2025/12/04

實驗十二

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

※注意

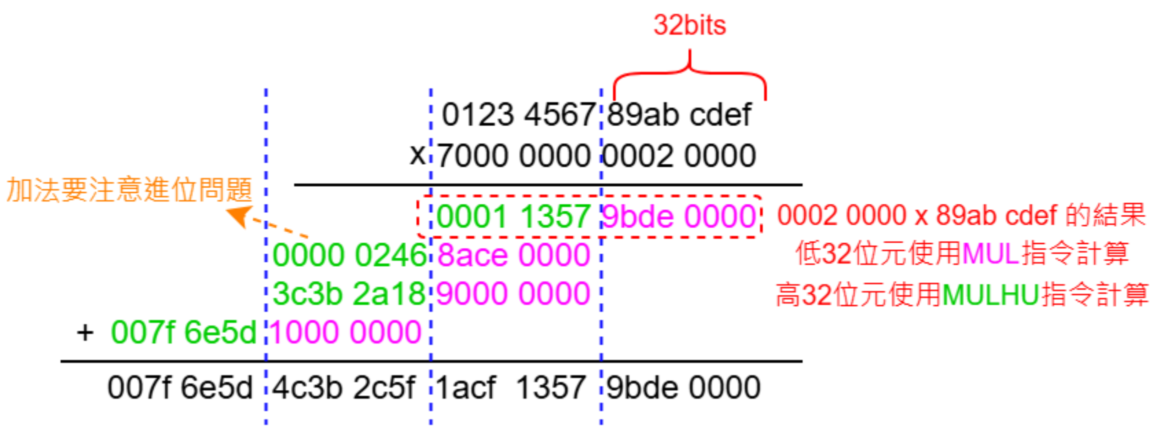
1. 繳交時一律轉 **PDF** 檔
2. 繳交期限為下周上課前
3. 一人繳交一份
4. 檔名請按照作業檔名格式進行填寫，未依照格式不予批改
5. 檔名範例：學號_姓名_HW12

1、 組語撰寫練習—64 位元乘法

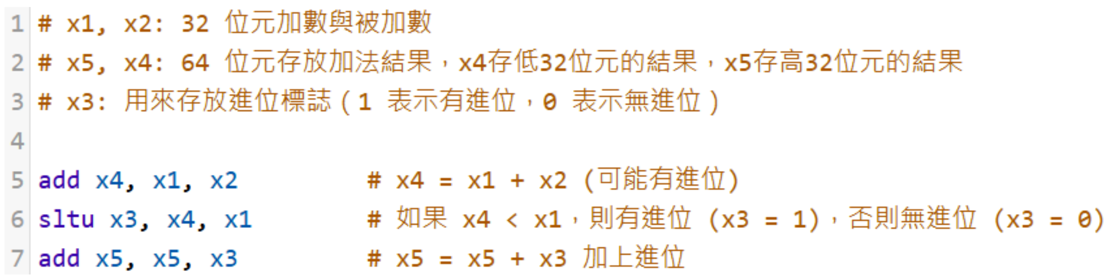
●實驗說明：

1. 請透過當前課堂所學的組合語言撰寫一個 64 位元的有號數乘法，計算 0x0123_4567_89ab_cdef 乘以 0x7000_0000_0002_0000 的結果，如下圖一所示。程式碼須加註解。
   $$0x1234\_567\_89ab\_cdef \times 0x7000\_0000\_0002\_0000$$
   $$= 0x007f\_6e5d\_4c3b\_2c5f\_1acf\_1357\_9bde\_0000$$
2. 64 位元的被乘數與乘數分別使用兩個暫存器儲存。
3. 在相乘前須先判斷被乘數與乘數是否為負數，若是則先取二補數。(Hint：較高的 32bits 取反向，較低的 32bits 反向加一，並判斷是否須進位)
4. 有無進位可以使用圖二中的組合語言來判斷，若'和'小於加數與被加數則表示有進位發生。

5. 最終計算出來的結果要依據乘數與被乘數的符號來決定是否要取二補數。
6. 所有暫存器皆可使用，但請將結果位置由小到大載入到 x31 ~ x28 暫存器中，如 0x9bde_0000 存入 x31。
7. 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。
8. 請將所有用到的暫存器都加入至波形中以方便觀察。
9. 請注意資料危障的問題。



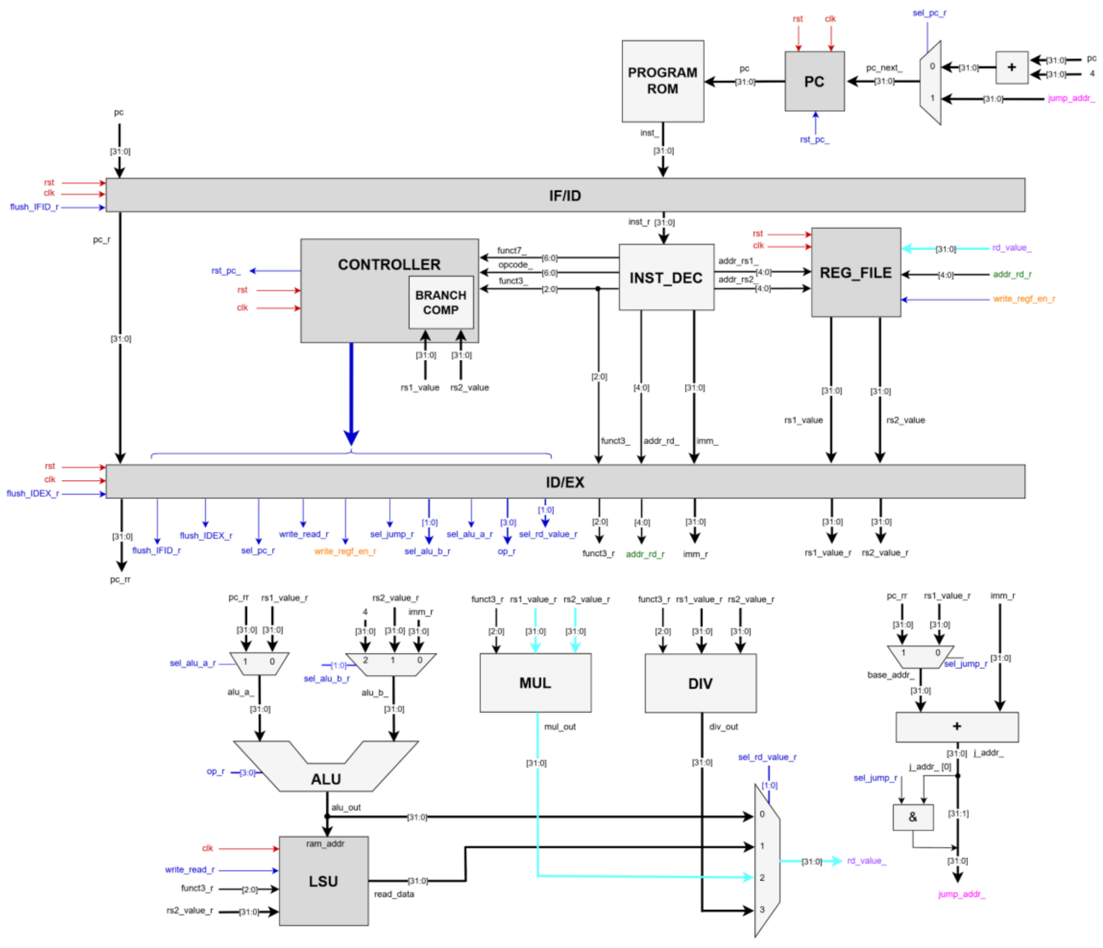
圖一



```
# x1, x2: 32 位元加數與被加數
# x5, x4: 64 位元存放加法結果，x4存低32位元的結果，x5存高32位元的結果
# x3: 用來存放進位標誌（1 表示有進位，0 表示無進位）

add x4, x1, x2          # x4 = x1 + x2 (可能有進位)
sltu x3, x4, x1         # 如果 x4 < x1，則有進位 (x3 = 1)，否則無進位 (x3 = 0)
add x5, x5, x3          # x5 = x5 + x3 加上進位
```

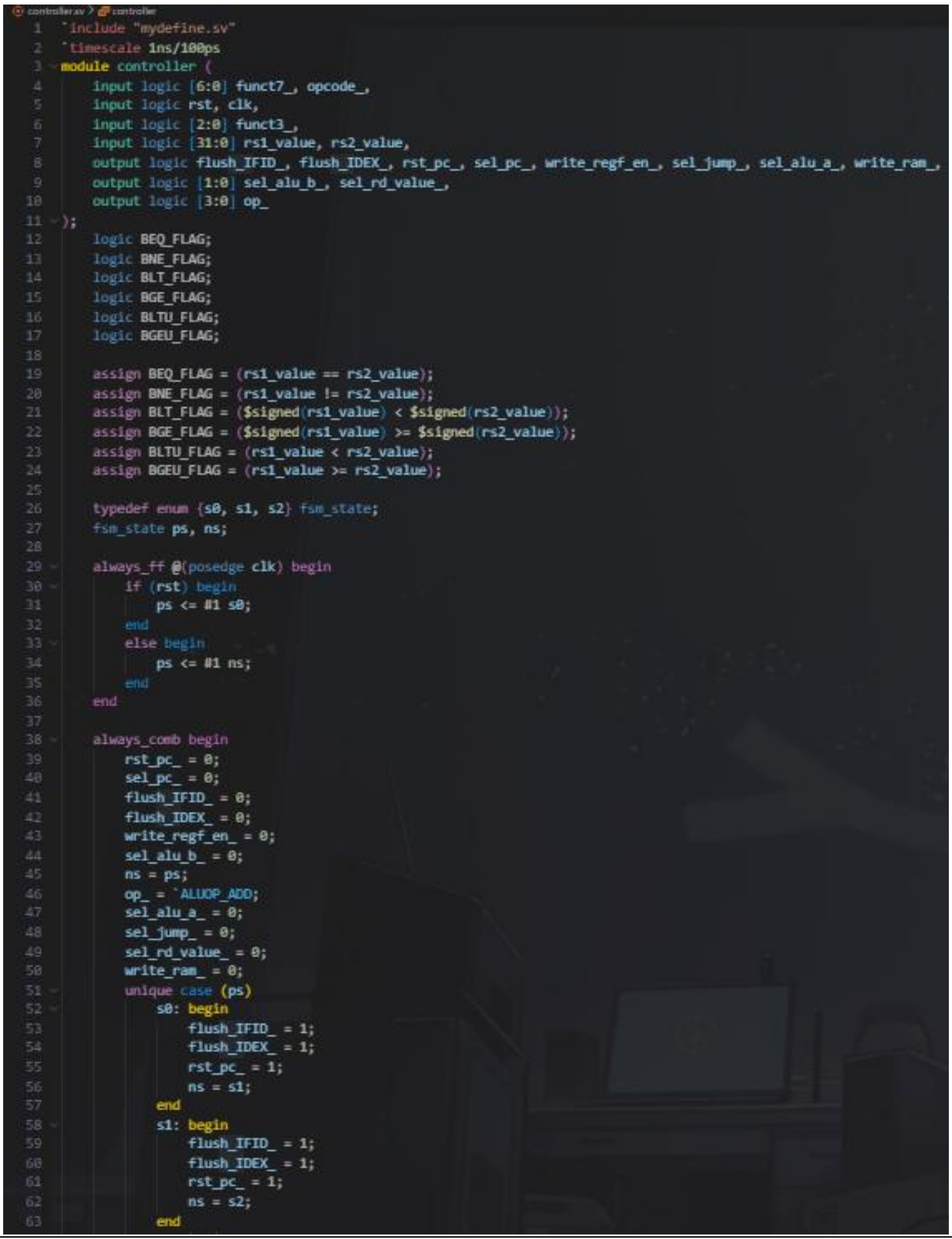
圖二

●系統硬體架構方塊圖（接線圖）：


PROGRAM ROM
PC
rst
clk
sel_pc_r
pc_next_
rst_pc_
pc
4
jump_addr_
inst_
IF/ID
flush_IFID_r
pc_r
inst_r
CONTROLLER
BRANCH COMP
INST_DEC
REG_FILE
funct7_
opcode_
funct3_
addr_rs1_
addr_rs2_
rd_value_
addr_rd_r
write_regf_en_r
rs1_value
rs2_value
funct3_
addr_rd_
imm_
ID/EX
flush_IDEX_r
flush_IFID_r
sel_pc_r
write_read_r
write_regf_en_r
sel_jump_r
sel_alu_b_r
sel_alu_a_r
op_r
sel_rd_value_r
funct3_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_
imm_r
ALU
alu_out
op_r
LSU
ram_addr
write_read_r
read_data
MUL
mul_out
DIV
div_out
sel_rd_value_r
rd_value_
base_addr_
sel_jump_r
j_addr_
j_addr_[0]
&
jump_addr_

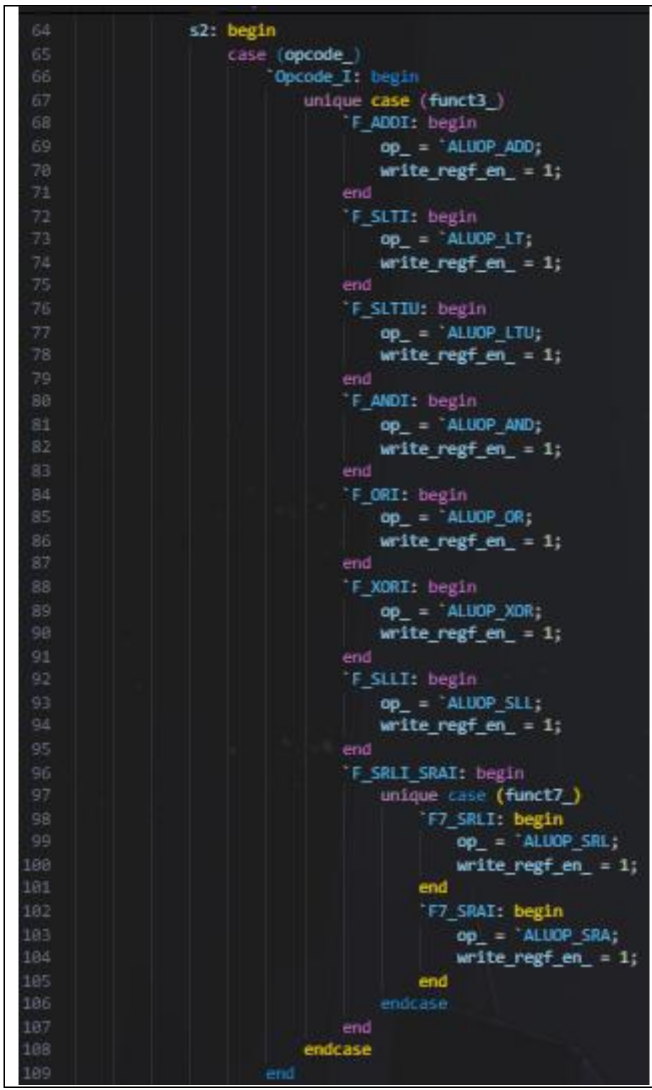
●系統架構程式碼、測試資料程式碼與程式碼說明
截圖請善用 **win+shift+S**

Controller.sv

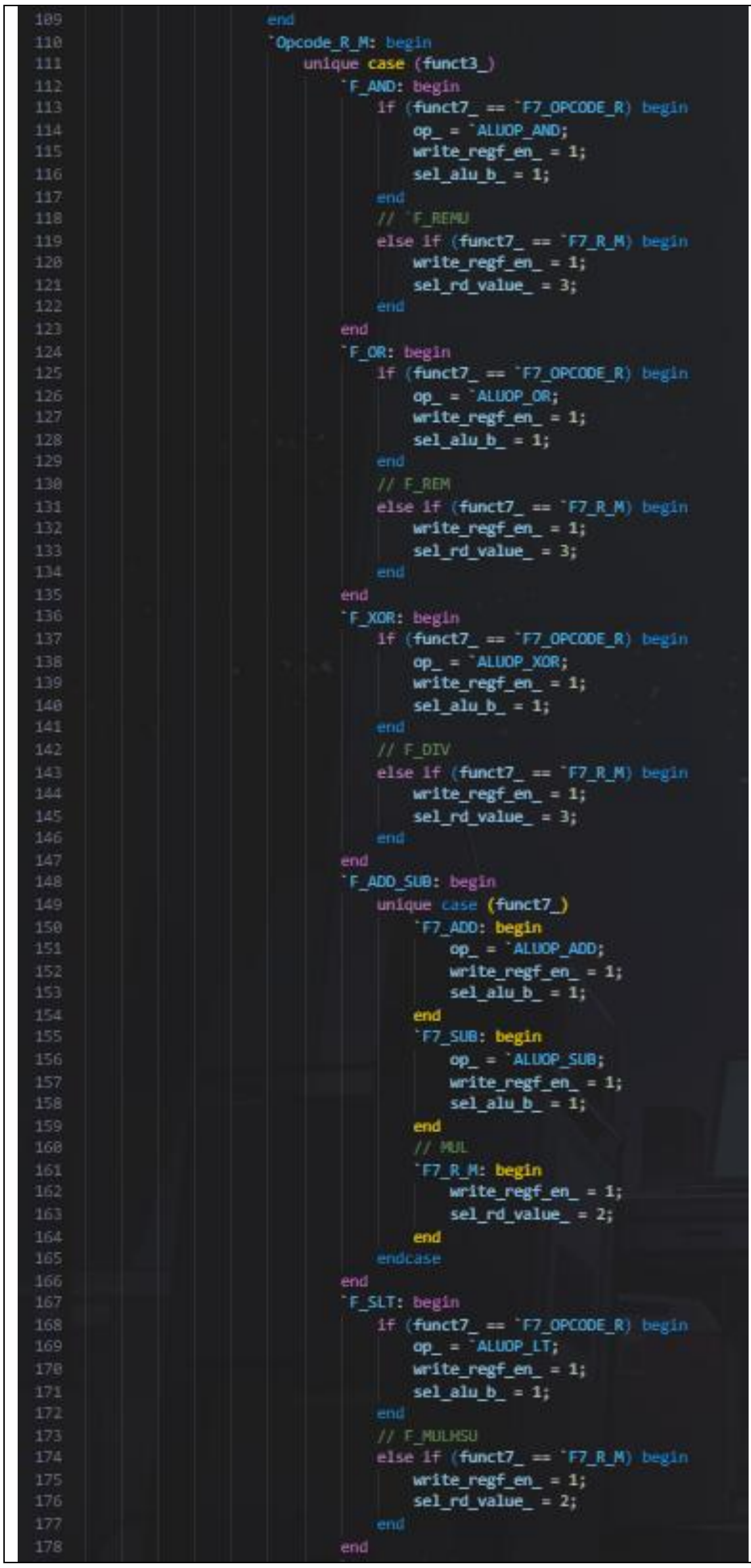


controller.sv > controller

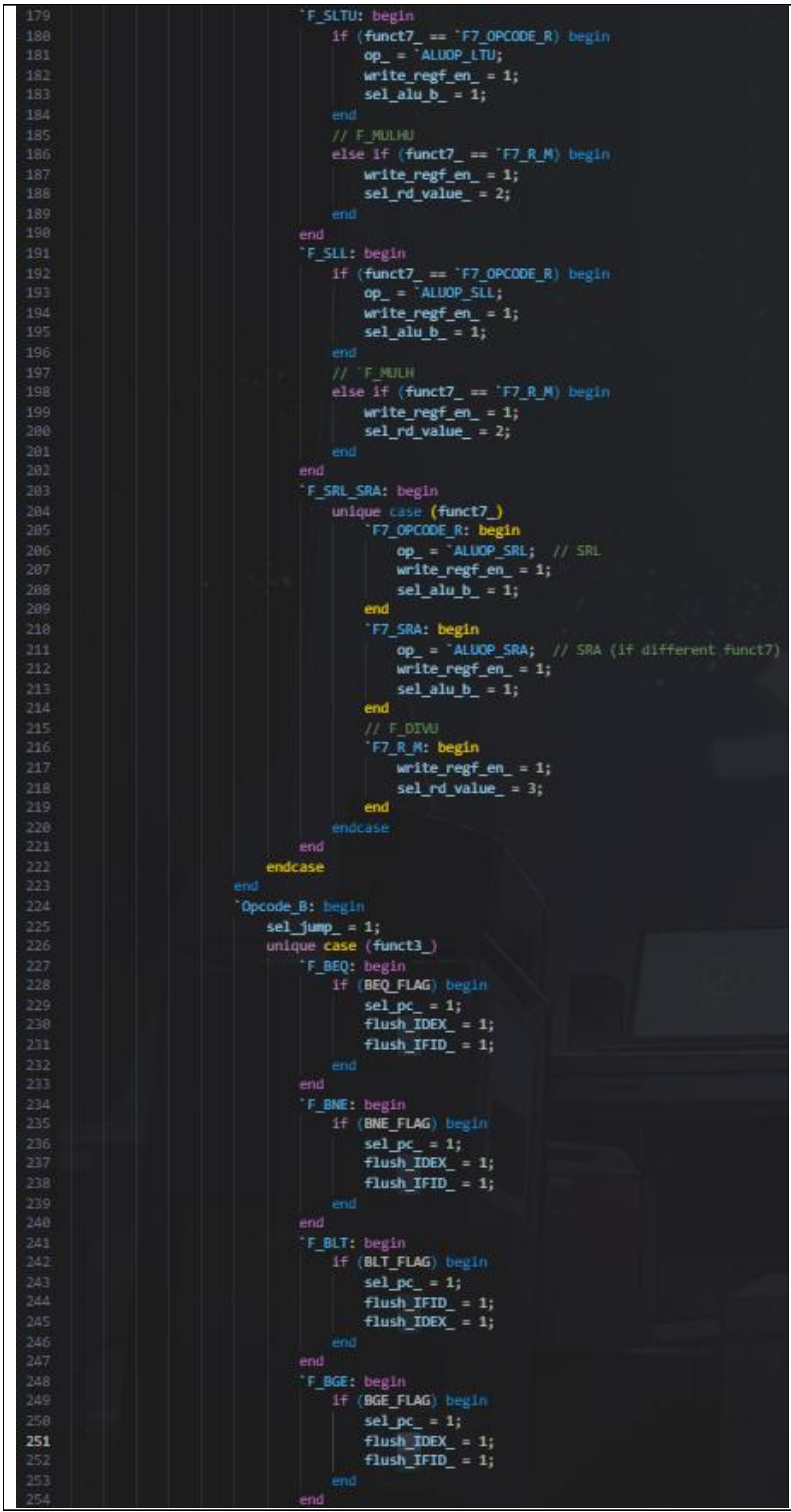
```
`include "mydefine.sv"
`timescale 1ns/100ps
module controller (
    input logic [6:0] funct7_, opcode_,
    input logic rst, clk,
    input logic [2:0] funct3_,
    input logic [31:0] rs1_value, rs2_value,
    output logic flush_IFID_, flush_IDEX_, rst_pc_, sel_pc_, write_regf_en_, sel_jump_, sel_alu_a_,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



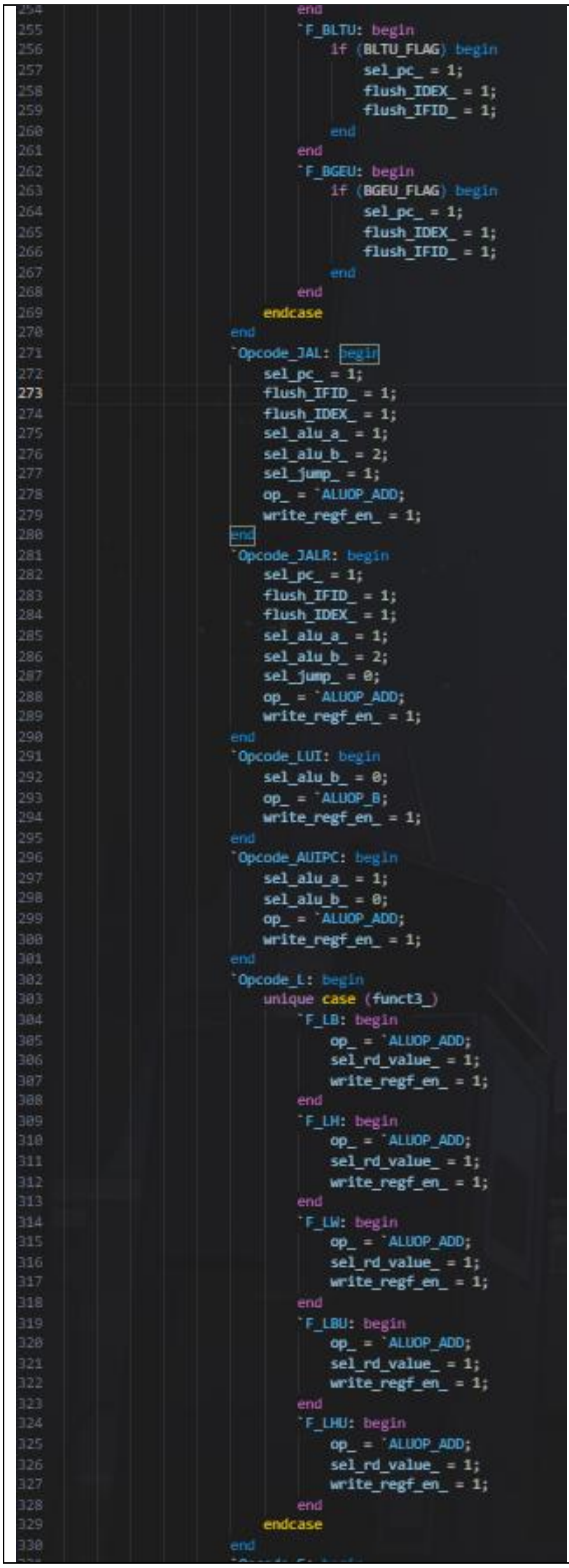
```
            s2: begin
                case (opcode_)
                    `Opcode_I: begin
                        unique case (funct3_)
                            `F_ADDI: begin
                                op_ = `ALUOP_ADD;
                                write_regf_en_ = 1;
                            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
```



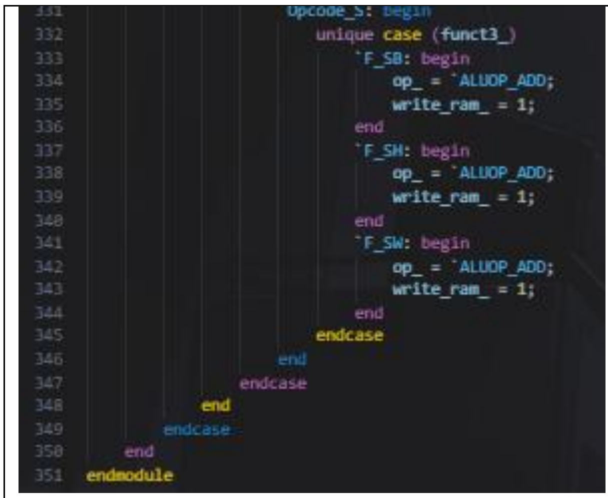
```
                    end
                    `Opcode_R_M: begin
                        unique case (funct3_)
                            `F_AND: begin
                                if (funct7_ == `F7_OPCODE_R) begin
                                    op_ = `ALUOP_AND;
                                    write_regf_en_ = 1;
                                    sel_alu_b_ = 1;
                                end
                                // `F_REMU
                                else if (funct7_ == `F7_R_M) begin
                                    write_regf_en_ = 1;
                                    sel_rd_value_ = 3;
                                end
                            end
                            `F_OR: begin
                                if (funct7_ == `F7_OPCODE_R) begin
                                    op_ = `ALUOP_OR;
                                    write_regf_en_ = 1;
                                    sel_alu_b_ = 1;
                                end
                                // F_REM
                                else if (funct7_ == `F7_R_M) begin
                                    write_regf_en_ = 1;
                                    sel_rd_value_ = 3;
                                end
                            end
                            `F_XOR: begin
                                if (funct7_ == `F7_OPCODE_R) begin
                                    op_ = `ALUOP_XOR;
                                    write_regf_en_ = 1;
                                    sel_alu_b_ = 1;
                                end
                                // F_DIV
                                else if (funct7_ == `F7_R_M) begin
                                    write_regf_en_ = 1;
                                    sel_rd_value_ = 3;
                                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
                                    // MUL
                                    `F7_R_M: begin
                                        write_regf_en_ = 1;
                                        sel_rd_value_ = 2;
                                    end
                                endcase
                            end
                            `F_SLT: begin
                                if (funct7_ == `F7_OPCODE_R) begin
                                    op_ = `ALUOP_LT;
                                    write_regf_en_ = 1;
                                    sel_alu_b_ = 1;
                                end
                                // F_MULHSU
                                else if (funct7_ == `F7_R_M) begin
                                    write_regf_en_ = 1;
                                    sel_rd_value_ = 2;
                                end
                            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
```

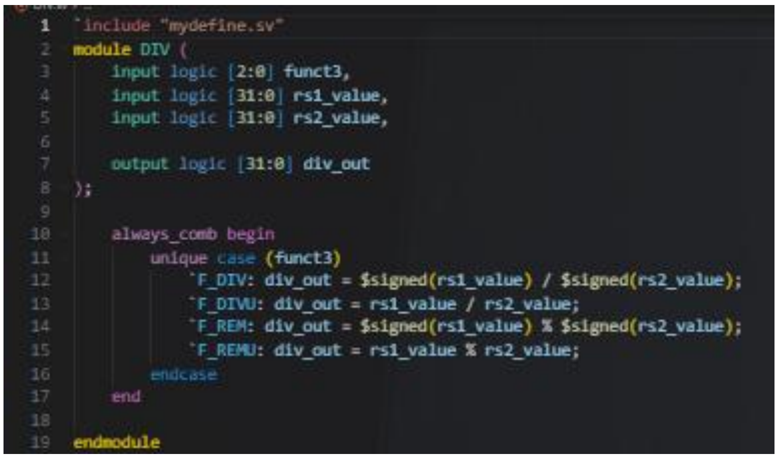


```
                                end
                                `F_BLTU: begin
                                    if (BLTU_FLAG) begin
                                        sel_pc_ = 1;
                                        flush_IDEX_ = 1;
                                        flush_IFID_ = 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
```



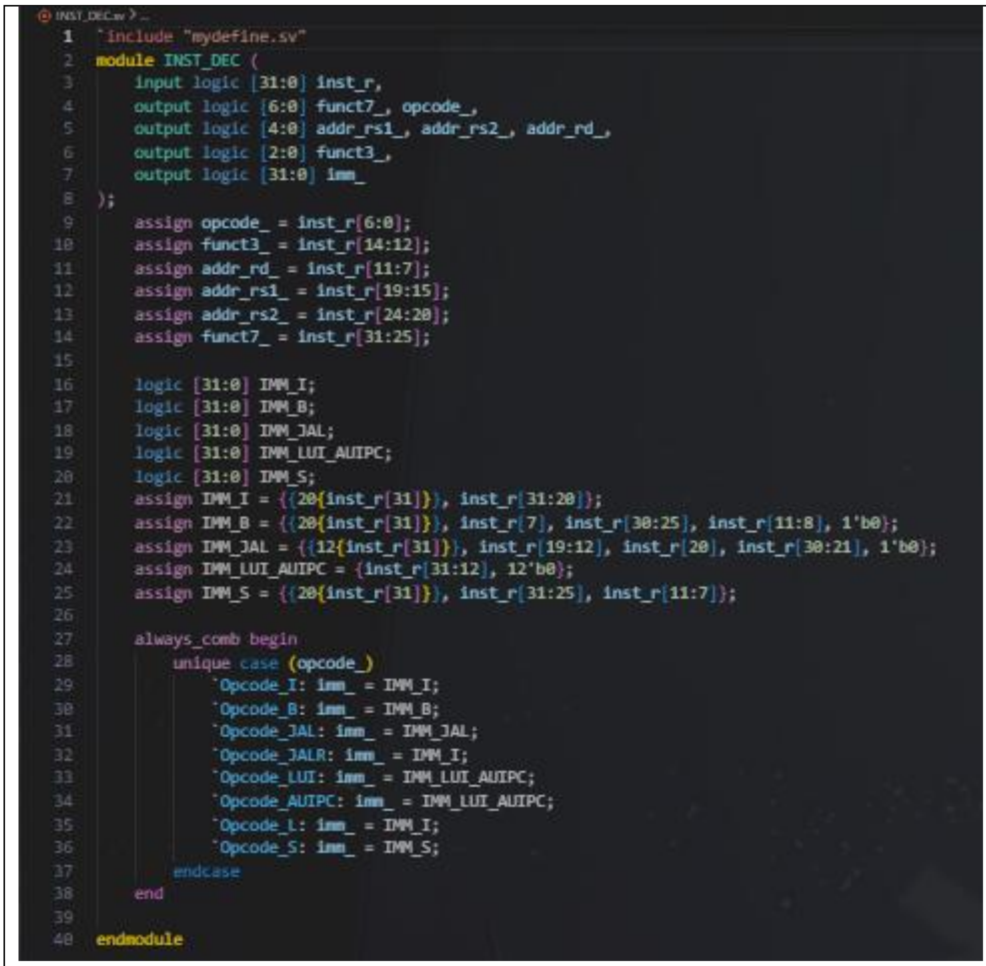
```
                    `Opcode_S: begin
                        unique case (funct3_)
                            `F_SB: begin
                                op_ = `ALUOP_ADD;
                                write_ram_ = 1;
                            end
                            `F_SH: begin
 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

div.sv



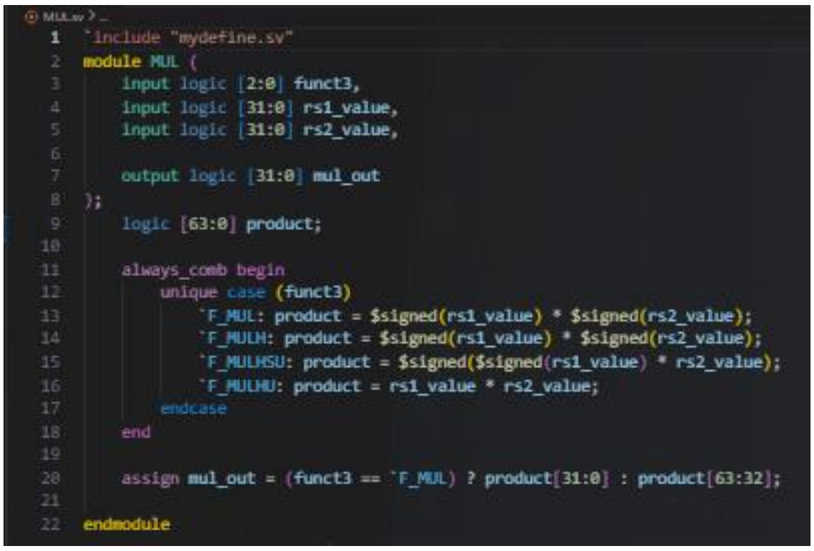
```
`include "mydefine.sv"
module DIV (
    input logic [2:0] funct3,
    input logic [31:0] rs1_value,
    input logic [31:0] rs2_value,

    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



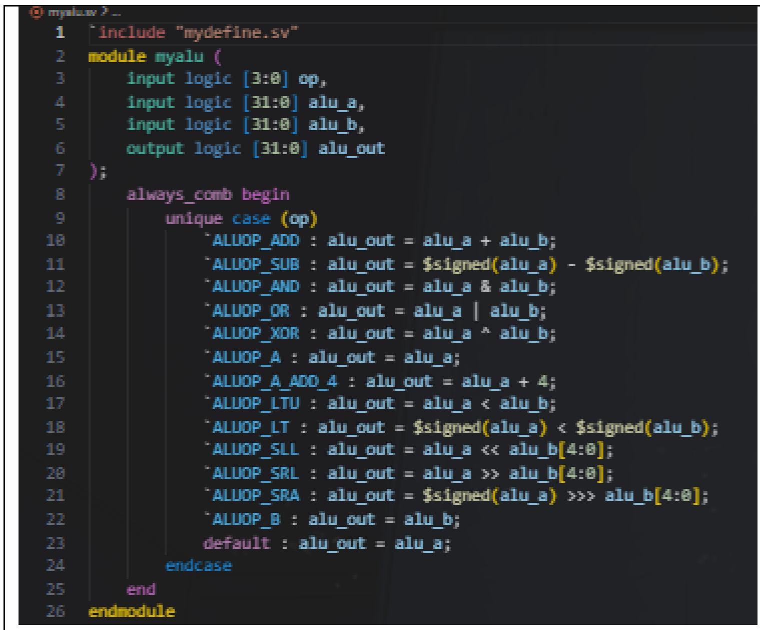
```
INST_DEC.sv > ...
`include "mydefine.sv"
module INST_DEC (
    input logic [31:0] inst_r,
    output logic [6:0] funct7_, opcode_,
    output logic [4:0] addr_rs1_, addr_rs2_, addr_rd_,
    output logic [2:0] funct3_,
    output logic [31:0] imm_
);
    assign opcode_ = inst_r[6:0];
    assign funct3_ = 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



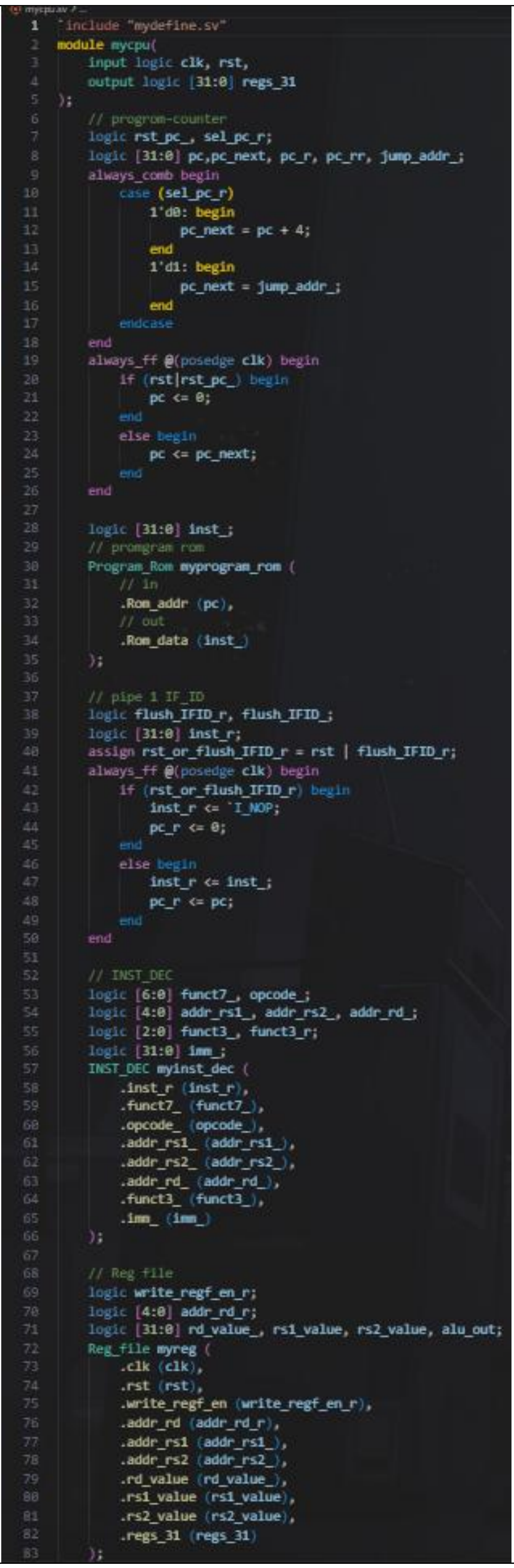
```
MUL.sv > ...
`include "mydefine.sv"
module MUL (
    input logic [2:0] funct3,
    input logic [31:0] rs1_value,
    input logic [31:0] rs2_value,

    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

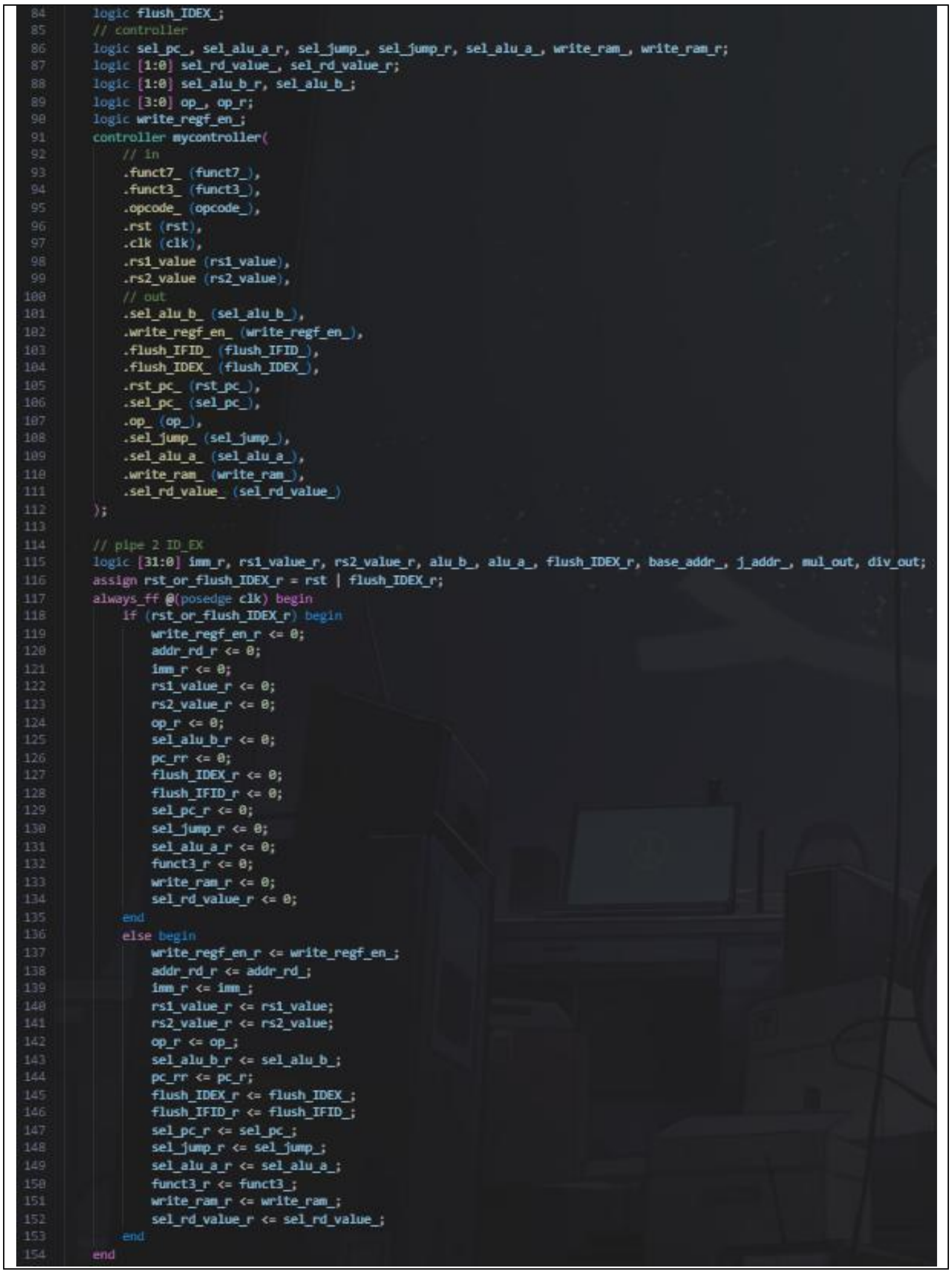


```
`include "mydefine.sv"
module myalu (
    input logic [3:0] op,
    input logic [31:0] alu_a,
    input logic [31:0] alu_b,
    output logic [31:0] alu_out
);
    always_comb begin
        unique case (op)
            `ALUOP_ADD : alu_out = alu_a + alu_b;
            `ALUOP_SUB : alu_out = $signed(alu_a) - $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
```

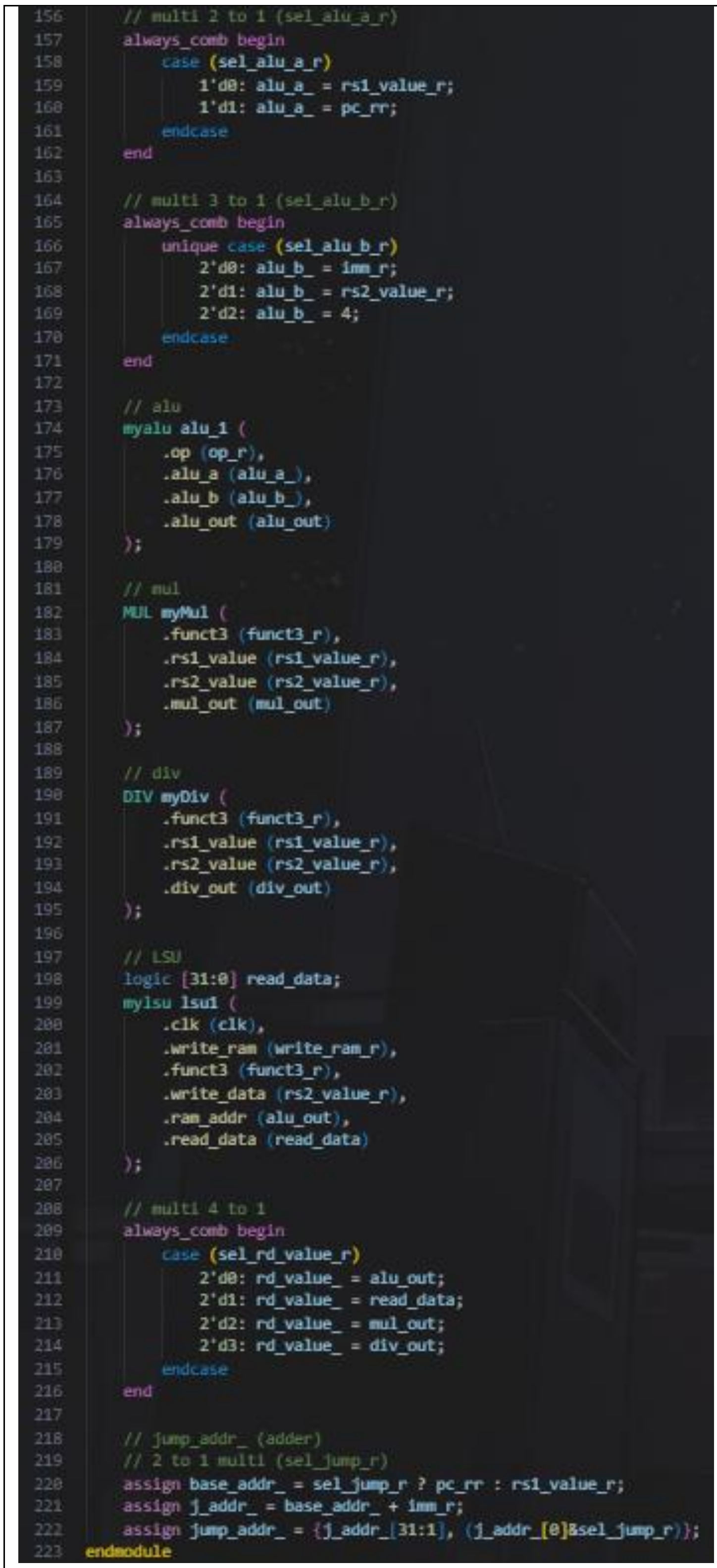
Mycpu.sv



```
`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)
    );
```

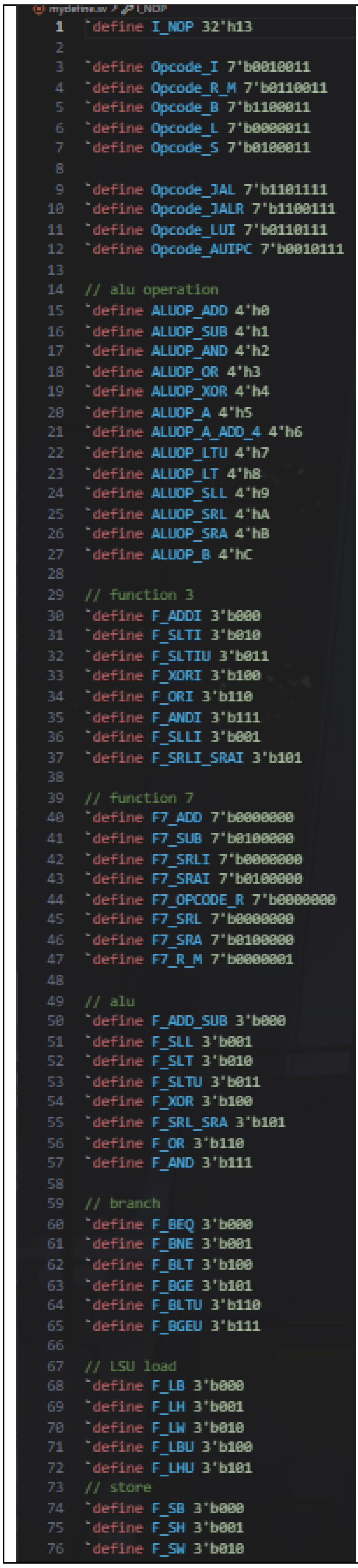


```
logic flush_IDEX_;
// controller
logic sel_pc_, sel_alu_a_r, sel_jump_, sel_jump_r, sel_alu_a_, write_ram_, write_ram_r;
logic [1:0] sel_rd_value_, sel_rd_value_r;
logic [1:0] sel_alu_b_r, sel_alu_b_;
logic [3:0] op_, op_r;
logic write_regf_en_;
controller mycontroller(
    // in
    .funct7_ (funct7_),
    .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_, mul_out, div_out;
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
```

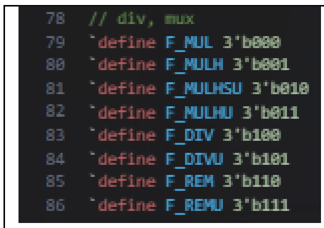


```
    // multi 2 to 1 (sel_alu_a_r)
    always_comb begin
        case (sel_alu_a_r)
            1'd0: alu_a_ = rs1_value_r;
            1'd1: alu_a_ = pc_rr;
        endcase
    end

    // 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)
    );

    // mul
    MUL myMul (
        .funct3 (funct3_r),
        .rs1_value (rs1_value_r),
        .rs2_value (rs2_value_r),
        .mul_out (mul_out)
    );

    // div
    DIV myDiv (
        .funct3 (funct3_r),
        .rs1_value (rs1_value_r),
        .rs2_value (rs2_value_r),
        .div_out (div_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)
    );

    // multi 4 to 1
    always_comb begin
        case (sel_rd_value_r)
            2'd0: rd_value_ = alu_out;
            2'd1: rd_value_ = read_data;
            2'd2: rd_value_ = mul_out;
            2'd3: rd_value_ = div_out;
        endcase
    end

    // 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
```

myDefine.sv

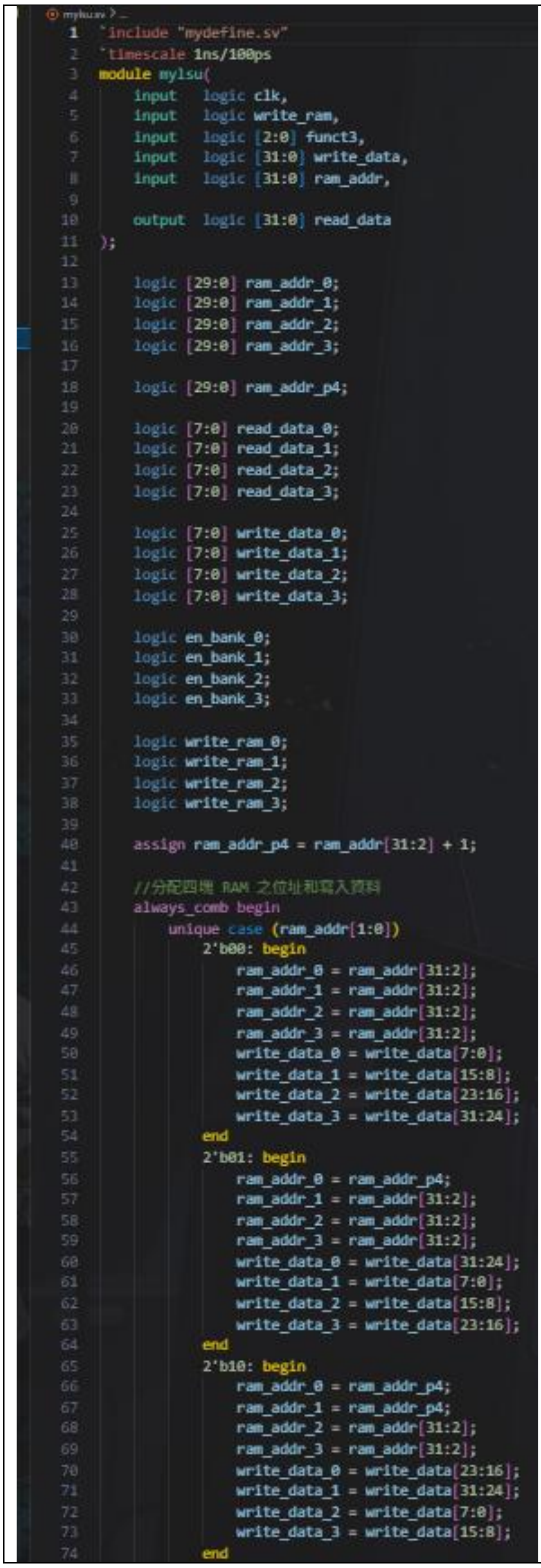


```verilog
`define I_NOP 32'h13

`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
```



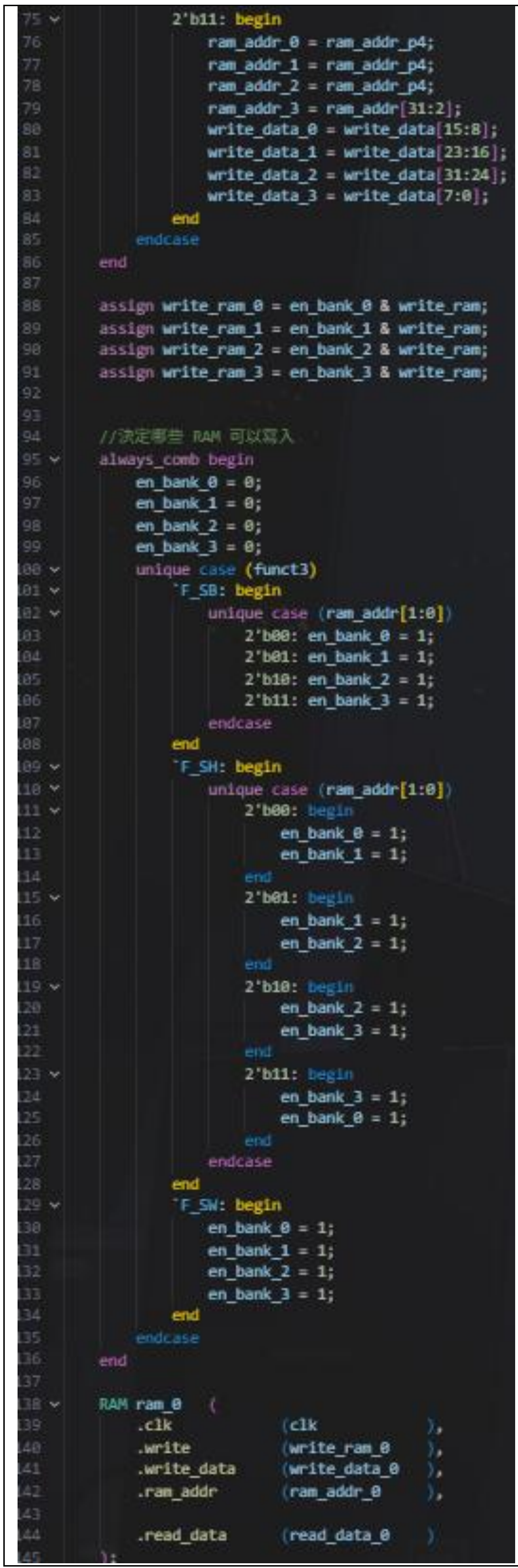
```
// div, mux
`define F_MUL 3'b000
`define F_MULH 3'b001
`define F_MULHSU 3'b010
`define F_MULHU 3'b011
`define F_DIV 3'b100
`define F_DIVU 3'b101
`define F_REM 3'b110
`define F_REMU 3'b111
```

Mylsu.sv

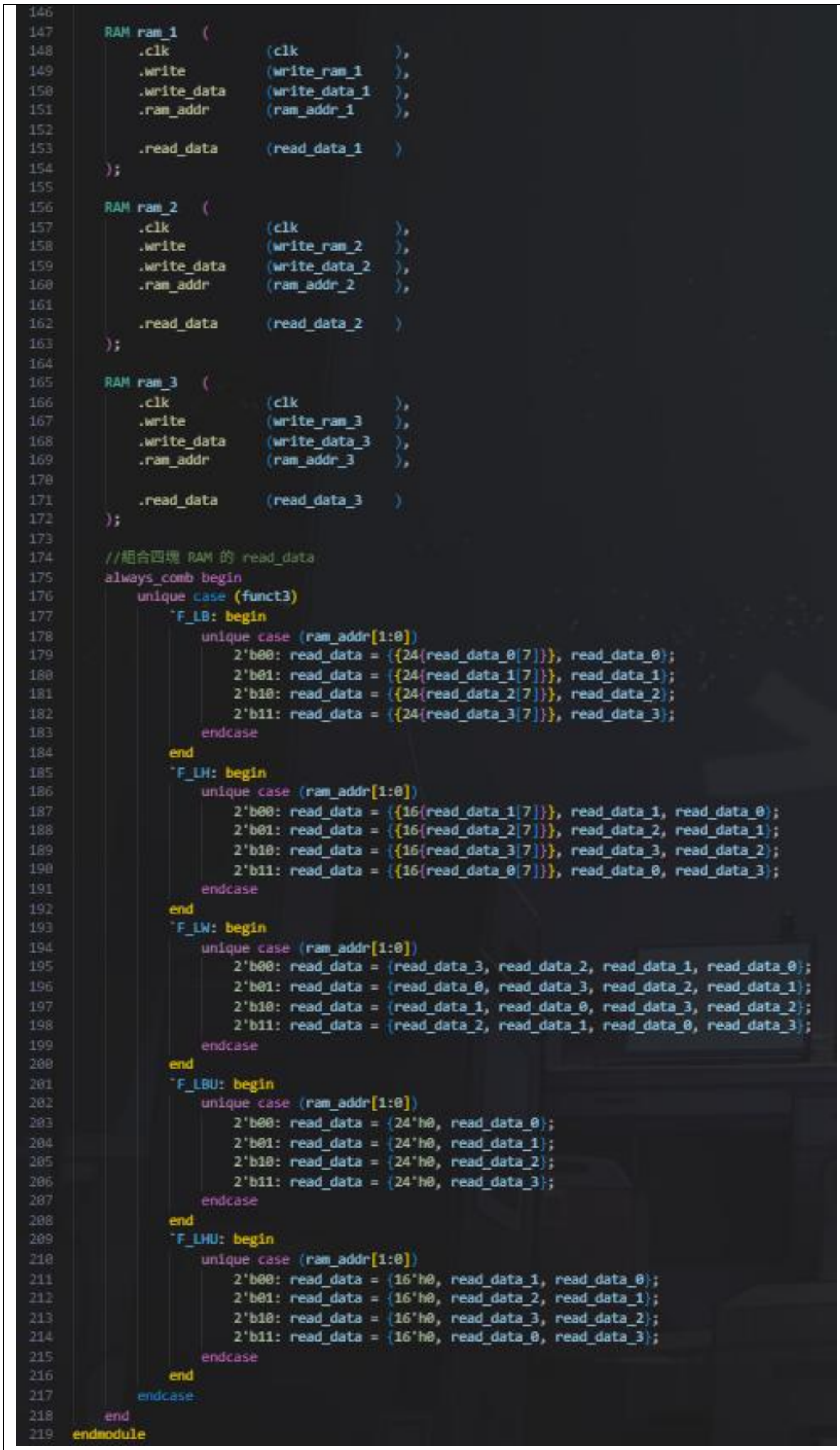


mylsu.sv > ...

```
`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
```

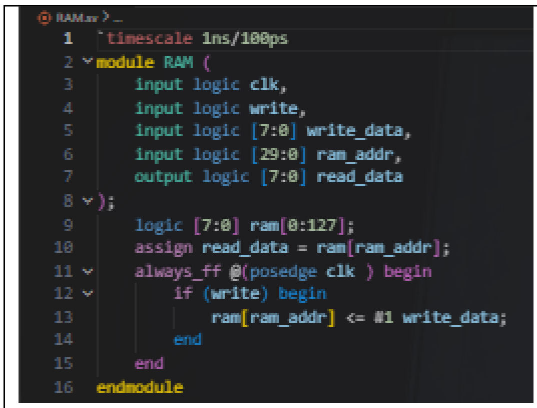


```
            2'b11: begin
                ram_addr_0 = ram_addr_p4;
                ram_addr_1 = ram_addr_p4;
                ram_addr_2 = ram_addr_p4;
                ram_addr_3 = ram_addr[31:2];
                write_data_0 = write_data[15:8];
                write_data_1 = write_data[23:16];
                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;


    //決定哪些 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    )
    );
```



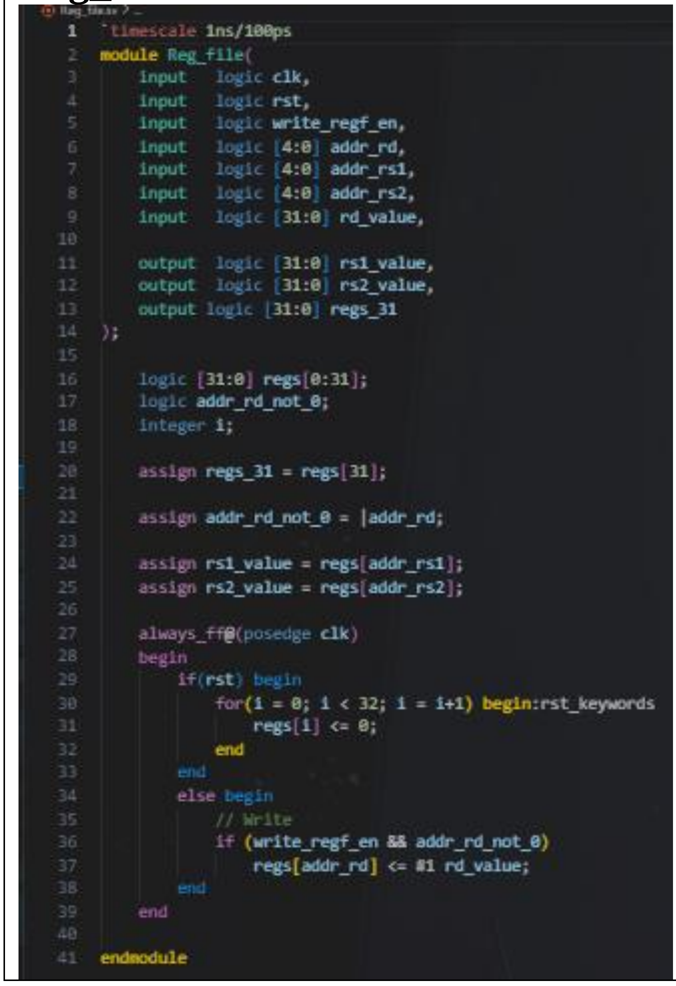
```systemverilog

    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    )
    );

    //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



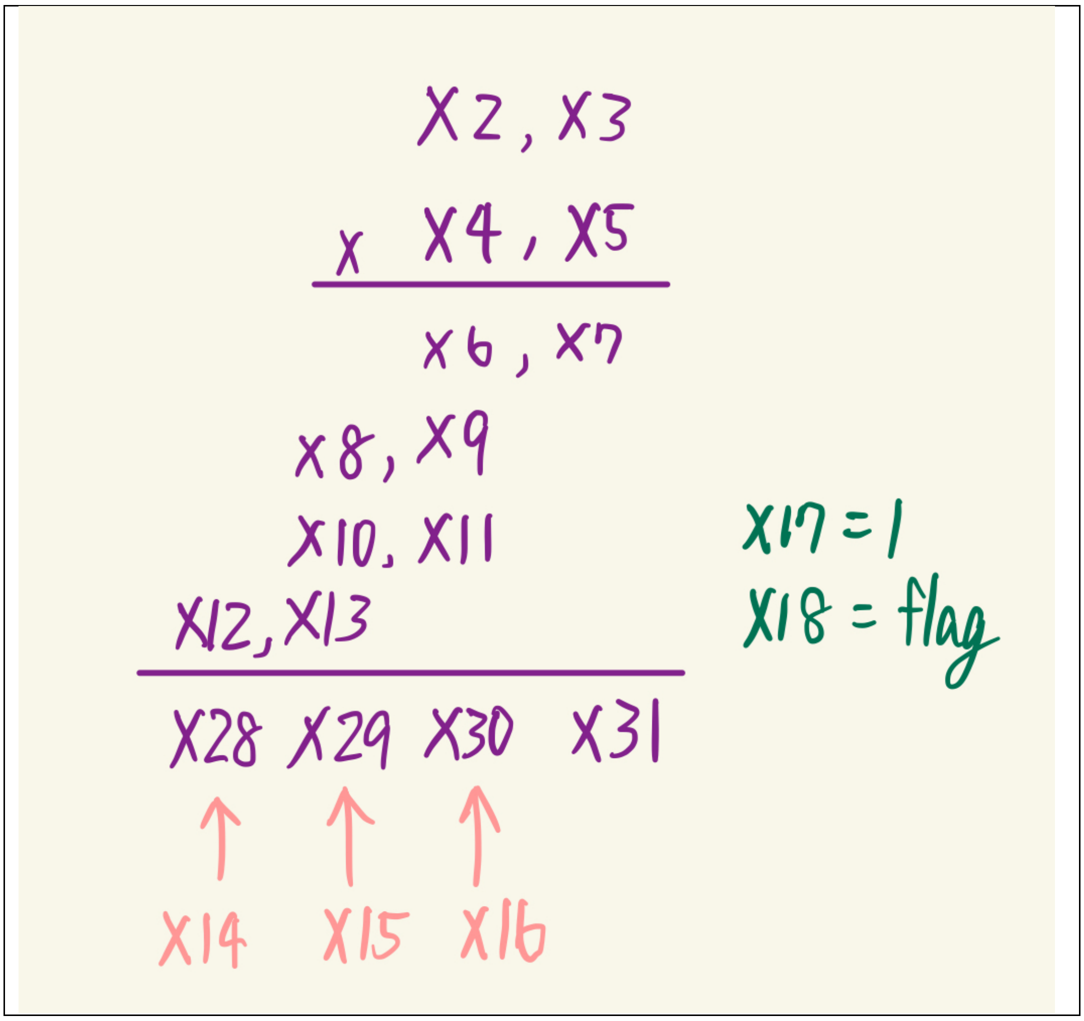
```
`timescale 1ns/100ps
module RAM (
    input logic clk,
    input logic write,
    input logic [7:0] write_data,
    input logic [29:0] ram_addr,
    output logic [7:0] read_data
);
    logic [7:0] ram[0:127];
    assign read_data = ram[ram_addr];
    always_ff @(posedge clk ) begin
        if (write)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

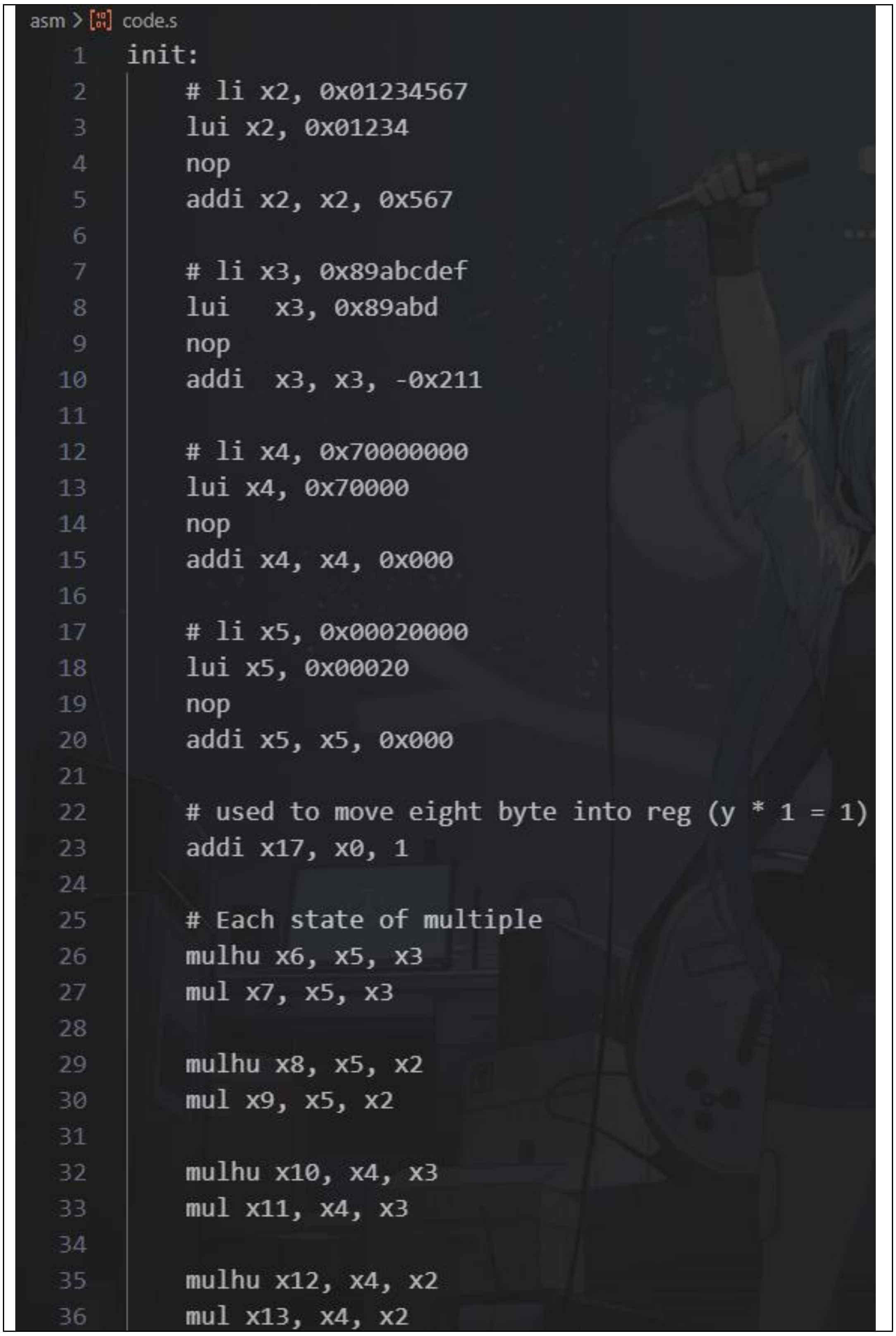
Reg_file.sv



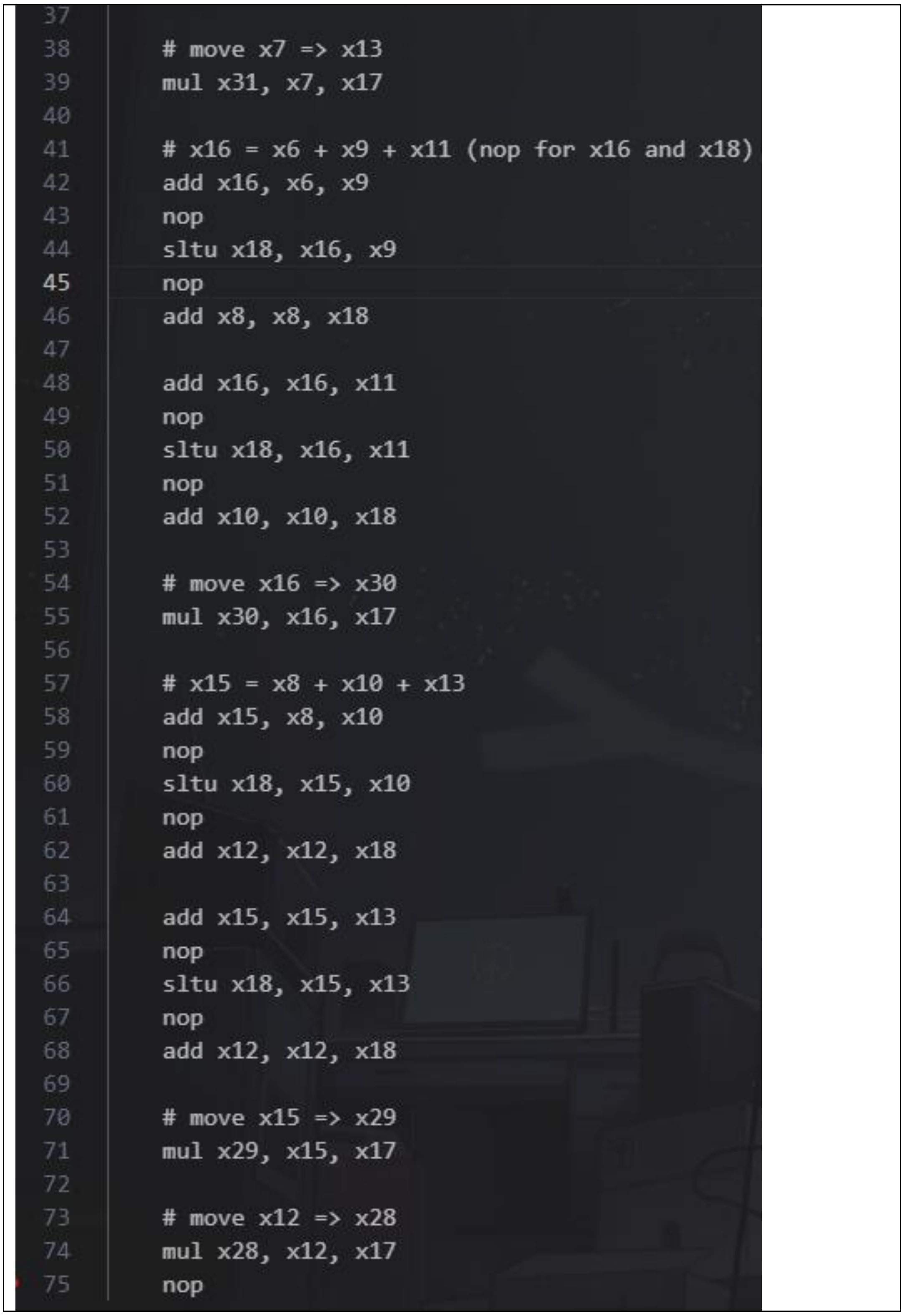
```
`timescale 1ns/100ps
module Reg_file(
    input   logic clk,
    input   logic rst,
    input   logic write_regf_en,
    input   logic [4:0] addr_rd,
    input   logic [4:0] addr_rs1,
    input   logic [4:0] addr_rs2,
    input   logic [31:0] rd_value,

    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
```

Program_rom + assembly

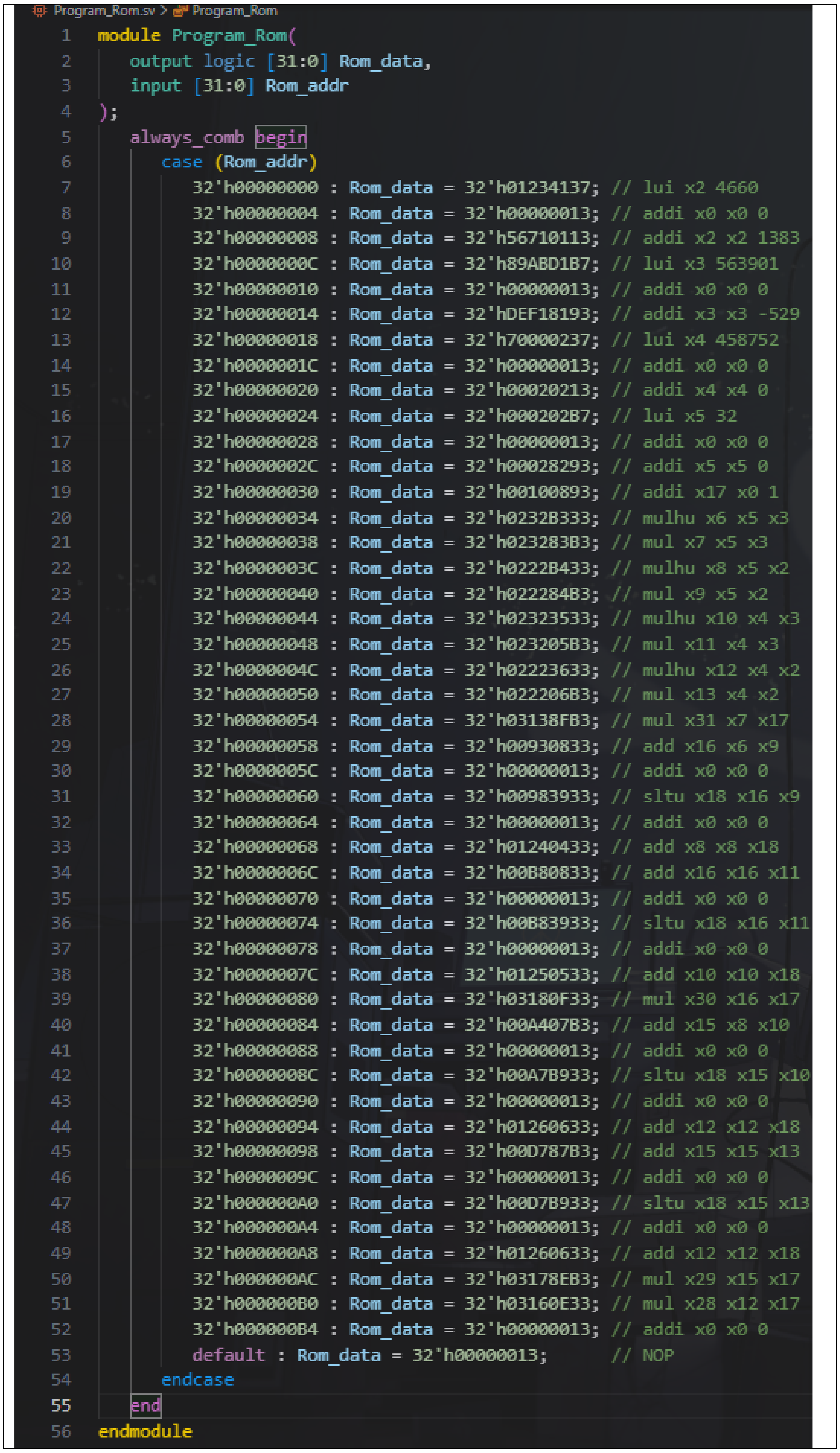

X2, X3
x X4, X5
X6, X7
X8, X9
X10, X11
X12, X13
X28 X29 X30 X31
X14 X15 X16
X17 = 1
X18 = flag


asm > code.s

```
init:
    # li x2, 0x01234567
    lui x2, 0x01234
    nop
    addi x2, x2, 0x567

    # li x3, 0x89abcdef
    lui   x3, 0x89abd
    nop
    addi  x3, x3, -0x211

    # li x4, 0x70000000
    lui x4, 0x70000
    nop
    addi x4, x4, 0x000

    # li x5, 0x00020000
    lui x5, 0x00020
    nop
    addi x5, x5, 0x000

    # used to move eight byte into reg (y * 1 = 1)
    addi x17, x0, 1

    # Each state of multiple
    mulhu x6, x5, x3
    mul x7, x5, x3

    mulhu x8, x5, x2
    mul x9, x5, x2

    mulhu x10, x4, x3
    mul x11, x4, x3

    mulhu x12, x4, x2
    mul x13, x4, x2
```

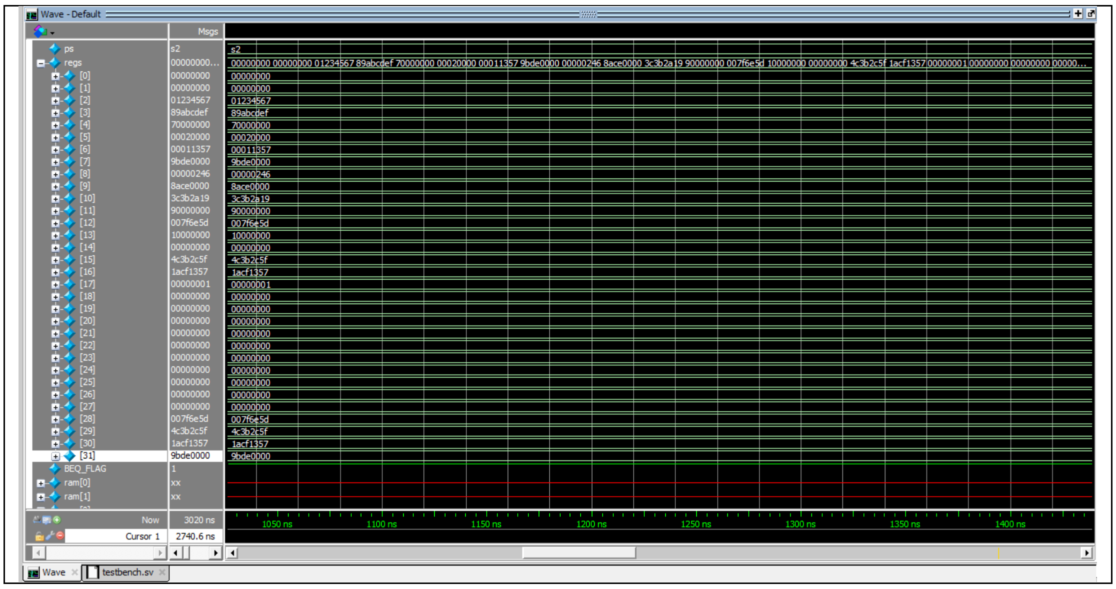


```

# move x7 => x13
mul x31, x7, x17

# x16 = x6 + x9 + x11 (nop for x16 and x18)
add x16, x6, x9
nop
sltu x18, x16, x9
nop
add x8, x8, x18

add x16, x16, x11
nop
sltu x18, x16, x11
nop
add x10, x10, x18

# move x16 => x30
mul x30, x16, x17

# x15 = x8 + x10 + x13
add x15, x8, x10
nop
sltu x18, x15, x10
nop
add x12, x12, x18

add x15, x15, x13
nop
sltu x18, x15, x13
nop
add x12, x12, x18

# move x15 => x29
mul x29, x15, x17

# move x12 => x28
mul x28, x12, x17
nop
```



```systemverilog
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'h01234137; // lui x2 4660
            32'h00000004 : Rom_data = 32'h00000013; // addi x0 x0 0
            32'h00000008 : Rom_data = 32'h56710113; // addi x2 x2 1383
            32'h0000000C : Rom_data = 32'h89ABD1B7; // lui x3 563901
            32'h00000010 : Rom_data = 32'h00000013; // addi x0 x0 0
            32'h00000014 : Rom_data = 32'hDEF18193; // addi x3 x3 -529
            32'h00000018 : Rom_data = 32'h70000237; // lui x4 458752
            32'h0000001C : Rom_data = 32'h00000013; // addi x0 x0 0
            32'h00000020 : Rom_data = 32'h00020213; // addi x4 x4 0
            32'h00000024 : Rom_data = 32'h000202B7; // lui x5 32
            32'h00000028 : Rom_data = 32'h00000013; // addi x0 x0 0
            32'h0000002C : Rom_data = 32'h00028293; // addi x5 x5 0
            32'h00000030 : Rom_data = 32'h00100893; // addi x17 x0 1
            32'h00000034 : Rom_data = 32'h0232B333; // mulhu x6 x5 x3
            32'h00000038 : Rom_data = 32'h023283B3; // mul x7 x5 x3
            32'h0000003C : Rom_data = 32'h0222B433; // mulhu x8 x5 x2
            32'h00000040 : Rom_data = 32'h022284B3; // mul x9 x5 x2
            32'h00000044 : Rom_data = 32'h02323533; // mulhu x10 x4 x3
            32'h00000048 : Rom_data = 32'h023205B3; // mul x11 x4 x3
            32'h0000004C : Rom_data = 32'h02223633; // mulhu x12 x4 x2
            32'h00000050 : Rom_data = 32'h022206B3; // mul x13 x4 x2
            32'h00000054 : Rom_data = 32'h03138FB3; // mul x31 x7 x17
            32'h00000058 : Rom_data = 32'h00930833; // add x16 x6 x9
            32'h0000005C : Rom_data = 32'h00000013; // addi x0 x0 0
            32'h00000060 : Rom_data = 32'h00983933; // sltu x18 x16 x9
            32'h00000064 : Rom_data = 32'h00000013; // addi x0 x0 0
            32'h00000068 : Rom_data = 32'h01240433; // add x8 x8 x18
            32'h0000006C : Rom_data = 32'h00B80833; // add x16 x16 x11
            32'h00000070 : Rom_data = 32'h00000013; // addi x0 x0 0
            32'h00000074 : Rom_data = 32'h00B83933; // sltu x18 x16 x11
            32'h00000078 : Rom_data = 32'h00000013; // addi x0 x0 0
            32'h0000007C : Rom_data = 32'h01250533; // add x10 x10 x18
            32'h00000080 : Rom_data = 32'h03180F33; // mul x30 x16 x17
            32'h00000084 : Rom_data = 32'h00A407B3; // add x15 x8 x10
            32'h00000088 : Rom_data = 32'h00000013; // addi x0 x0 0
            32'h0000008C : Rom_data = 32'h00A7B933; // sltu x18 x15 x10
            32'h00000090 : Rom_data = 32'h00000013; // addi x0 x0 0
            32'h00000094 : Rom_data = 32'h01260633; // add x12 x12 x18
            32'h00000098 : Rom_data = 32'h00D787B3; // add x15 x15 x13
            32'h0000009C : Rom_data = 32'h00000013; // addi x0 x0 0
            32'h000000A0 : Rom_data = 32'h00D7B933; // sltu x18 x15 x13
            32'h000000A4 : Rom_data = 32'h00000013; // addi x0 x0 0
            32'h000000A8 : Rom_data = 32'h01260633; // add x12 x12 x18
            32'h000000AC : Rom_data = 32'h03178EB3; // mul x29 x15 x17
            32'h000000B0 : Rom_data = 32'h03160E33; // mul x28 x12 x17
            32'h000000B4 : 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：



```
Wave - Default
                Msgs
ps              s2           s2
regs            00000000...  00000000 00000000 01234567 89abcdef 70000000 00020000 00011357 9bde0000 00000246 8ace0000 3c3b2a19 90000000 007f6e5d 10000000 00000000 4c3b2c5f 1acf1357 00000001 00000000 00000000 00000...
  [0]           00000000     00000000
  [1]           00000000     00000000
  [2]           01234567     01234567
  [3]           89abcdef     89abcdef
  [4]           70000000     70000000
  [5]           00020000     00020000
  [6]           00011357     00011357
  [7]           9bde0000     9bde0000
  [8]           00000246     00000246
  [9]           8ace0000     8ace0000
  [10]          3c3b2a19     3c3b2a19
  [11]          90000000     90000000
  [12]          007f6e5d     007f6e5d
  [13]          10000000     10000000
  [14]          00000000     00000000
  [15]          4c3b2c5f     4c3b2c5f
  [16]          1acf1357     1acf1357
  [17]          00000001     00000001
  [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]          007f6e5d     007f6e5d
  [29]          4c3b2c5f     4c3b2c5f
  [30]          1acf1357     1acf1357
  [31]          9bde0000     9bde0000
BEQ_FLAG        1
ram[0]          xx
ram[1]          xx
            Now 3020 ns      1050 ns  1100 ns  1150 ns  1200 ns  1250 ns  1300 ns  1350 ns  1400 ns
       Cursor 1 2740.6 ns
Wave  testbench.sv
```

●結論與心得：

　　我知道乘法組語不是那麼簡單，但我也不想再往後檢查，向前一位檢察溢位已經是我的極限了，我不想一直考慮到最開頭的地方了，沒有每次那麼剛好從做後加一到開頭一路進位的：）