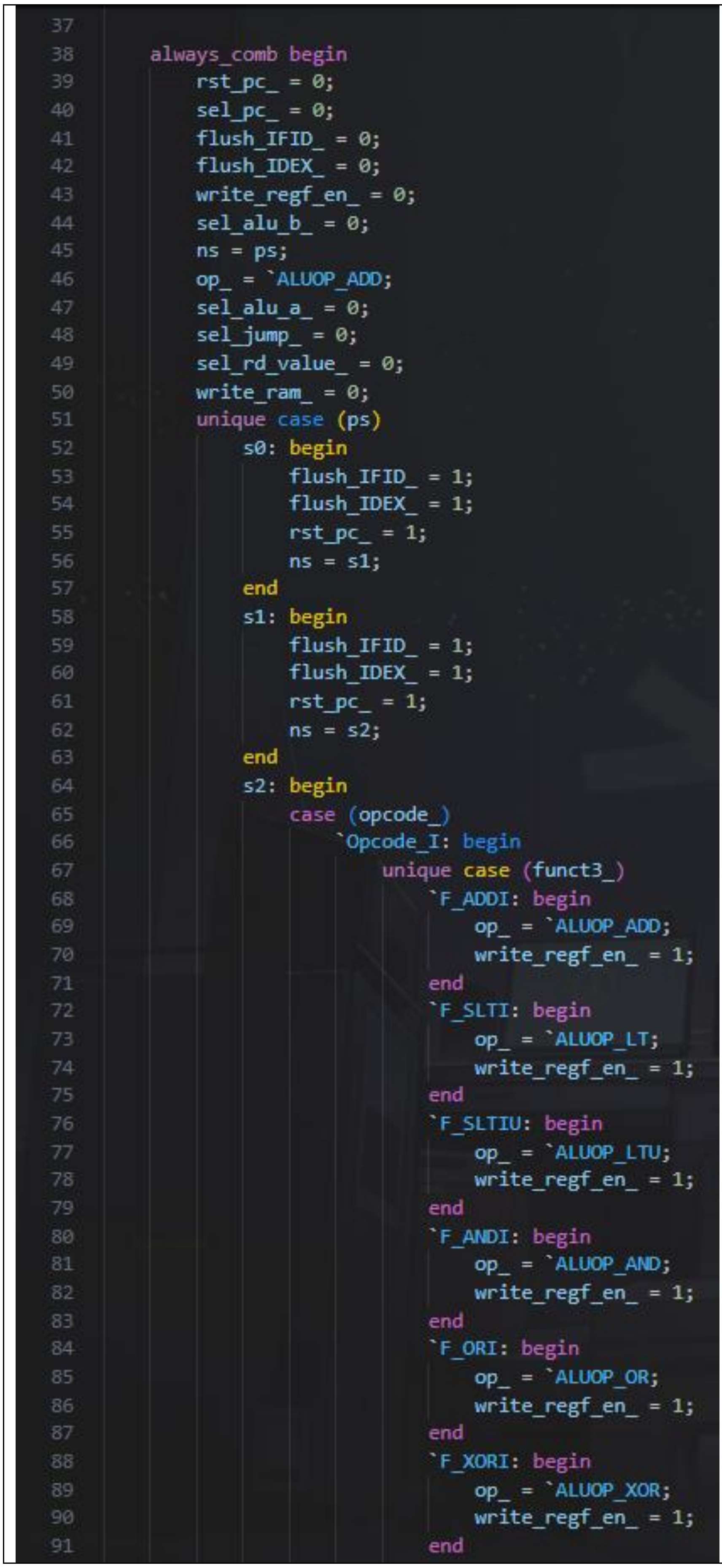


```
    // multi 2 to 1 (sel_alu_a_r)
    always_comb begin
        case (sel_alu_a_r)
            1'd0: alu_a_ = rs1_value_r;
            1'd1: alu_a_ = pc_rr;
        endcase
    end

    // multi 3 to 1 (sel_alu_b_r)
    always_comb begin
        unique case (sel_alu_b_r)
            2'd0: alu_b_ = imm_r;
            2'd1: alu_b_ = rs2_value_r;
            2'd2: alu_b_ = 4;
        endcase
    end

    // alu
    myalu alu_1 (
        .op (op_r),
        .alu_a (alu_a_),
        .alu_b (alu_b_),
        .alu_out (alu_out)
    );

    // LSU
    logic [31:0] read_data;
    mylsu lsu1 (
        .clk (clk),
        .write_ram (write_ram_r),
        .funct3 (funct3_r),
        .write_data (rs2_value_r),
        .ram_addr (alu_out),
        .read_data (read_data)
    );

    assign rd_value_ = sel_rd_value_r ? read_data : alu_out;

    // jump_addr_ (adder)
    // 2 to 1 multi (sel_jump_r)
    assign base_addr_ = sel_jump_r ? pc_rr : rs1_value_r;
    assign j_addr_ = base_addr_ + imm_r;
    assign jump_addr_ = {j_addr_[31:1], (j_addr_[0]&sel_jump_r)};
endmodule
```

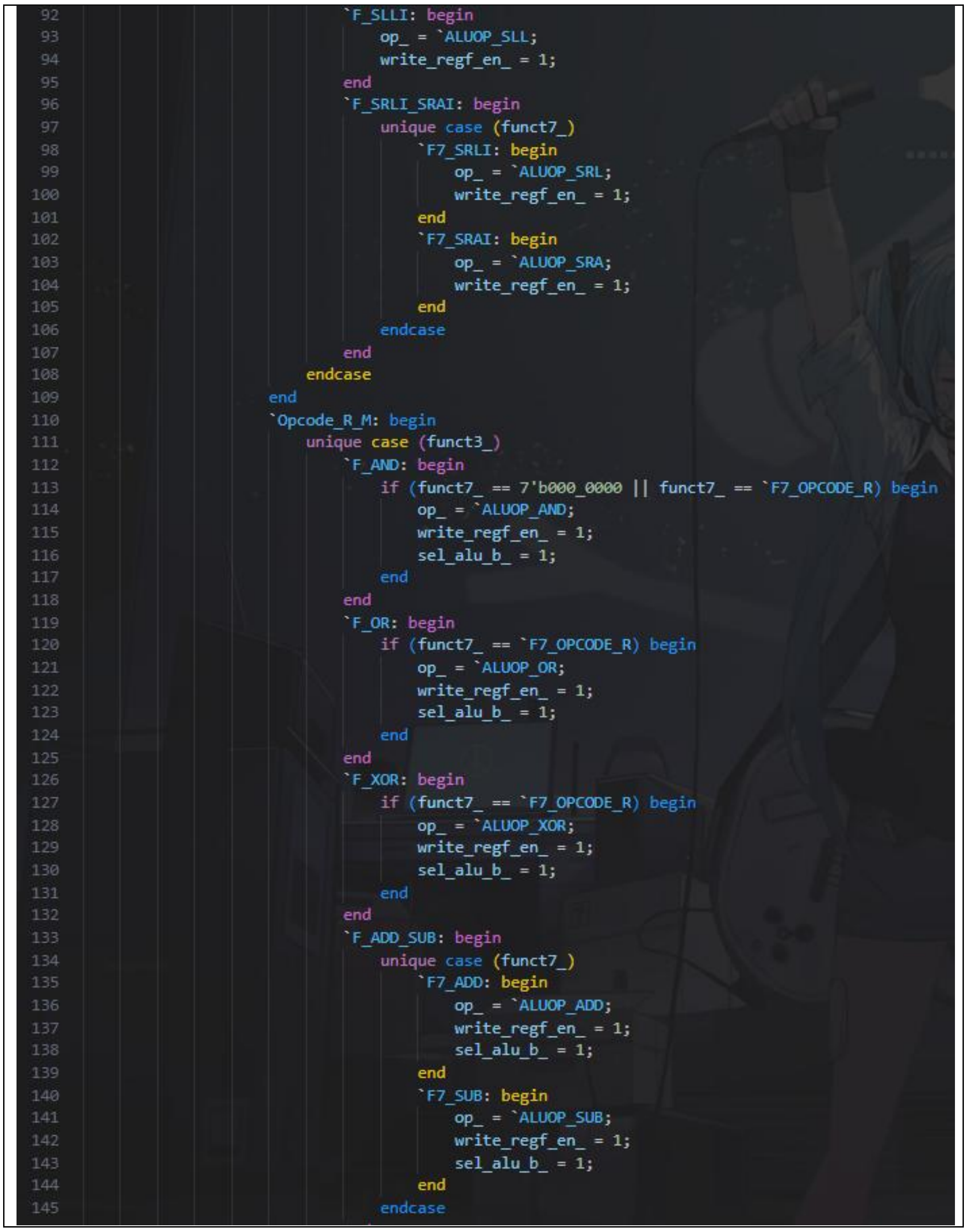
**Controller.sv**


controller.sv > controller

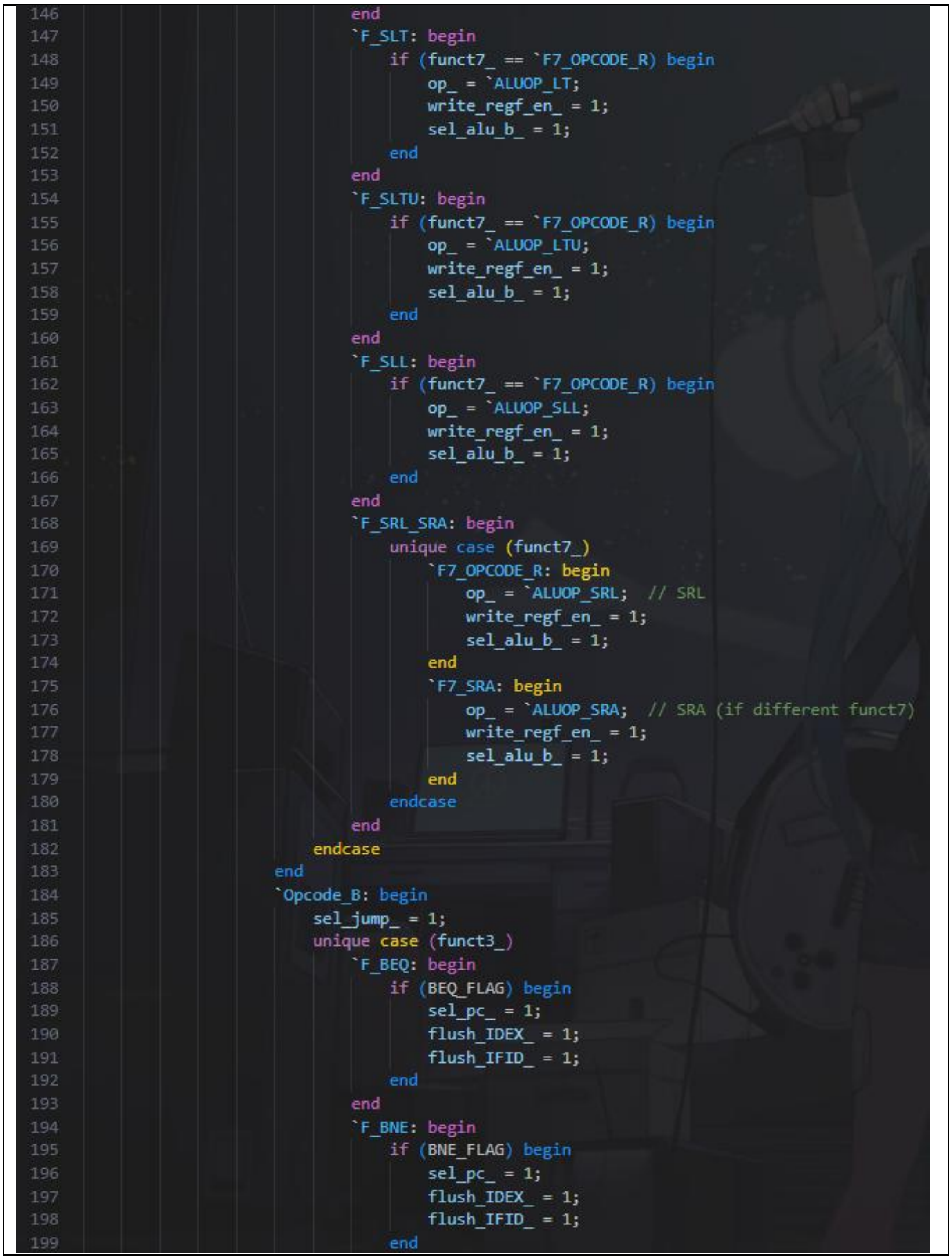
```systemverilog
`include "mydefine.sv"
`timescale 1ns/100ps
module controller (
    input logic [6:0] funct7_, opcode_,
    input logic rst, clk,
    input logic [2:0] funct3_,
    input logic [31:0] rs1_value, rs2_value,
    output logic flush_IFID_, flush_IDEX_, rst_pc_, sel_pc_, write_regf_en_, sel_jump_, sel_alu_a_, write_ram_, sel_rd_value_,
    output logic [1:0] sel_alu_b_,
    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
```



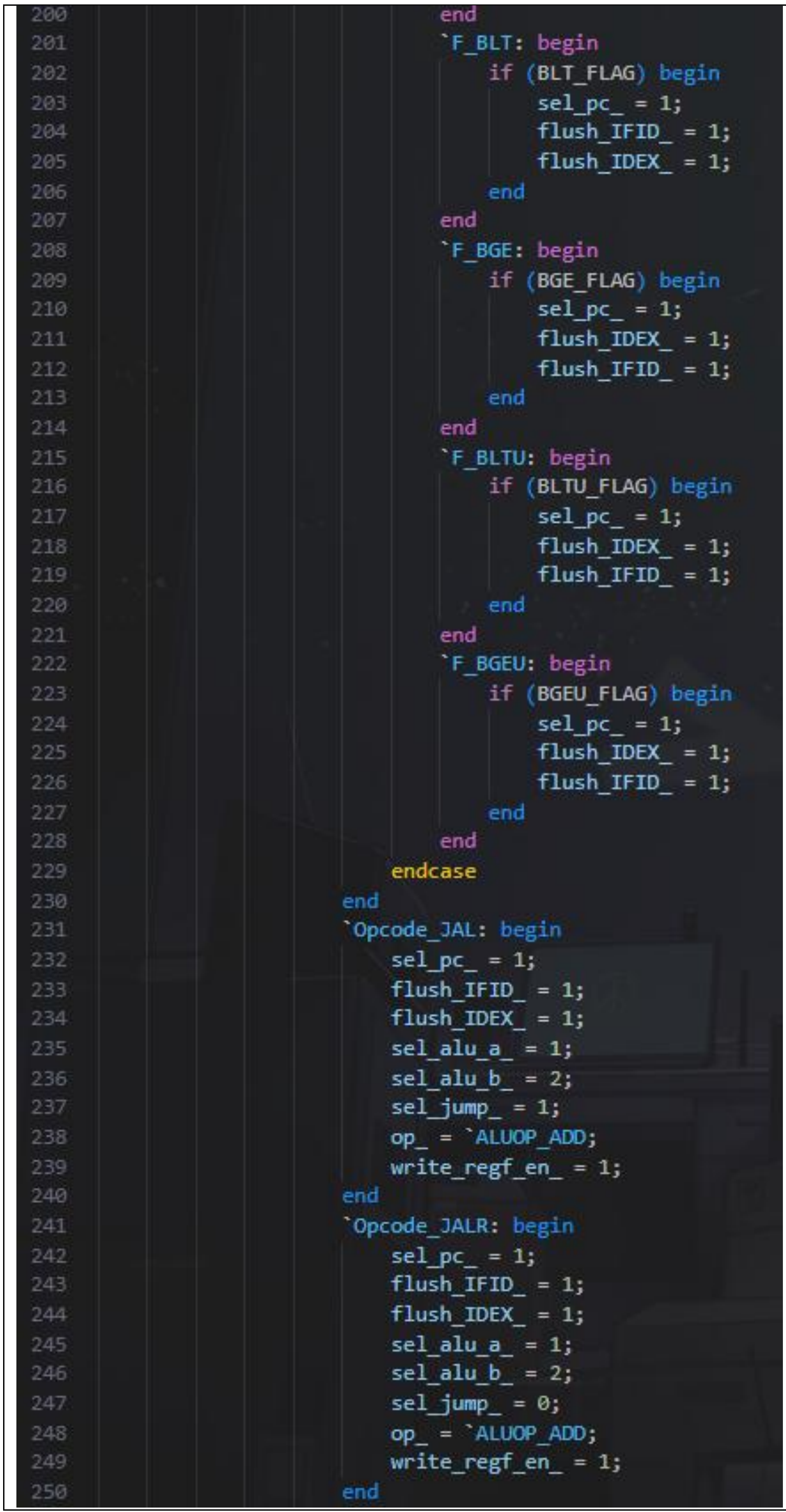
```
    always_comb begin
        rst_pc_ = 0;
        sel_pc_ = 0;
        flush_IFID_ = 0;
        flush_IDEX_ = 0;
        write_regf_en_ = 0;
        sel_alu_b_ = 0;
        ns = ps;
        op_ = `ALUOP_ADD;
        sel_alu_a_ = 0;
        sel_jump_ = 0;
        sel_rd_value_ = 0;
        write_ram_ = 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
            s2: begin
                case (opcode_)
                    `Opcode_I: begin
                        unique case (funct3_)
                            `F_ADDI: begin
                                op_ = `ALUOP_ADD;
                                write_regf_en_ = 1;
                            end
                            `F_SLTI: begin
                                op_ = `ALUOP_LT;
                                write_regf_en_ = 1;
                            end
                            `F_SLTIU: begin
                                op_ = `ALUOP_LTU;
                                write_regf_en_ = 1;
                            end
                            `F_ANDI: begin
                                op_ = `ALUOP_AND;
                                write_regf_en_ = 1;
                            end
                            `F_ORI: begin
                                op_ = `ALUOP_OR;
                                write_regf_en_ = 1;
                            end
                            `F_XORI: begin
                                op_ = `ALUOP_XOR;
                                write_regf_en_ = 1;
                            end
```



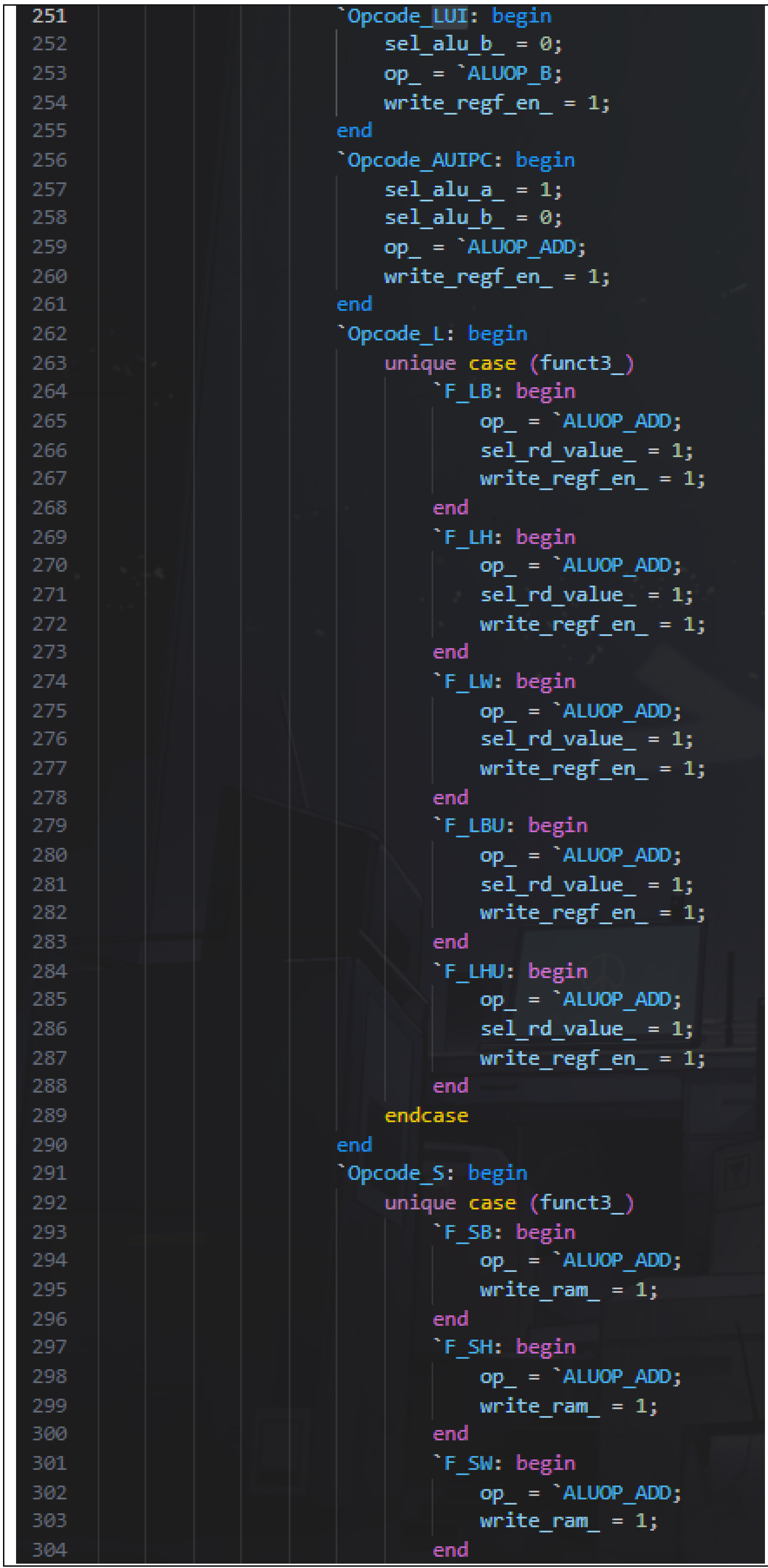
```
                            `F_SLLI: begin
                                op_ = `ALUOP_SLL;
                                write_regf_en_ = 1;
                            end
                            `F_SRLI_SRAI: begin
                                unique case (funct7_)
                                    `F7_SRLI: begin
                                        op_ = `ALUOP_SRL;
                                        write_regf_en_ = 1;
                                    end
                                    `F7_SRAI: begin
                                        op_ = `ALUOP_SRA;
                                        write_regf_en_ = 1;
                                    end
                                endcase
                            end
                        endcase
                    end
                    `Opcode_R_M: begin
                        unique case (funct3_)
                            `F_AND: begin
                                if (funct7_ == 7'b000_0000 || funct7_ == `F7_OPCODE_R) begin
                                    op_ = `ALUOP_AND;
                                    write_regf_en_ = 1;
                                    sel_alu_b_ = 1;
                                end
                            end
                            `F_OR: begin
                                if (funct7_ == `F7_OPCODE_R) begin
                                    op_ = `ALUOP_OR;
                                    write_regf_en_ = 1;
                                    sel_alu_b_ = 1;
                                end
                            end
                            `F_XOR: begin
                                if (funct7_ == `F7_OPCODE_R) begin
                                    op_ = `ALUOP_XOR;
                                    write_regf_en_ = 1;
                                    sel_alu_b_ = 1;
                                end
                            end
                            `F_ADD_SUB: begin
                                unique case (funct7_)
                                    `F7_ADD: begin
                                        op_ = `ALUOP_ADD;
                                        write_regf_en_ = 1;
                                        sel_alu_b_ = 1;
                                    end
                                    `F7_SUB: begin
                                        op_ = `ALUOP_SUB;
                                        write_regf_en_ = 1;
                                        sel_alu_b_ = 1;
                                    end
                                endcase
```



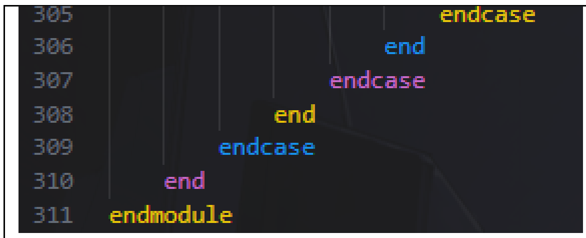
```
                            end
                            `F_SLT: begin
                                if (funct7_ == `F7_OPCODE_R) begin
                                    op_ = `ALUOP_LT;
                                    write_regf_en_ = 1;
                                    sel_alu_b_ = 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
                        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
                        `F_BLTU: begin
                            if (BLTU_FLAG) begin
                                sel_pc_ = 1;
                                flush_IDEX_ = 1;
                                flush_IFID_ = 1;
                            end
                        end
                        `F_BGEU: begin
                            if (BGEU_FLAG) begin
                                sel_pc_ = 1;
                                flush_IDEX_ = 1;
                                flush_IFID_ = 1;
                            end
                        end
                    endcase
                end
                `Opcode_JAL: begin
                    sel_pc_ = 1;
                    flush_IFID_ = 1;
                    flush_IDEX_ = 1;
                    sel_alu_a_ = 1;
                    sel_alu_b_ = 2;
                    sel_jump_ = 1;
                    op_ = `ALUOP_ADD;
                    write_regf_en_ = 1;
                end
                `Opcode_JALR: begin
                    sel_pc_ = 1;
                    flush_IFID_ = 1;
                    flush_IDEX_ = 1;
                    sel_alu_a_ = 1;
                    sel_alu_b_ = 2;
                    sel_jump_ = 0;
                    op_ = `ALUOP_ADD;
                    write_regf_en_ = 1;
                end
```

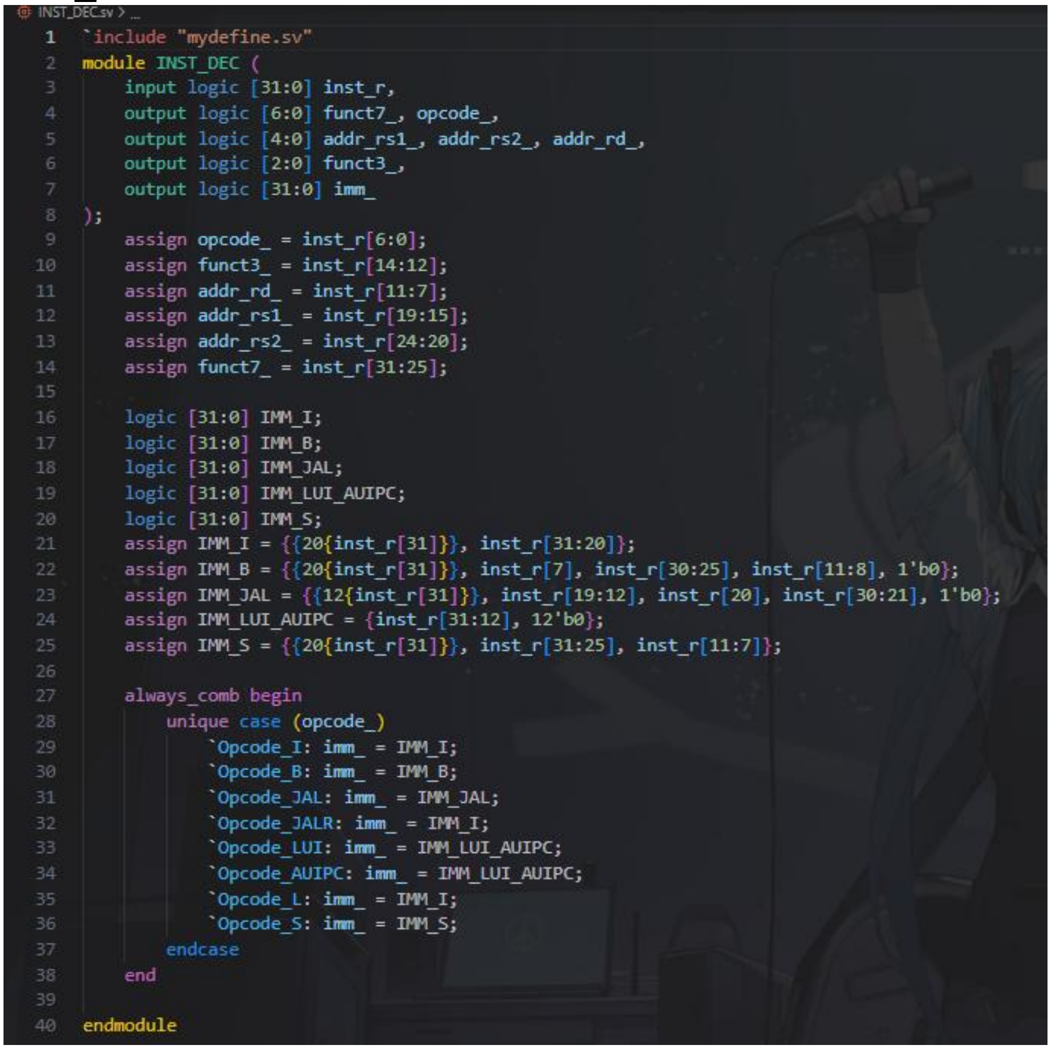


```
                    `Opcode_LUI: begin
                        sel_alu_b_ = 0;
                        op_ = `ALUOP_B;
                        write_regf_en_ = 1;
                    end
                    `Opcode_AUIPC: begin
                        sel_alu_a_ = 1;
                        sel_alu_b_ = 0;
                        op_ = `ALUOP_ADD;
                        write_regf_en_ = 1;
                    end
                    `Opcode_L: begin
                        unique case (funct3_)
                            `F_LB: begin
                                op_ = `ALUOP_ADD;
                                sel_rd_value_ = 1;
                                write_regf_en_ = 1;
                            end
                            `F_LH: begin
                                op_ = `ALUOP_ADD;
                                sel_rd_value_ = 1;
                                write_regf_en_ = 1;
                            end
                            `F_LW: begin
                                op_ = `ALUOP_ADD;
                                sel_rd_value_ = 1;
                                write_regf_en_ = 1;
                            end
                            `F_LBU: begin
                                op_ = `ALUOP_ADD;
                                sel_rd_value_ = 1;
                                write_regf_en_ = 1;
                            end
                            `F_LHU: begin
                                op_ = `ALUOP_ADD;
                                sel_rd_value_ = 1;
                                write_regf_en_ = 1;
                            end
                        endcase
                    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
```



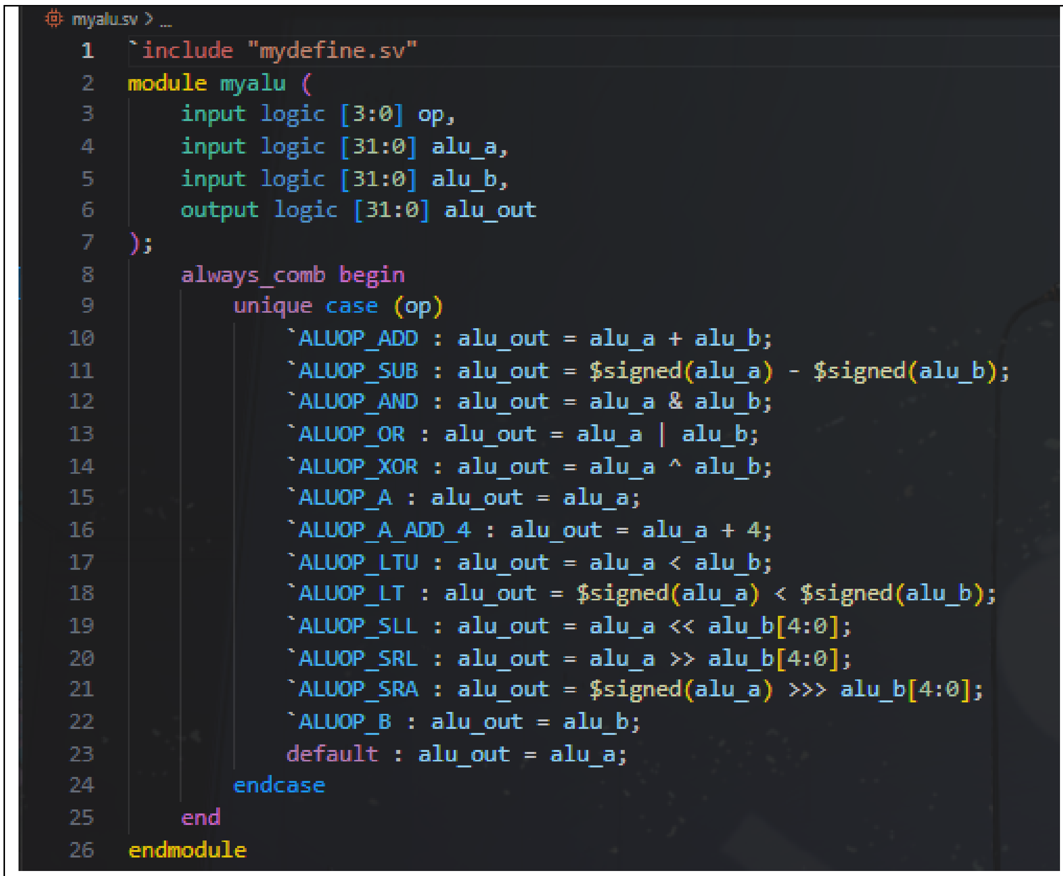
```
                        endcase
                    end
                endcase
            end
        endcase
    end
endmodule
```

## 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
```

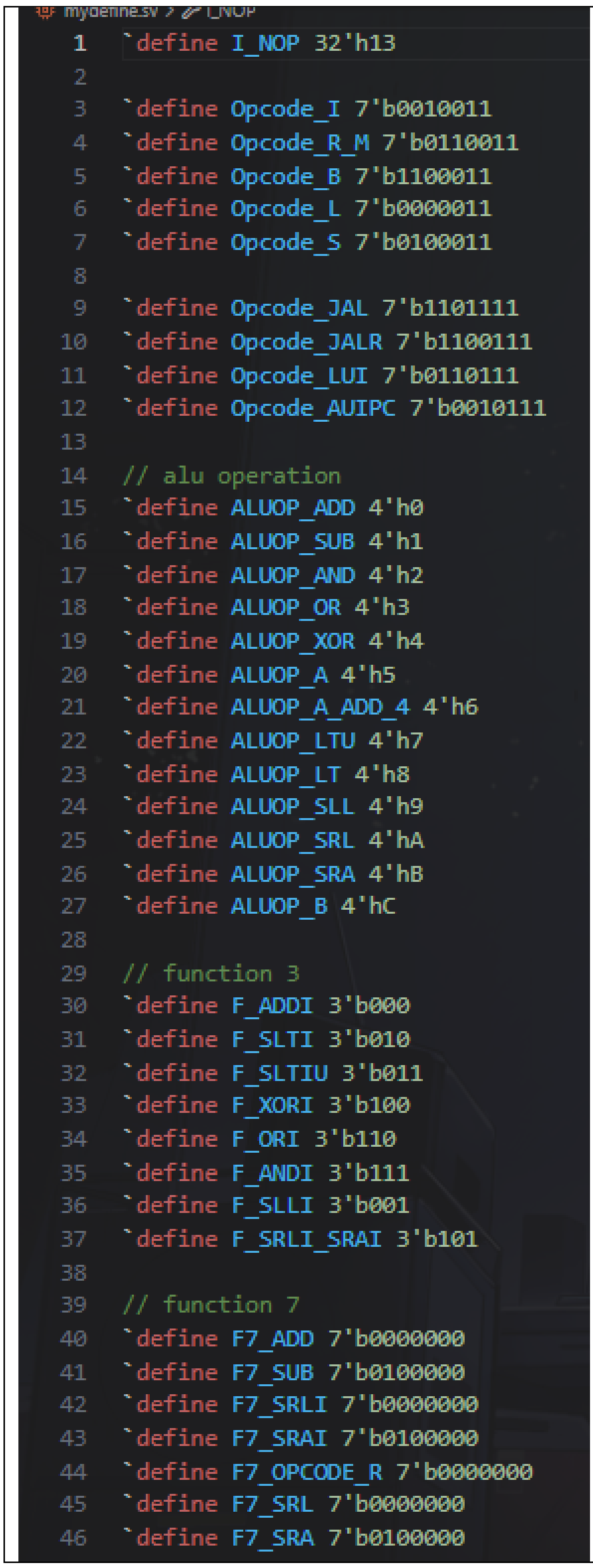
## Myalu.sv



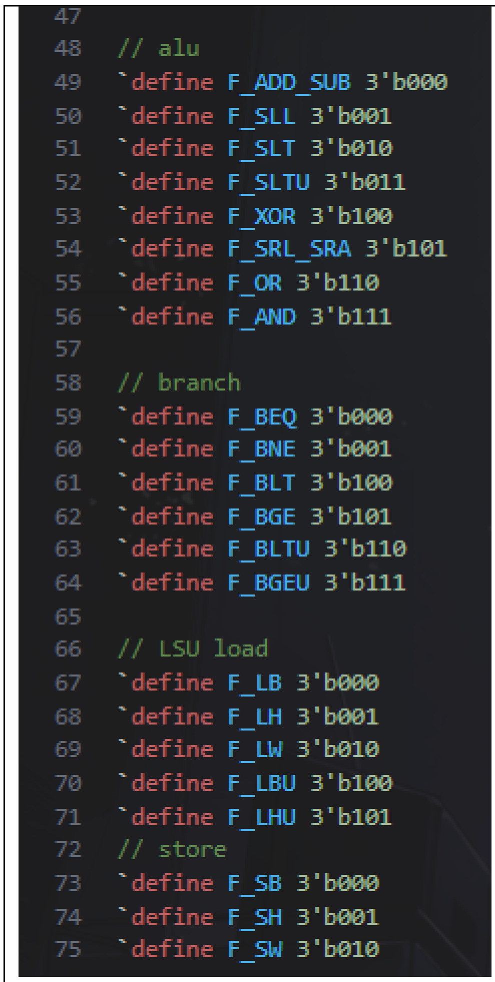
myalu.sv > ...

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

**Mydefine.sv**

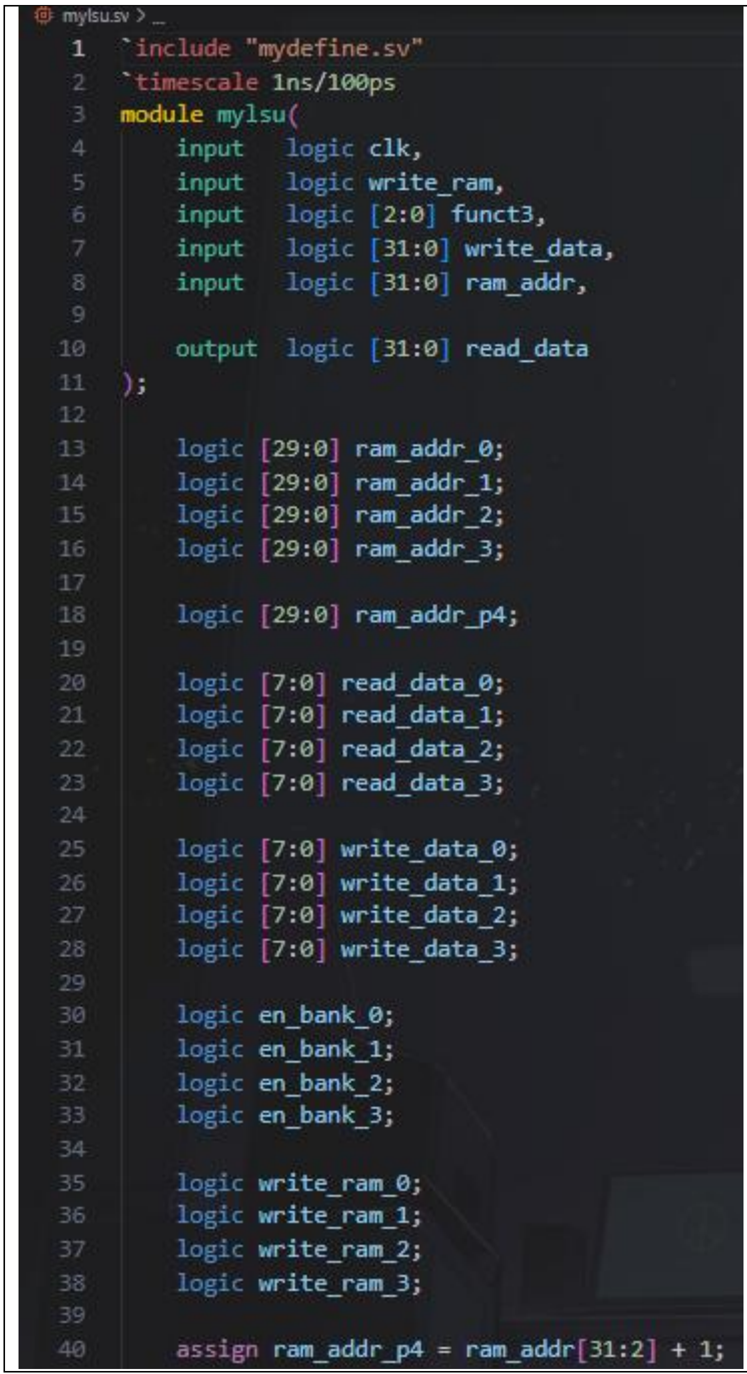


```
`define I_NOP 32'h13

`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
```

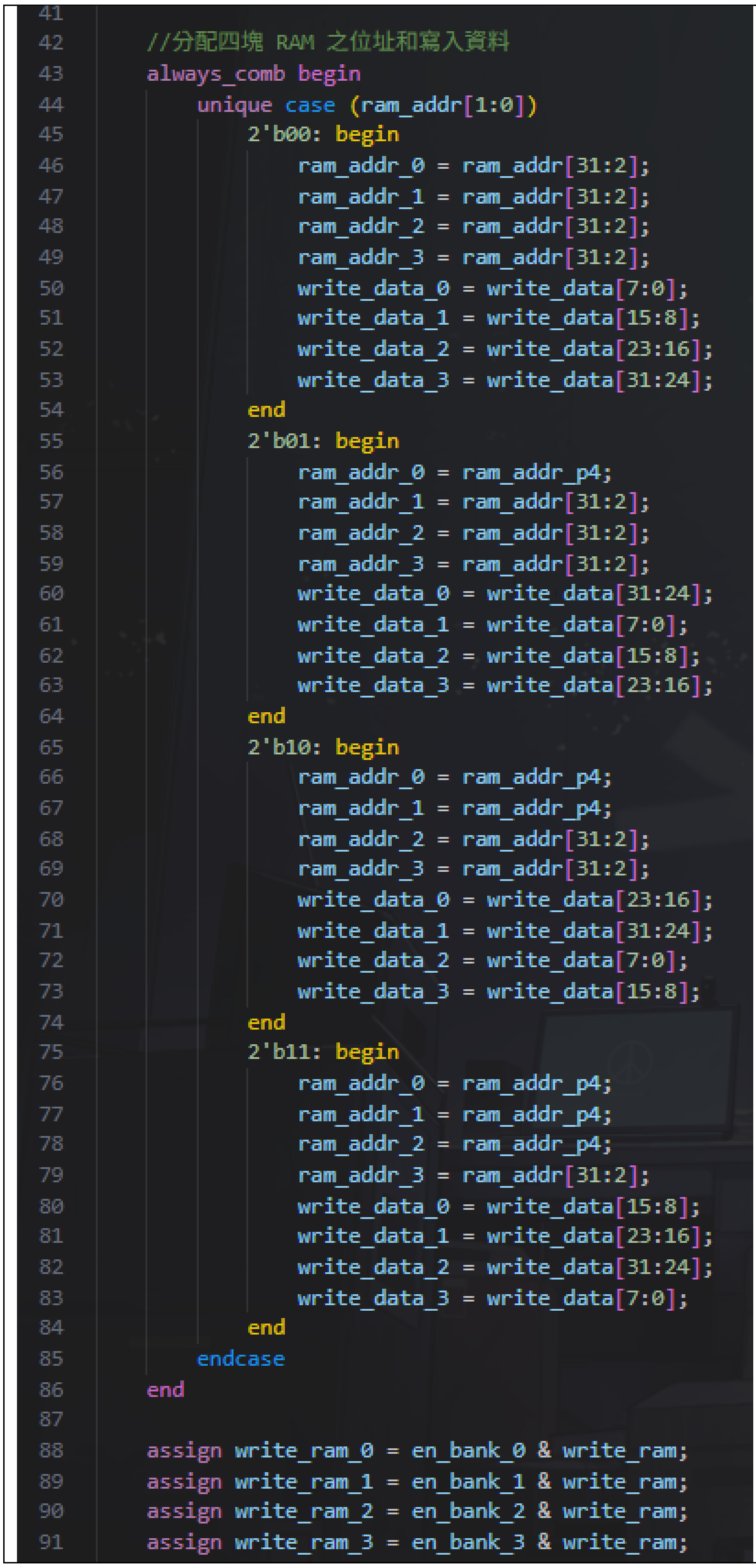


```
// alu
`define F_ADD_SUB 3'b000
`define F_SLL 3'b001
`define F_SLT 3'b010
`define F_SLTU 3'b011
`define F_XOR 3'b100
`define F_SRL_SRA 3'b101
`define F_OR 3'b110
`define F_AND 3'b111

// 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
```

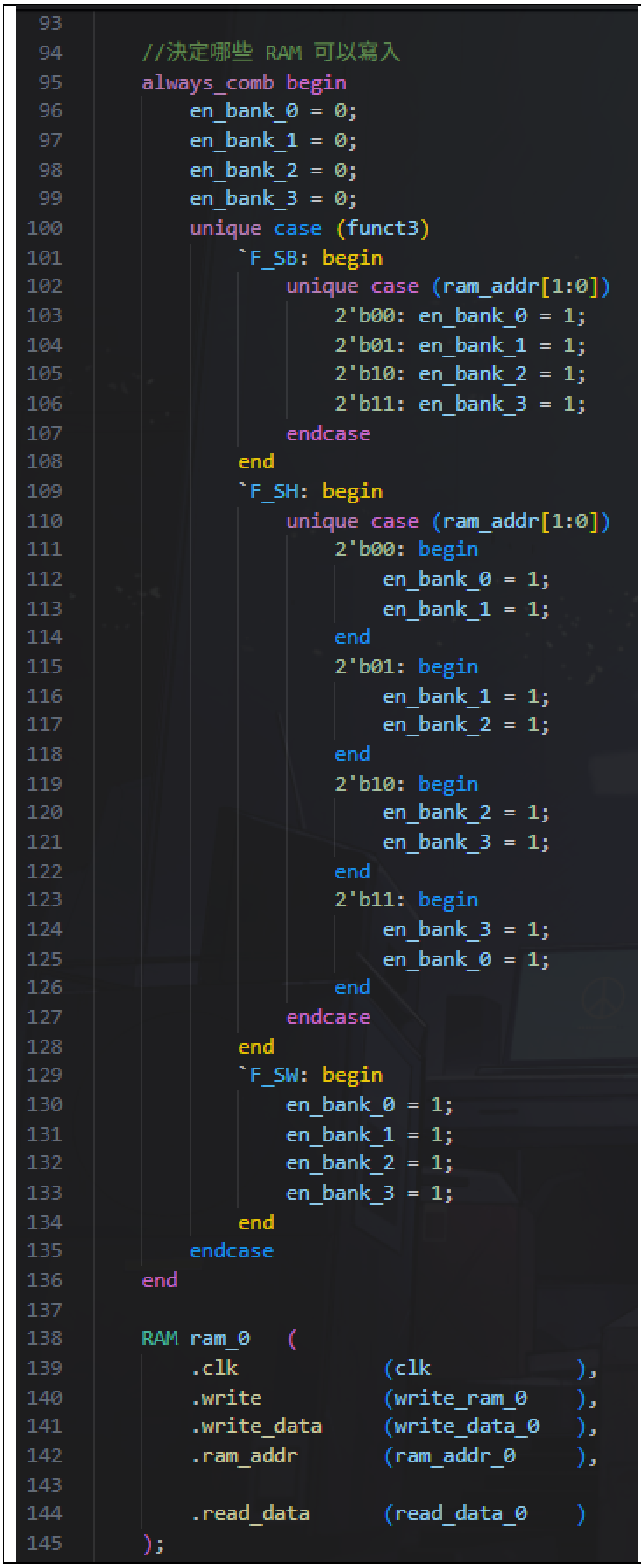
**Mylsu.sv**


mylsu.sv > ...

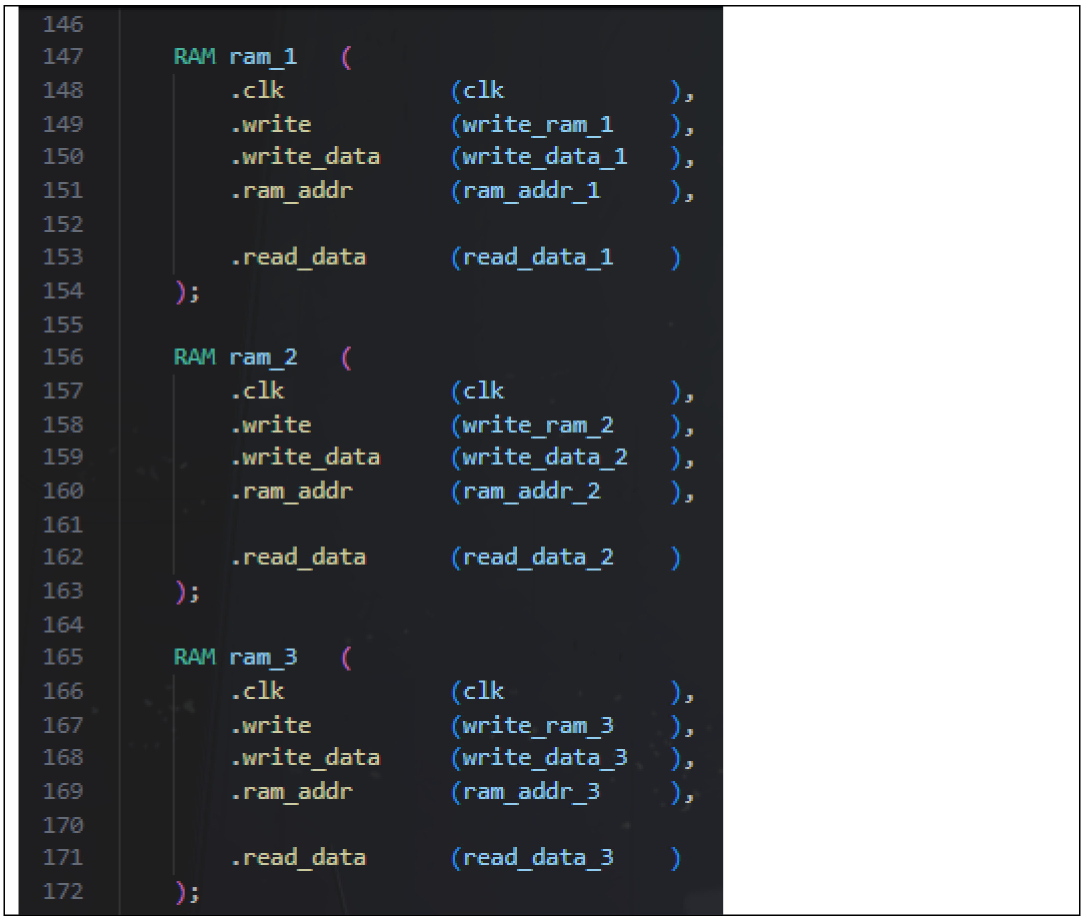
```
`include "mydefine.sv"
`timescale 1ns/100ps
module mylsu(
    input   logic clk,
    input   logic write_ram,
    input   logic [2:0] funct3,
    input   logic [31:0] write_data,
    input   logic [31:0] ram_addr,

    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;
```



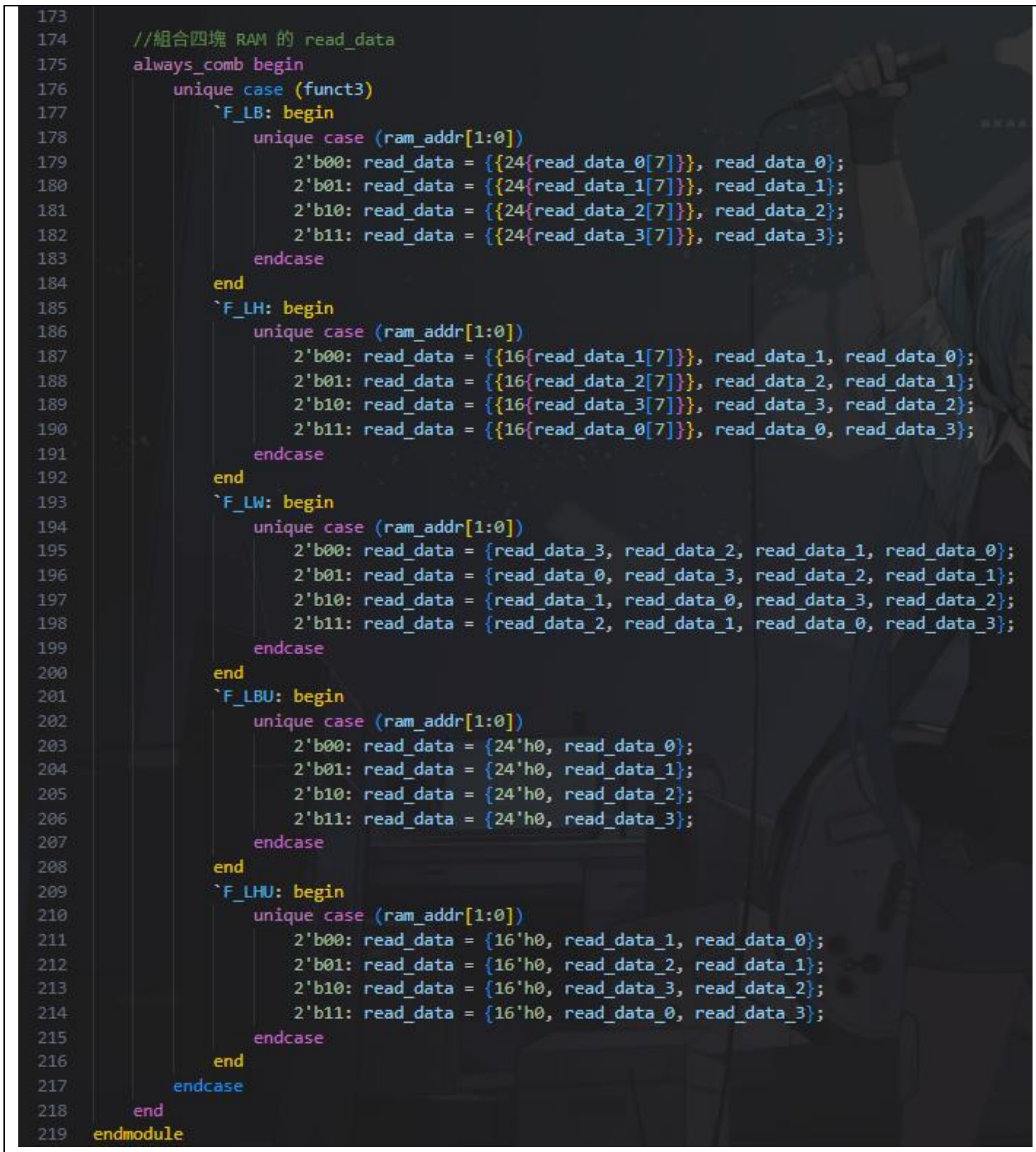
```
    //分配四塊 RAM 之位址和寫入資料
    always_comb begin
        unique case (ram_addr[1:0])
            2'b00: begin
                ram_addr_0 = ram_addr[31:2];
                ram_addr_1 = ram_addr[31:2];
                ram_addr_2 = ram_addr[31:2];
                ram_addr_3 = ram_addr[31:2];
                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 可以寫入
    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    )
    );
```

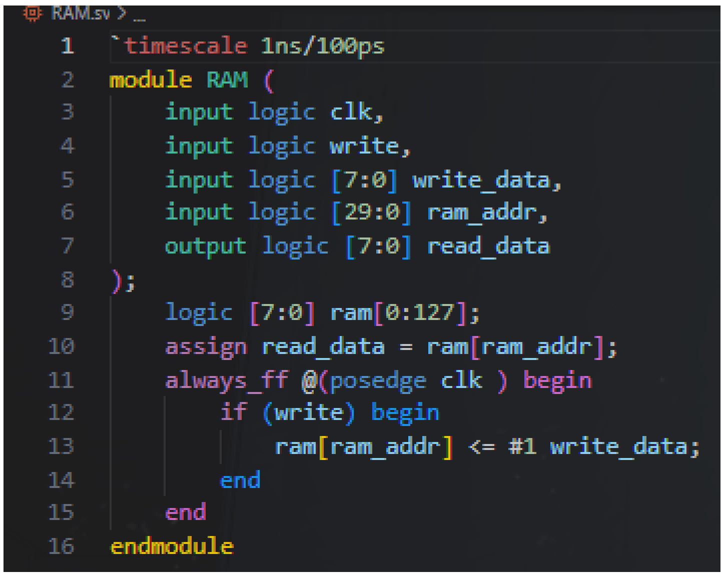


```

    RAM ram_1   (
        .clk            (clk            ),
        .write          (write_ram_1    ),
        .write_data     (write_data_1   ),
        .ram_addr       (ram_addr_1     ),

        .read_data      (read_data_1    )
    );

    RAM ram_2   (
        .clk            (clk            ),
        .write          (write_ram_2    ),
        .write_data     (write_data_2   ),
        .ram_addr       (ram_addr_2     ),

        .read_data      (read_data_2    )
    );

    RAM ram_3   (
        .clk            (clk            ),
        .write          (write_ram_3    ),
        .write_data     (write_data_3   ),
        .ram_addr       (ram_addr_3     ),

        .read_data      (read_data_3    )
    );
```



```systemverilog

    //組合四塊 RAM 的 read_data
    always_comb begin
        unique case (funct3)
            `F_LB: begin
                unique case (ram_addr[1:0])
                    2'b00: read_data = {{24{read_data_0[7]}}, read_data_0};
                    2'b01: read_data = {{24{read_data_1[7]}}, read_data_1};
                    2'b10: read_data = {{24{read_data_2[7]}}, read_data_2};
                    2'b11: read_data = {{24{read_data_3[7]}}, read_data_3};
                endcase
            end
            `F_LH: begin
                unique case (ram_addr[1:0])
                    2'b00: read_data = {{16{read_data_1[7]}}, read_data_1, read_data_0};
                    2'b01: read_data = {{16{read_data_2[7]}}, read_data_2, read_data_1};
                    2'b10: read_data = {{16{read_data_3[7]}}, read_data_3, read_data_2};
                    2'b11: read_data = {{16{read_data_0[7]}}, read_data_0, read_data_3};
                endcase
            end
            `F_LW: begin
                unique case (ram_addr[1:0])
                    2'b00: read_data = {read_data_3, read_data_2, read_data_1, read_data_0};
                    2'b01: read_data = {read_data_0, read_data_3, read_data_2, read_data_1};
                    2'b10: read_data = {read_data_1, read_data_0, read_data_3, read_data_2};
                    2'b11: read_data = {read_data_2, read_data_1, read_data_0, read_data_3};
                endcase
            end
            `F_LBU: begin
                unique case (ram_addr[1:0])
                    2'b00: read_data = {24'h0, read_data_0};
                    2'b01: read_data = {24'h0, read_data_1};
                    2'b10: read_data = {24'h0, read_data_2};
                    2'b11: read_data = {24'h0, read_data_3};
                endcase
            end
            `F_LHU: begin
                unique case (ram_addr[1:0])
                    2'b00: read_data = {16'h0, read_data_1, read_data_0};
                    2'b01: read_data = {16'h0, read_data_2, read_data_1};
                    2'b10: read_data = {16'h0, read_data_3, read_data_2};
                    2'b11: read_data = {16'h0, read_data_0, read_data_3};
                endcase
            end
        endcase
    end
endmodule
```

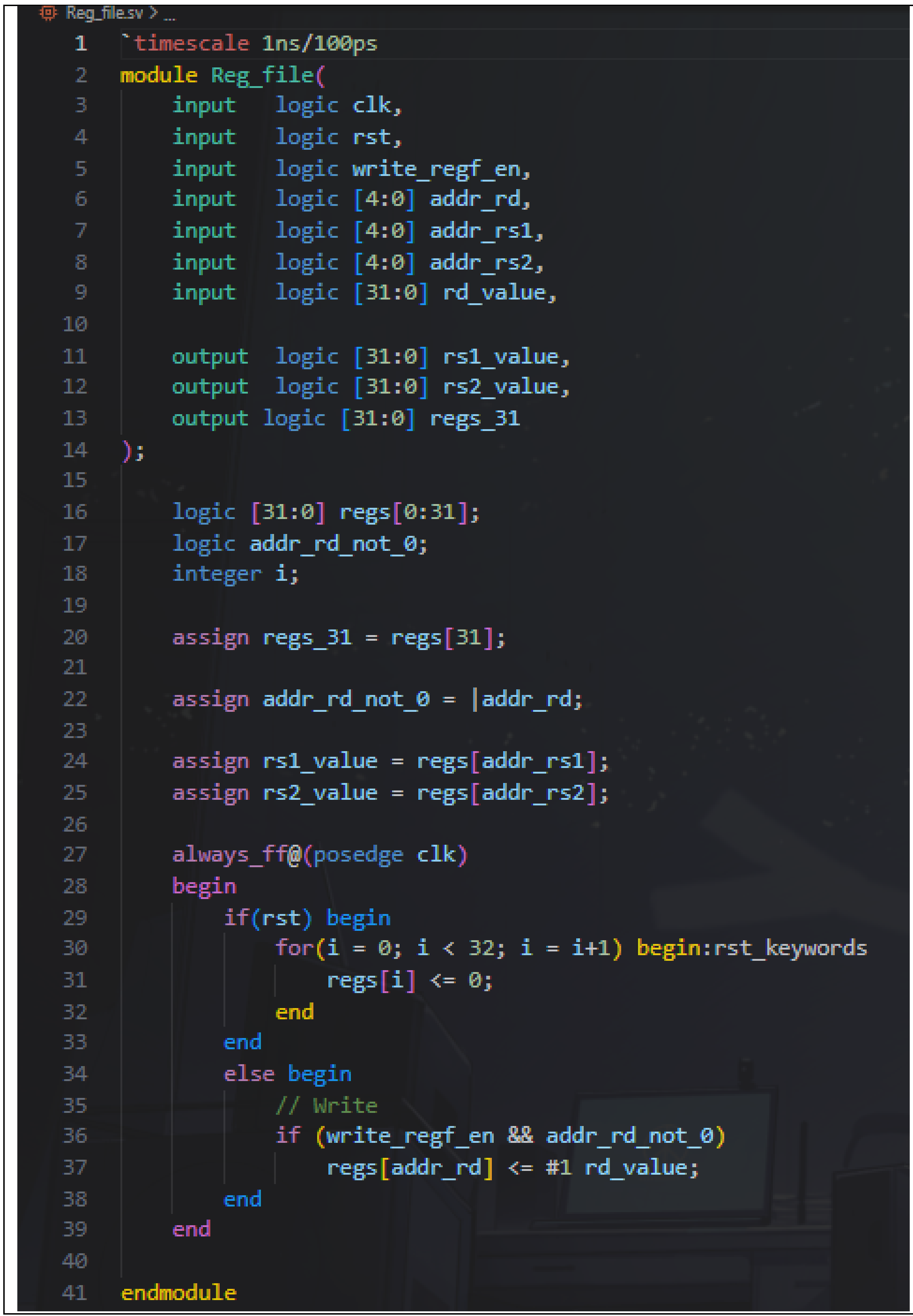
**Ram.sv**



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
```

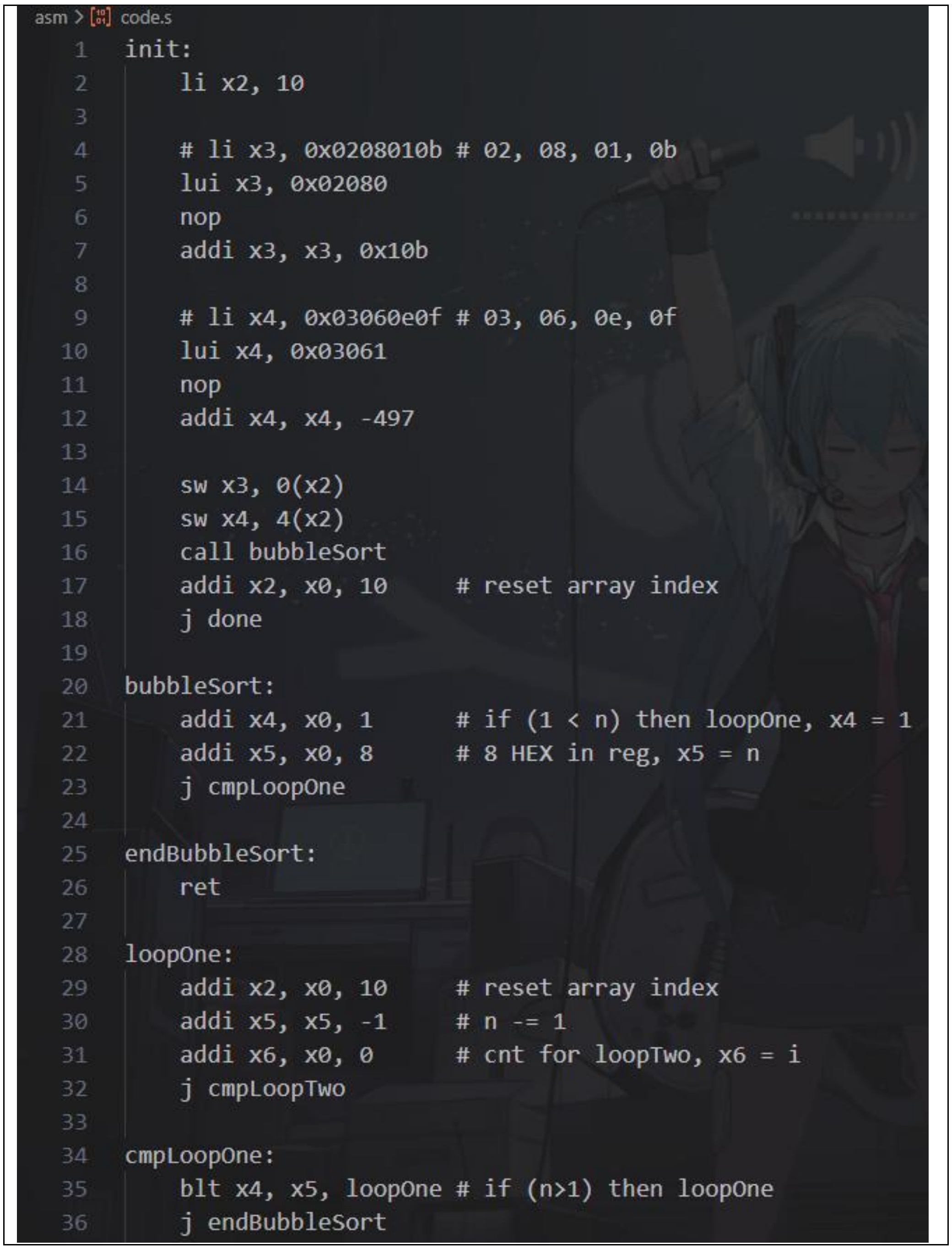
**Reg_file.sv**



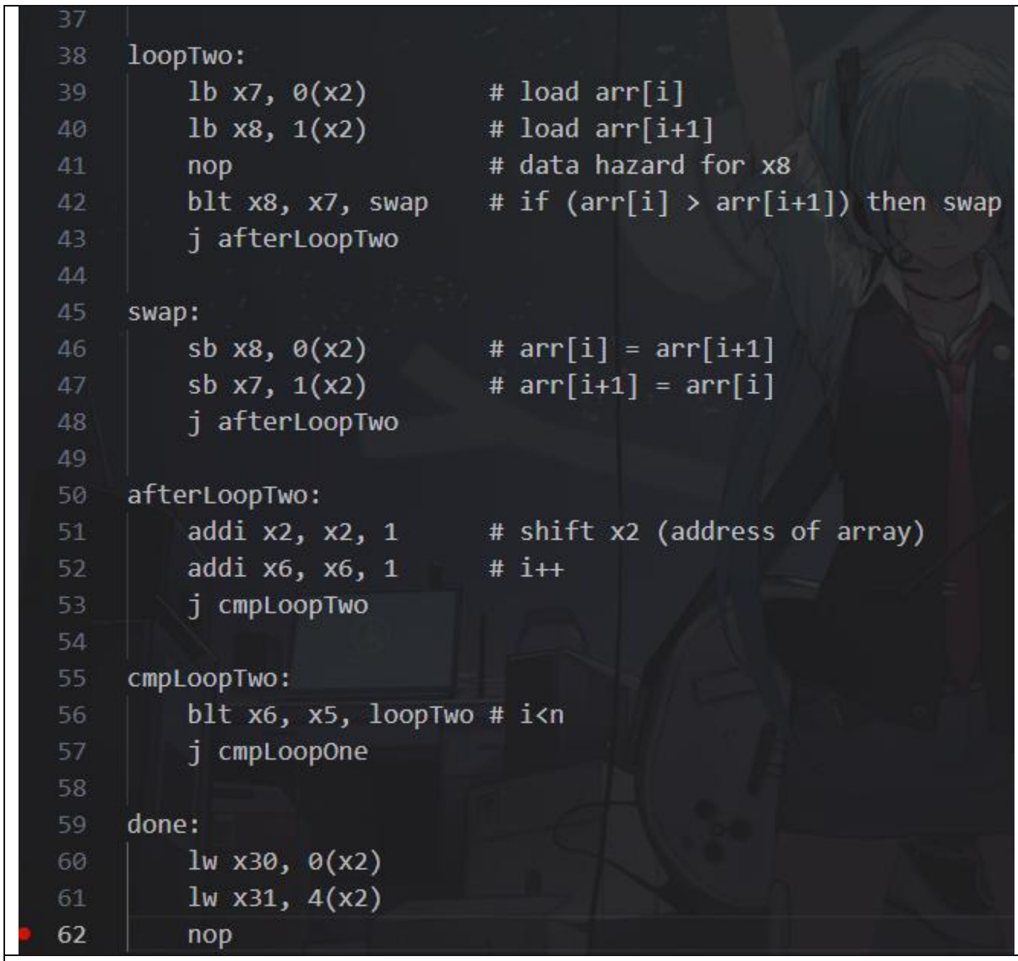
Reg_file.sv > ...

```
`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
```

**(第一題) Program_rom**

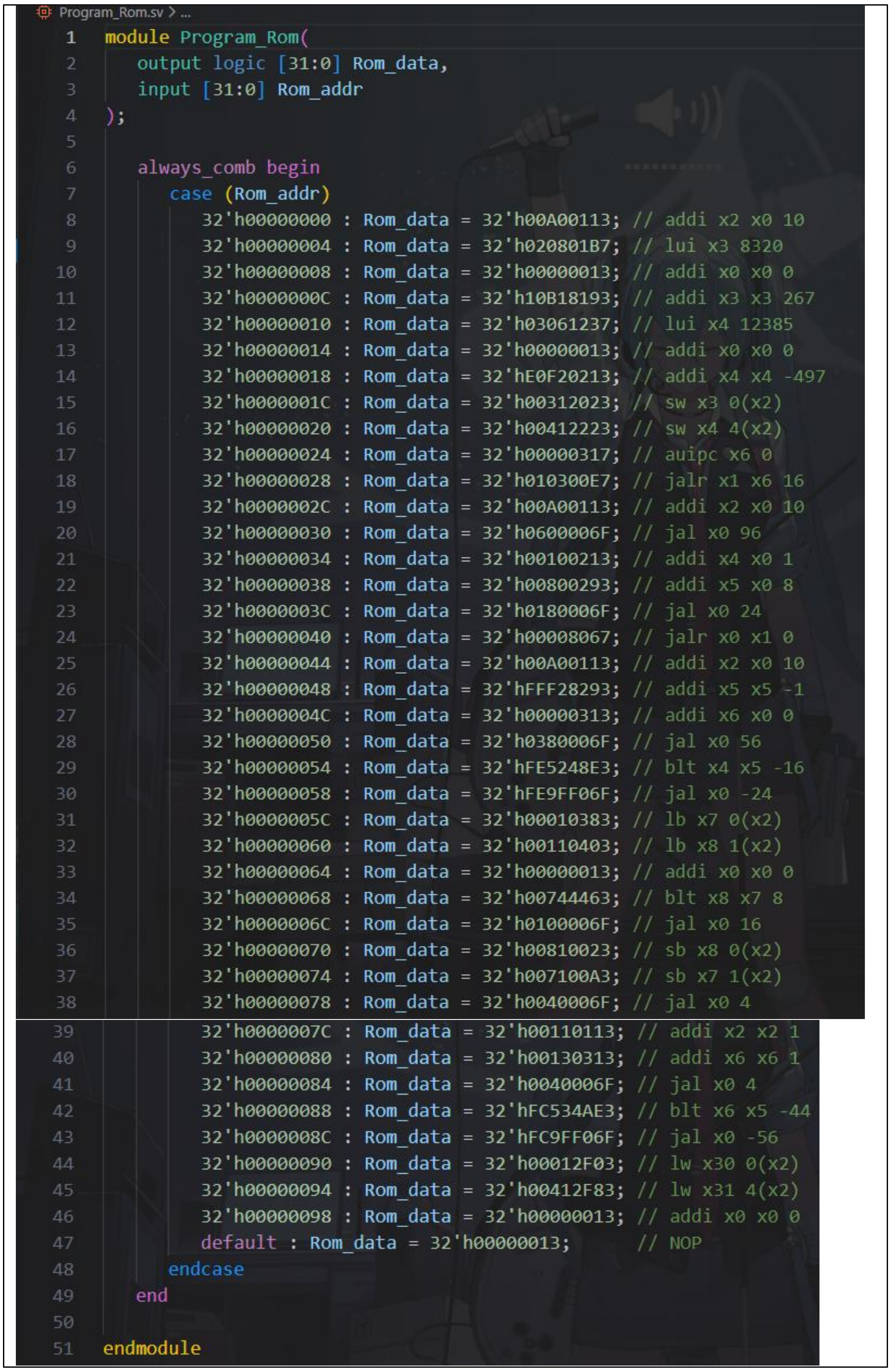

asm > code.s

```
init:
    li x2, 10

    # li x3, 0x0208010b # 02, 08, 01, 0b
    lui x3, 0x02080
    nop
    addi x3, x3, 0x10b

    # li x4, 0x03060e0f # 03, 06, 0e, 0f
    lui x4, 0x03061
    nop
    addi x4, x4, -497

    sw x3, 0(x2)
    sw x4, 4(x2)
    call bubbleSort
    addi x2, x0, 10     # reset array index
    j done

bubbleSort:
    addi x4, x0, 1      # if (1 < n) then loopOne, x4 = 1
    addi x5, x0, 8      # 8 HEX in reg, x5 = n
    j cmpLoopOne

endBubbleSort:
    ret

loopOne:
    addi x2, x0, 10     # reset array index
    addi x5, x5, -1     # n -= 1
    addi x6, x0, 0      # cnt for loopTwo, x6 = i
    j cmpLoopTwo

cmpLoopOne:
    blt x4, x5, loopOne # if (n>1) then loopOne
    j endBubbleSort
```



```

loopTwo:
    lb x7, 0(x2)        # load arr[i]
    lb x8, 1(x2)        # load arr[i+1]
    nop                 # data hazard for x8
    blt x8, x7, swap    # if (arr[i] > arr[i+1]) then swap
    j afterLoopTwo

swap:
    sb x8, 0(x2)        # arr[i] = arr[i+1]
    sb x7, 1(x2)        # arr[i+1] = arr[i]
    j afterLoopTwo

afterLoopTwo:
    addi x2, x2, 1      # shift x2 (address of array)
    addi x6, x6, 1      # i++
    j cmpLoopTwo

cmpLoopTwo:
    blt x6, x5, loopTwo # i<n
    j cmpLoopOne

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

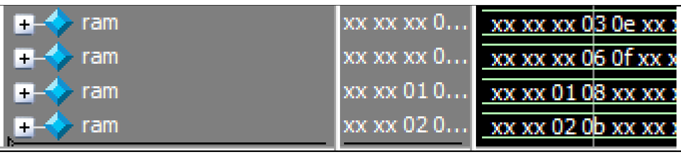
可以看到最開始 li 不能用，因為兩個步驟中出現 data hazard，我真的會謝：）



Program_Rom.sv > ...

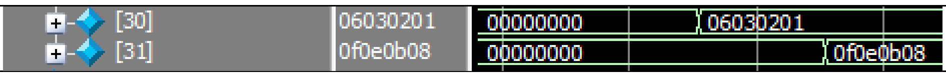
```
module Program_Rom(
   output logic [31:0] Rom_data,
   input [31:0] Rom_addr
);

   always_comb begin
      case (Rom_addr)
         32'h00000000 : Rom_data = 32'h00A00113; // addi x2 x0 10
         32'h00000004 : Rom_data = 32'h020801B7; // lui x3 8320
         32'h00000008 : Rom_data = 32'h00000013; // addi x0 x0 0
         32'h0000000C : Rom_data = 32'h10B18193; // addi x3 x3 267
         32'h00000010 : Rom_data = 32'h03061237; // lui x4 12385
         32'h00000014 : Rom_data = 32'h00000013; // addi x0 x0 0
         32'h00000018 : Rom_data = 32'hE0F20213; // addi x4 x4 -497
         32'h0000001C : Rom_data = 32'h00312023; // sw x3 0(x2)
         32'h00000020 : Rom_data = 32'h00412223; // sw x4 4(x2)
         32'h00000024 : Rom_data = 32'h00000317; // auipc x6 0
         32'h00000028 : Rom_data = 32'h010300E7; // jalr x1 x6 16
         32'h0000002C : Rom_data = 32'h00A00113; // addi x2 x0 10
         32'h00000030 : Rom_data = 32'h0600006F; // jal x0 96
         32'h00000034 : Rom_data = 32'h00100213; // addi x4 x0 1
         32'h00000038 : Rom_data = 32'h00800293; // addi x5 x0 8
         32'h0000003C : Rom_data = 32'h0180006F; // jal x0 24
         32'h00000040 : Rom_data = 32'h00008067; // jalr x0 x1 0
         32'h00000044 : Rom_data = 32'h00A00113; // addi x2 x0 10
         32'h00000048 : Rom_data = 32'hFFF28293; // addi x5 x5 -1
         32'h0000004C : Rom_data = 32'h00000313; // addi x6 x0 0
         32'h00000050 : Rom_data = 32'h0380006F; // jal x0 56
         32'h00000054 : Rom_data = 32'hFE5248E3; // blt x4 x5 -16
         32'h00000058 : Rom_data = 32'hFE9FF06F; // jal x0 -24
         32'h0000005C : Rom_data = 32'h00010383; // lb x7 0(x2)
         32'h00000060 : Rom_data = 32'h00110403; // lb x8 1(x2)
         32'h00000064 : Rom_data = 32'h00000013; // addi x0 x0 0
         32'h00000068 : Rom_data = 32'h00744463; // blt x8 x7 8
         32'h0000006C : Rom_data = 32'h0100006F; // jal x0 16
         32'h00000070 : Rom_data = 32'h00810023; // sb x8 0(x2)
         32'h00000074 : Rom_data = 32'h007100A3; // sb x7 1(x2)
         32'h00000078 : Rom_data = 32'h0040006F; // jal x0 4
         32'h0000007C : Rom_data = 32'h00110113; // addi x2 x2 1
         32'h00000080 : Rom_data = 32'h00130313; // addi x6 x6 1
         32'h00000084 : Rom_data = 32'h0040006F; // jal x0 4
         32'h00000088 : Rom_data = 32'hFC534AE3; // blt x6 x5 -44
         32'h0000008C : Rom_data = 32'hFC9FF06F; // jal x0 -56
         32'h00000090 : Rom_data = 32'h00012F03; // lw x30 0(x2)
         32'h00000094 : Rom_data = 32'h00412F83; // lw x31 4(x2)
         32'h00000098 : Rom_data = 32'h00000013; // addi x0 x0 0
         default : Rom_data = 32'h00000013;       // NOP
      endcase
   end

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—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 0x00000e0f) 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。

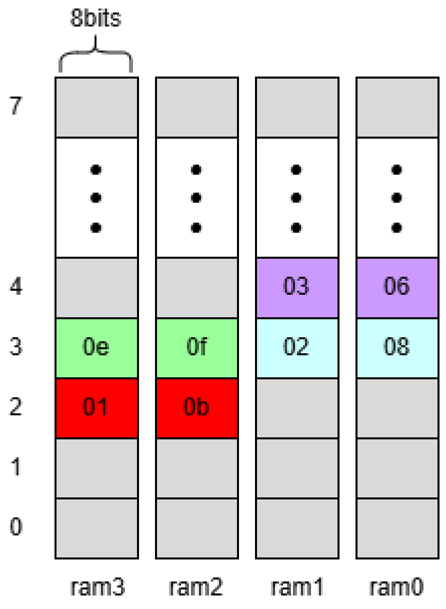
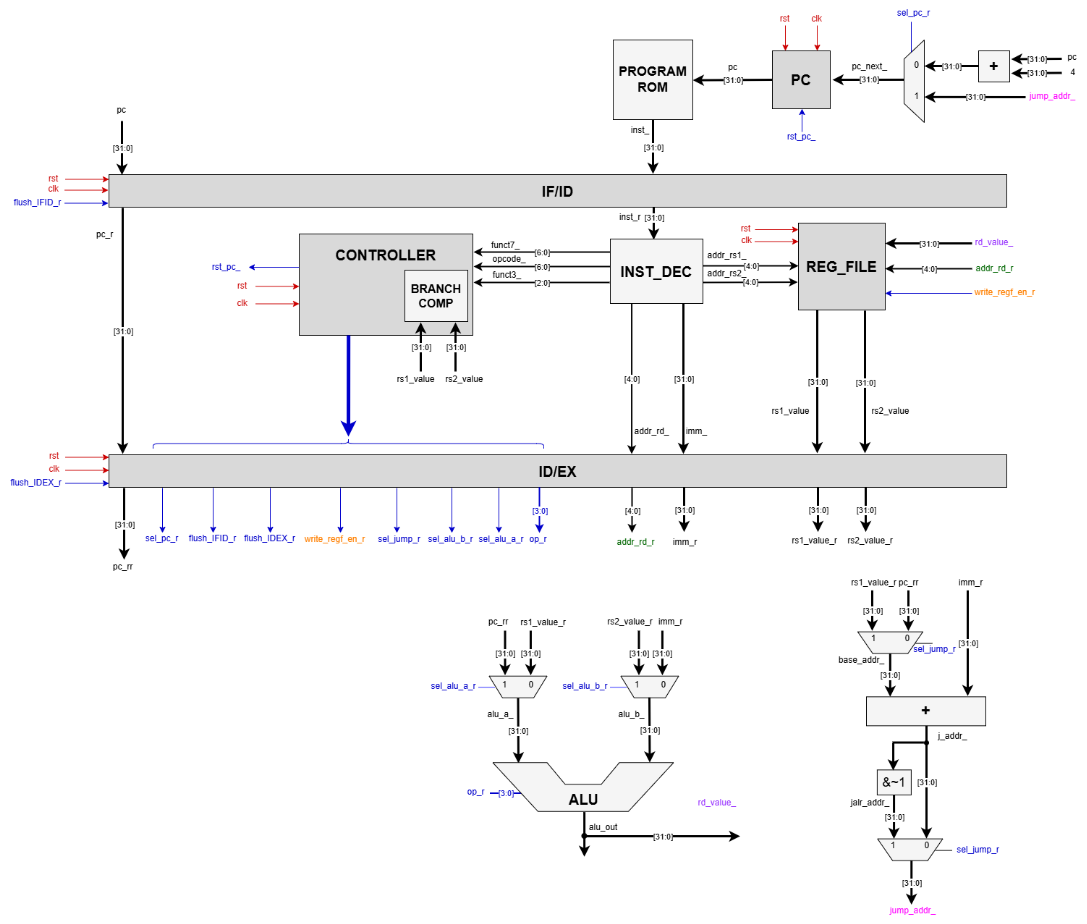


圖 3

```
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--------
```

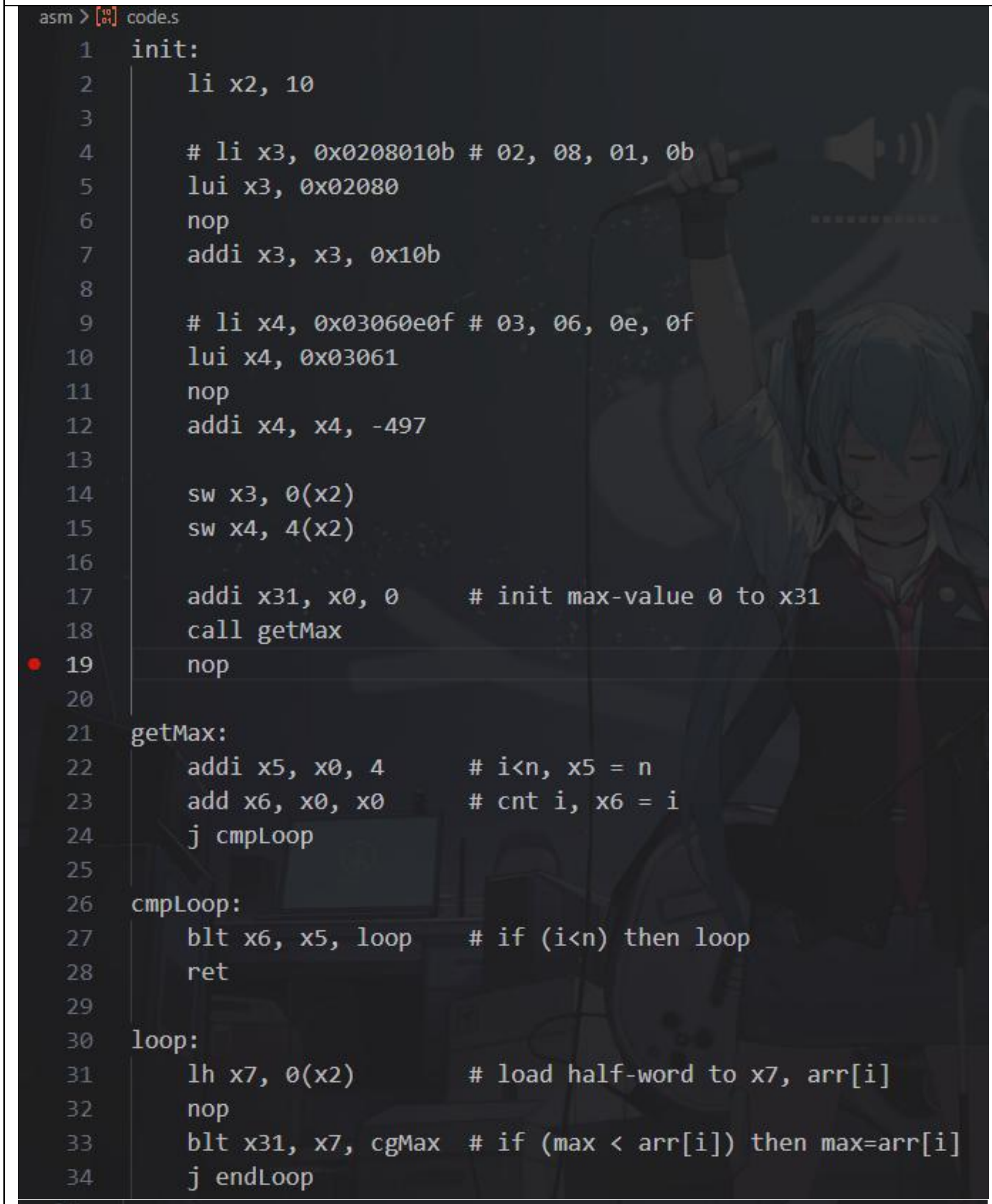
圖 4

## ●系統硬體架構方塊圖（接線圖）：

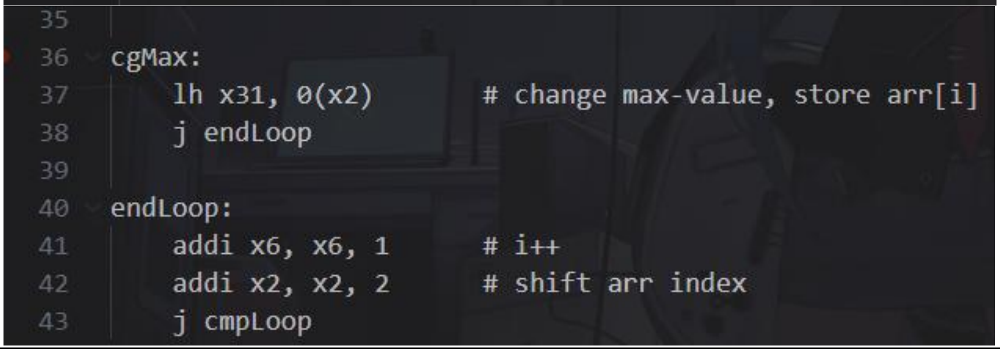


●系統架構程式碼、測試資料程式碼與程式碼說明
截圖請善用 **win+shift+S**

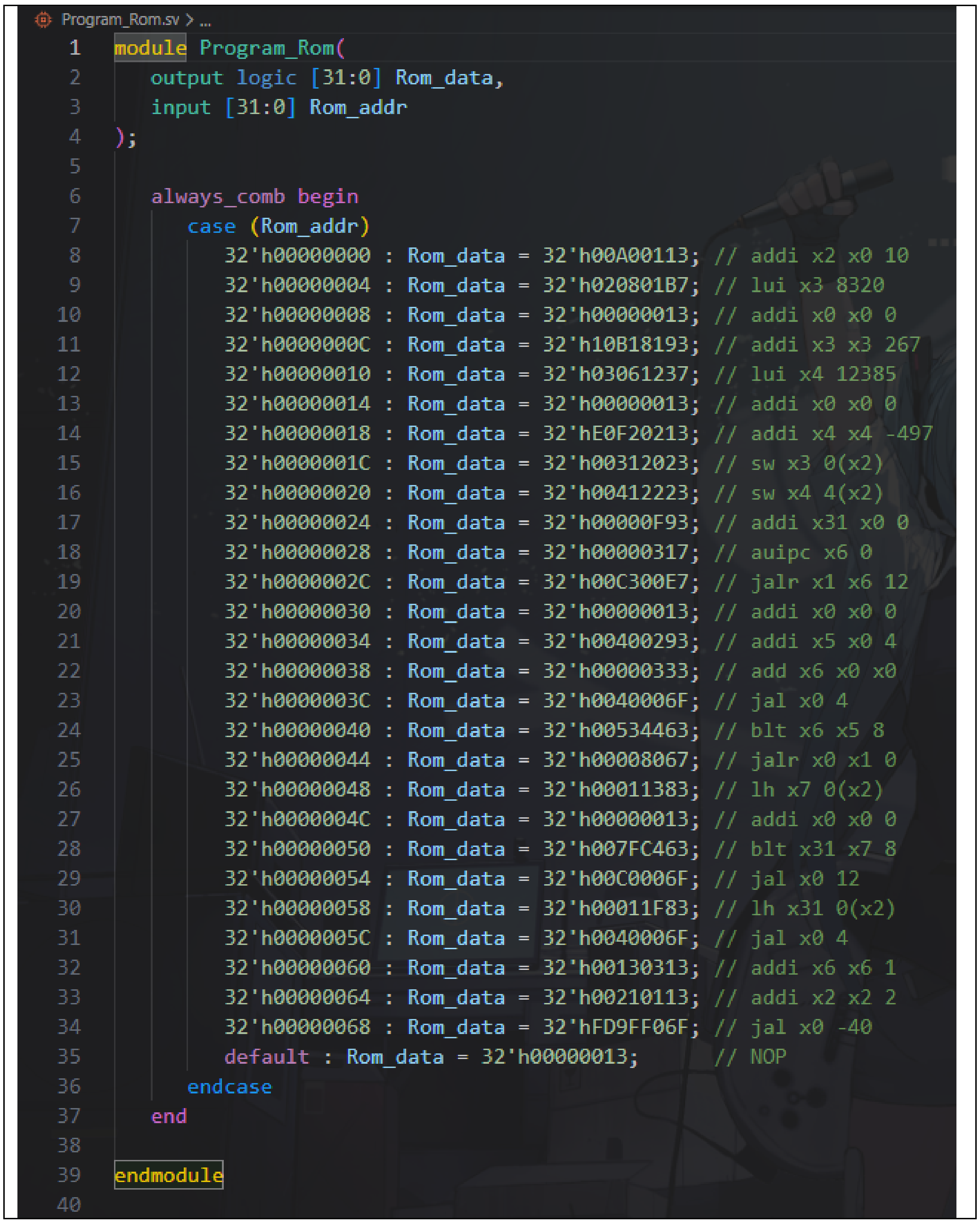
Program_rom.sv



```
init:
    li x2, 10

    # li x3, 0x0208010b # 02, 08, 01, 0b
    lui x3, 0x02080
    nop
    addi x3, x3, 0x10b

    # li x4, 0x03060e0f # 03, 06, 0e, 0f
    lui x4, 0x03061
    nop
    addi x4, x4, -497

    sw x3, 0(x2)
    sw x4, 4(x2)

    addi x31, x0, 0     # init max-value 0 to x31
    call getMax
    nop

getMax:
    addi x5, x0, 4      # i<n, x5 = n
    add x6, x0, x0      # cnt i, x6 = i
    j cmpLoop

cmpLoop:
    blt x6, x5, loop    # if (i<n) then loop
    ret

loop:
    lh x7, 0(x2)        # load half-word to x7, arr[i]
    nop
    blt x31, x7, cgMax  # if (max < arr[i]) then max=arr[i]
    j endLoop
```



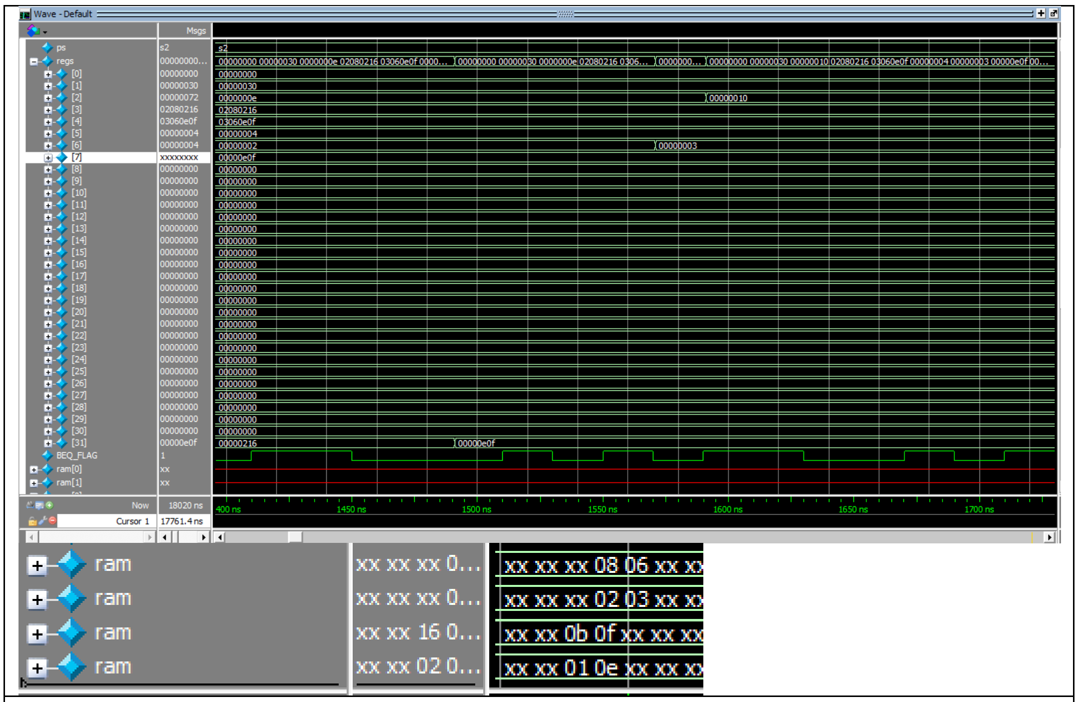
```

cgMax:
    lh x31, 0(x2)       # change max-value, store arr[i]
    j endLoop

endLoop:
    addi x6, x6, 1      # i++
    addi x2, x2, 2      # shift arr index
    j cmpLoop
```



Program_Rom.sv > ...

```
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'h00A00113; // addi x2 x0 10
            32'h00000004 : Rom_data = 32'h020801B7; // lui x3 8320
            32'h00000008 : Rom_data = 32'h00000013; // addi x0 x0 0
            32'h0000000C : Rom_data = 32'h10B18193; // addi x3 x3 267
            32'h00000010 : Rom_data = 32'h03061237; // lui x4 12385
            32'h00000014 : Rom_data = 32'h00000013; // addi x0 x0 0
            32'h00000018 : Rom_data = 32'hE0F20213; // addi x4 x4 -497
            32'h0000001C : Rom_data = 32'h00312023; // sw x3 0(x2)
            32'h00000020 : Rom_data = 32'h00412223; // sw x4 4(x2)
            32'h00000024 : Rom_data = 32'h00000F93; // addi x31 x0 0
            32'h00000028 : Rom_data = 32'h00000317; // auipc x6 0
            32'h0000002C : Rom_data = 32'h00C300E7; // jalr x1 x6 12
            32'h00000030 : Rom_data = 32'h00000013; // addi x0 x0 0
            32'h00000034 : Rom_data = 32'h00400293; // addi x5 x0 4
            32'h00000038 : Rom_data = 32'h00000333; // add x6 x0 x0
            32'h0000003C : Rom_data = 32'h0040006F; // jal x0 4
            32'h00000040 : Rom_data = 32'h00534463; // blt x6 x5 8
            32'h00000044 : Rom_data = 32'h00008067; // jalr x0 x1 0
            32'h00000048 : Rom_data = 32'h00011383; // lh x7 0(x2)
            32'h0000004C : Rom_data = 32'h00000013; // addi x0 x0 0
            32'h00000050 : Rom_data = 32'h007FC463; // blt x31 x7 8
            32'h00000054 : Rom_data = 32'h00C0006F; // jal x0 12
            32'h00000058 : Rom_data = 32'h00011F83; // lh x31 0(x2)
            32'h0000005C : Rom_data = 32'h0040006F; // jal x0 4
            32'h00000060 : Rom_data = 32'h00130313; // addi x6 x6 1
            32'h00000064 : Rom_data = 32'h00210113; // addi x2 x2 2
            32'h00000068 : Rom_data = 32'hFD9FF06F; // jal x0 -40
            default : Rom_data = 32'h00000013;      // NOP
        endcase
    end

endmodule
```

●模擬結果與結果說明：



```
Wave - Default
                Msgs
ps              s2          s2
regs            00000000... 00000000 00000030 0000000e 02080216 03060e0f 0000... | 00000000 00000030 0000000e 02080216 0306... | 0000000... | 00000000 00000030 00000010 02080216 03060e0f 00000004 00000003 00000e0f 00...
 [0]            00000000    00000000
 [1]            00000030    00000030
 [2]            00000072    0000000e | 00000010
 [3]            02080216    02080216
 [4]            03060e0f    03060e0f
 [5]            00000004    00000004
 [6]            00000004    00000002 | 00000003
 [7]            xxxxxxxx    00000e0f
 [8]            00000000    00000000
 [9]            00000000    00000000
 [10]           00000000    00000000
 [11]           00000000    00000000
 [12]           00000000    00000000
 [13]           00000000    00000000
 [14]           00000000    00000000
 [15]           00000000    00000000
 [16]           00000000    00000000
 [17]           00000000    00000000
 [18]           00000000    00000000
 [19]           00000000    00000000
 [20]           00000000    00000000
 [21]           00000000    00000000
 [22]           00000000    00000000
 [23]           00000000    00000000
 [24]           00000000    00000000
 [25]           00000000    00000000
 [26]           00000000    00000000
 [27]           00000000    00000000
 [28]           00000000    00000000
 [29]           00000000    00000000
 [30]           00000000    00000000
 [31]           00000e0f    00000216 | 00000e0f
BEQ_FLAG        1
ram[0]          xx
ram[1]          xx
Now             18020 ns    400 ns   1450 ns   1500 ns   1550 ns   1600 ns   1650 ns   1700 ns
Cursor 1        17761.4 ns

ram             xx xx xx 0...   xx xx xx 08 06 xx xx
ram             xx xx xx 0...   xx xx xx 02 03 xx xx
ram             xx xx 16 0...   xx xx 0b 0f xx xx xx
ram             xx xx 02 0...   xx xx 01 0e xx xx xx
```

以上為此題結果，可以看到 x31 為最大的 half-word

## ●結論與心得：

這是一場博弈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？