2025/11/13
實驗九

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

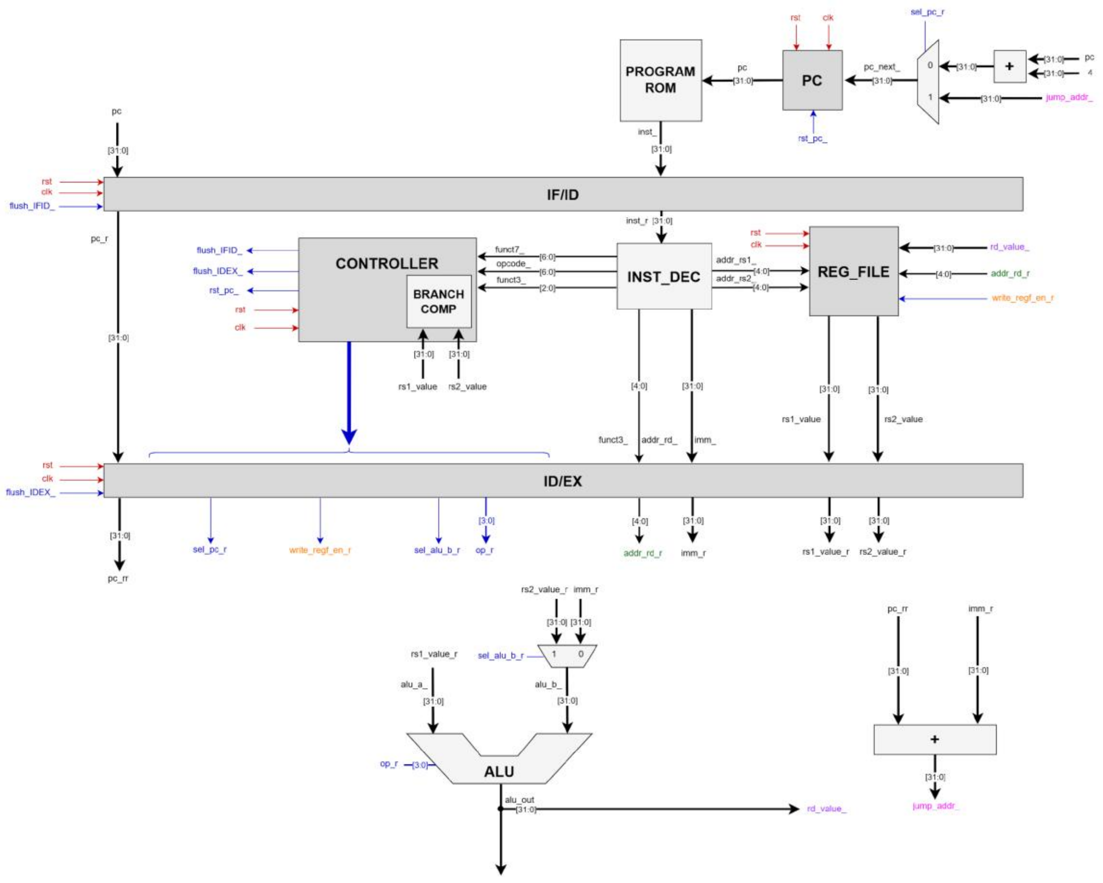
※注意
1. 繳交時一律轉 **PDF** 檔
2. 繳交期限為下周上課前
3. 一人繳交一份
4. 檔名請按照作業檔名格式進行填寫，未依照格式不予批改
5. 檔名範例：學號_姓名_**HW9**

1、 組語撰寫練習—累加

●實驗說明：

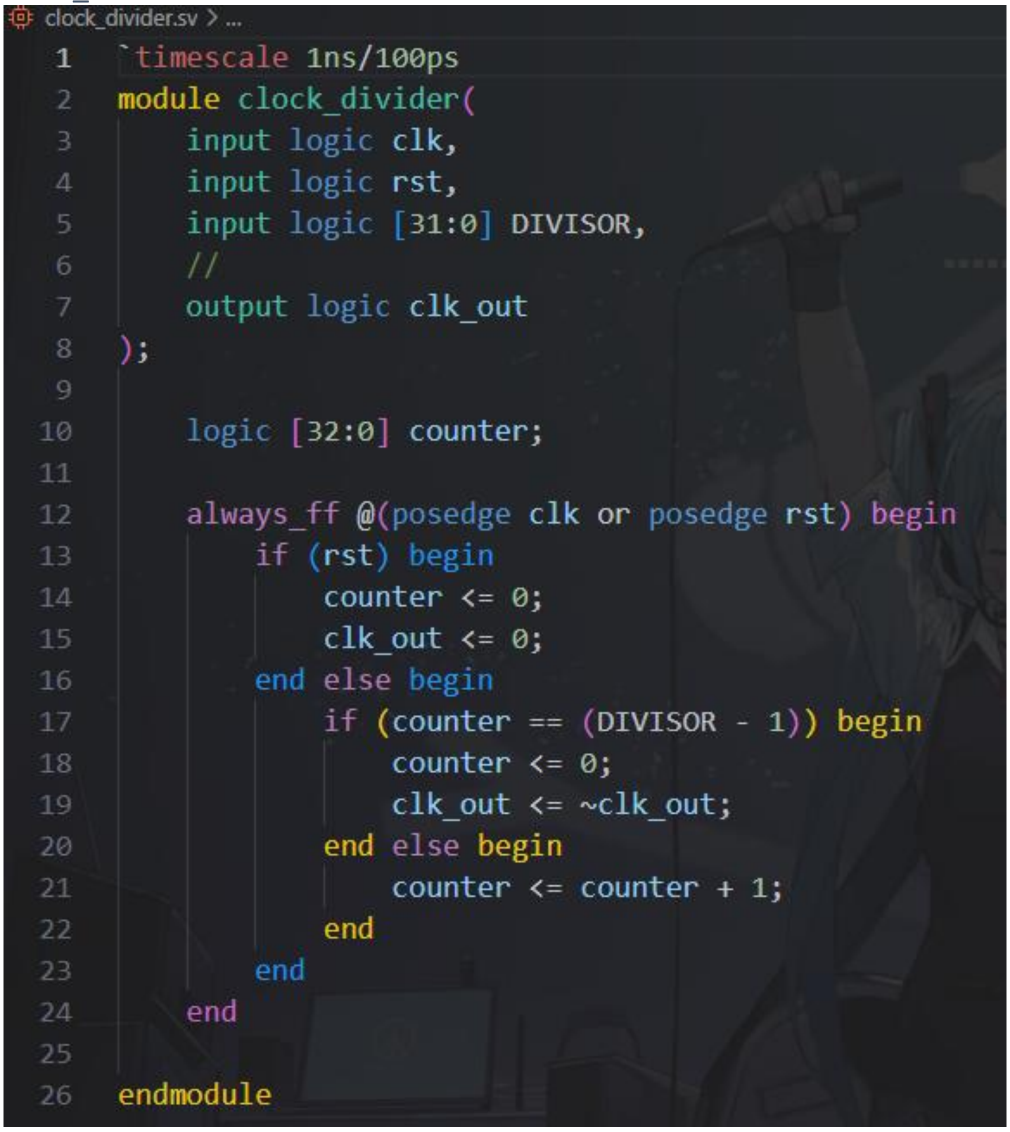
1. 透過當前課堂所學的所有組合語言撰寫一個 1 累加到 10 的程式，須加註解。
2. 所有暫存器皆可使用，但請將結果存儲至暫存器 x31。
3. 程式撰寫完成後透過 Tool Chain 將其轉換成 Program_ROM.sv 檔，並使用 ModelSim 軟體觀察波形是否執行正確。
4. 請將所有用到的暫存器都加入至波形中以方便觀察。
5. 請注意資料危障的問題。

●系統硬體架構方塊圖（接線圖）：


PROGRAM ROM
PC
pc_next_
sel_pc_r
jump_addr_
rst_pc_
inst_
IF/ID
flush_IFID_
inst_r
CONTROLLER
BRANCH COMP
flush_IDEX_
rst_pc_
funct7_
opcode_
funct3_
INST_DEC
addr_rs1_
addr_rs2_
REG_FILE
rd_value_
addr_rd_r
write_regf_en_r
rs1_value
rs2_value
pc_r
funct3_
addr_rd_
imm_
ID/EX
sel_pc_r
write_regf_en_r
sel_alu_b_r
op_r
addr_rd_r
imm_r
rs1_value_r
rs2_value_r
pc_rr
rs2_value_r
imm_r
rs1_value_r
sel_alu_b_r
alu_a_
alu_b_
ALU
op_r
alu_out
rd_value_
pc_rr
imm_r
jump_addr_

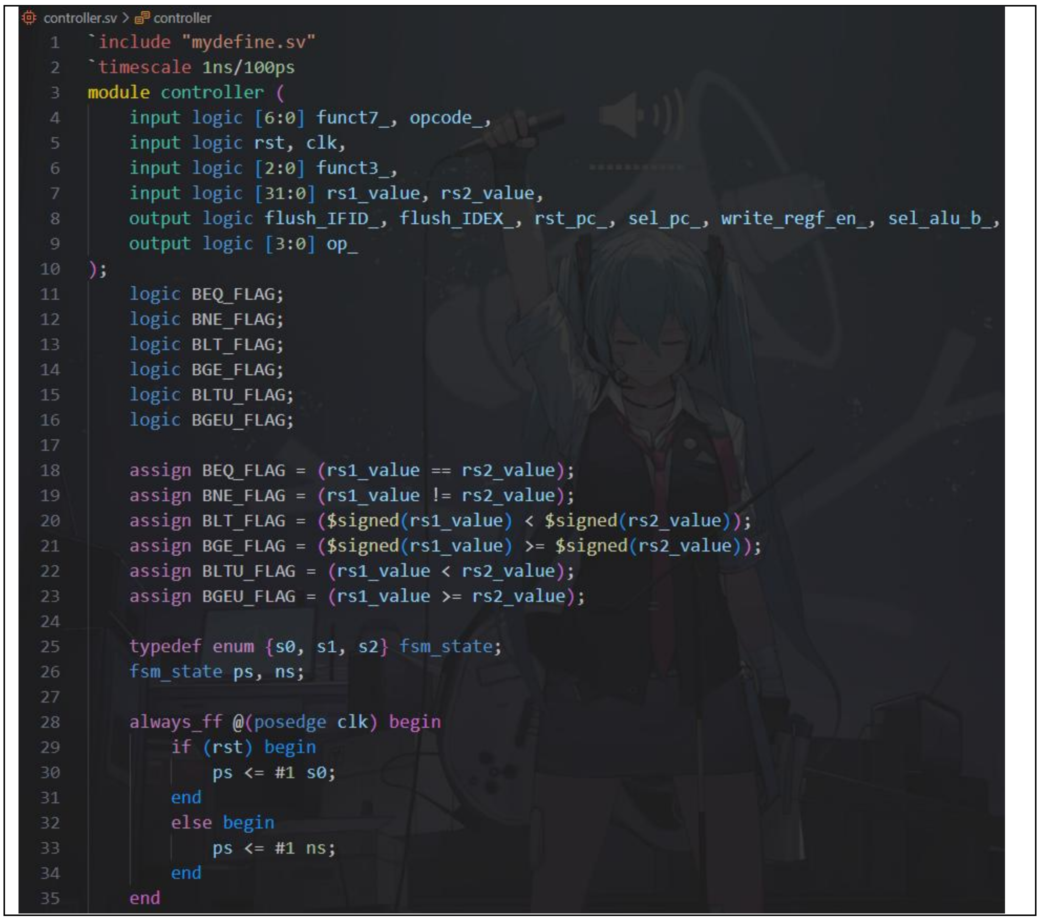
●系統架構程式碼、測試資料程式碼與程式碼說明
截圖請善用 **win+shift+S**

Clock_divider.sv

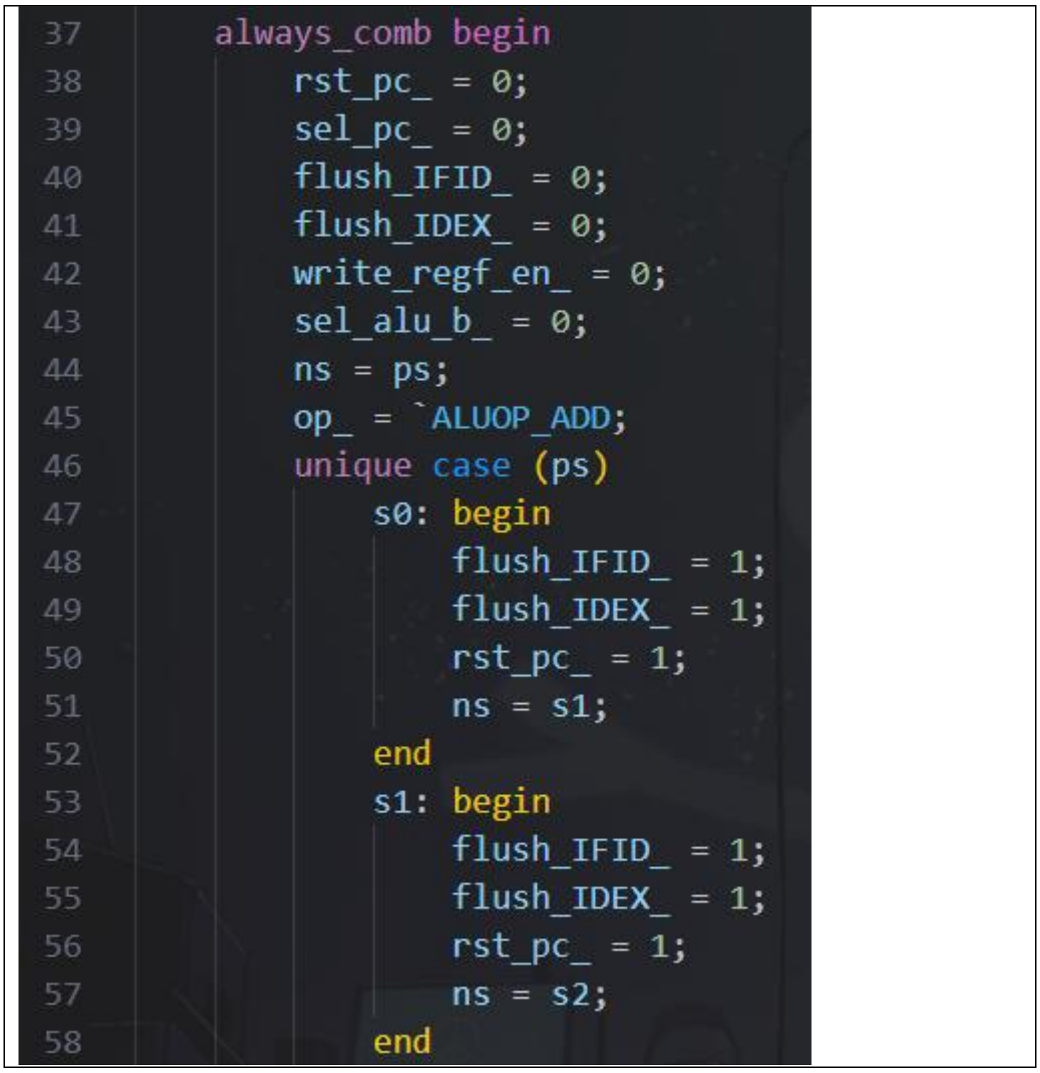


```
`timescale 1ns/100ps
module clock_divider(
    input logic clk,
    input logic rst,
    input logic [31:0] DIVISOR,
    //
    output logic clk_out
);

    logic [32:0] counter;

    always_ff @(posedge clk or posedge rst) begin
        if (rst) begin
            counter <= 0;
            clk_out <= 0;
        end else begin
            if (counter == (DIVISOR - 1)) begin
                counter <= 0;
                clk_out <= ~clk_out;
            end else begin
                counter <= counter + 1;
            end
        end
    end

endmodule
```

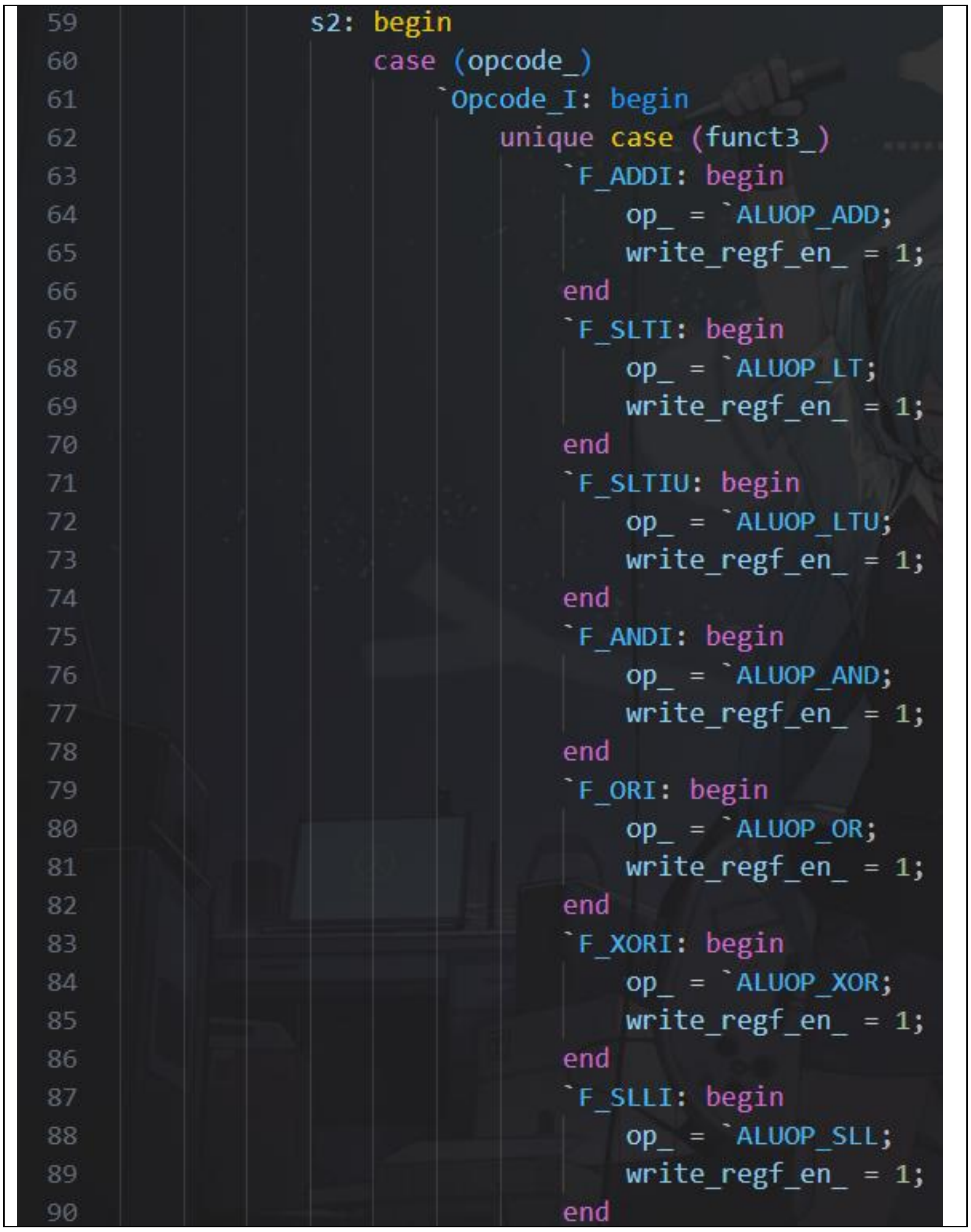
Controller.sv



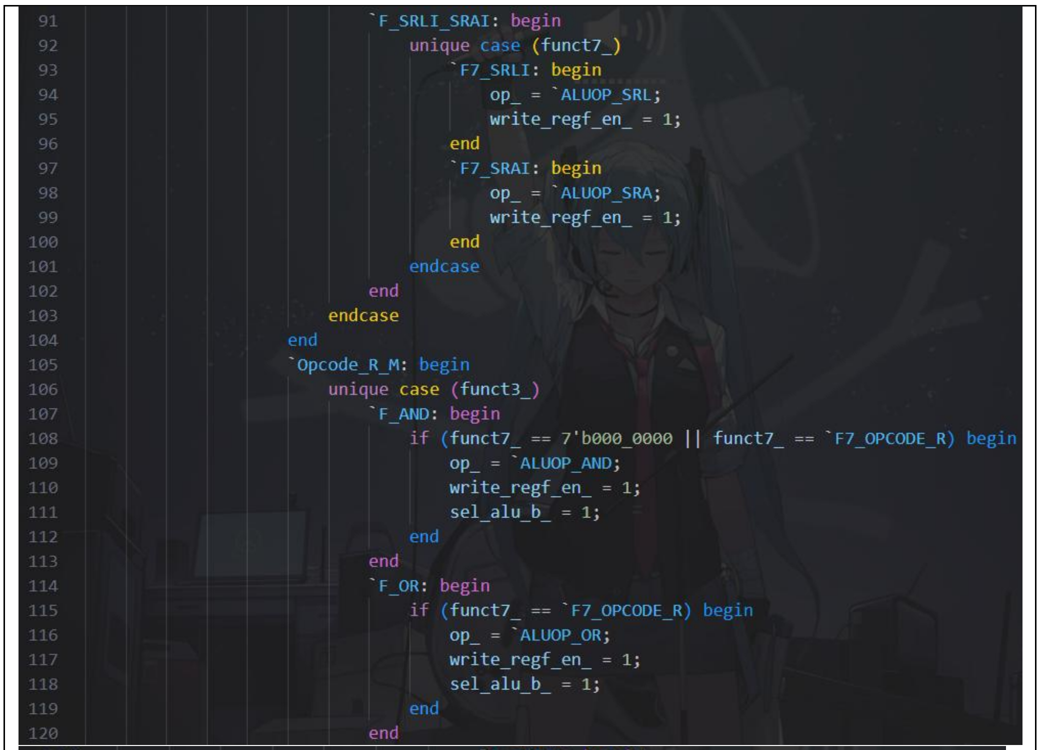
```
`include "mydefine.sv"
`timescale 1ns/100ps
module controller (
    input logic [6:0] funct7_, opcode_,
    input logic rst, clk,
    input logic [2:0] funct3_,
    input logic [31:0] rs1_value, rs2_value,
    output logic flush_IFID_, flush_IDEX_, rst_pc_, sel_pc_, write_regf_en_, sel_alu_b_,
 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



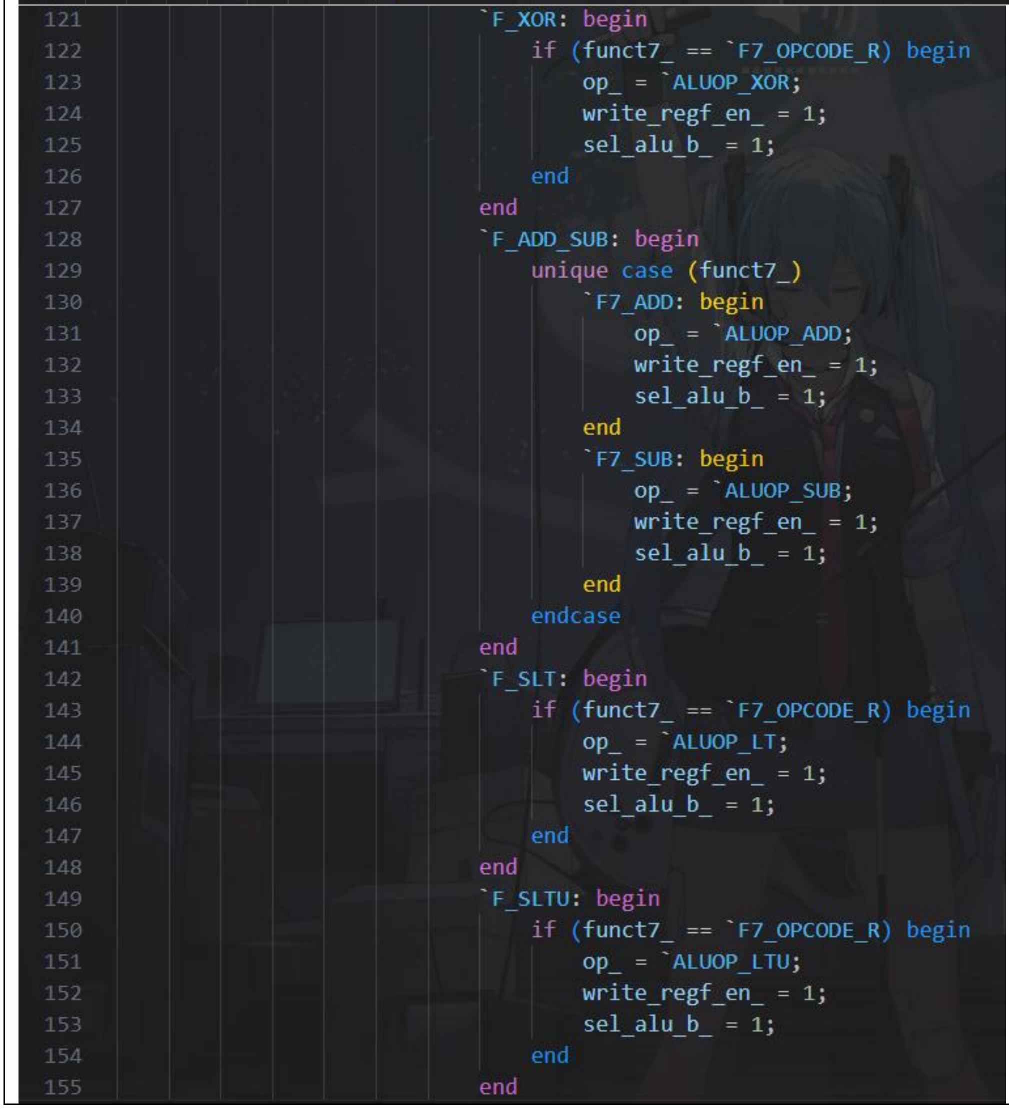
```
    always_comb begin
        rst_pc_ = 0;
        sel_pc_ = 0;
        flush_IFID_ = 0;
        flush_IDEX_ = 0;
        write_regf_en_ = 0;
        sel_alu_b_ = 0;
        ns = ps;
        op_ = `ALUOP_ADD;
        unique case (ps)
            s0: begin
                flush_IFID_ = 1;
                flush_IDEX_ = 1;
                rst_pc_ = 1;
                ns = s1;
            end
            s1: begin
                flush_IFID_ = 1;
                flush_IDEX_ = 1;
                rst_pc_ = 1;
                ns = s2;
            end
```



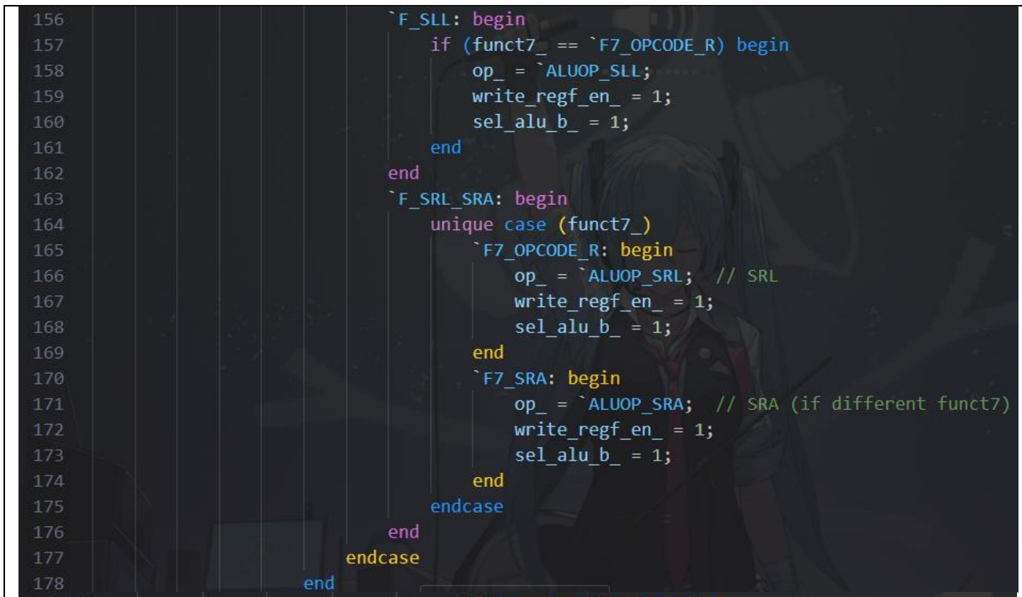
```
            s2: begin
                case (opcode_)
                    `Opcode_I: begin
                        unique case (funct3_)
                            `F_ADDI: begin
                                op_ = `ALUOP_ADD;
                                write_regf_en_ = 1;
               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



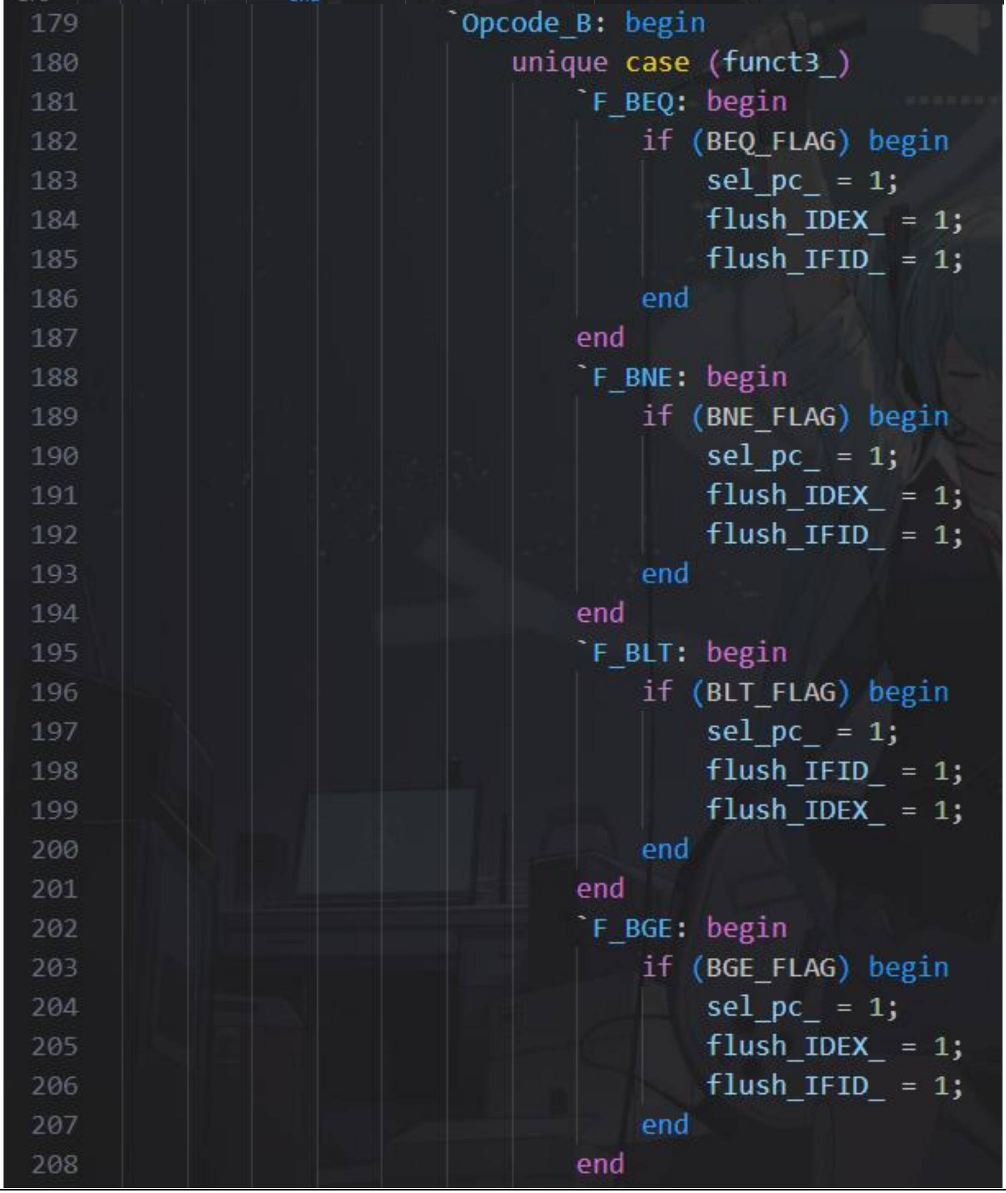
```verilog
                            `F_SRLI_SRAI: begin
                                unique case (funct7_)
                                    `F7_SRLI: begin
                                        op_ = `ALUOP_SRL;
                                        write_regf_en_ = 1;
                                    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
```



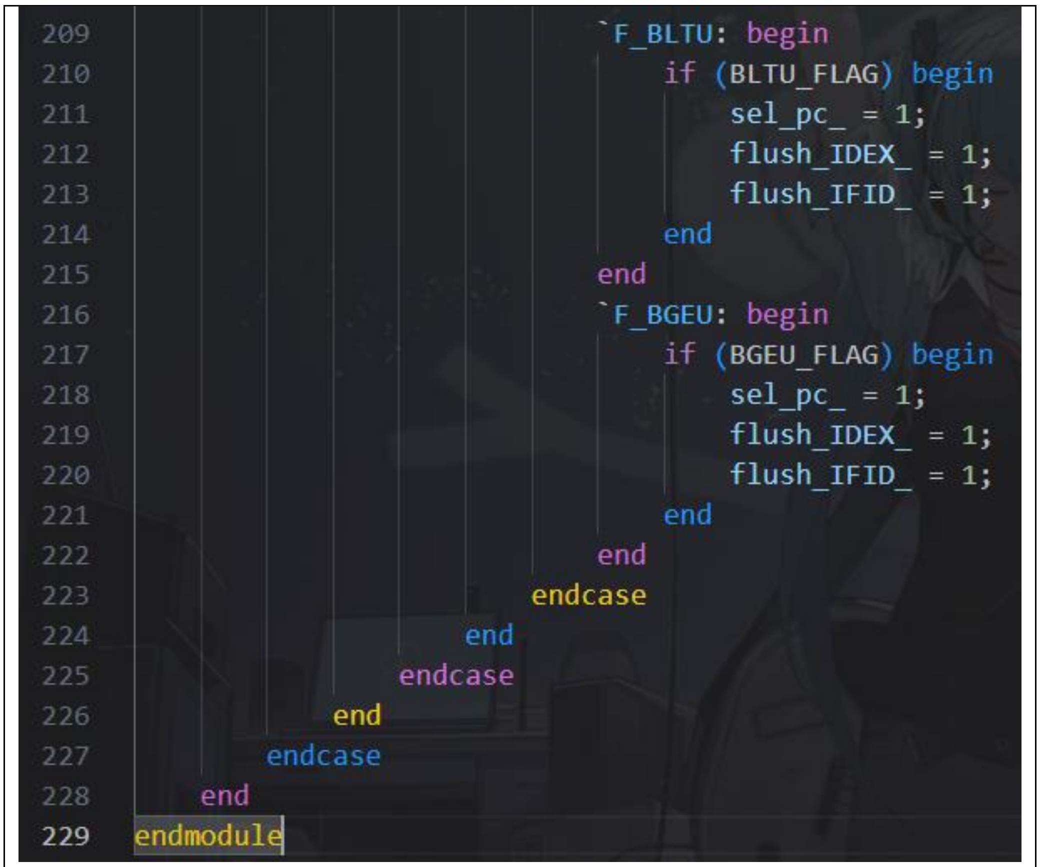
```verilog
                            `F_XOR: begin
                                if (funct7_ == `F7_OPCODE_R) begin
                                    op_ = `ALUOP_XOR;
                                    write_regf_en_ = 1;
                                    sel_alu_b_ = 1;
                                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
```

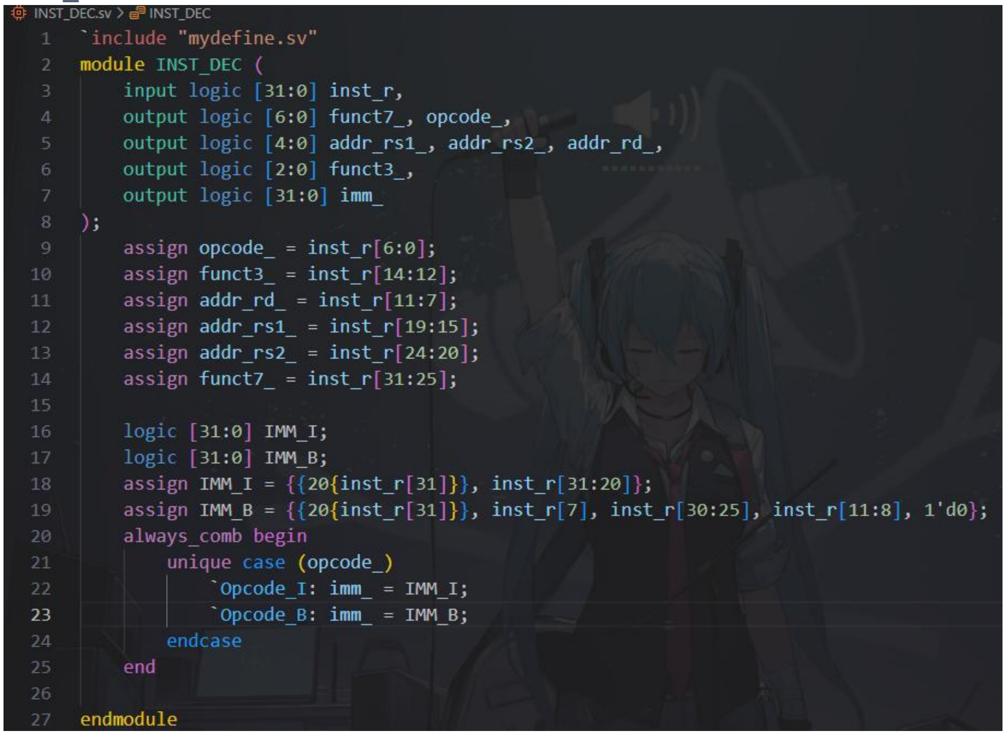


```
                    `Opcode_B: begin
                        unique case (funct3_)
                            `F_BEQ: begin
                                if (BEQ_FLAG) begin
                                    sel_pc_ = 1;
                                    flush_IDEX_ = 1;
                                    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
                endcase
            end
        endcase
    end
endmodule
```

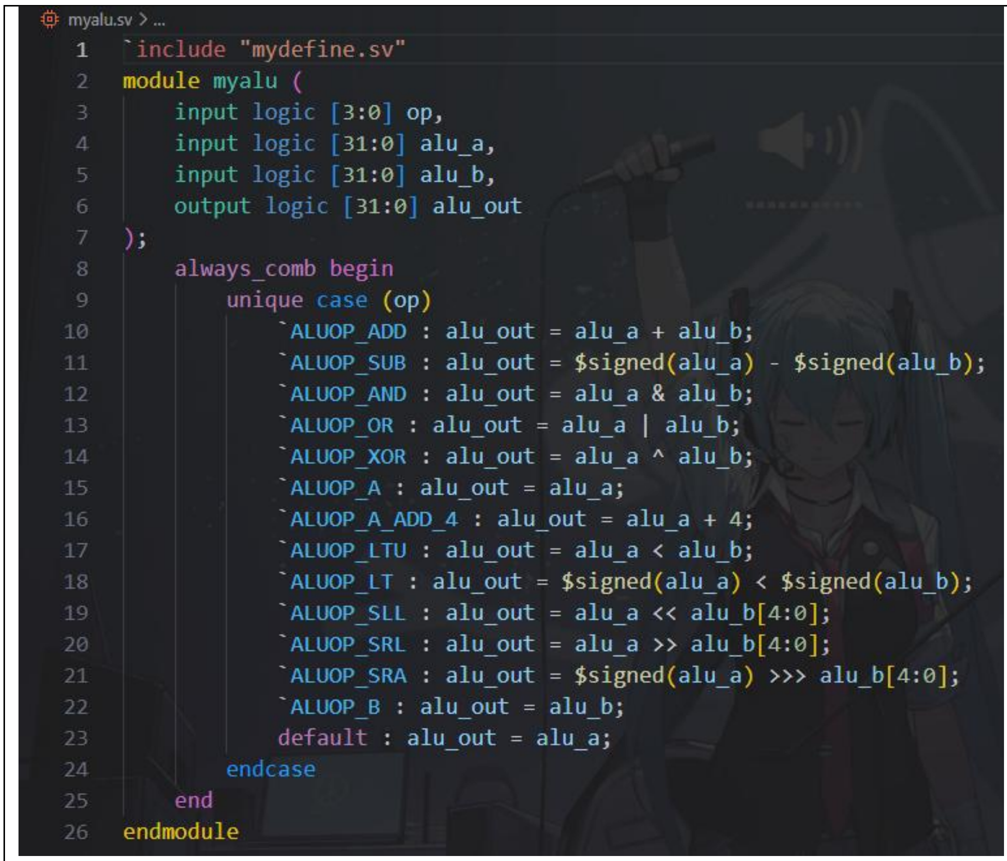
INST_DEC.sv



INST_DEC.sv > INST_DEC

```
`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;
    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'd0};
    always_comb begin
        unique case (opcode_)
            `Opcode_I: imm_ = IMM_I;
            `Opcode_B: imm_ = IMM_B;
        endcase
    end

endmodule
```

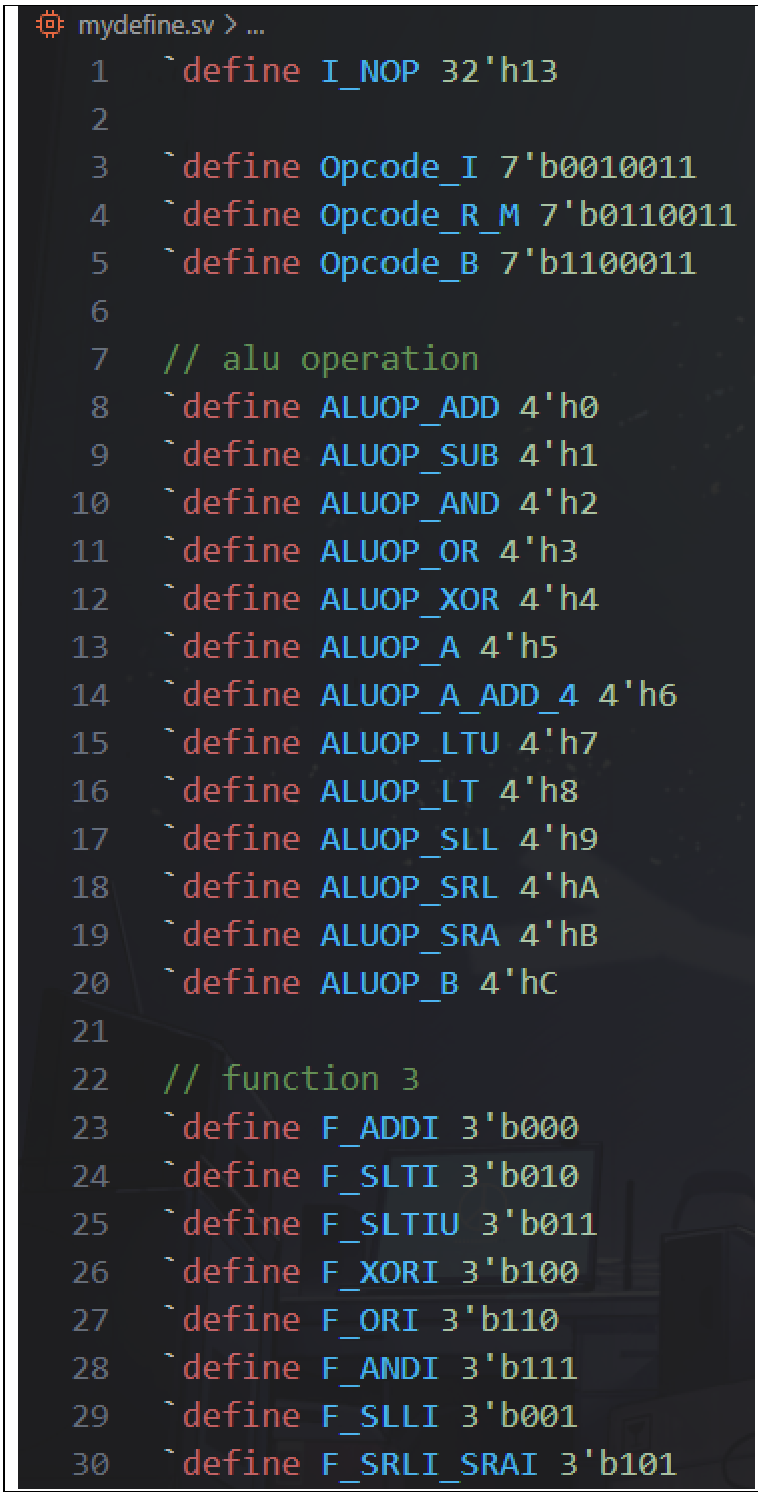
Myalu.sv



myalu.sv > ...

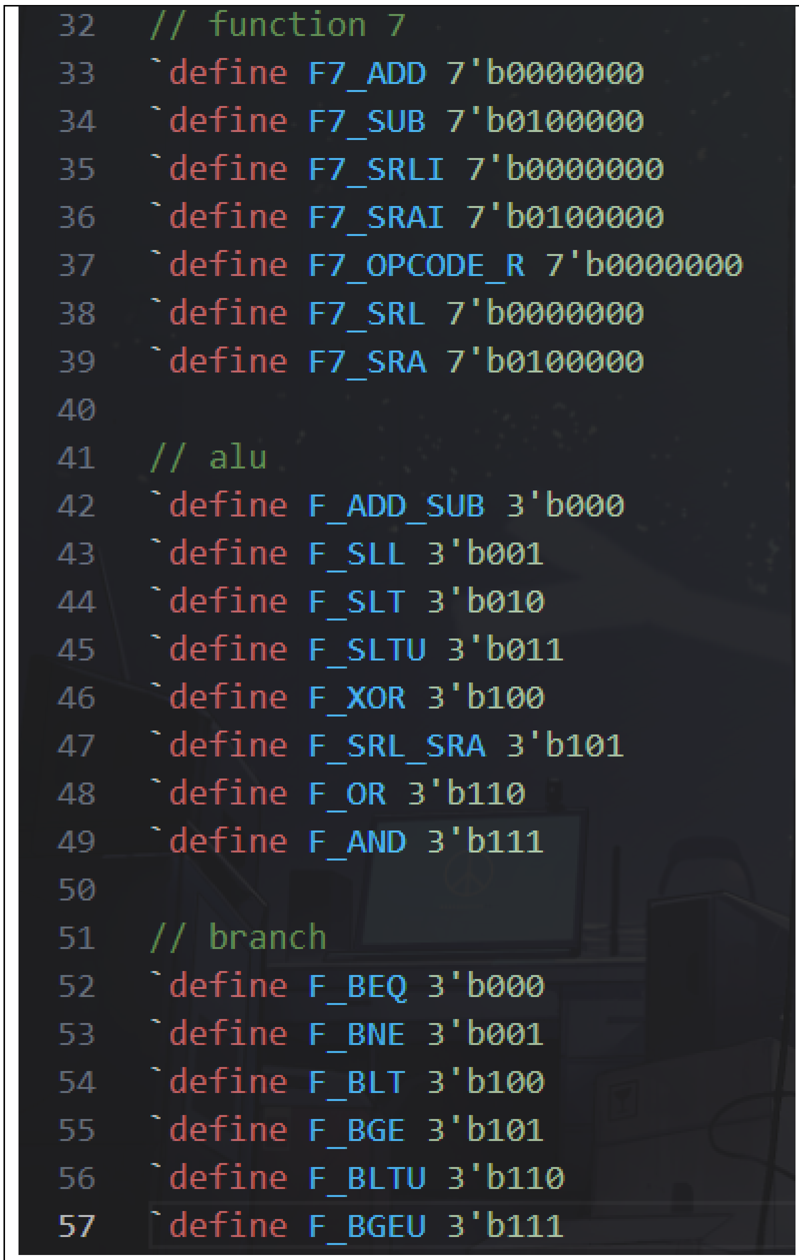
```
`include "mydefine.sv"
module myalu (
    input logic [3:0] op,
    input logic [31:0] alu_a,
    input logic [31:0] alu_b,
    output logic [31:0] alu_out
);
    always_comb begin
        unique case (op)
            `ALUOP_ADD : alu_out = alu_a + alu_b;
            `ALUOP_SUB : alu_out = $signed(alu_a)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



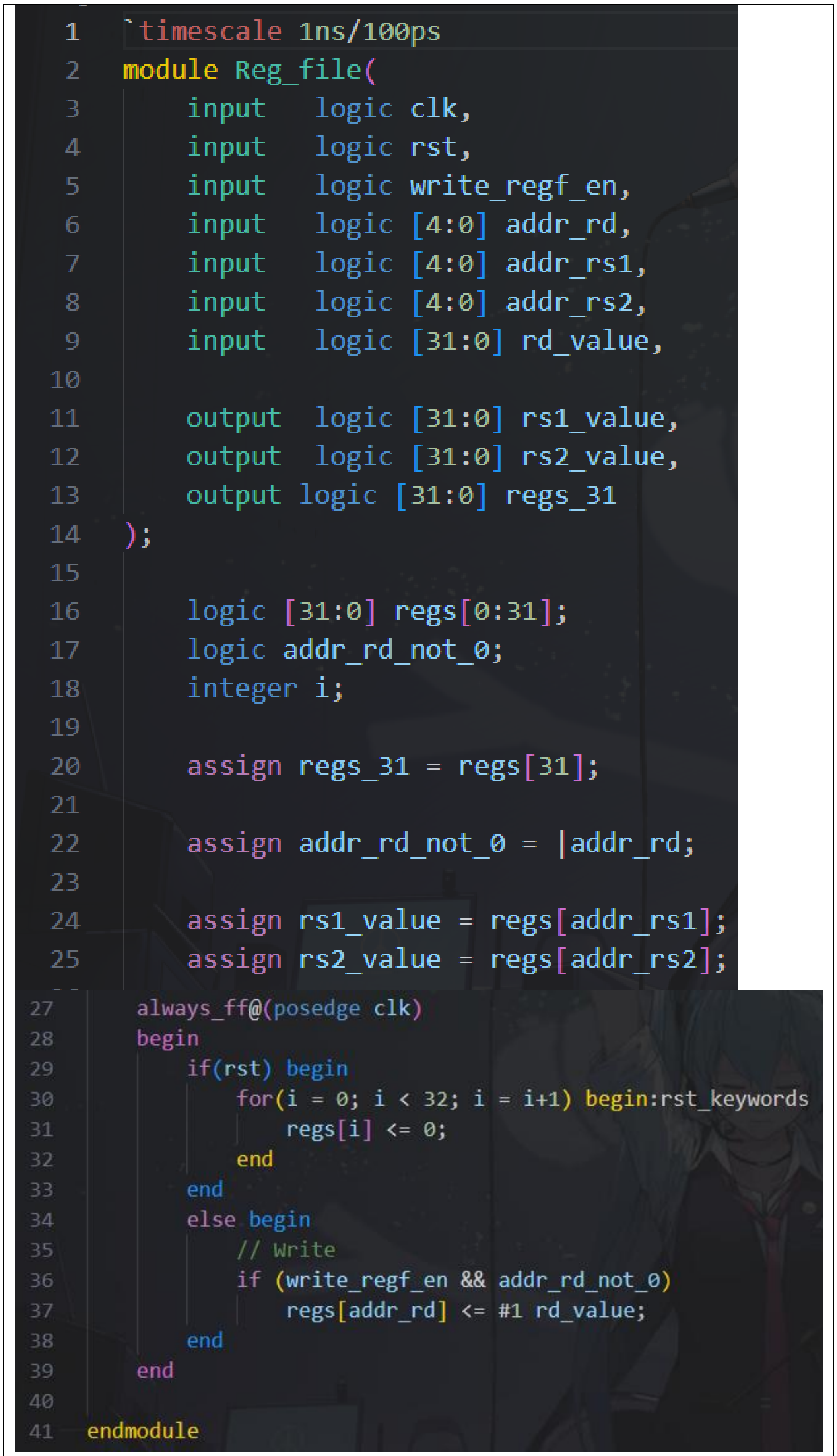
mydefine.sv > ...

```
`define I_NOP 32'h13

`define Opcode_I 7'b0010011
`define Opcode_R_M 7'b0110011
`define Opcode_B 7'b1100011

// 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
```



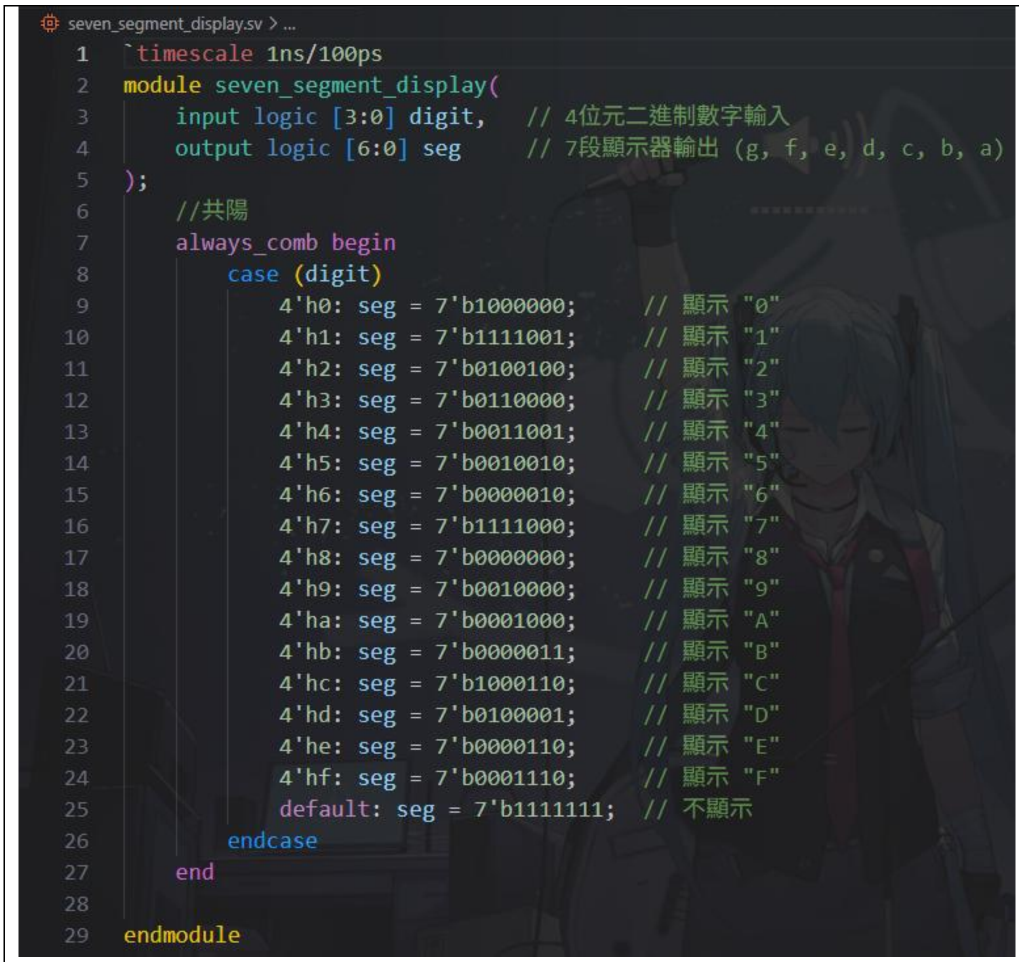
```
// function 7
`define F7_ADD 7'b0000000
`define F7_SUB 7'b0100000
`define F7_SRLI 7'b0000000
`define F7_SRAI 7'b0100000
`define F7_OPCODE_R 7'b0000000
`define F7_SRL 7'b0000000
`define F7_SRA 7'b0100000

// 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
```

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
```

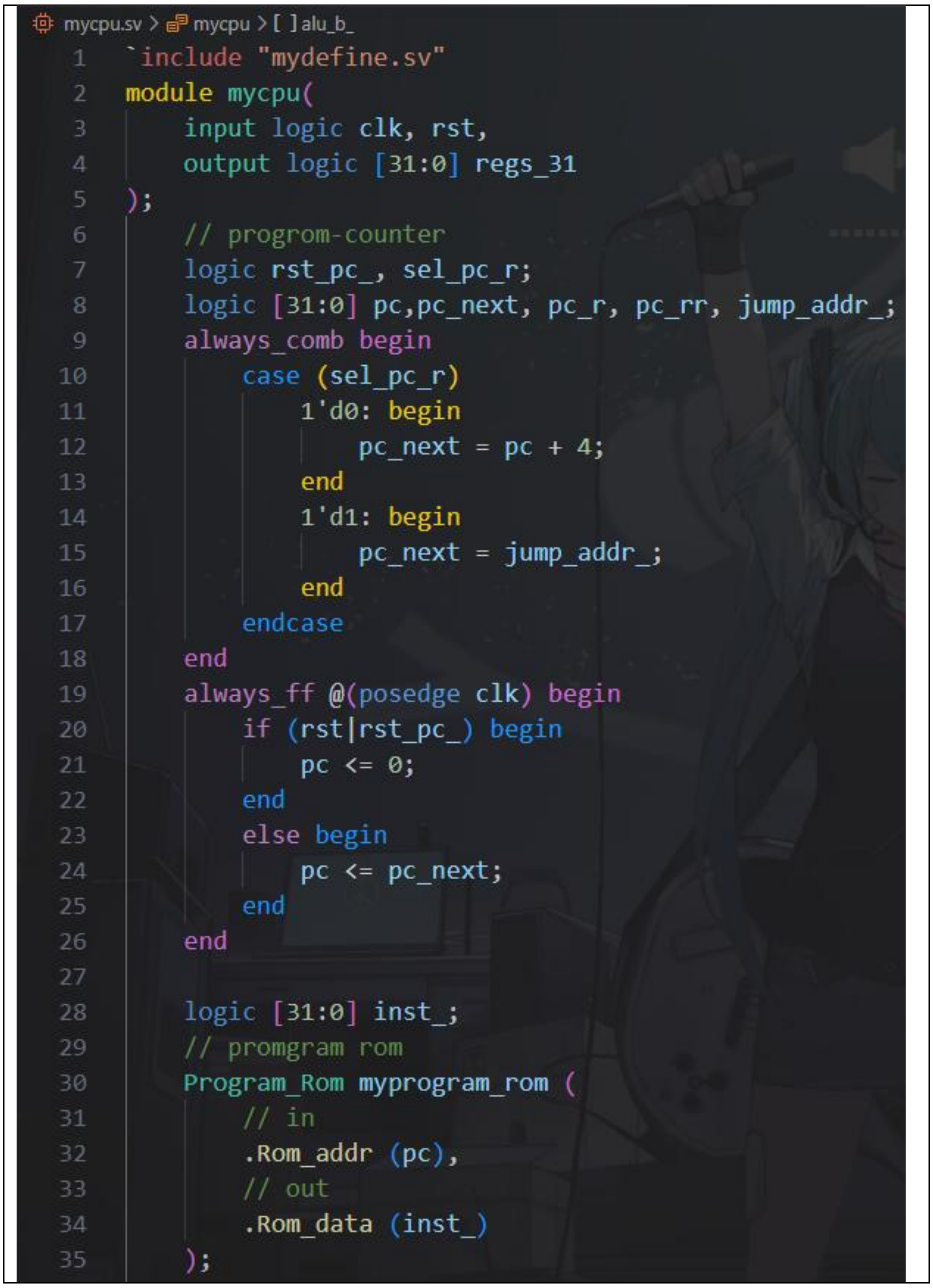
Seven_segment_display.sv



seven_segment_display.sv > ...

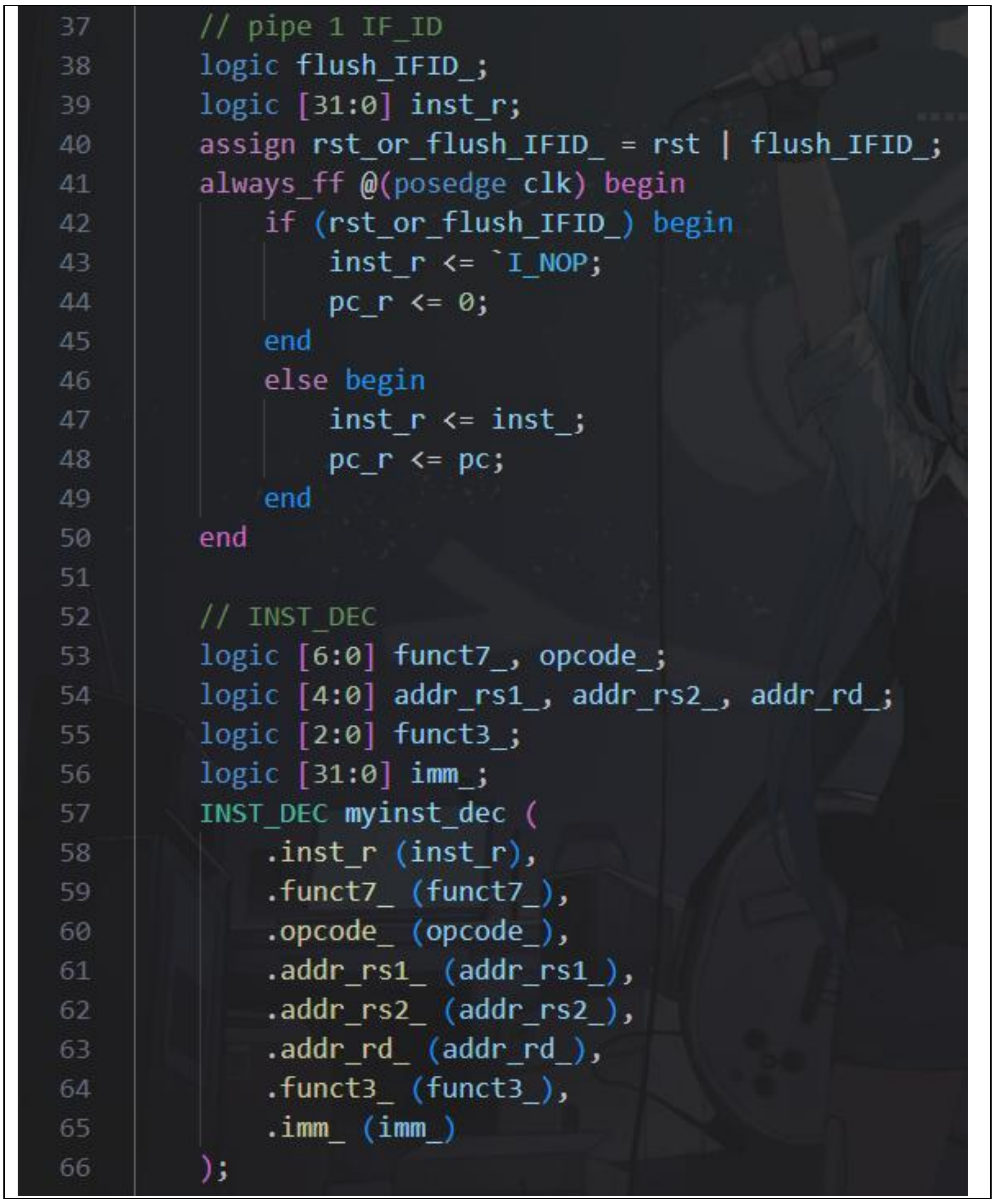
```systemverilog
`timescale 1ns/100ps
module seven_segment_display(
    input logic [3:0] digit,   // 4位元二進制數字輸入
    output logic [6:0] seg     // 7段顯示器輸出 (g, f, e, d, c, b, a)
);
    //共陽
    always_comb begin
        case (digit)
            4'h0: seg = 7'b1000000;     // 顯示 "0"
            4'h1: seg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

endmodule
```

Mycpu.sv

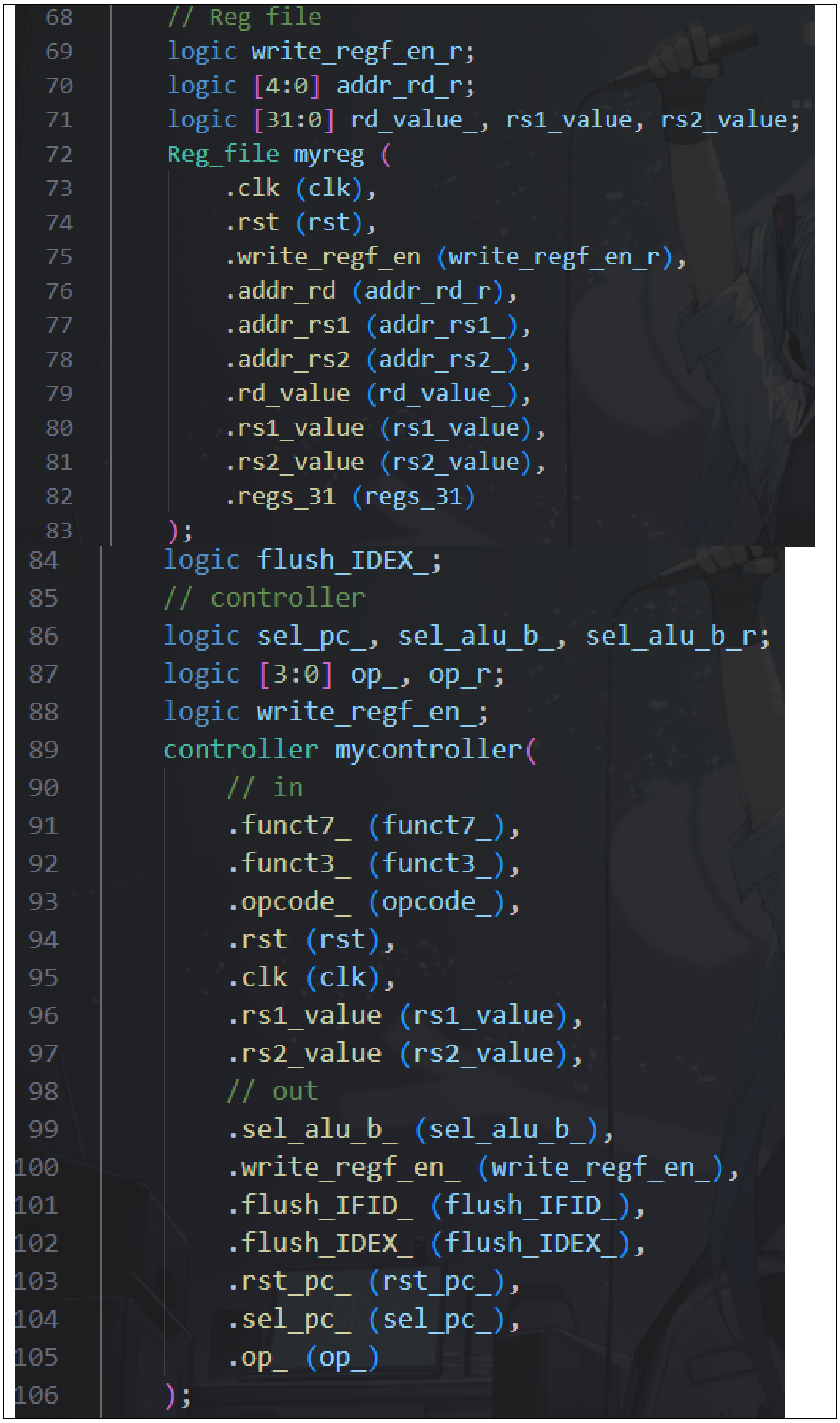


mycpu.sv > mycpu > [ ] alu_b_

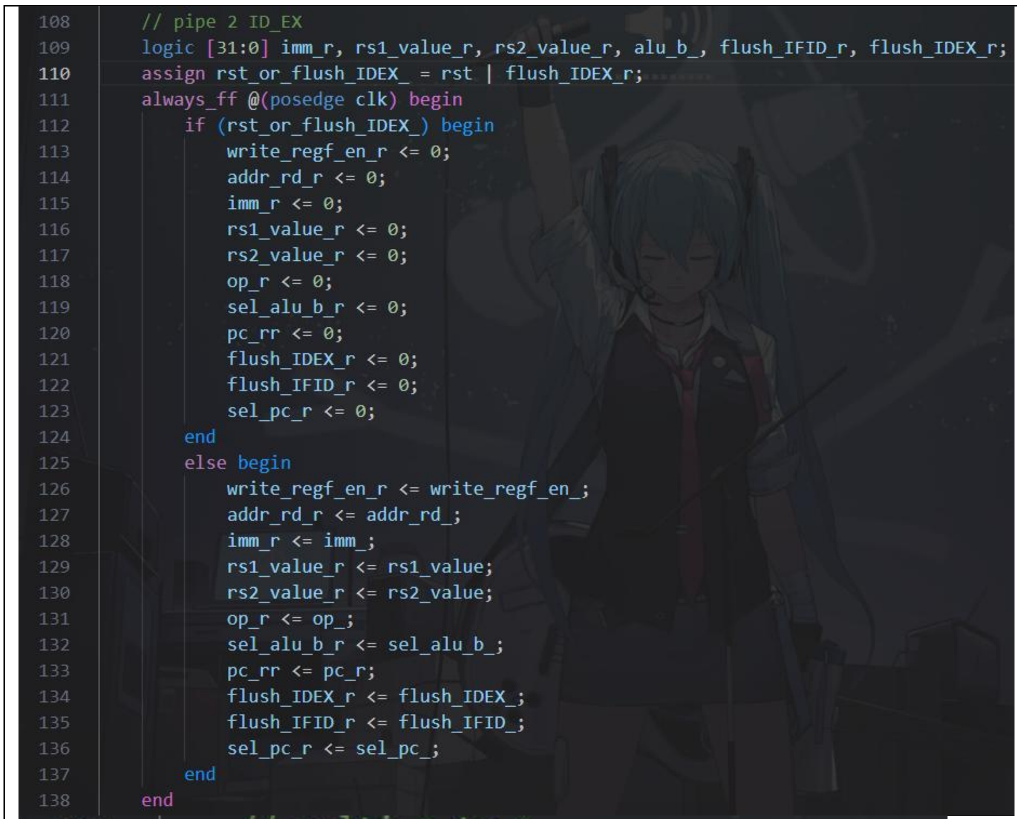
```
`include "mydefine.sv"
module mycpu(
    input logic clk, rst,
    output logic [31:0] regs_31
);
    // progrom-counter
    logic rst_pc_, sel_pc_r;
    logic [31:0] pc,pc_next, pc_r, pc_rr, jump_addr_;
    always_comb begin
        case (sel_pc_r)
            1'd0: begin
                pc_next = 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_)
    );
```



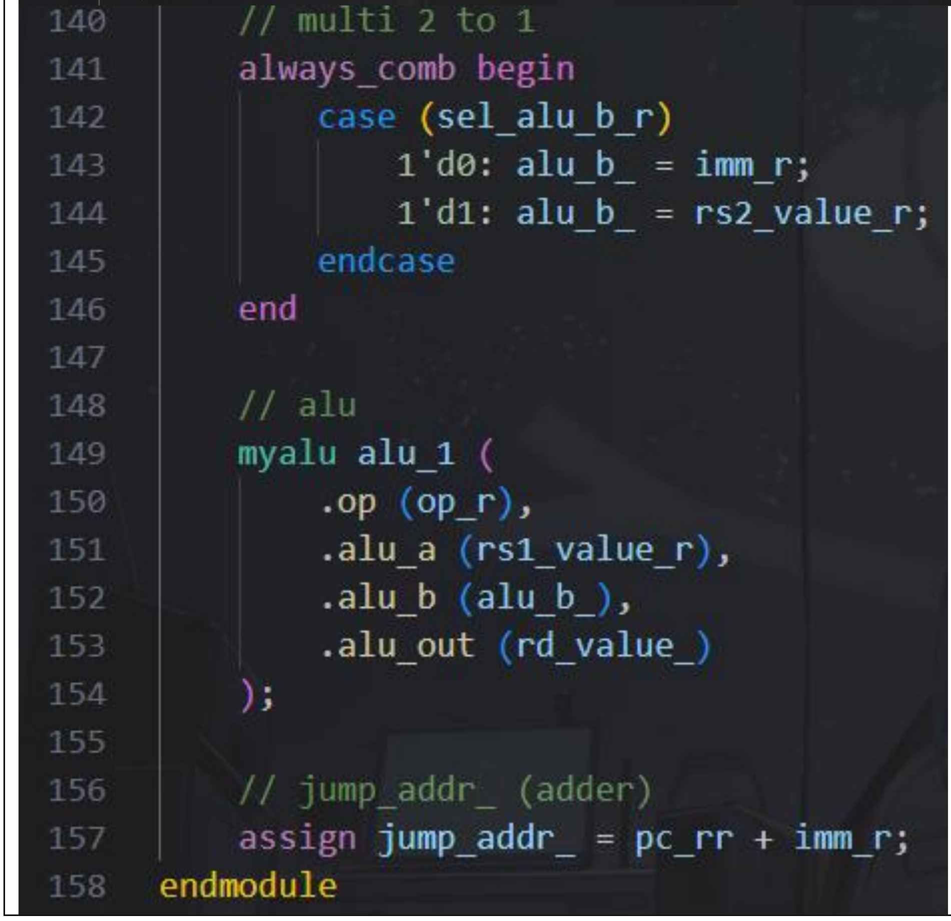
```
// pipe 1 IF_ID
logic flush_IFID_;
logic [31:0] inst_r;
assign rst_or_flush_IFID_ = rst | flush_IFID_;
always_ff @(posedge clk) begin
    if (rst_or_flush_IFID_) begin
        inst_r <= `I_NOP;
        pc_r <= 0;
    end
    else begin
        inst_r <= inst_;
        pc_r <= pc;
    end
end

// INST_DEC
logic [6:0] funct7_, opcode_;
logic [4:0] addr_rs1_, addr_rs2_, addr_rd_;
logic [2:0] funct3_;
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_)
);
```



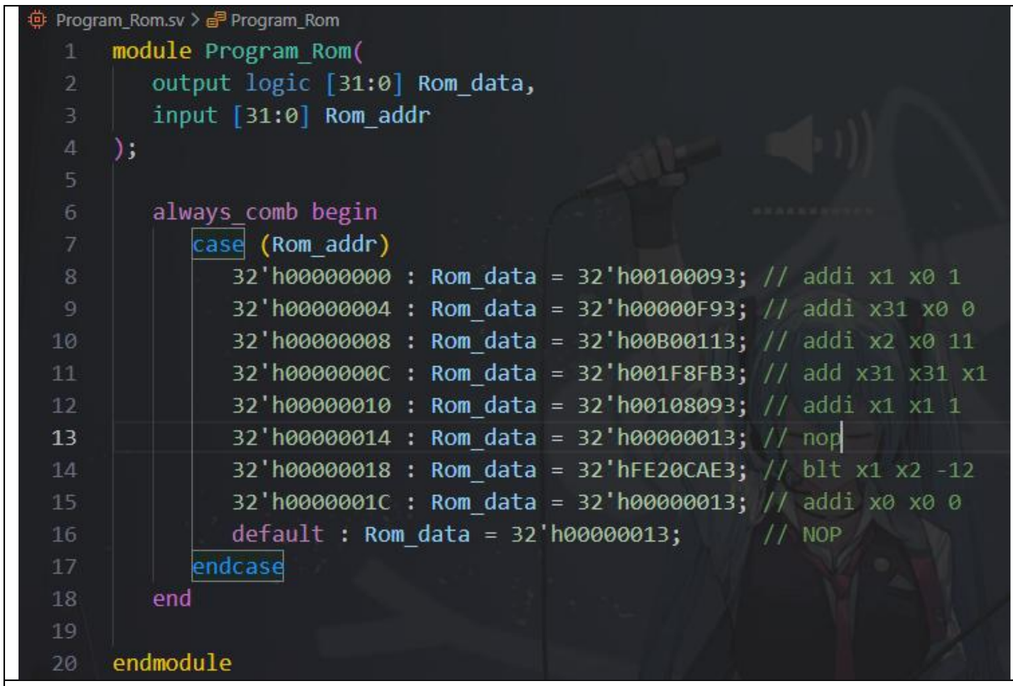
```
    // Reg file
    logic write_regf_en_r;
    logic [4:0] addr_rd_r;
    logic [31:0] rd_value_, rs1_value, rs2_value;
    Reg_file myreg (
        .clk (clk),
        .rst (rst),
        .write_regf_en (write_regf_en_r),
        .addr_rd (addr_rd_r),
        .addr_rs1 (addr_rs1_),
        .addr_rs2 (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_b_, sel_alu_b_r;
    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_)
    );
```



```
    // pipe 2 ID_EX
    logic [31:0] imm_r, rs1_value_r, rs2_value_r, alu_b_, flush_IFID_r, flush_IDEX_r;
    assign rst_or_flush_IDEX_ = rst | flush_IDEX_r;
    always_ff @(posedge clk) begin
        if (rst_or_flush_IDEX_) 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;
        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_;
        end
    end
```



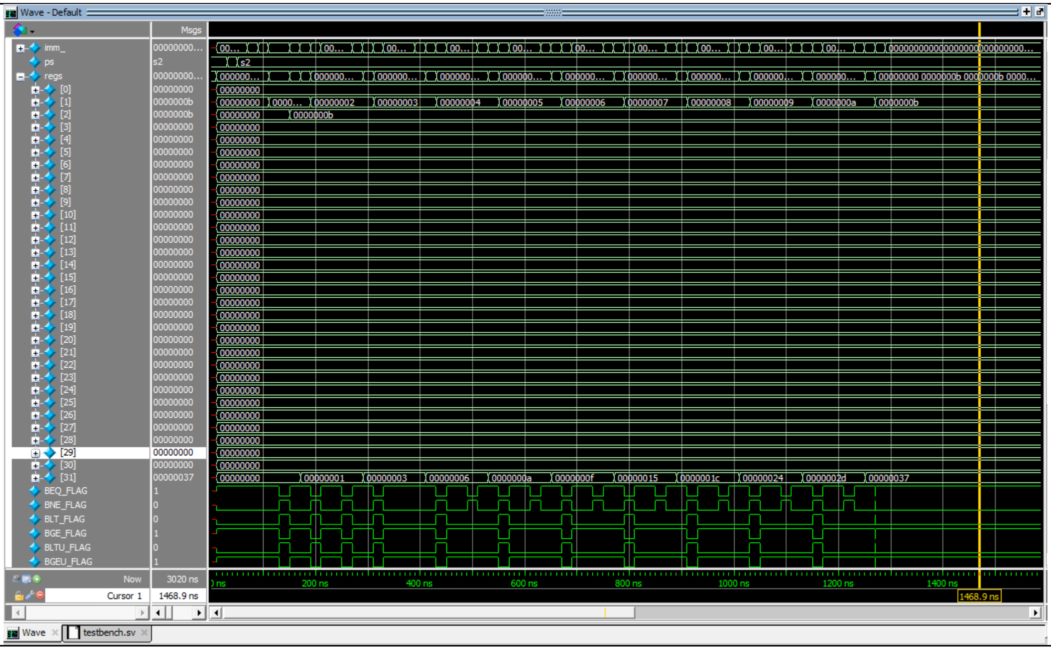
```
    // multi 2 to 1
    always_comb begin
        case (sel_alu_b_r)
            1'd0: alu_b_ = imm_r;
            1'd1: alu_b_ = rs2_value_r;
        endcase
    end

    // alu
    myalu alu_1 (
        .op (op_r),
        .alu_a (rs1_value_r),
        .alu_b (alu_b_),
        .alu_out (rd_value_)
    );

    // jump_addr_ (adder)
    assign jump_addr_ = pc_rr + imm_r;
endmodule
```



```
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'h00100093; // addi x1 x0 1
            32'h00000004 : Rom_data = 32'h00000F93; // addi x31 x0 0
            32'h00000008 : Rom_data = 32'h00B00113; // addi x2 x0 11
            32'h0000000C : Rom_data = 32'h001F8FB3; // add x31 x31 x1
            32'h00000010 : Rom_data = 32'h00108093; // addi x1 x1 1
            32'h00000014 : Rom_data = 32'h00000013; // nop
            32'h00000018 : Rom_data = 32'hFE20CAE3; // blt x1 x2 -12
            32'h0000001C : Rom_data = 32'h00000013; // addi x0 x0 0
            default : Rom_data = 32'h00000013;         // NOP
        endcase
    end

endmodule
```

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

●模擬結果與結果說明：

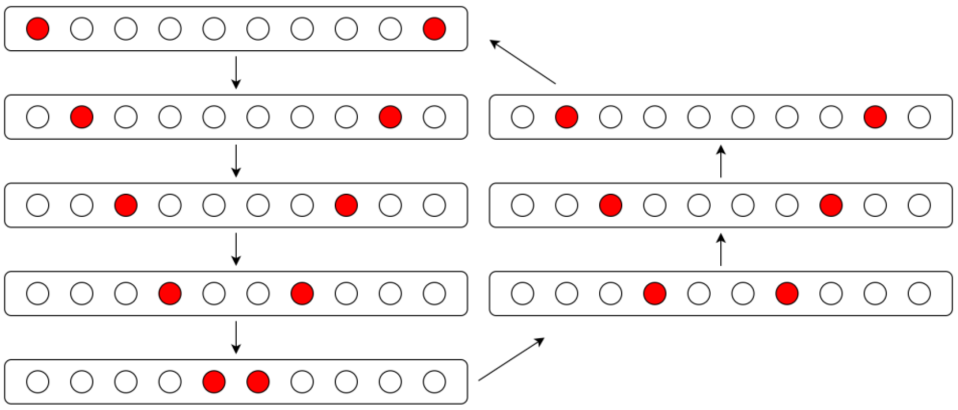


由 reg[31]可看出確實從 1 加至 55(Hex: 37)

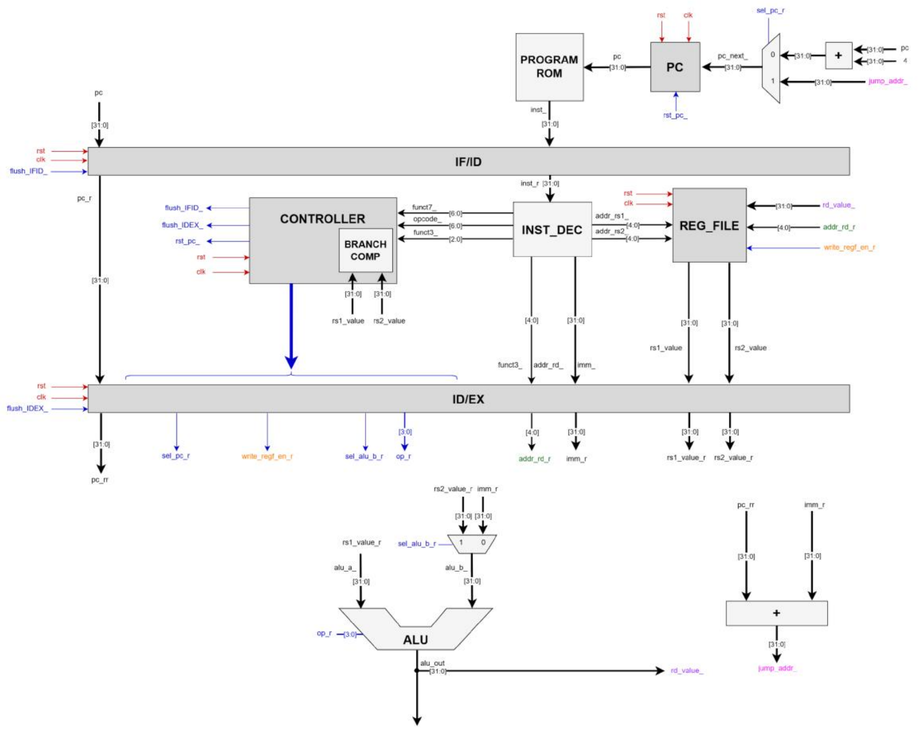
2、 組語撰寫練習—LED

●實驗說明：

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



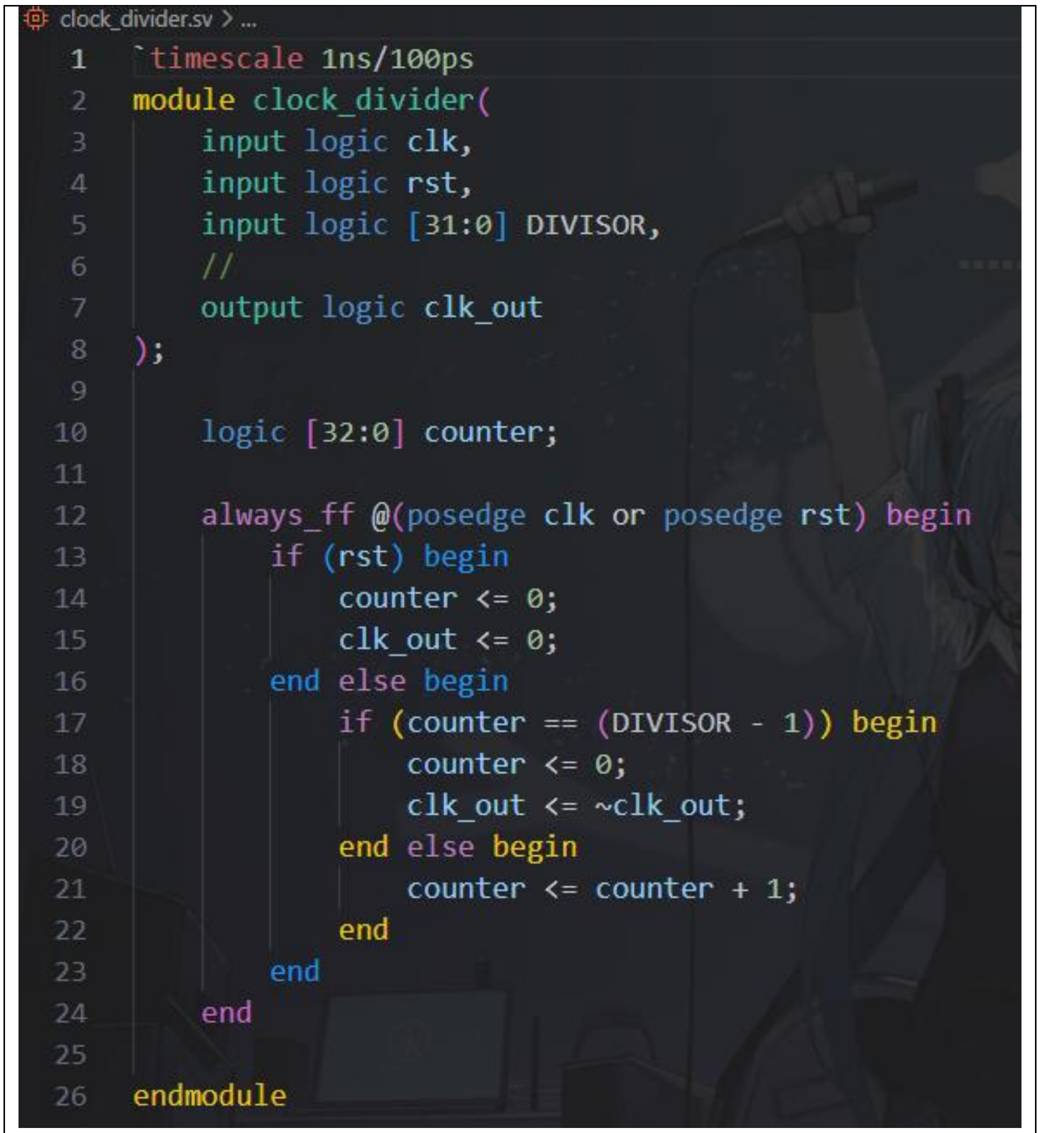
●系統硬體架構方塊圖（接線圖）：



●系統架構程式碼、測試資料程式碼與程式碼說明

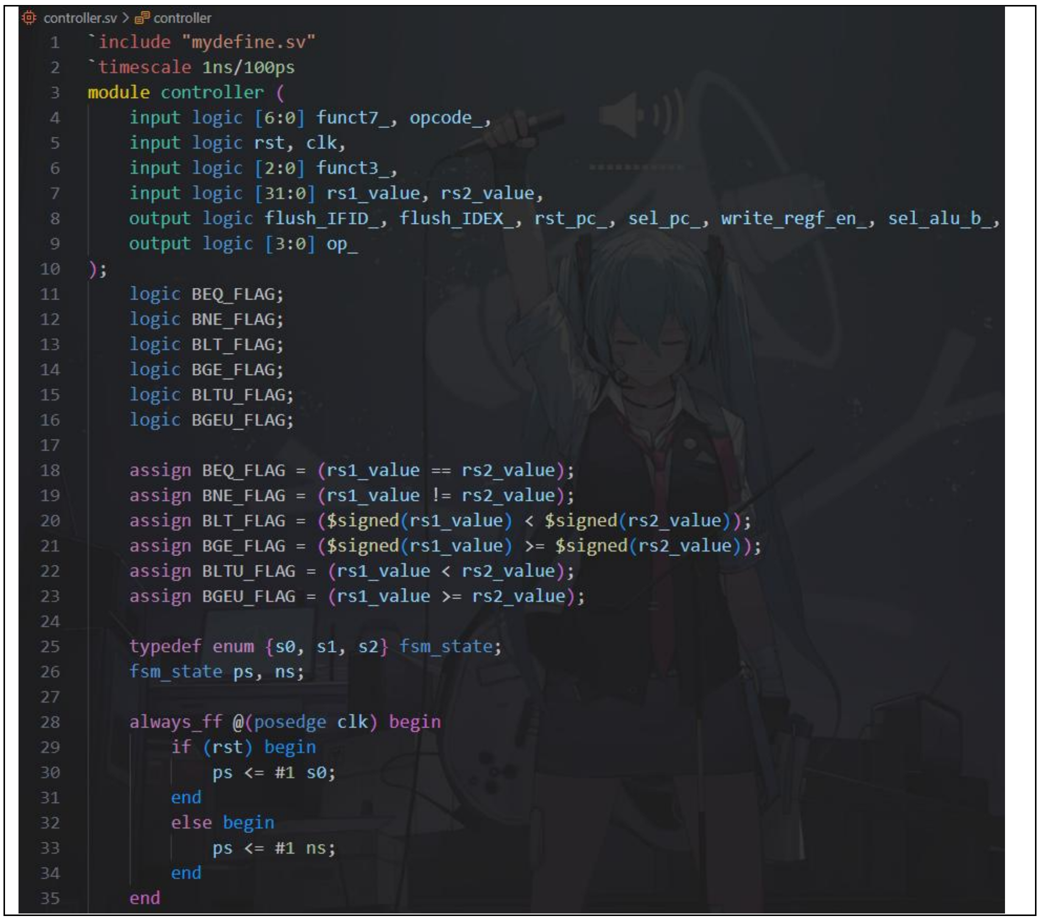
截圖請善用 **win+shift+S**

Clock_divider.sv

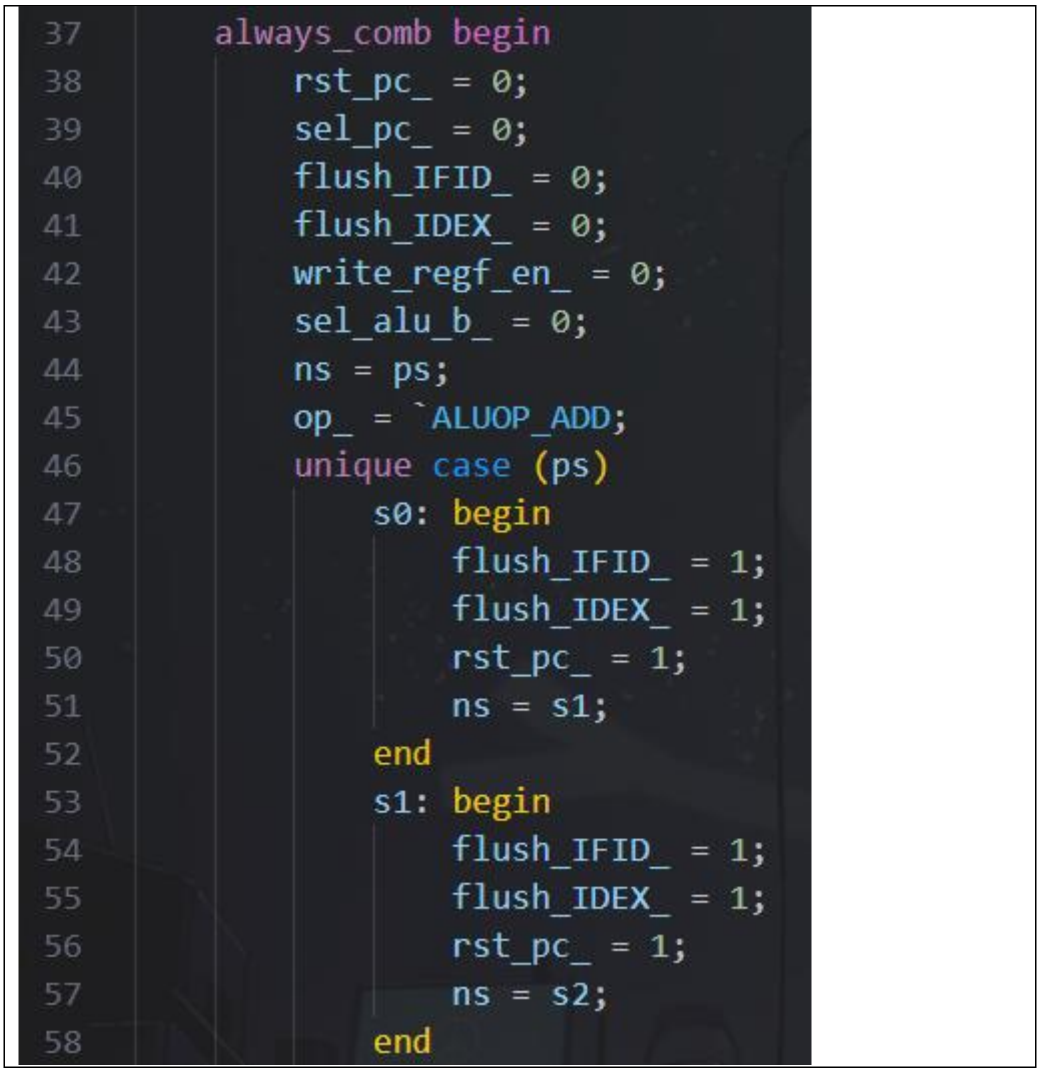

clock_divider.sv > ...

```
`timescale 1ns/100ps
module clock_divider(
    input logic clk,
    input logic rst,
    input logic [31:0] DIVISOR,
    //
    output logic clk_out
);

    logic [32:0] counter;

    always_ff @(posedge clk or posedge rst) begin
        if (rst) begin
            counter <= 0;
            clk_out <= 0;
        end else begin
            if (counter == (DIVISOR - 1)) begin
                counter <= 0;
                clk_out <= ~clk_out;
            end else begin
                counter <= counter + 1;
            end
        end
    end

endmodule
```

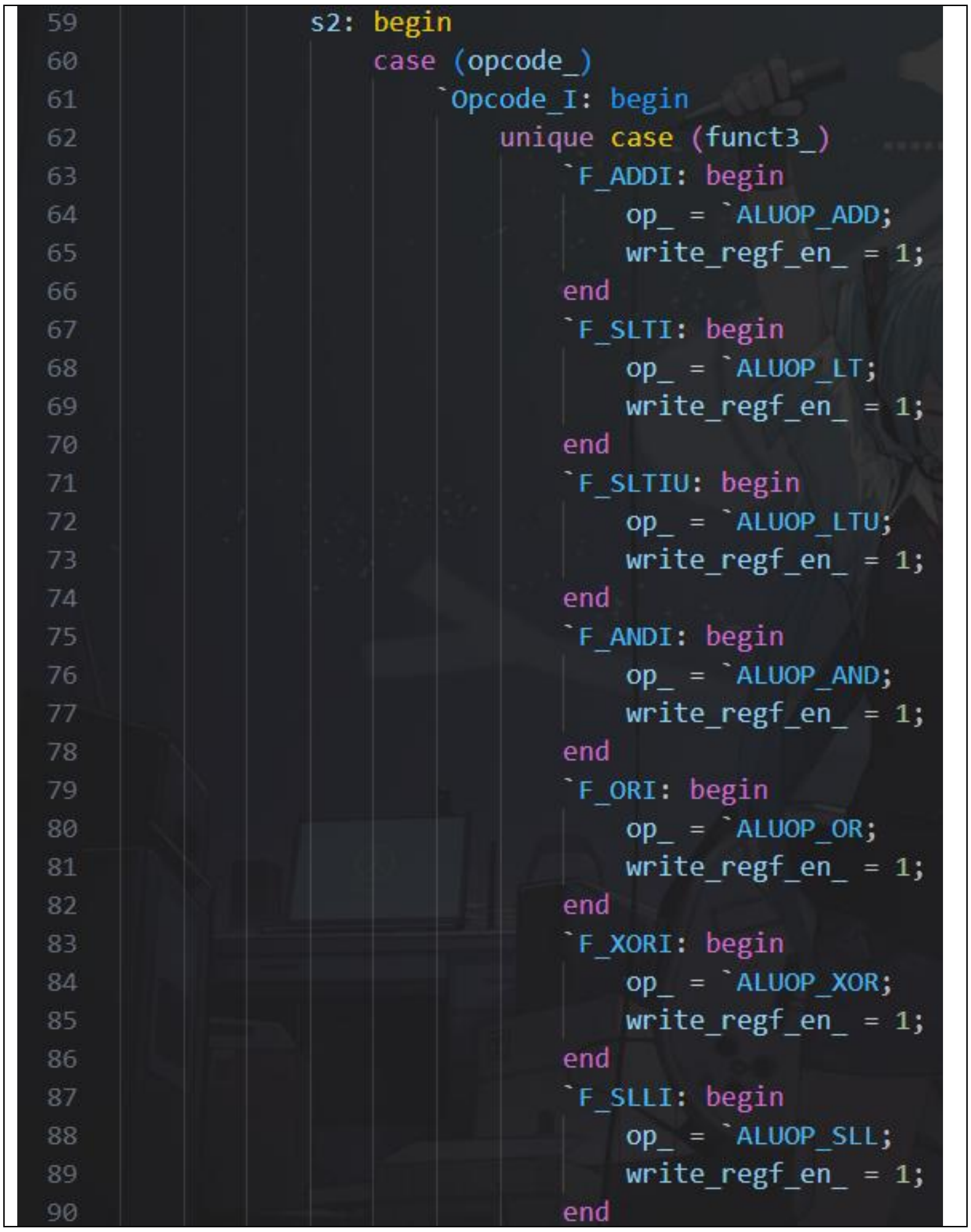
Controller.sv



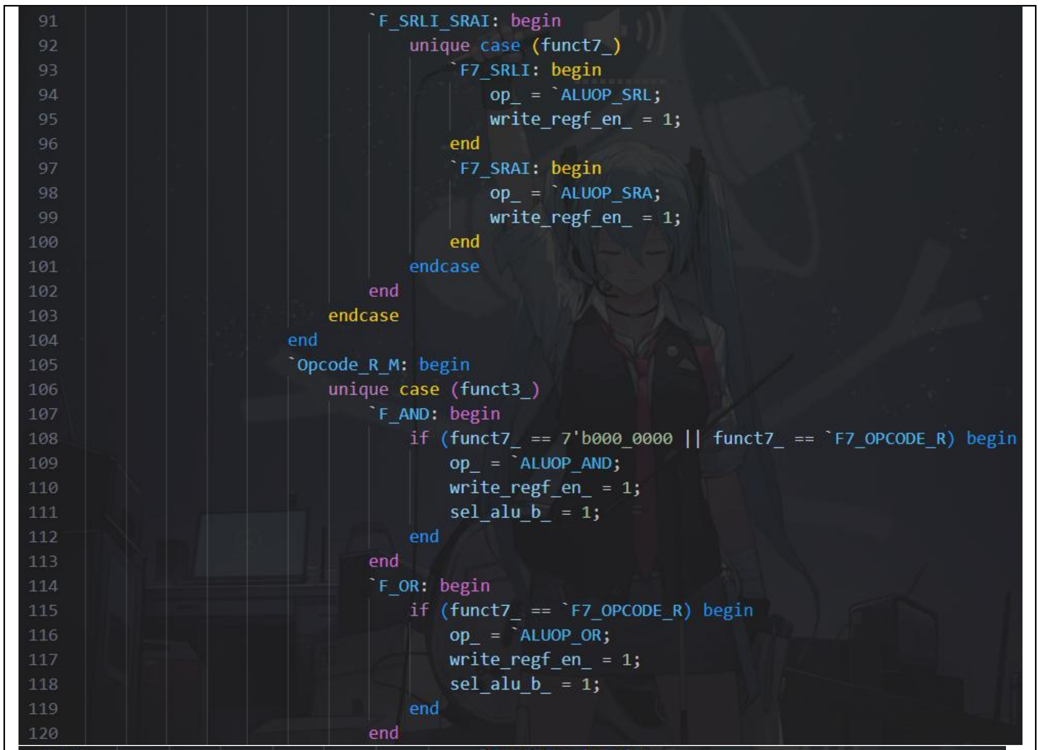
```
`include "mydefine.sv"
`timescale 1ns/100ps
module controller (
    input logic [6:0] funct7_, opcode_,
    input logic rst, clk,
    input logic [2:0] funct3_,
    input logic [31:0] rs1_value, rs2_value,
    output logic flush_IFID_, flush_IDEX_, rst_pc_, sel_pc_, write_regf_en_, sel_alu_b_,
 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



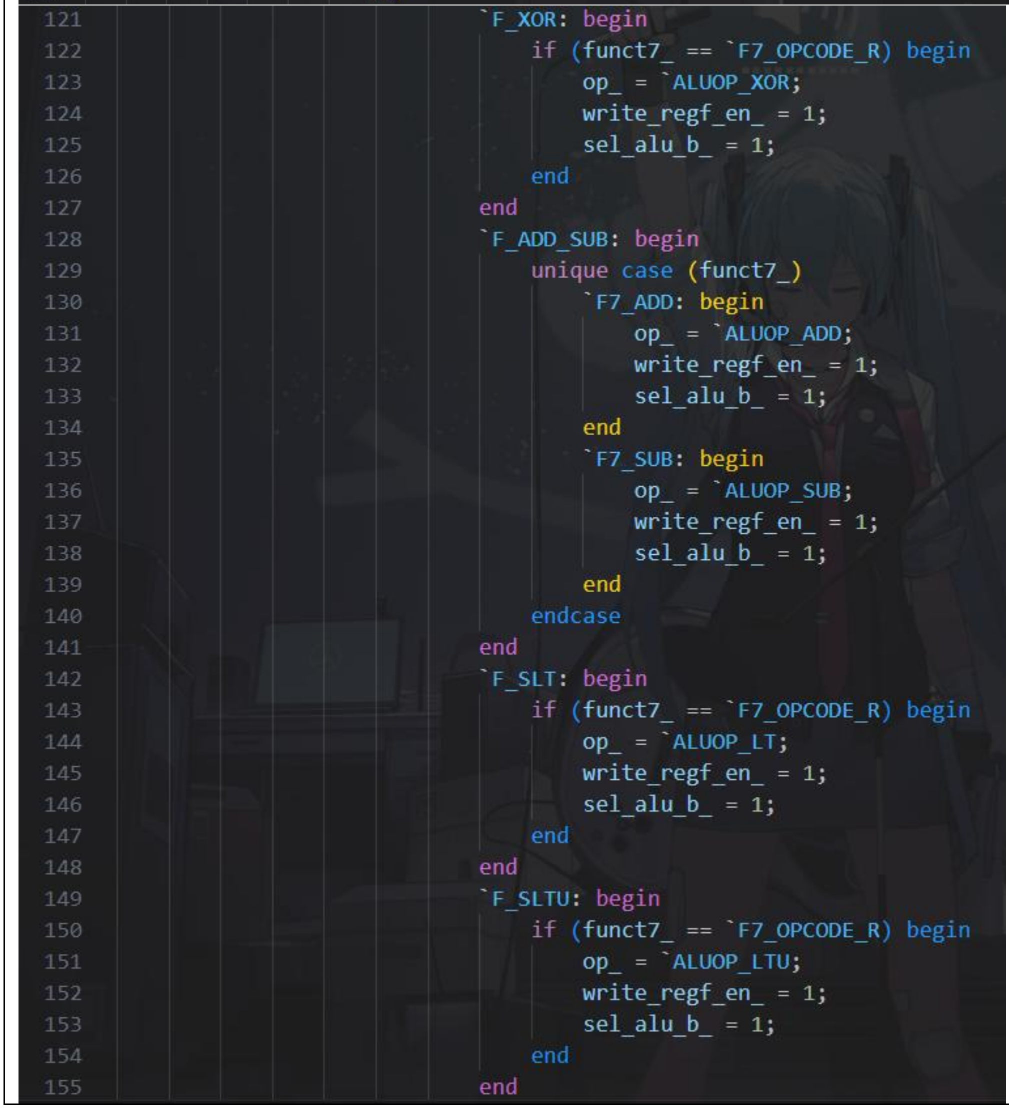
```
    always_comb begin
        rst_pc_ = 0;
        sel_pc_ = 0;
        flush_IFID_ = 0;
        flush_IDEX_ = 0;
        write_regf_en_ = 0;
        sel_alu_b_ = 0;
        ns = ps;
        op_ = `ALUOP_ADD;
        unique case (ps)
            s0: begin
                flush_IFID_ = 1;
                flush_IDEX_ = 1;
                rst_pc_ = 1;
                ns = s1;
            end
            s1: begin
                flush_IFID_ = 1;
                flush_IDEX_ = 1;
                rst_pc_ = 1;
                ns = s2;
            end
```



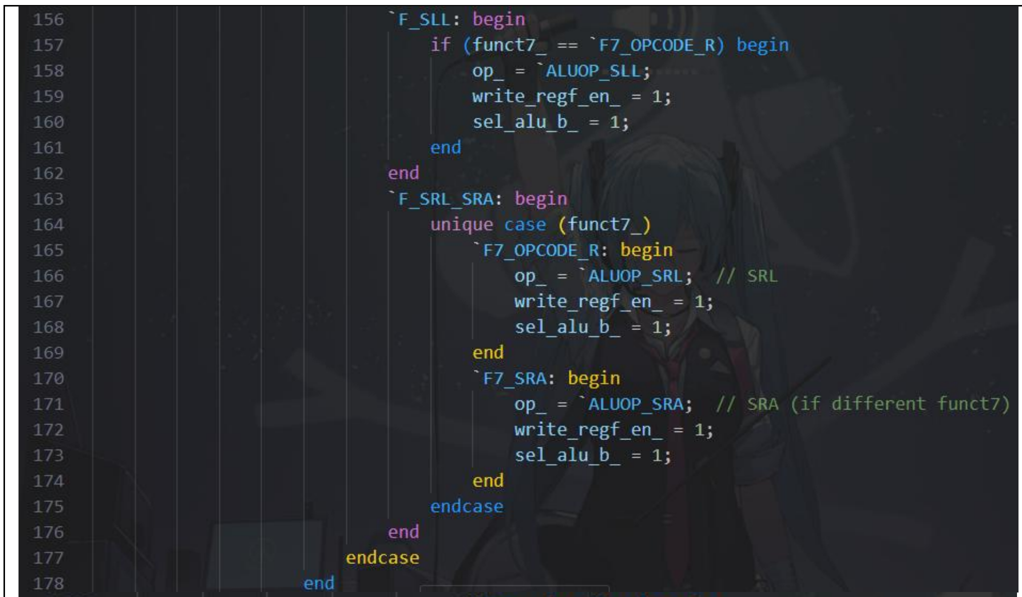
```
                s2: begin
                    case (opcode_)
                        `Opcode_I: begin
                            unique case (funct3_)
                                `F_ADDI: begin
                                    op_ = `ALUOP_ADD;
                                    write_regf_en_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



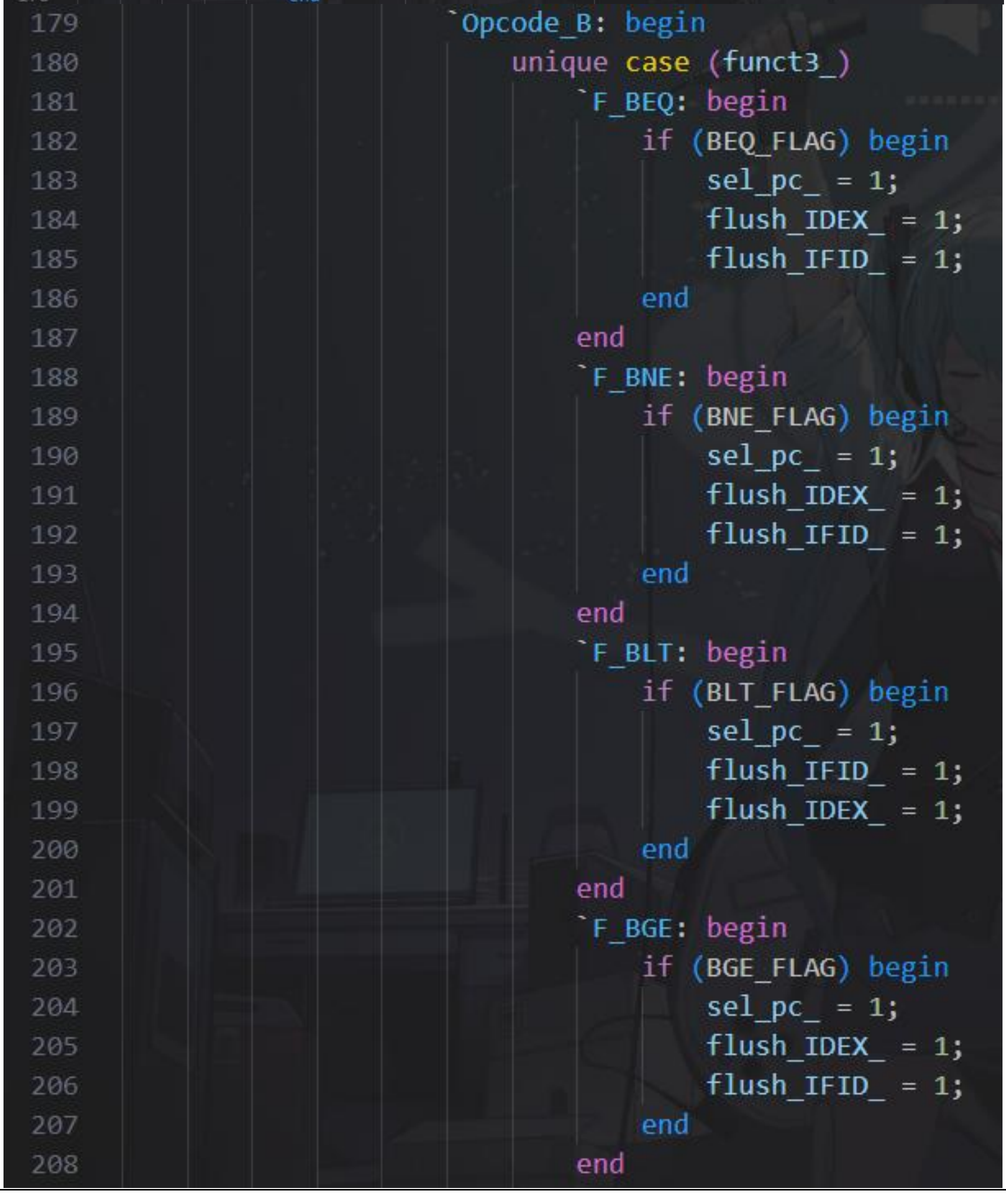
```
                            `F_SRLI_SRAI: begin
                                unique case (funct7_)
                                    `F7_SRLI: begin
                                        op_ = `ALUOP_SRL;
                                        write_regf_en_ = 1;
                                    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
```



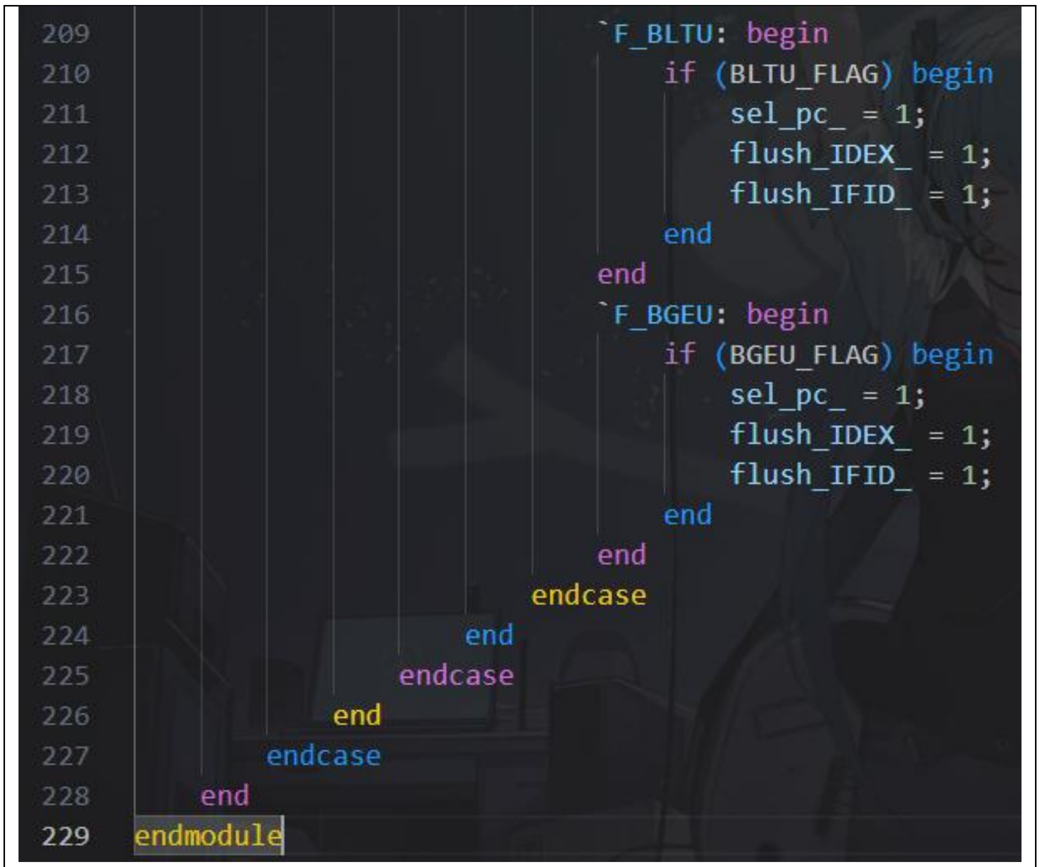
```
                            `F_XOR: begin
                                if (funct7_ == `F7_OPCODE_R) begin
                                    op_ = `ALUOP_XOR;
                                    write_regf_en_ = 1;
                                    sel_alu_b_ = 1;
                                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
```

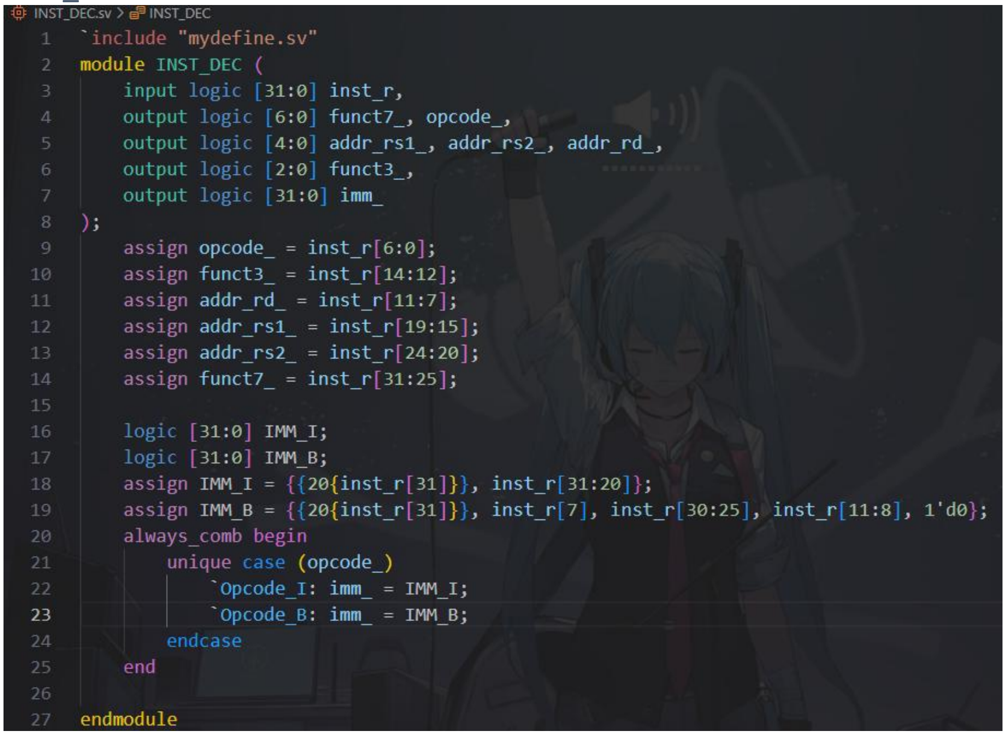


```
                    `Opcode_B: begin
                        unique case (funct3_)
                            `F_BEQ: begin
                                if (BEQ_FLAG) begin
                                    sel_pc_ = 1;
                                    flush_IDEX_ = 1;
                                    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
```



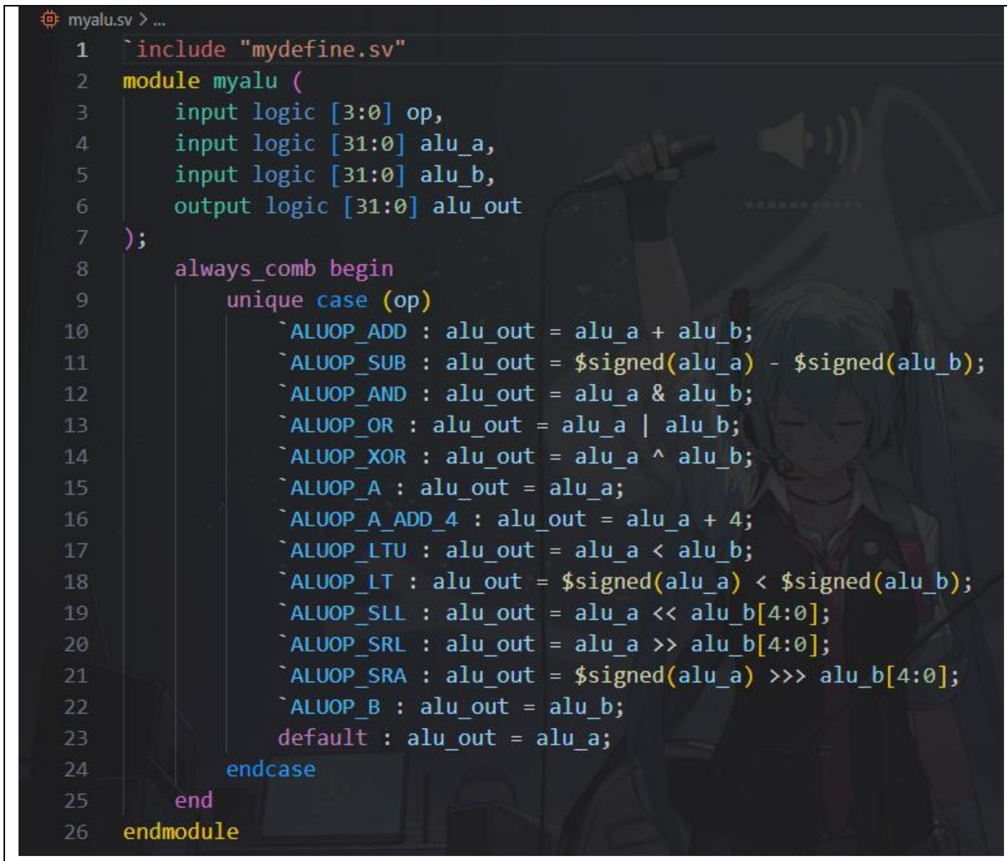
```
                                `F_BLTU: begin
                                    if (BLTU_FLAG) begin
                                        sel_pc_ = 1;
                                        flush_IDEX_ = 1;
                                        flush_IFID_ = 1;
                                    end
                                end
                                `F_BGEU: begin
                                    if (BGEU_FLAG) begin
                                        sel_pc_ = 1;
                                        flush_IDEX_ = 1;
                                        flush_IFID_ = 1;
                                    end
                                end
                            endcase
                        end
                    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;
    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'd0};
    always_comb begin
        unique case (opcode_)
            `Opcode_I: imm_ = IMM_I;
            `Opcode_B: imm_ = IMM_B;
        endcase
    end

endmodule
```

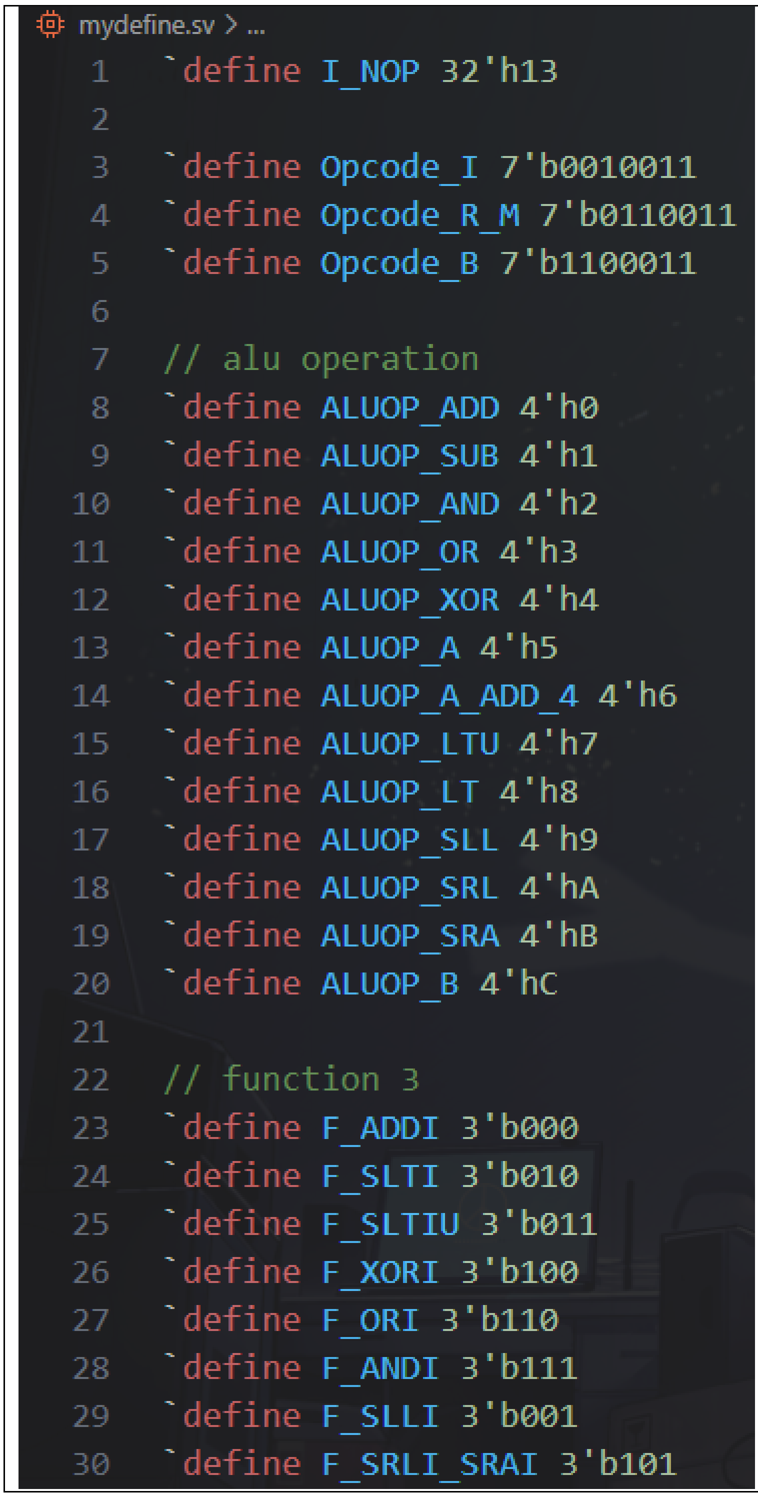
Myalu.sv



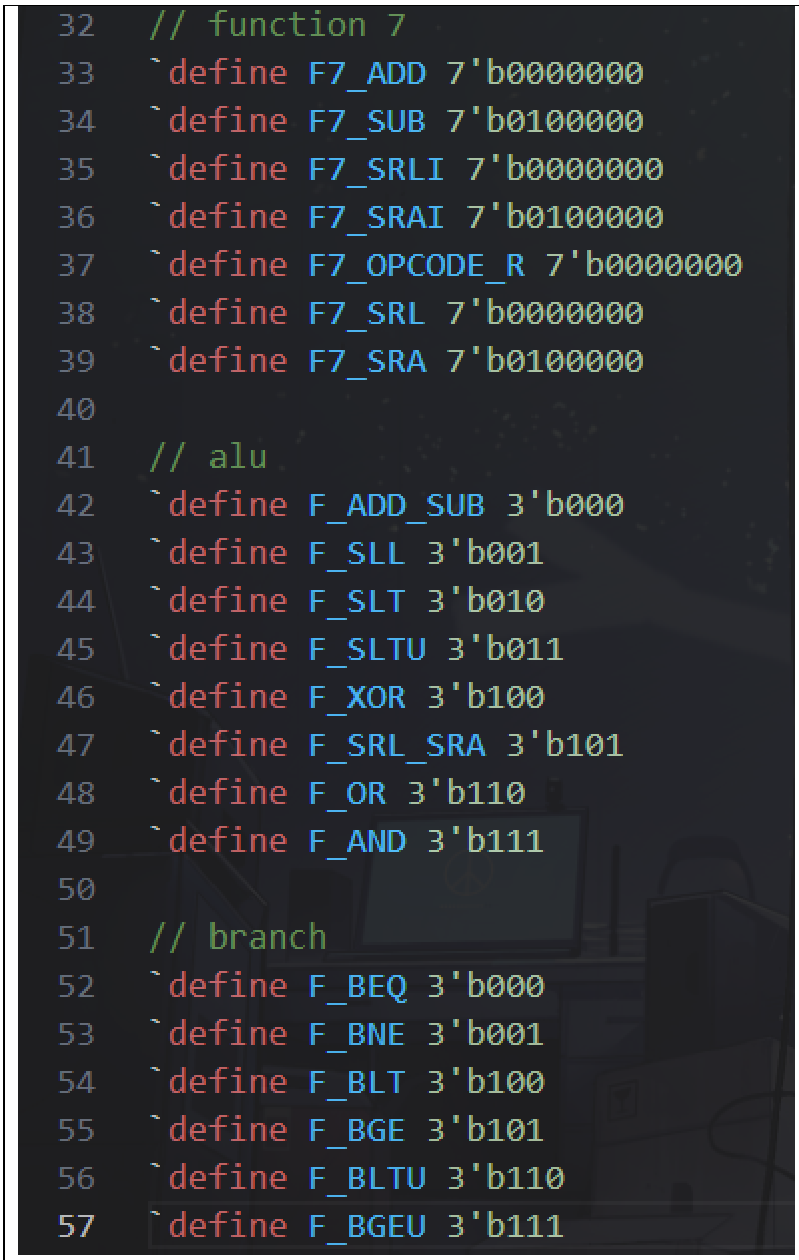
myalu.sv > ...

```
`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

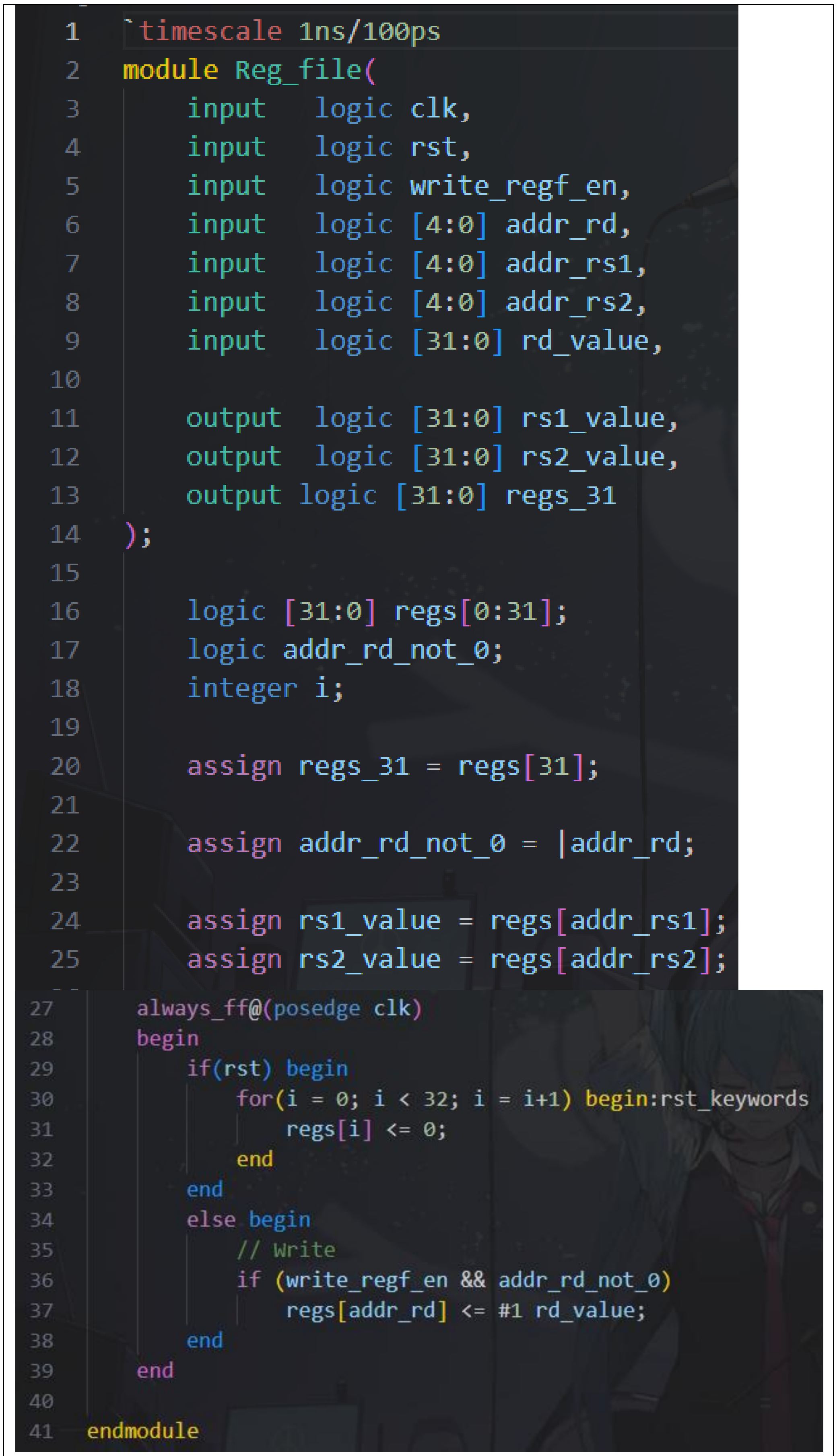

mydefine.sv > ...

```systemverilog
`define I_NOP 32'h13

`define Opcode_I 7'b0010011
`define Opcode_R_M 7'b0110011
`define Opcode_B 7'b1100011

// 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
```



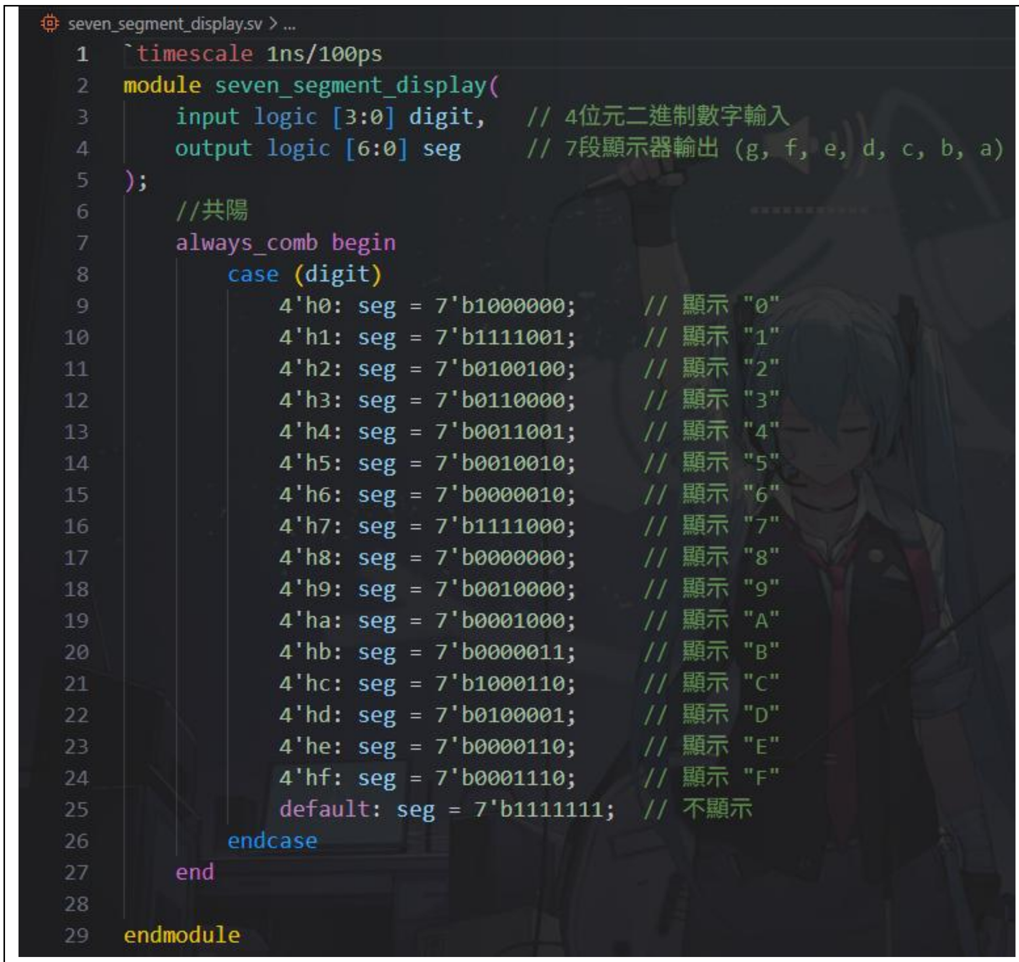
```
// function 7
`define F7_ADD 7'b0000000
`define F7_SUB 7'b0100000
`define F7_SRLI 7'b0000000
`define F7_SRAI 7'b0100000
`define F7_OPCODE_R 7'b0000000
`define F7_SRL 7'b0000000
`define F7_SRA 7'b0100000

// 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
```

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
```

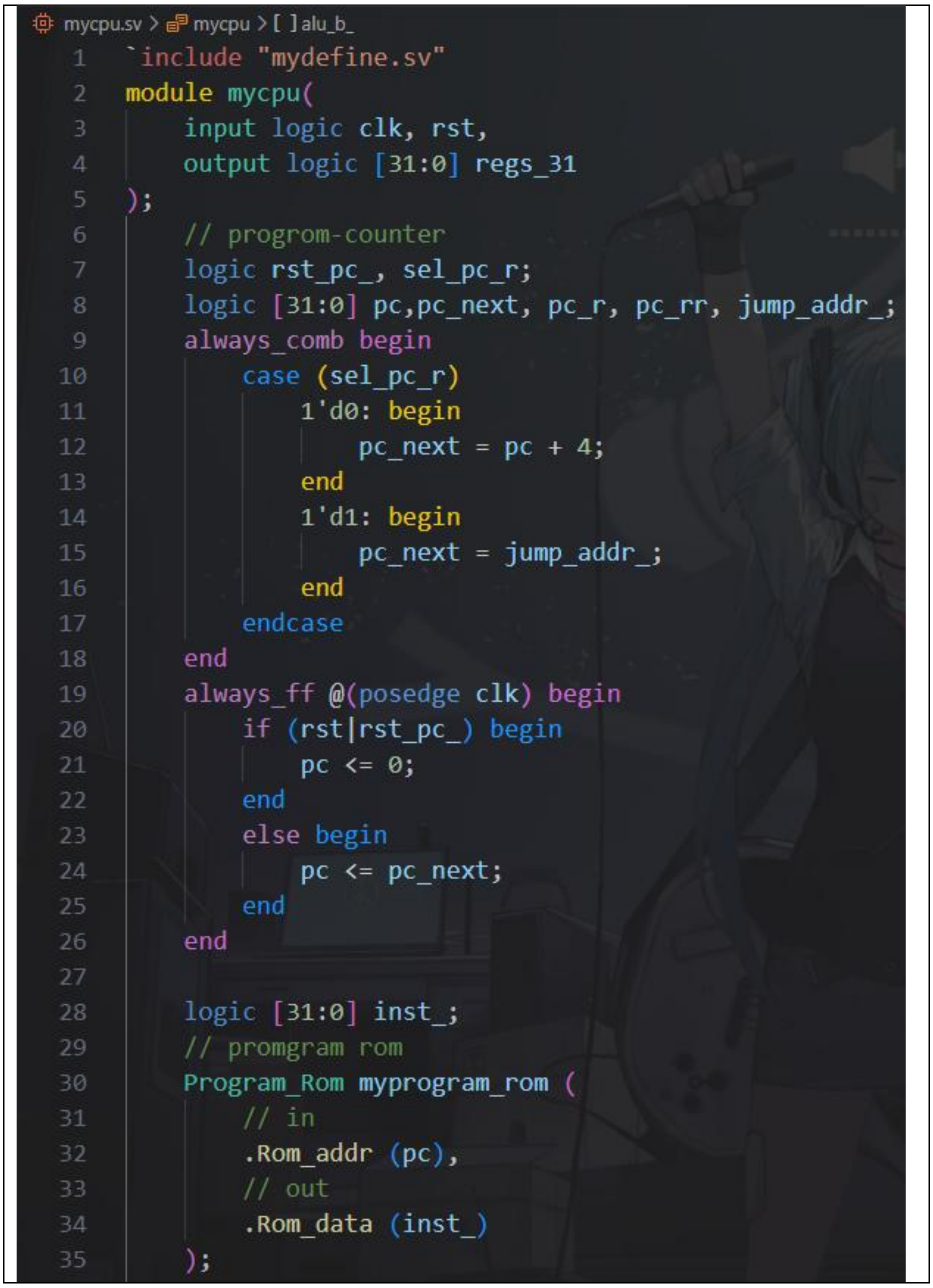
Seven_segment_display.sv



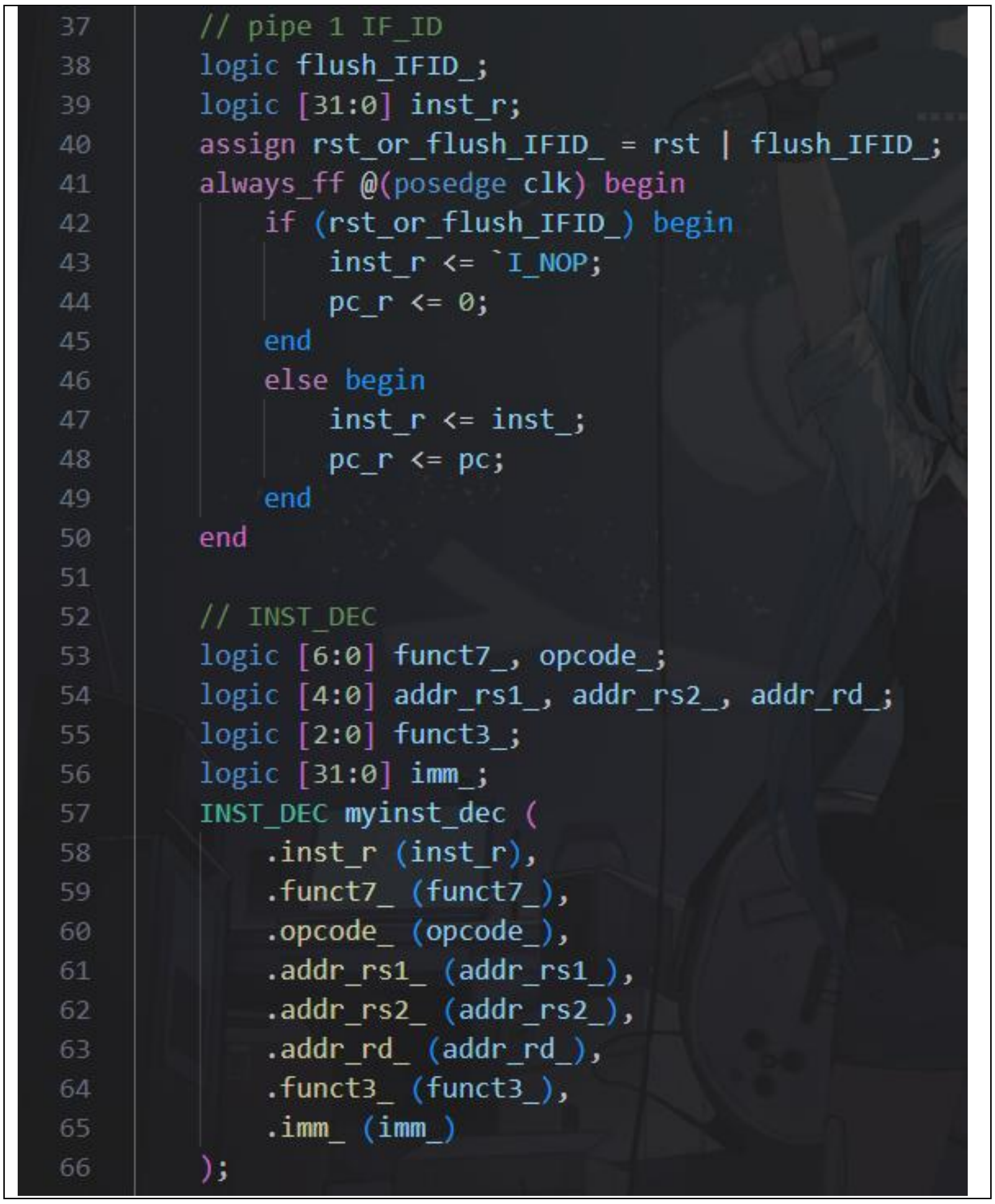
seven_segment_display.sv > ...

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

endmodule
```

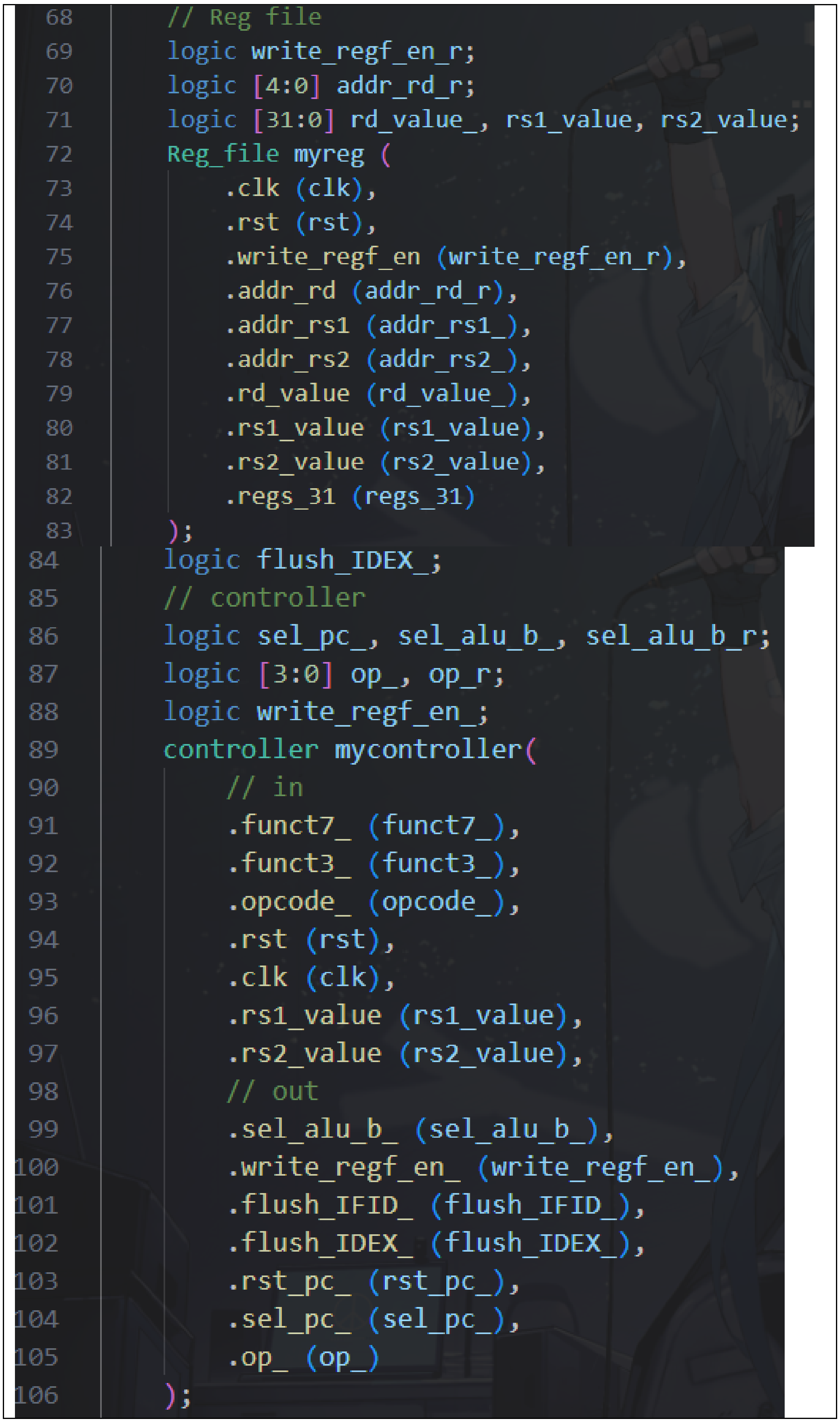
Mycpu.sv

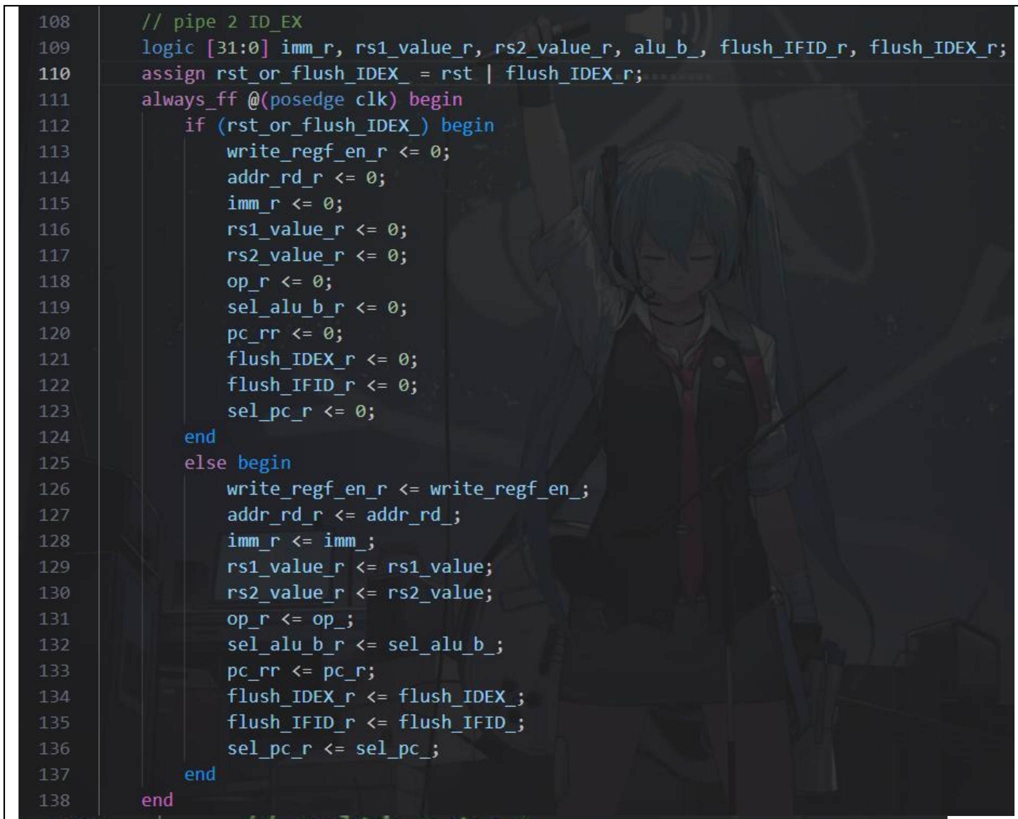

mycpu.sv > mycpu > [ ] alu_b_

```
`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_)
    );
```

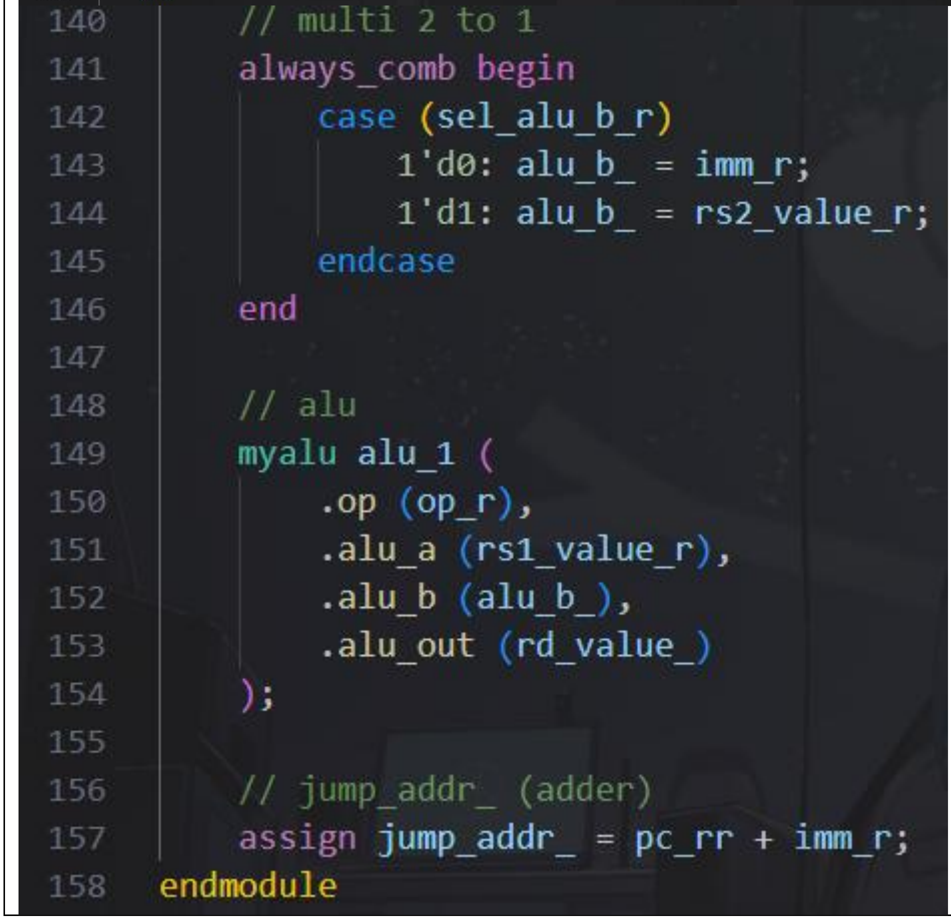


```
    // pipe 1 IF_ID
    logic flush_IFID_;
    logic [31:0] inst_r;
    assign rst_or_flush_IFID_ = rst | flush_IFID_;
    always_ff @(posedge clk) begin
        if (rst_or_flush_IFID_) begin
            inst_r <= `I_NOP;
            pc_r <= 0;
        end
        else begin
            inst_r <=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

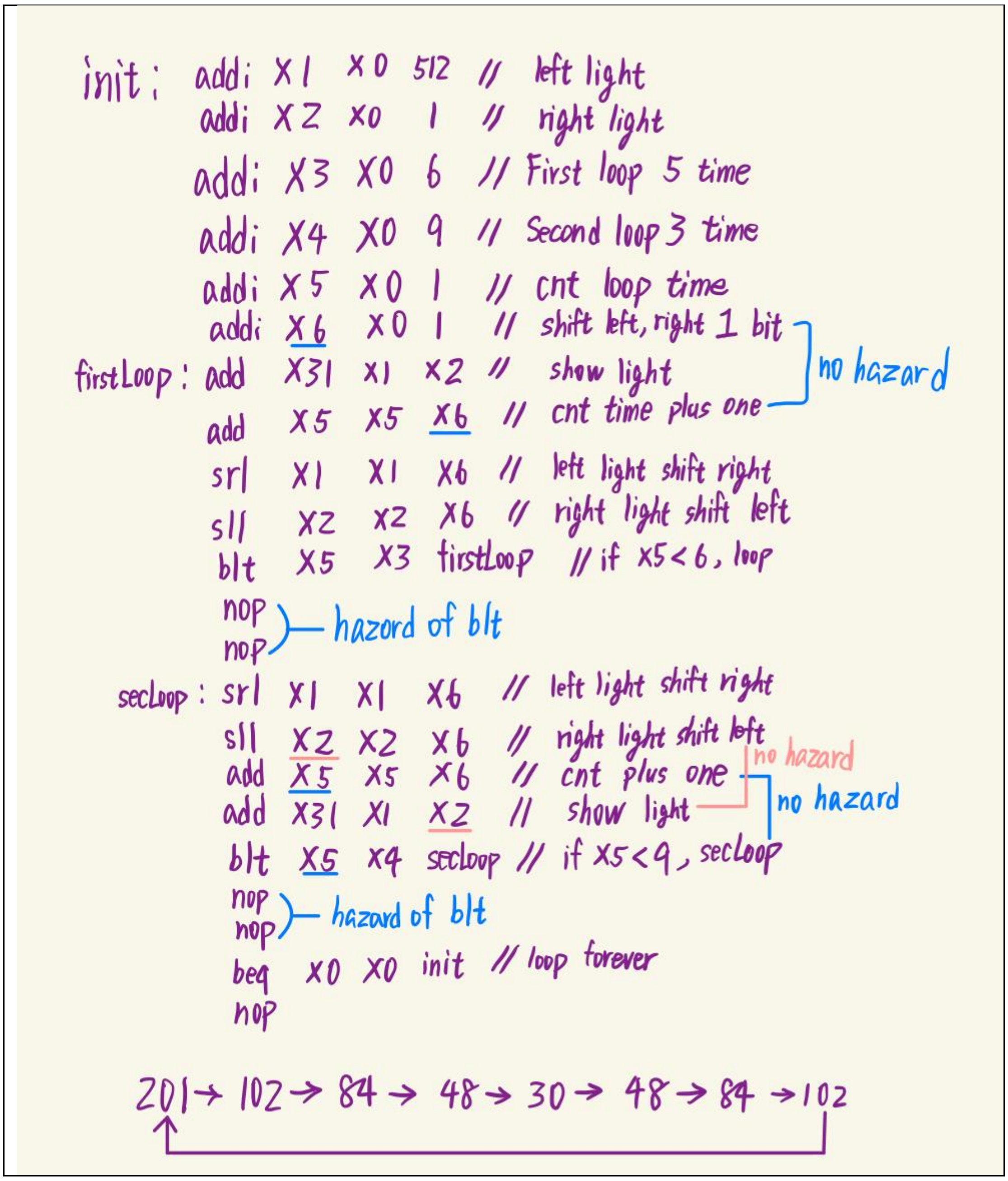

```
    // Reg file
    logic write_regf_en_r;
    logic [4:0] addr_rd_r;
    logic [31:0] rd_value_, rs1_value, rs2_value;
    Reg_file myreg (
        .clk (clk),
        .rst (rst),
        .write_regf_en (write_regf_en_r),
        .addr_rd (addr_rd_r),
        .addr_rs1 (addr_rs1_),
        .addr_rs2 (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_b_, sel_alu_b_r;
    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_)
    );
```



```
    // pipe 2 ID_EX
    logic [31:0] imm_r, rs1_value_r, rs2_value_r, alu_b_, flush_IFID_r, flush_IDEX_r;
    assign rst_or_flush_IDEX_ = rst | flush_IDEX_r;
    always_ff @(posedge clk) begin
        if (rst_or_flush_IDEX_) 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;
        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_;
        end
    end
```

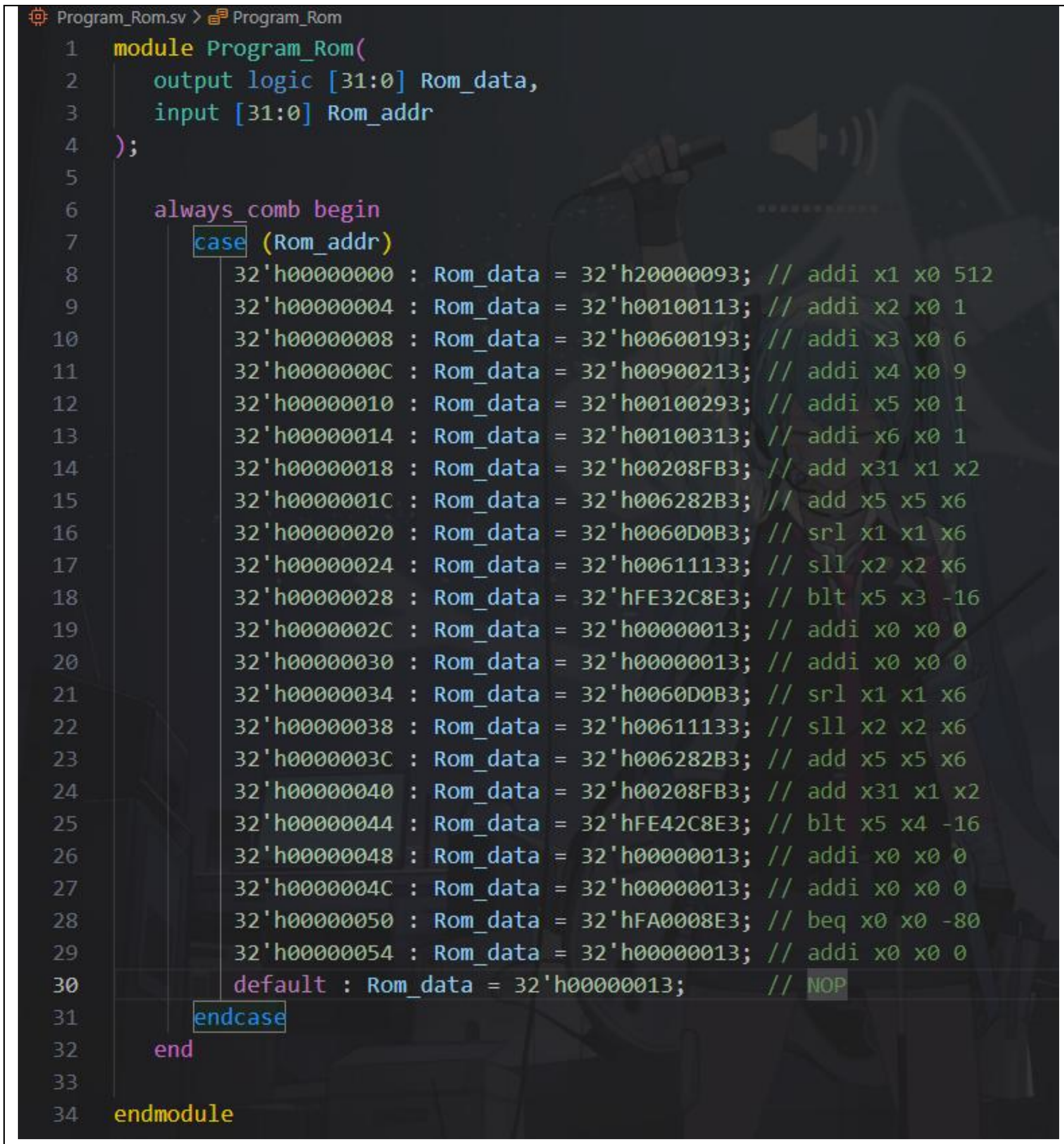


```
    // multi 2 to 1
    always_comb begin
        case (sel_alu_b_r)
            1'd0: alu_b_ = imm_r;
            1'd1: alu_b_ = rs2_value_r;
        endcase
    end

    // alu
    myalu alu_1 (
        .op (op_r),
        .alu_a (rs1_value_r),
        .alu_b (alu_b_),
        .alu_out (rd_value_)
    );

    // jump_addr_ (adder)
    assign jump_addr_ = pc_rr + imm_r;
endmodule
```



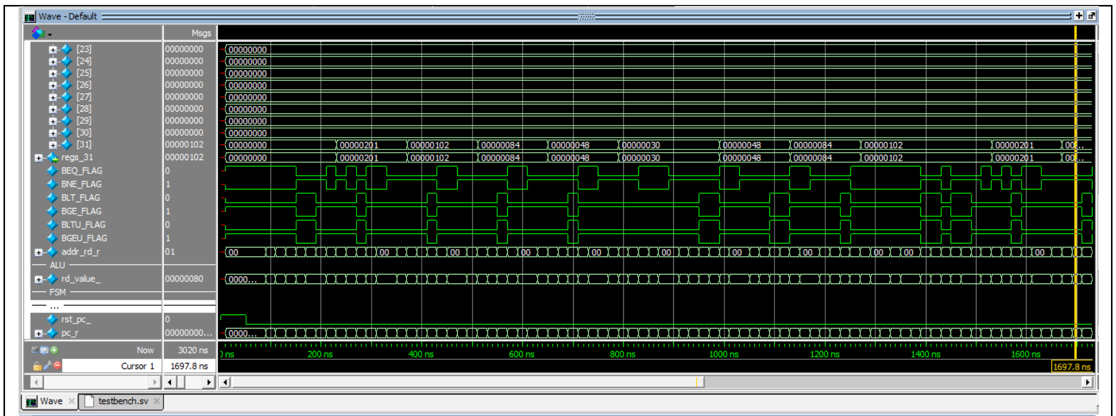
```
init:      addi X1  X0  512   // left light
           addi X2  X0  1     // right light
           addi X3  X0  6     // First loop 5 time
           addi X4  X0  9     // Second loop 3 time
           addi X5  X0  1     // cnt loop time
           addi X6  X0  1     // shift left, right 1 bit  ┐
firstLoop: add  X31 X1  X2    // show light               │ no hazard
           add  X5  X5  X6    // cnt time plus one        ┘
           srl  X1  X1  X6    // left light shift right
           sll  X2  X2  X6    // right light shift left
           blt  X5  X3  firstLoop  // if X5<6, loop
           nop  ) hazard of blt
           nop  )
secLoop:   srl  X1  X1  X6    // left light shift right
           sll  X2  X2  X6    // right light shift left
           add  X5  X5  X6    // cnt plus one   ┐ no hazard ┐
           add  X31 X1  X2    // show light     ┘           │ no hazard
           blt  X5  X4  secloop  // if X5<9, secLoop        ┘
           nop  ) hazard of blt
           nop  )
           beq  X0  X0  init  // loop forever
           nop
```

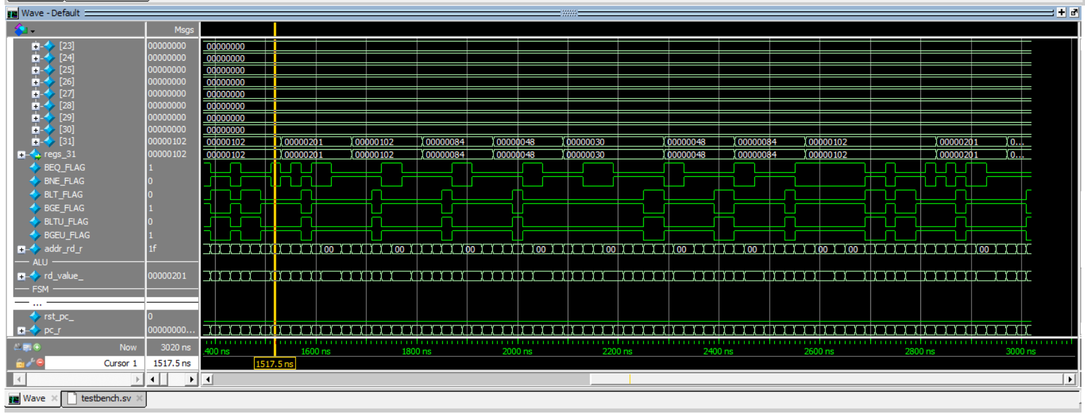
$$\begin{tikzcd} 201 \arrow[r] & 102 \arrow[r] & 84 \arrow[r] & 48 \arrow[r] & 30 \arrow[r] & 48 \arrow[r] & 84 \arrow[r] & 102 \arrow[lllllll, bend left] \end{tikzcd}$$


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

    always_comb begin
        case (Rom_addr)
            32'h00000000 : Rom_data = 32'h20000093; // addi x1 x0 512
            32'h00000004 : Rom_data = 32'h00100113; // addi x2 x0 1
            32'h00000008 : Rom_data = 32'h00600193; // addi x3 x0 6
            32'h0000000C : Rom_data = 32'h00900213; // addi x4 x0 9
            32'h00000010 : Rom_data = 32'h00100293; // addi x5 x0 1
            32'h00000014 : Rom_data = 32'h00100313; // addi x6 x0 1
            32'h00000018 : Rom_data = 32'h00208FB3; // add x31 x1 x2
            32'h0000001C : Rom_data = 32'h006282B3; // add x5 x5 x6
            32'h00000020 : Rom_data = 32'h0060D0B3; // srl x1 x1 x6
            32'h00000024 : Rom_data = 32'h00611133; // sll x2 x2 x6
            32'h00000028 : Rom_data = 32'hFE32C8E3; // blt x5 x3 -16
            32'h0000002C : Rom_data = 32'h00000013; // addi x0 x0 0
            32'h00000030 : Rom_data = 32'h00000013; // addi x0 x0 0
            32'h00000034 : Rom_data = 32'h0060D0B3; // srl x1 x1 x6
            32'h00000038 : Rom_data = 32'h00611133; // sll x2 x2 x6
            32'h0000003C : Rom_data = 32'h006282B3; // add x5 x5 x6
            32'h00000040 : Rom_data = 32'h00208FB3; // add x31 x1 x2
            32'h00000044 : Rom_data = 32'hFE42C8E3; // blt x5 x4 -16
            32'h00000048 : Rom_data = 32'h00000013; // addi x0 x0 0
            32'h0000004C : Rom_data = 32'h00000013; // addi x0 x0 0
            32'h00000050 : Rom_data = 32'hFA0008E3; // beq x0 x0 -80
            32'h00000054 : Rom_data = 32'h00000013; // addi x0 x0 0
            default : Rom_data = 32'h00000013;      // NOP
        endcase
    end

endmodule
```

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

●模擬結果與結果說明：





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

●結論與心得：

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