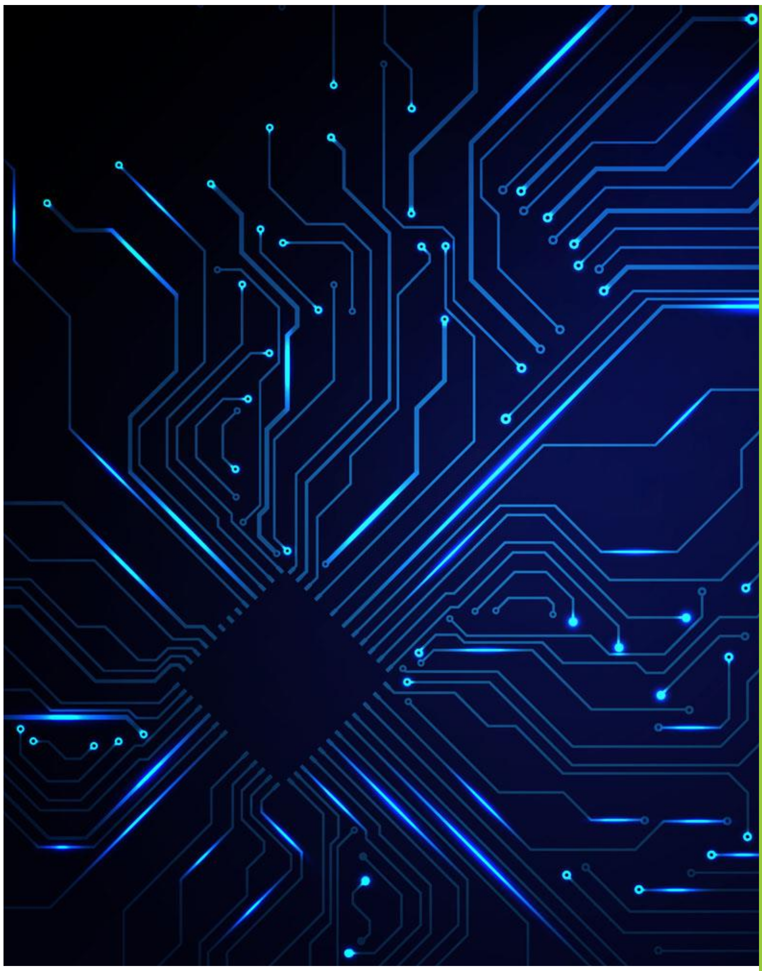


注意

1. 繳交時一律轉 PDF 檔
2. 一人繳交一份
3. 檔名：學號_HW?.pdf
   檔名請按照作業檔名格式進行填寫
   未依照格式不予批改

2022/12/20

實驗十五

PIPELINE

姓名：張銀軒　　　學號：00957050

班級：資工 3A

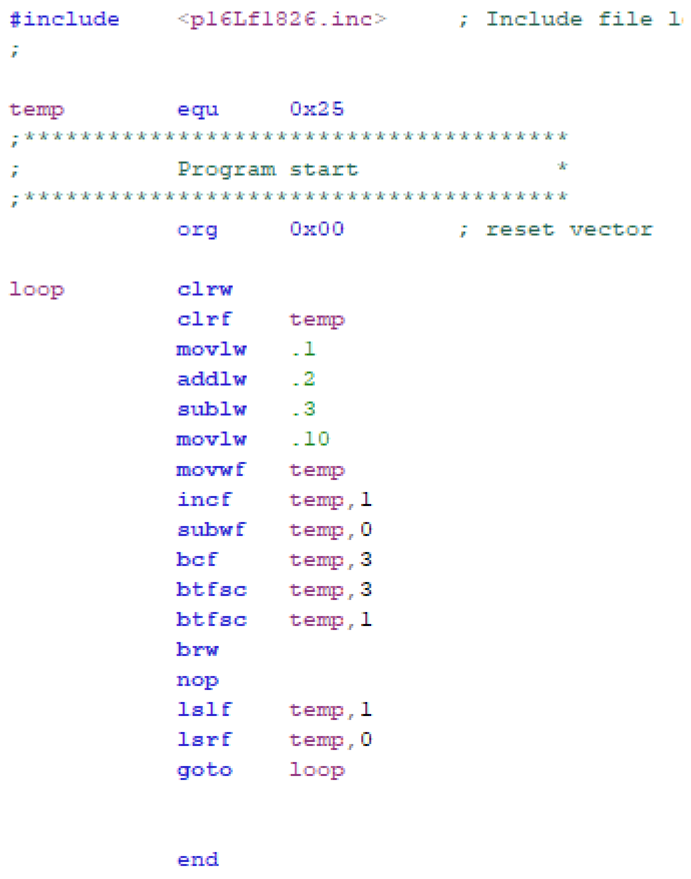
E-mail：00957050@mail.ntou.edu.tw

2022/12/20

● 實驗說明：

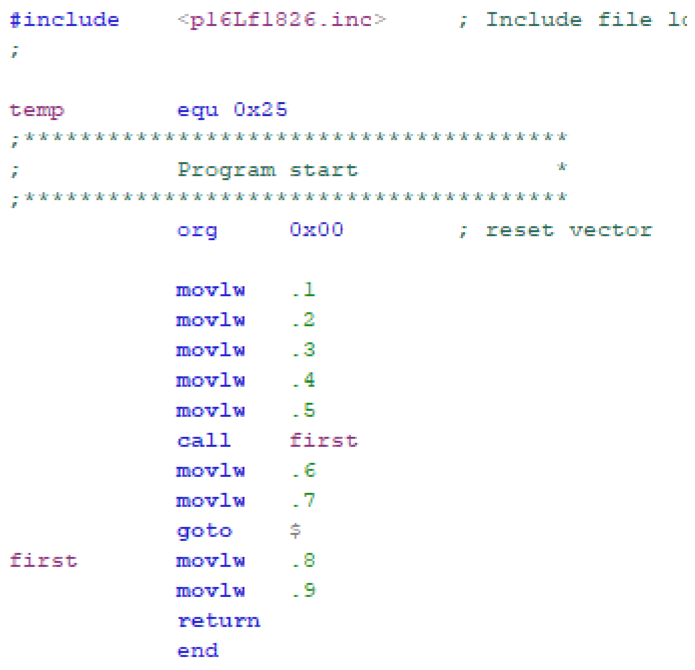
將 PIC MCU 改成 pipeline 的架構後執行以下測試檔。

PIPELINE 測試程式 1:



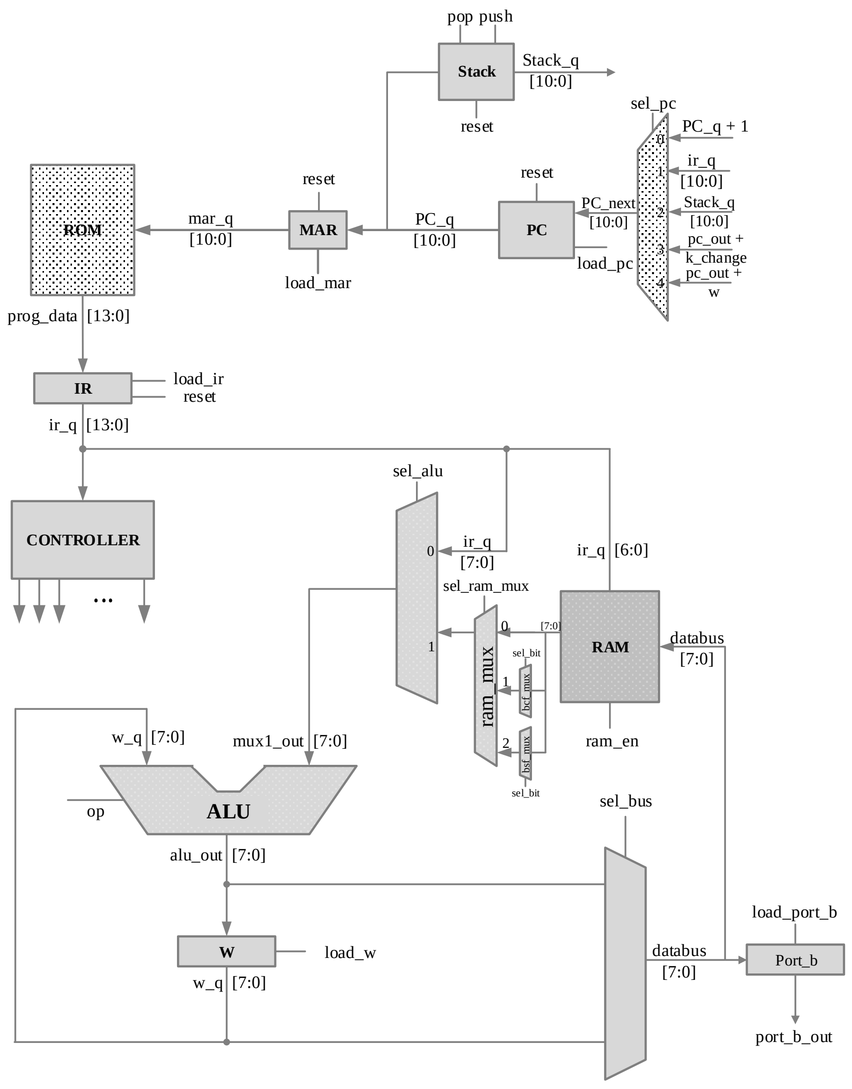
```
#include     <p16Lf1826.inc>      ; Include file l
;

temp         equ      0x25
;****************************************
;            Program start               *
;****************************************
             org      0x00         ; reset vector

loop         clrw
             clrf     temp
             movlw    .1
             addlw    .2
             sublw    .3
             movlw    .10
             movwf    temp
             incf     temp,1
             subwf    temp,0
             bcf      temp,3
             btfsc    temp,3
             btfsc    temp,1
             brw
             nop
             lslf     temp,1
             lsrf     temp,0
             goto     loop


             end
```

PIPELINE 測試程式 2:



```
#include     <p16Lf1826.inc>      ; Include file l
;

temp         equ 0x25
;****************************************
;            Program start               *
;****************************************
             org      0x00         ; reset vector

             movlw    .1
             movlw    .2
             movlw    .3
             movlw    .4
             movlw    .5
             call     first
             movlw    .6
             movlw    .7
             goto     $
first        movlw    .8
             movlw    .9
             return
             end
```

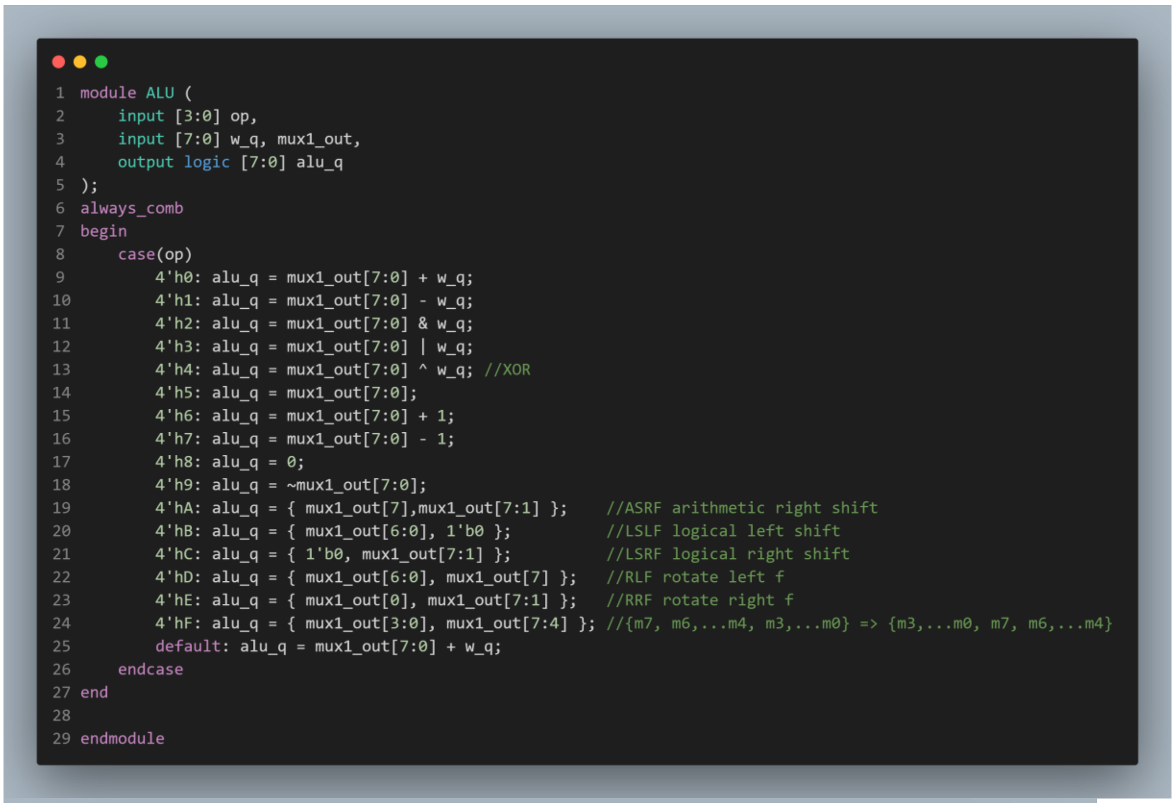
● 系統硬體架構方塊圖（接線圖）：



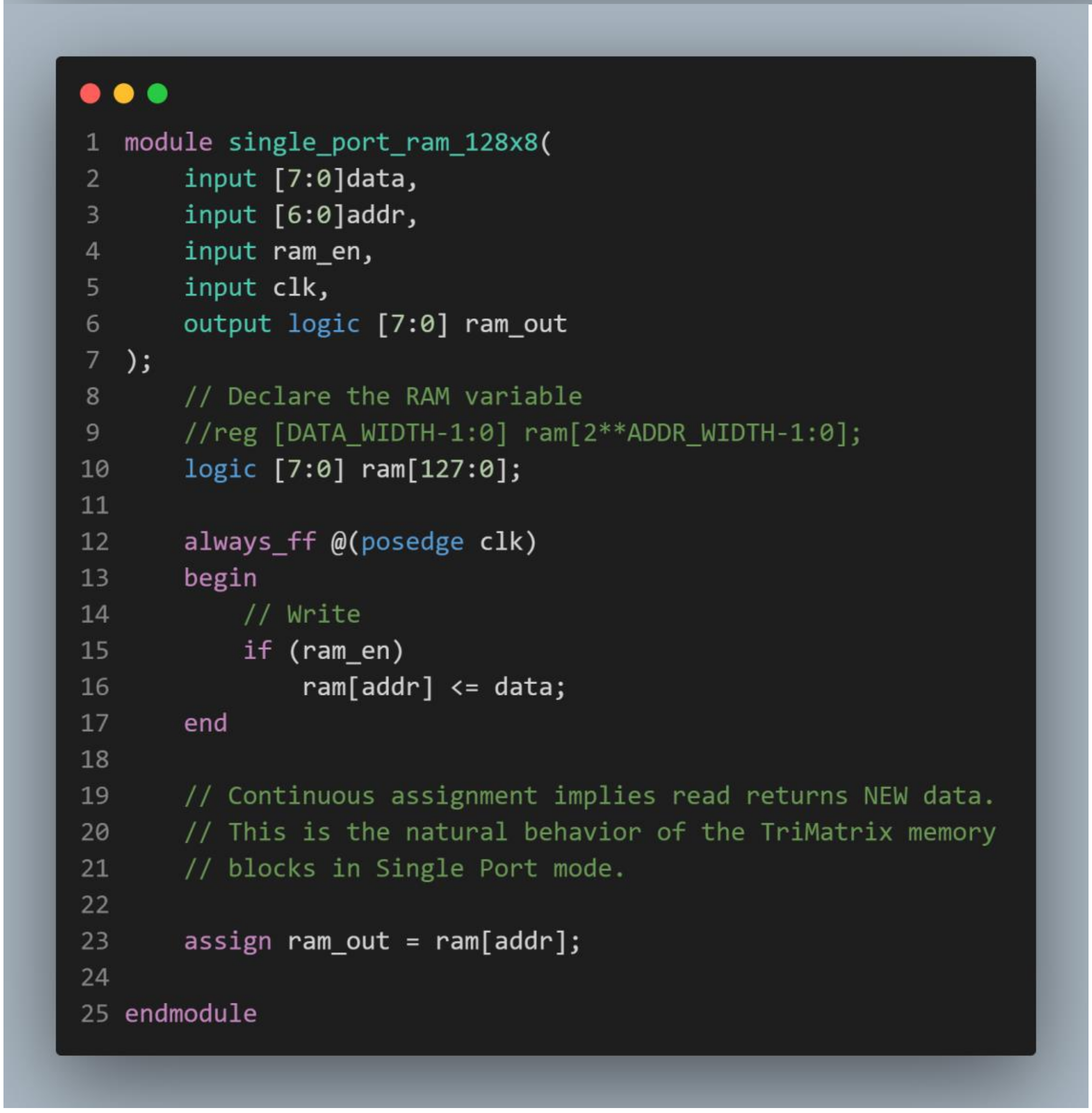
架構圖

● 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

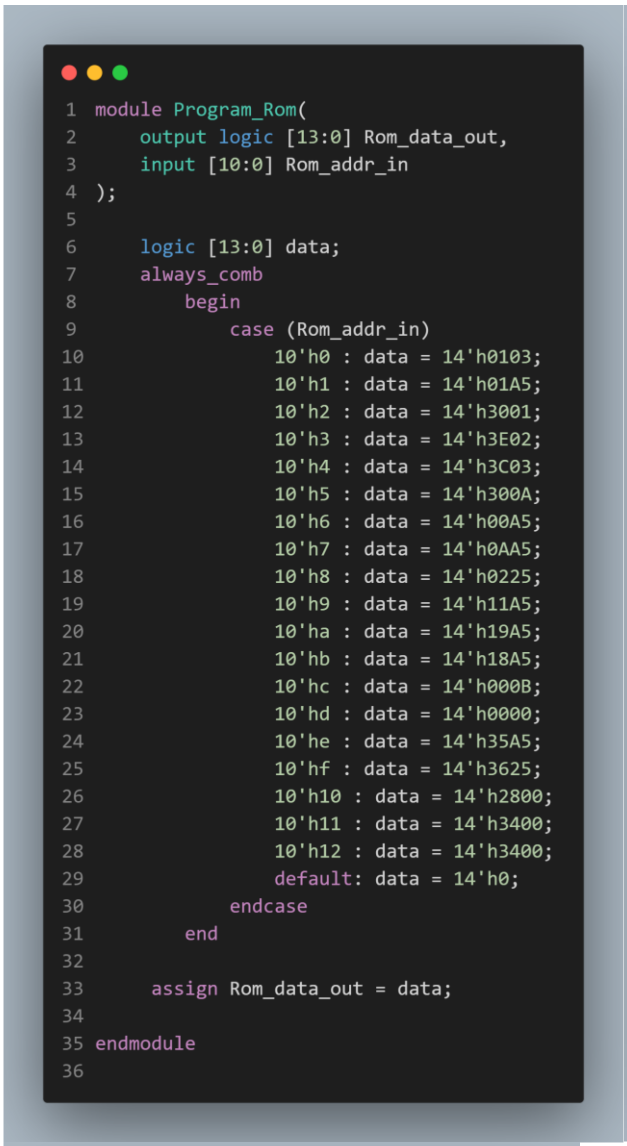


```systemverilog
module ALU (
    input [3:0] op,
    input [7:0] w_q, mux1_out,
    output logic [7:0] alu_q
);
always_comb
begin
    case(op)
        4'h0: alu_q = mux1_out[7:0] + w_q;
        4'h1: alu_q = mux1_out[7:0] - w_q;
        4'h2: alu_q = mux1_out[7:0] & w_q;
        4'h3: alu_q = mux1_out[7:0] | w_q;
        4'h4: alu_q = mux1_out[7:0] ^ w_q; //XOR
        4'h5: alu_q = mux1_out[7:0];
        4'h6: alu_q = mux1_out[7:0] + 1;
        4'h7: alu_q = mux1_out[7:0] - 1;
        4'h8: alu_q = 0;
        4'h9: alu_q = ~mux1_out[7:0];
        4'hA: alu_q = { mux1_out[7],mux1_out[7:1] };    //ASRF arithmetic right shift
        4'hB: alu_q = { mux1_out[6:0], 1'b0 };          //LSLF logical left shift
        4'hC: alu_q = { 1'b0, mux1_out[7:1] };          //LSRF logical right shift
        4'hD: alu_q = { mux1_out[6:0], mux1_out[7] };   //RLF rotate left f
        4'hE: alu_q = { mux1_out[0], mux1_out[7:1] };   //RRF rotate right f
        4'hF: alu_q = { mux1_out[3:0], mux1_out[7:4] }; //{m7, m6,...m4, m3,...m0} => {m3,...m0, m7, m6,...m4}
        default: alu_q = mux1_out[7:0] + w_q;
    endcase
end

endmodule
```



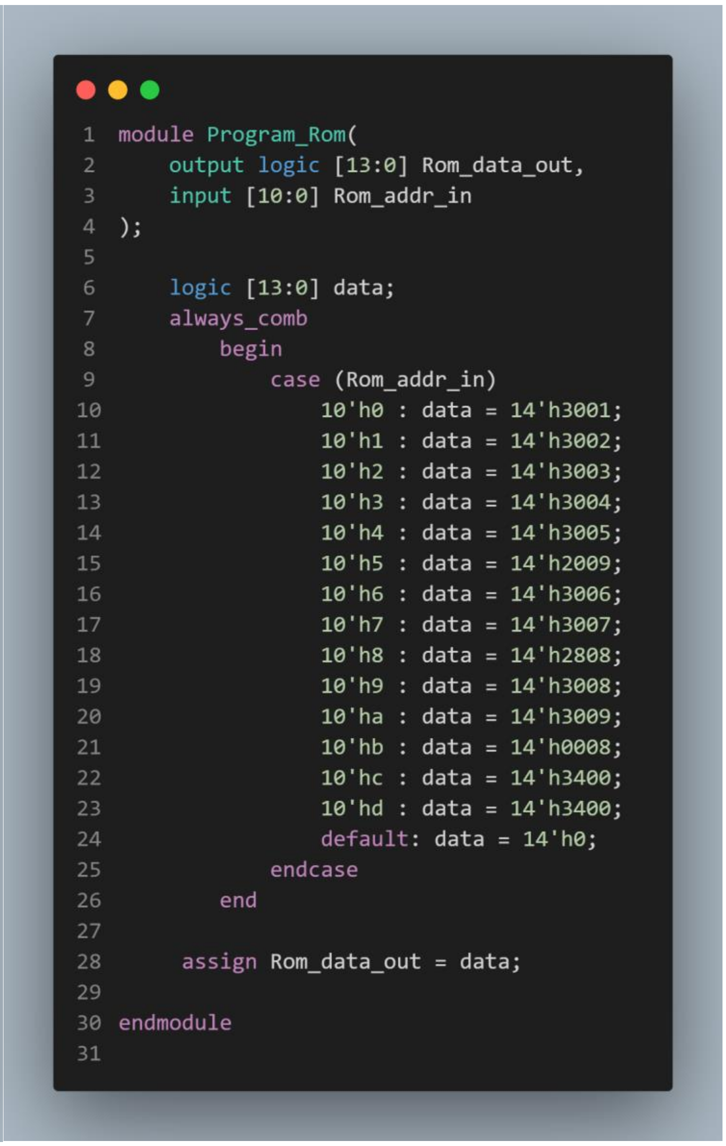
```systemverilog
module single_port_ram_128x8(
    input [7:0]data,
    input [6:0]addr,
    input ram_en,
    input clk,
    output logic [7:0] ram_out
);
    // Declare the RAM variable
    //reg [DATA_WIDTH-1:0] ram[2**ADDR_WIDTH-1:0];
    logic [7:0] ram[127:0];

    always_ff @(posedge clk)
    begin
        // Write
        if (ram_en)
            ram[addr] <= data;
    end

    // Continuous assignment implies read returns NEW data.
    // This is the natural behavior of the TriMatrix memory
    // blocks in Single Port mode.

    assign ram_out = ram[addr];

endmodule
```

測試程式 1

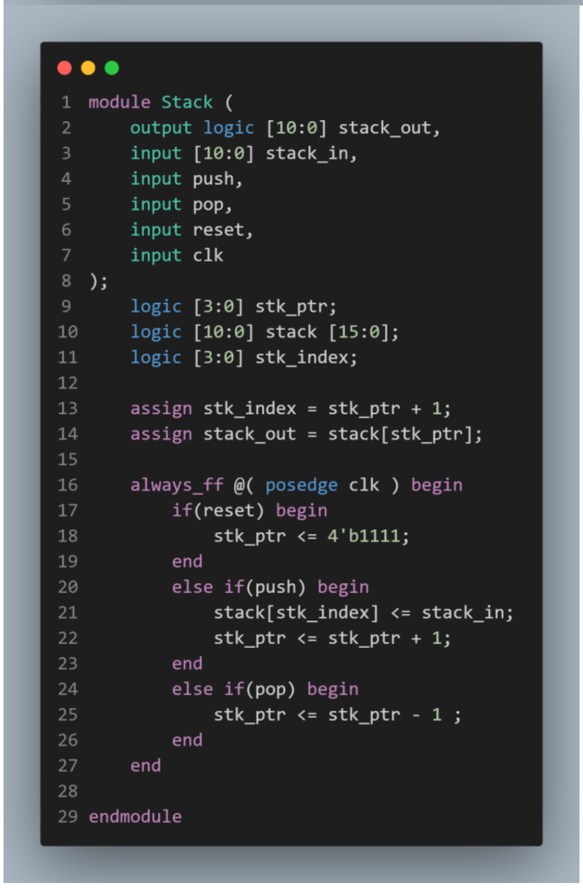


```
module Program_Rom(
    output logic [13:0] Rom_data_out,
    input [10:0] Rom_addr_in
);

    logic [13:0] data;
    always_comb
        begin
            case (Rom_addr_in)
                10'h0 : data = 14'h0103;
                10'h1 : data = 14'h01A5;
                10'h2 : data = 14'h3001;
                10'h3 : data = 14'h3E02;
                10'h4 : data = 14'h3C03;
                10'h5 : data = 14'h300A;
                10'h6 : data = 14'h00A5;
                10'h7 : data = 14'h0AA5;
                10'h8 : data = 14'h0225;
                10'h9 : data = 14'h11A5;
                10'ha : data = 14'h19A5;
                10'hb : data = 14'h18A5;
                10'hc : data = 14'h000B;
                10'hd : data = 14'h0000;
                10'he : data = 14'h35A5;
                10'hf : data = 14'h3625;
                10'h10 : data = 14'h2800;
                10'h11 : data = 14'h3400;
                10'h12 : data = 14'h3400;
                default: data = 14'h0;
            endcase
        end

     assign Rom_data_out = data;

endmodule
```

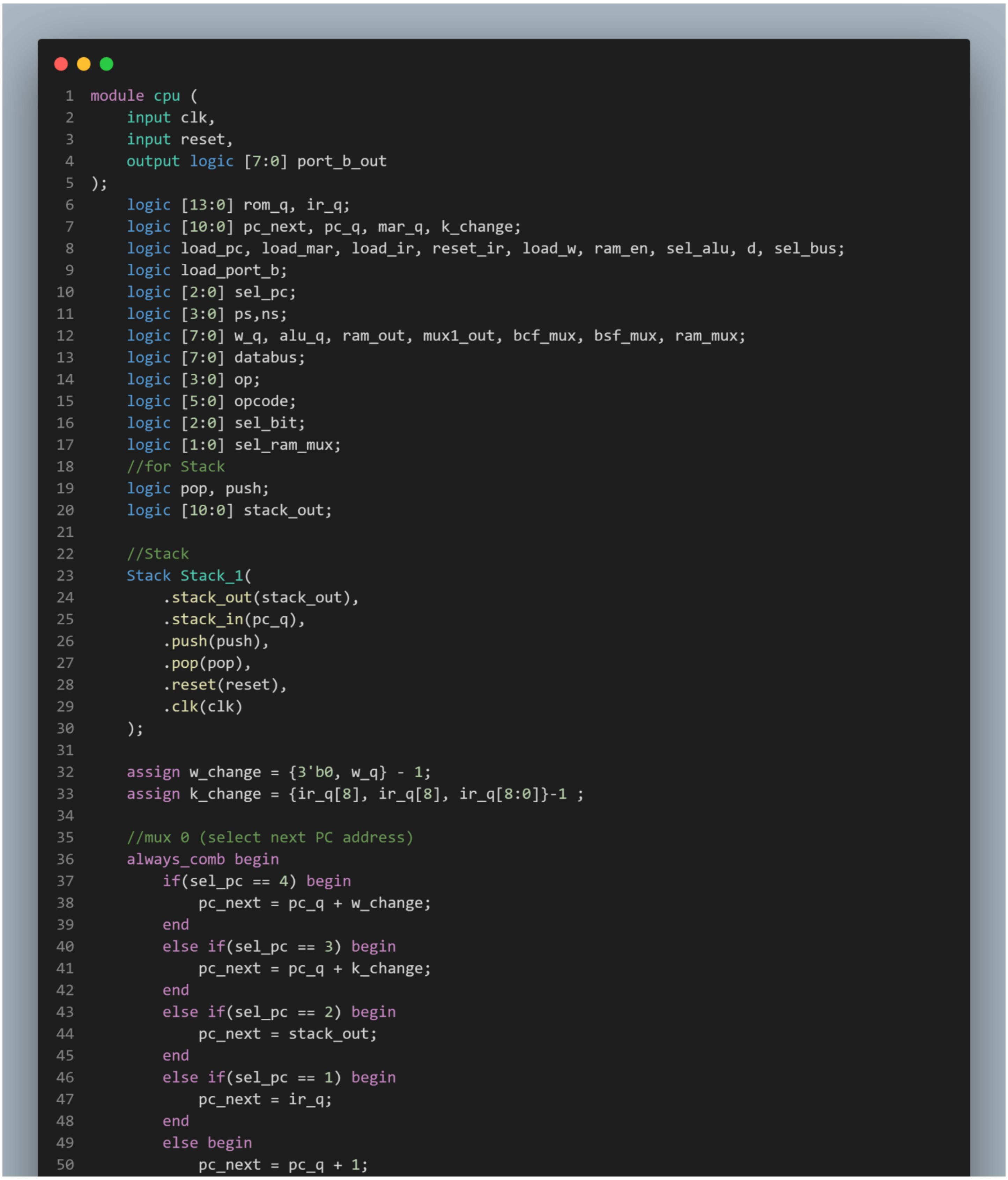
測試程式 2



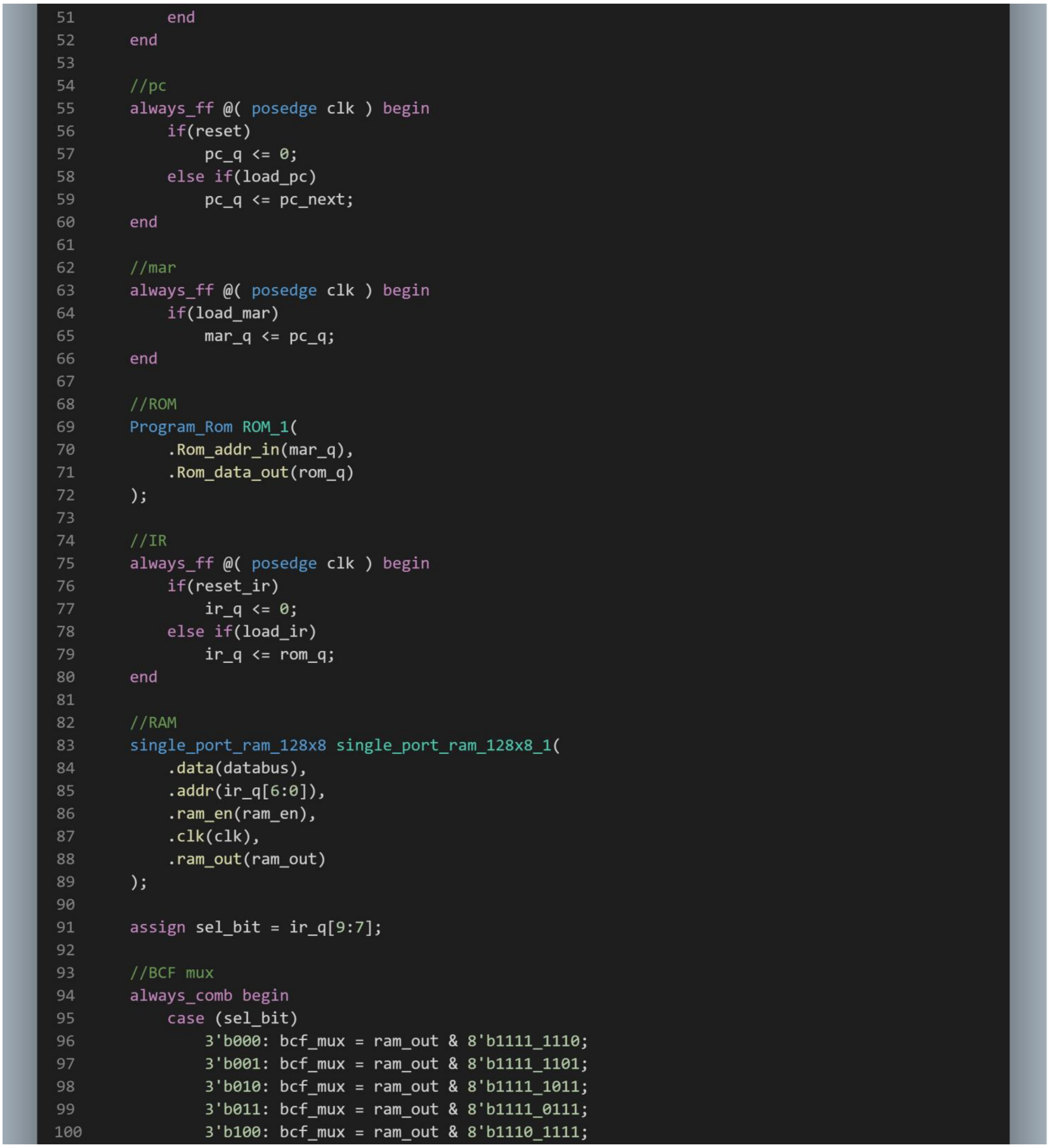
```
module Program_Rom(
    output logic [13:0] Rom_data_out,
    input [10:0] Rom_addr_in
);

    logic [13:0] data;
    always_comb
        begin
            case (Rom_addr_in)
                10'h0 : data = 14'h3001;
                10'h1 : data = 14'h3002;
                10'h2 : data = 14'h3003;
                10'h3 : data = 14'h3004;
                10'h4 : data = 14'h3005;
                10'h5 : data = 14'h2009;
                10'h6 : data = 14'h3006;
                10'h7 : data = 14'h3007;
                10'h8 : data = 14'h2808;
                10'h9 : data = 14'h3008;
                10'ha : data = 14'h3009;
                10'hb : data = 14'h0008;
                10'hc : data = 14'h3400;
                10'hd : data = 14'h3400;
                default: data = 14'h0;
            endcase
        end

     assign Rom_data_out = data;

endmodule
```



```
module Stack (
    output logic [10:0] stack_out,
    input [10:0] stack_in,
    input push,
    input pop,
    input reset,
    input clk
);
    logic [3:0] stk_ptr;
    logic [10:0] stack [15:0];
    logic [3:0] stk_index;

    assign stk_index = stk_ptr + 1;
    assign stack_out = stack[stk_ptr];

    always_ff @( posedge clk ) begin
        if(reset) begin
            stk_ptr <= 4'b1111;
        end
        else if(push) begin
            stack[stk_index] <= stack_in;
            stk_ptr <= stk_ptr + 1;
        end
        else if(pop) begin
            stk_ptr <= stk_ptr - 1 ;
        end
    end

endmodule
```

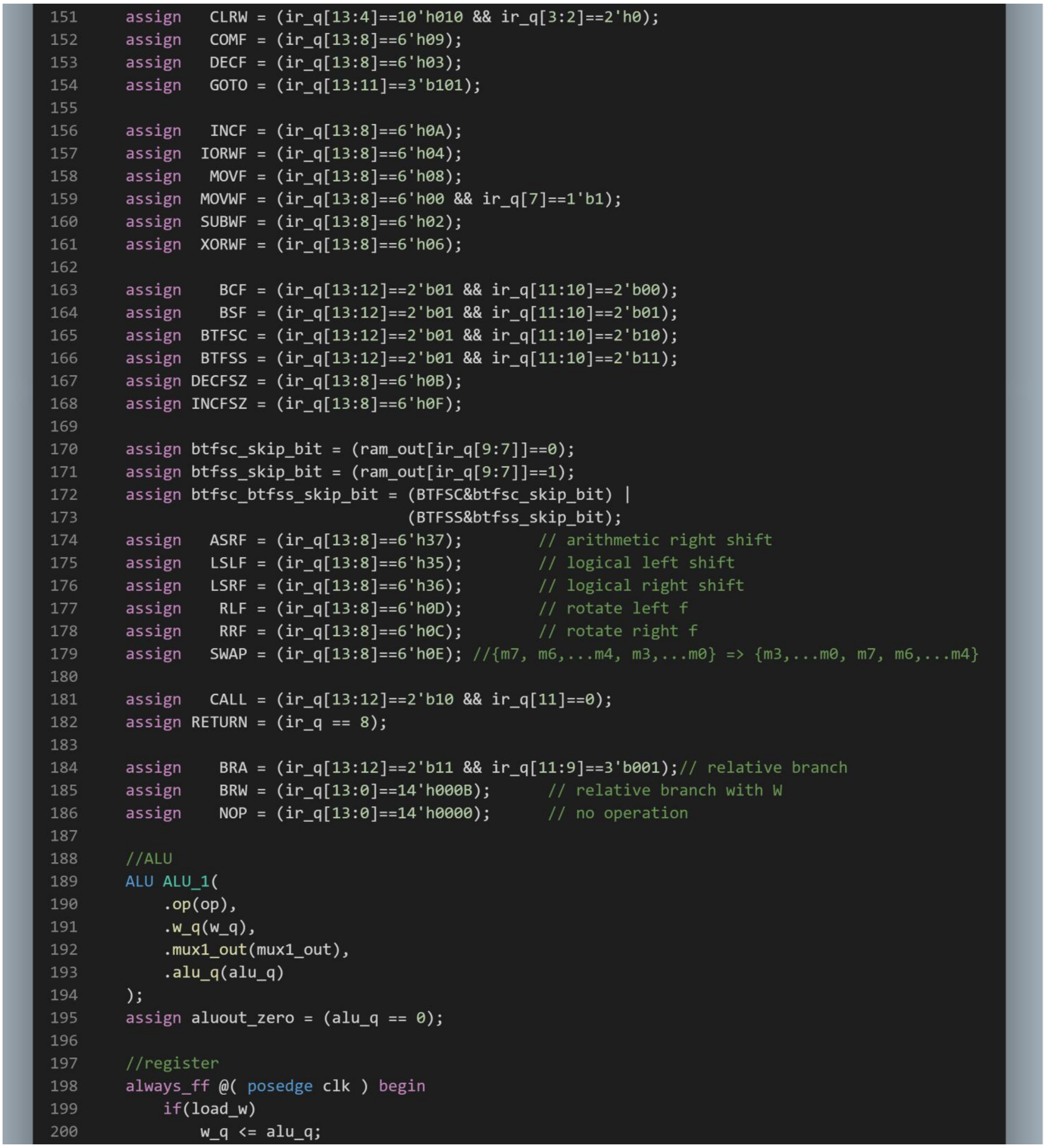


```
module cpu (
    input clk,
    input reset,
    output logic [7:0] port_b_out
);
    logic [13:0] rom_q, ir_q;
    logic [10:0] pc_next, pc_q, mar_q, k_change;
    logic load_pc, load_mar, load_ir, reset_ir, load_w, ram_en, sel_alu, d, sel_bus;
    logic load_port_b;
    logic [2:0] sel_pc;
    logic [3:0] ps,ns;
    logic [7:0] w_q, alu_q, ram_out, mux1_out, bcf_mux, bsf_mux, ram_mux;
    logic [7:0] databus;
    logic [3:0] op;
    logic [5:0] opcode;
    logic [2:0] sel_bit;
    logic [1:0] sel_ram_mux;
    //for Stack
    logic pop, push;
    logic [10:0] stack_out;

    //Stack
    Stack Stack_1(
        .stack_out(stack_out),
        .stack_in(pc_q),
        .push(push),
        .pop(pop),
        .reset(reset),
        .clk(clk)
    );

    assign w_change = {3'b0, w_q} - 1;
    assign k_change = {ir_q[8], ir_q[8], ir_q[8:0]}-1 ;

    //mux 0 (select next PC address)
    always_comb begin
        if(sel_pc == 4) begin
            pc_next = pc_q + w_change;
        end
        else if(sel_pc == 3) begin
            pc_next = pc_q + k_change;
        end
        else if(sel_pc == 2) begin
            pc_next = stack_out;
        end
        else if(sel_pc == 1) begin
            pc_next = ir_q;
        end
        else begin
            pc_next = pc_q + 1;
```

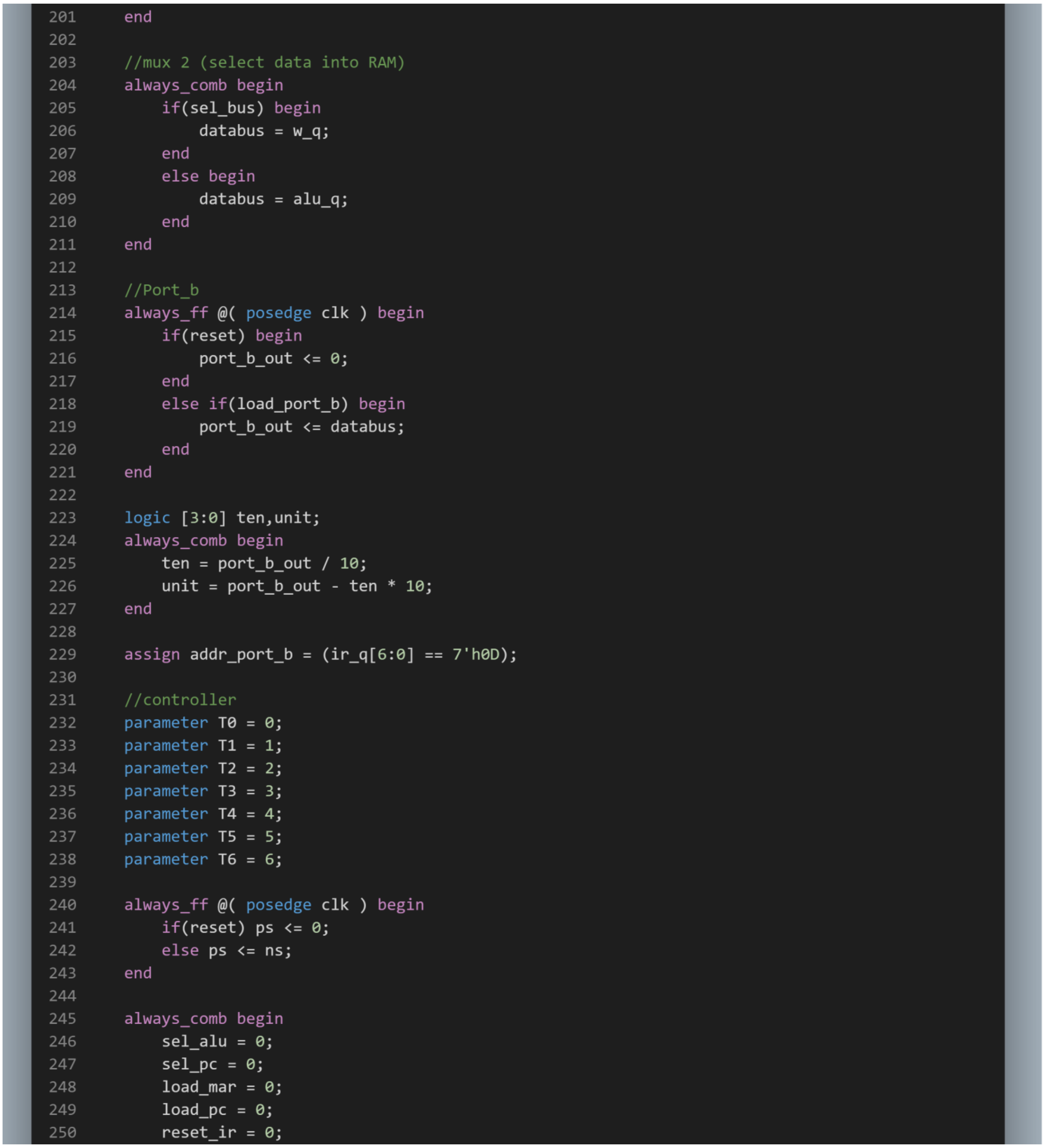


```
        end
    end

    //pc
    always_ff @( posedge clk ) begin
        if(reset)
            pc_q <= 0;
        else if(load_pc)
            pc_q <= pc_next;
    end

    //mar
    always_ff @( posedge clk ) begin
        if(load_mar)
            mar_q <= pc_q;
    end

    //ROM
    Program_Rom ROM_1(
        .Rom_addr_in(mar_q),
        .Rom_data_out(rom_q)
    );

    //IR
    always_ff @( posedge clk ) begin
        if(reset_ir)
            ir_q <= 0;
        else if(load_ir)
            ir_q <= rom_q;
    end

    //RAM
    single_port_ram_128x8 single_port_ram_128x8_1(
        .data(databus),
        .addr(ir_q[6:0]),
        .ram_en(ram_en),
        .clk(clk),
        .ram_out(ram_out)
    );

    assign sel_bit = ir_q[9:7];

    //BCF mux
    always_comb begin
        case (sel_bit)
            3'b000: bcf_mux = ram_out & 8'b1111_1110;
            3'b001: bcf_mux = ram_out & 8'b1111_1101;
            3'b010: bcf_mux = ram_out & 8'b1111_1011;
            3'b011: bcf_mux = ram_out & 8'b1111_0111;
            3'b100: bcf_mux = ram_out & 8'b1110_1111;
```

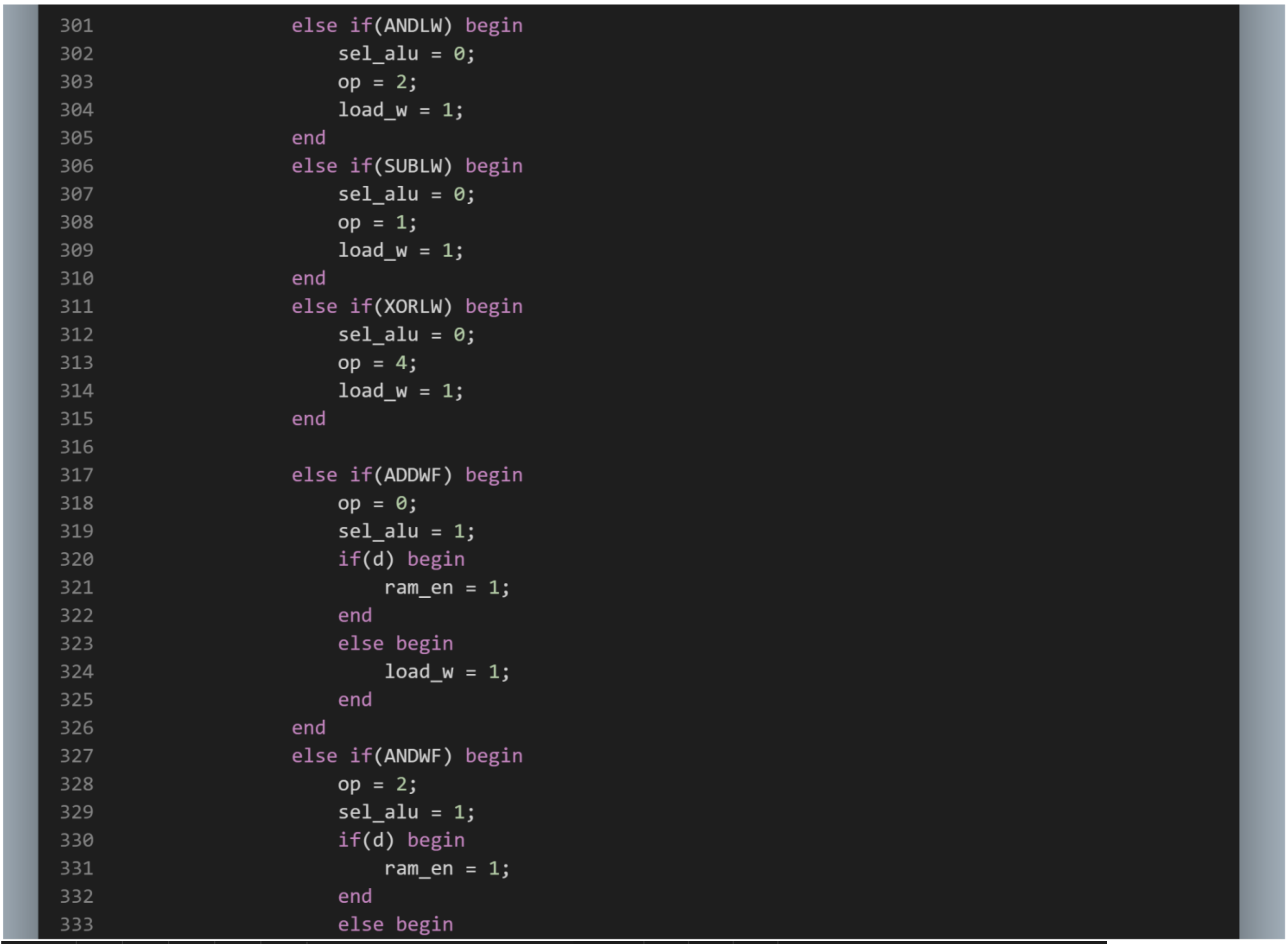
```verilog
            3'b101: bcf_mux = ram_out & 8'b1101_1111;
            3'b110: bcf_mux = ram_out & 8'b1011_1111;
            3'b111: bcf_mux = ram_out & 8'b0111_1111;
        endcase
    end

    //BSF mux
    always_comb begin
        case (sel_bit)
            3'b000: bsf_mux = ram_out | 8'b0000_0001;
            3'b001: bsf_mux = ram_out | 8'b0000_0010;
            3'b010: bsf_mux = ram_out | 8'b0000_0100;
            3'b011: bsf_mux = ram_out | 8'b0000_1000;
            3'b100: bsf_mux = ram_out | 8'b0001_0000;
            3'b101: bsf_mux = ram_out | 8'b0010_0000;
            3'b110: bsf_mux = ram_out | 8'b0100_0000;
            3'b111: bsf_mux = ram_out | 8'b1000_0000;
        endcase
    end

    //ram mux (select data into mux1)
    always_comb begin
        case (sel_ram_mux)
            0: ram_mux = ram_out;
            1: ram_mux = bcf_mux;
            2: ram_mux = bsf_mux;
        endcase
    end

    //mux 1 (select data into ALU)
    always_comb begin
        if(sel_alu) begin
            mux1_out = ram_mux;
        end
        else begin
            mux1_out = ir_q;
        end
    end

    assign d = ir_q[7];
    assign  MOVLW = (ir_q[13:8]==6'h30);
    assign  ADDLW = (ir_q[13:8]==6'h3E);
    assign  IORLW = (ir_q[13:8]==6'h38);
    assign  ANDLW = (ir_q[13:8]==6'h39);
    assign  SUBLW = (ir_q[13:8]==6'h3C);
    assign  XORLW = (ir_q[13:8]==6'h3A);

    assign  ADDWF = (ir_q[13:8]==6'h07);
    assign  ANDWF = (ir_q[13:8]==6'h05);
    assign   CLRF = (ir_q[13:8]==6'h01 && d==1);
```



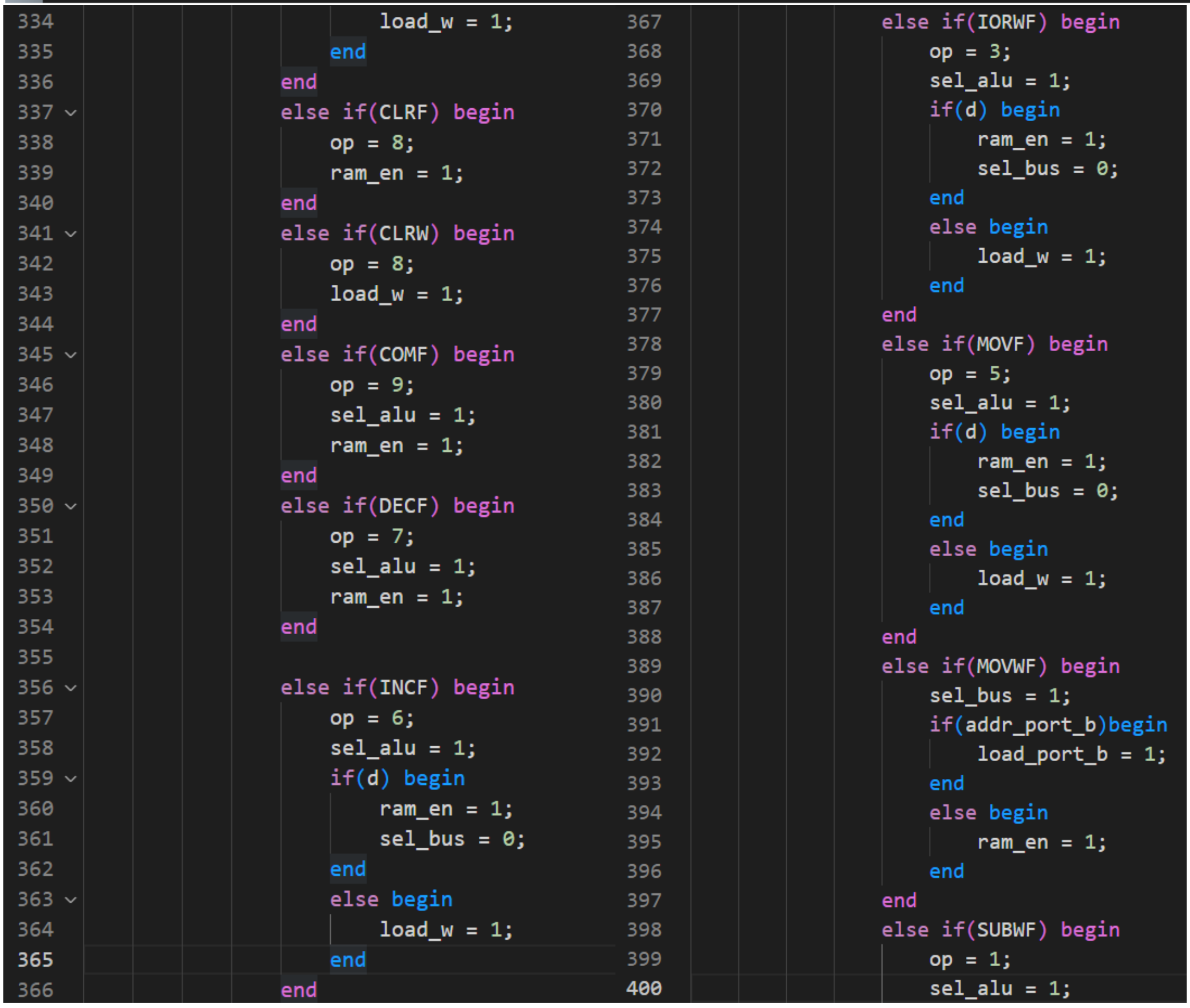
```
    assign   CLRW = (ir_q[13:4]==10'h010 && ir_q[3:2]==2'h0);
    assign   COMF = (ir_q[13:8]==6'h09);
    assign   DECF = (ir_q[13:8]==6'h03);
    assign   GOTO = (ir_q[13:11]==3'b101);

    assign   INCF = (ir_q[13:8]==6'h0A);
    assign  IORWF = (ir_q[13:8]==6'h04);
    assign   MOVF = (ir_q[13:8]==6'h08);
    assign  MOVWF = (ir_q[13:8]==6'h00 && ir_q[7]==1'b1);
    assign  SUBWF = (ir_q[13:8]==6'h02);
    assign  XORWF = (ir_q[13:8]==6'h06);

    assign    BCF = (ir_q[13:12]==2'b01 && ir_q[11:10]==2'b00);
    assign    BSF = (ir_q[13:12]==2'b01 && ir_q[11:10]==2'b01);
    assign  BTFSC = (ir_q[13:12]==2'b01 && ir_q[11:10]==2'b10);
    assign  BTFSS = (ir_q[13:12]==2'b01 && ir_q[11:10]==2'b11);
    assign DECFSZ = (ir_q[13:8]==6'h0B);
    assign INCFSZ = (ir_q[13:8]==6'h0F);

    assign btfsc_skip_bit = (ram_out[ir_q[9:7]]==0);
    assign btfss_skip_bit = (ram_out[ir_q[9:7]]==1);
    assign btfsc_btfss_skip_bit = (BTFSC&btfsc_skip_bit) |
                                  (BTFSS&btfss_skip_bit);
    assign   ASRF = (ir_q[13:8]==6'h37);        // arithmetic right shift
    assign   LSLF = (ir_q[13:8]==6'h35);        // logical left shift
    assign   LSRF = (ir_q[13:8]==6'h36);        // logical right shift
    assign    RLF = (ir_q[13:8]==6'h0D);        // rotate left f
    assign    RRF = (ir_q[13:8]==6'h0C);        // rotate right f
    assign   SWAP = (ir_q[13:8]==6'h0E); //{m7, m6,...m4, m3,...m0} => {m3,...m0, m7, m6,...m4}

    assign   CALL = (ir_q[13:12]==2'b10 && ir_q[11]==0);
    assign RETURN = (ir_q == 8);

    assign    BRA = (ir_q[13:12]==2'b11 && ir_q[11:9]==3'b001);// relative branch
    assign    BRW = (ir_q[13:0]==14'h000B);      // relative branch with W
    assign    NOP = (ir_q[13:0]==14'h0000);      // no operation

    //ALU
    ALU ALU_1(
        .op(op),
        .w_q(w_q),
        .mux1_out(mux1_out),
        .alu_q(alu_q)
    );
    assign aluout_zero = (alu_q == 0);

    //register
    always_ff @( posedge clk ) begin
        if(load_w)
            w_q <= alu_q;
```



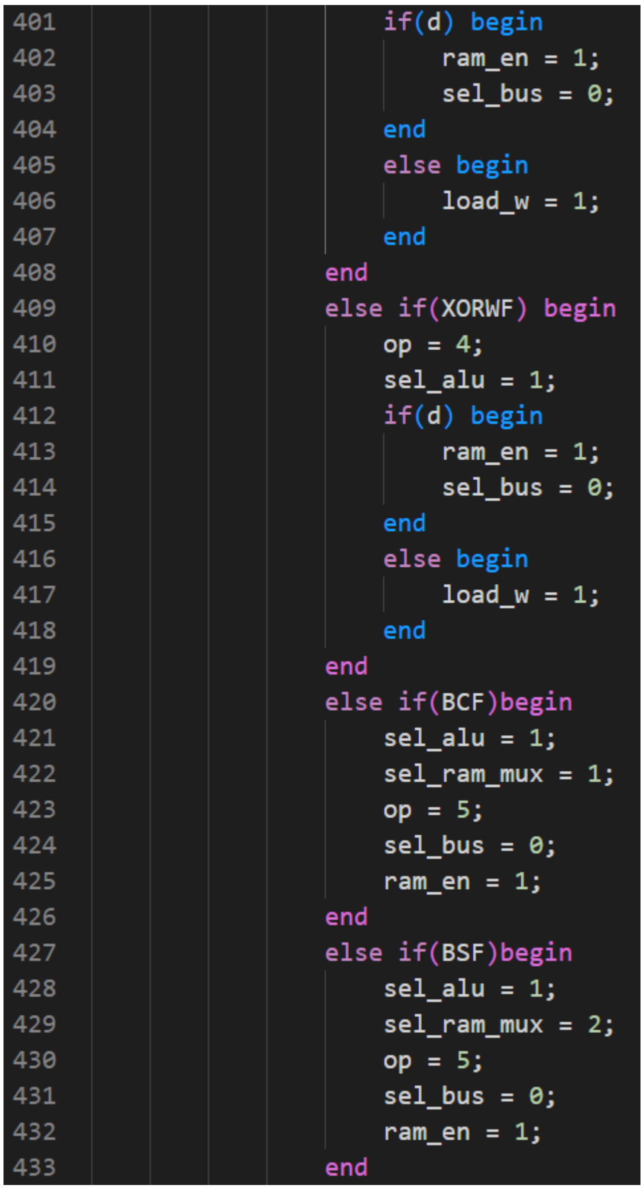
```
    end

    //mux 2 (select data into RAM)
    always_comb begin
        if(sel_bus) begin
            databus = w_q;
        end
        else begin
            databus = alu_q;
        end
    end

    //Port_b
    always_ff @( posedge clk ) begin
        if(reset) begin
            port_b_out <= 0;
        end
        else if(load_port_b) begin
            port_b_out <= databus;
        end
    end

    logic [3:0] ten,unit;
    always_comb begin
        ten = port_b_out / 10;
        unit = port_b_out - ten * 10;
    end

    assign addr_port_b = (ir_q[6:0] == 7'h0D);

    //controller
    parameter T0 = 0;
    parameter T1 = 1;
    parameter T2 = 2;
    parameter T3 = 3;
    parameter T4 = 4;
    parameter T5 = 5;
    parameter T6 = 6;

    always_ff @( posedge clk ) begin
        if(reset) ps <= 0;
        else ps <= ns;
    end

    always_comb begin
        sel_alu = 0;
        sel_pc = 0;
        load_mar = 0;
        load_pc = 0;
        reset_ir = 0;
        load_ir = 0;
        load_w = 0;
        ram_en = 0;
        op = 0;
        sel_ram_mux = 0;
        sel_bus = 0;
        load_port_b = 0;
        ns=0;
        //for Stack
        push = 0;
        pop = 0;
        case(ps)
            T0: begin
                ns = T1;
            end
            T1: begin
                load_mar = 1;
                sel_pc = 0;
                load_pc = 1;
                ns = T2;
            end
            T2: begin
                ns = T3;
            end
            T3: begin
                load_ir = 1;
                ns = T4;
            end
            T4: begin
                load_mar = 1;
                sel_pc = 2'b00;
                load_pc = 1;
                if(GOTO) begin//to skip the following instruction in T4
                    load_mar = 1;
                end
                else if(MOVLW) begin
                    sel_alu = 0;
                    op = 5;
                    load_w = 1;
                end
                else if(ADDLW) begin
                    sel_alu = 0;
                    op = 0;
                    load_w = 1;
                end
                else if(IORLW) begin
                    sel_alu = 0;
                    op = 3;
                    load_w = 1;
                end
```



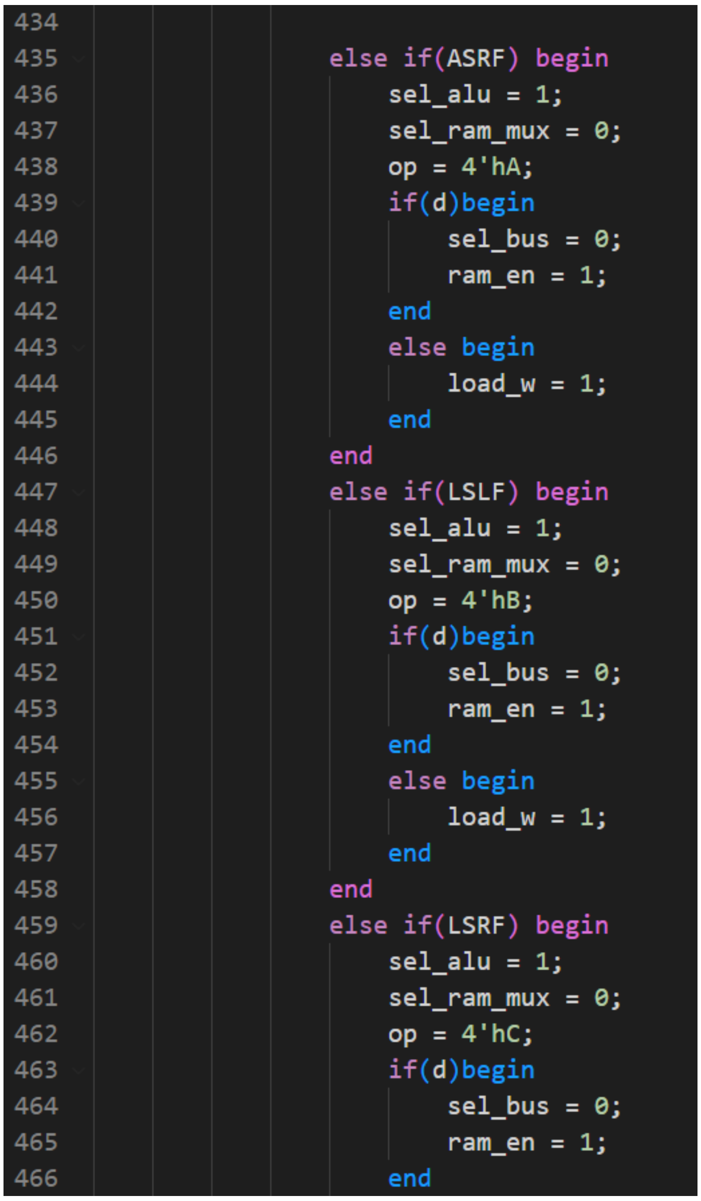
```verilog
        else if(ANDLW) begin
            sel_alu = 0;
            op = 2;
            load_w = 1;
        end
        else if(SUBLW) begin
            sel_alu = 0;
            op = 1;
            load_w = 1;
        end
        else if(XORLW) begin
            sel_alu = 0;
            op = 4;
            load_w = 1;
        end

        else if(ADDWF) begin
            op = 0;
            sel_alu = 1;
            if(d) begin
                ram_en = 1;
            end
            else begin
                load_w = 1;
            end
        end
        else if(ANDWF) begin
            op = 2;
            sel_alu = 1;
            if(d) begin
                ram_en = 1;
            end
            else begin
```



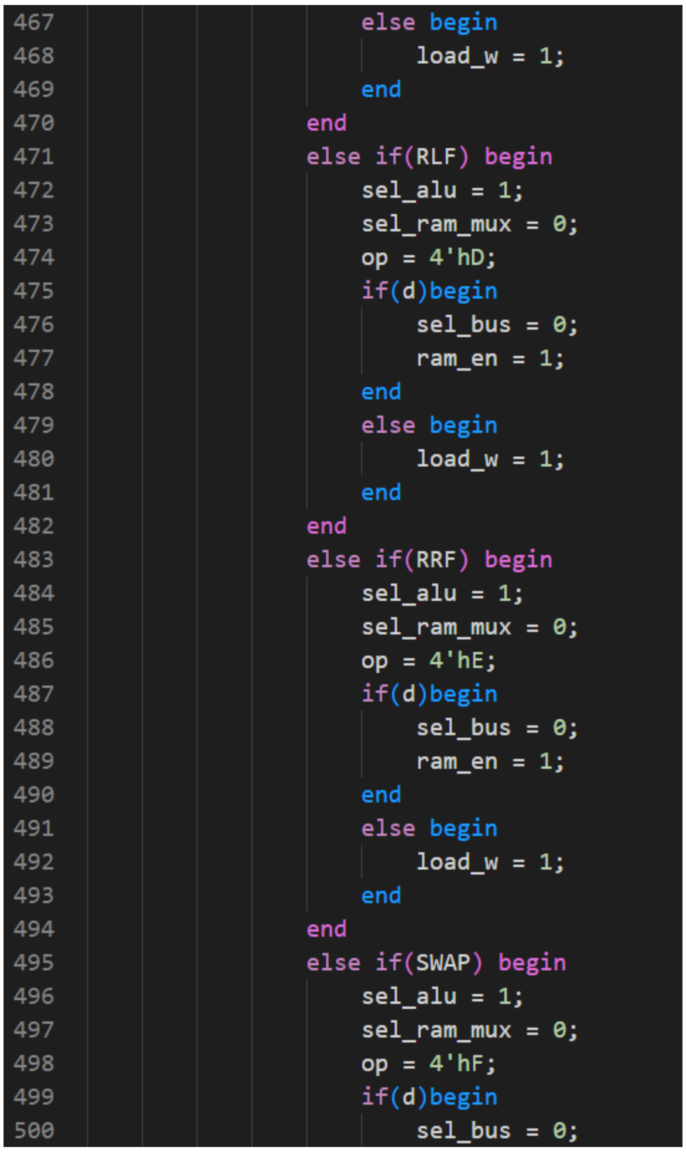
```verilog
                load_w = 1;
            end
        end
        else if(CLRF) begin
            op = 8;
            ram_en = 1;
        end
        else if(CLRW) begin
            op = 8;
            load_w = 1;
        end
        else if(COMF) begin
            op = 9;
            sel_alu = 1;
            ram_en = 1;
        end
        else if(DECF) begin
            op = 7;
            sel_alu = 1;
            ram_en = 1;
        end

        else if(INCF) begin
            op = 6;
            sel_alu = 1;
            if(d) begin
                ram_en = 1;
                sel_bus = 0;
            end
            else begin
                load_w = 1;
            end
        end
        else if(IORWF) begin
            op = 3;
            sel_alu = 1;
            if(d) begin
                ram_en = 1;
                sel_bus = 0;
            end
            else begin
                load_w = 1;
            end
        end
        else if(MOVF) begin
            op = 5;
            sel_alu = 1;
            if(d) begin
                ram_en = 1;
                sel_bus = 0;
            end
            else begin
                load_w = 1;
            end
        end
        else if(MOVWF) begin
            sel_bus = 1;
            if(addr_port_b)begin
                load_port_b = 1;
            end
            else begin
                ram_en = 1;
            end
        end
        else if(SUBWF) begin
            op = 1;
            sel_alu = 1;
```



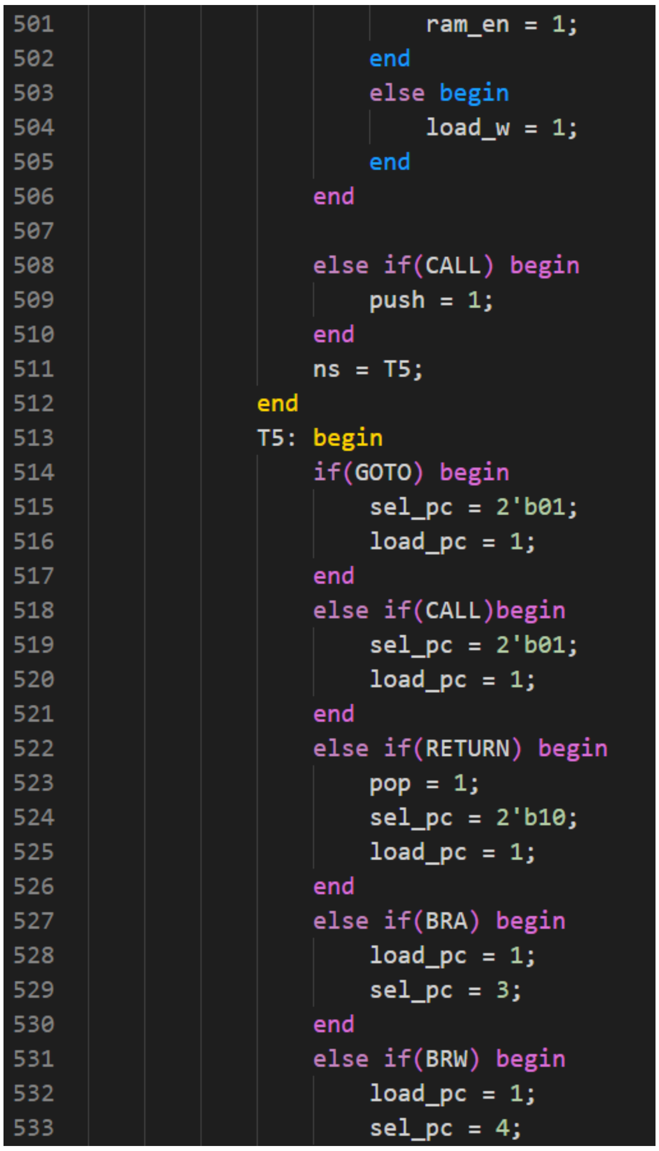
```
                    if(d) begin
                        ram_en = 1;
                        sel_bus = 0;
                    end
                    else begin
                        load_w = 1;
                    end
                end
                else if(XORWF) begin
                    op = 4;
                    sel_alu = 1;
                    if(d) begin
                        ram_en = 1;
                        sel_bus = 0;
                    end
                    else begin
                        load_w = 1;
                    end
                end
                else if(BCF)begin
                    sel_alu = 1;
                    sel_ram_mux = 1;
                    op = 5;
                    sel_bus = 0;
                    ram_en = 1;
                end
                else if(BSF)begin
                    sel_alu = 1;
                    sel_ram_mux = 2;
                    op = 5;
                    sel_bus = 0;
                    ram_en = 1;
                end
```



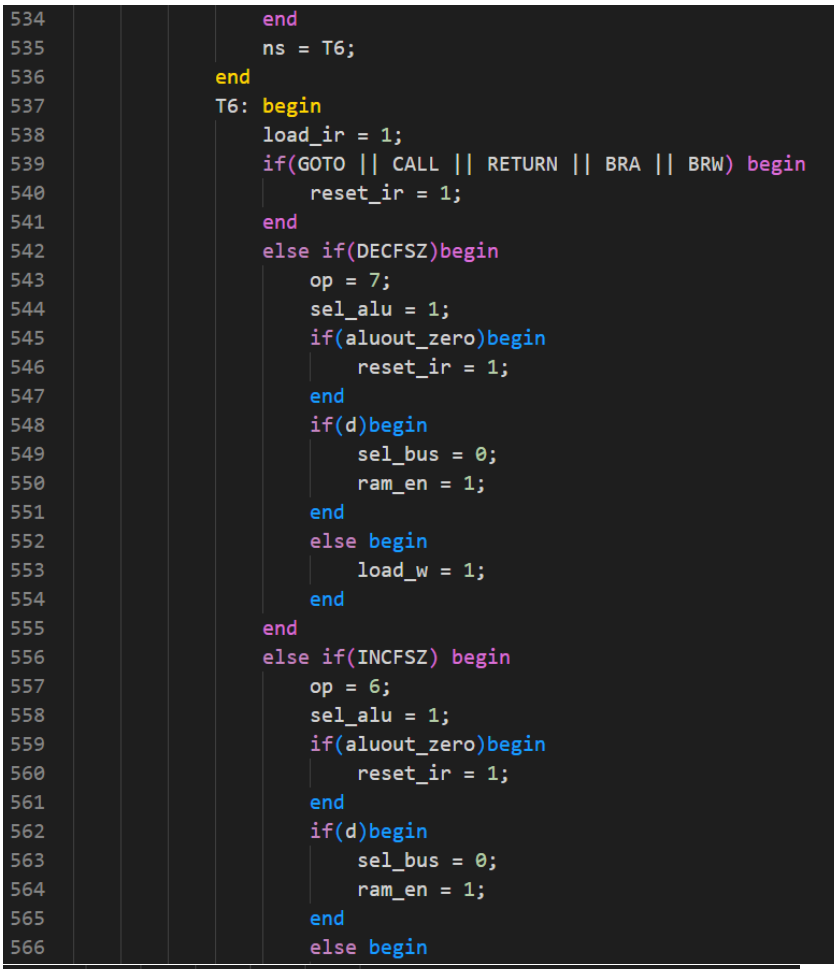
```

                else if(ASRF) begin
                    sel_alu = 1;
                    sel_ram_mux = 0;
                    op = 4'hA;
                    if(d)begin
                        sel_bus = 0;
                        ram_en = 1;
                    end
                    else begin
                        load_w = 1;
                    end
                end
                else if(LSLF) begin
                    sel_alu = 1;
                    sel_ram_mux = 0;
                    op = 4'hB;
                    if(d)begin
                        sel_bus = 0;
                        ram_en = 1;
                    end
                    else begin
                        load_w = 1;
                    end
                end
                else if(LSRF) begin
                    sel_alu = 1;
                    sel_ram_mux = 0;
                    op = 4'hC;
                    if(d)begin
                        sel_bus = 0;
                        ram_en = 1;
                    end
```



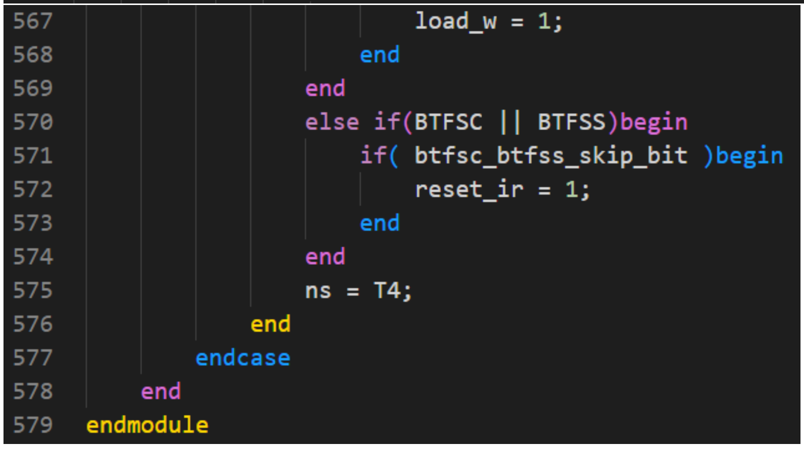
```
                    else begin
                        load_w = 1;
                    end
                end
                else if(RLF) begin
                    sel_alu = 1;
                    sel_ram_mux = 0;
                    op = 4'hD;
                    if(d)begin
                        sel_bus = 0;
                        ram_en = 1;
                    end
                    else begin
                        load_w = 1;
                    end
                end
                else if(RRF) begin
                    sel_alu = 1;
                    sel_ram_mux = 0;
                    op = 4'hE;
                    if(d)begin
                        sel_bus = 0;
                        ram_en = 1;
                    end
                    else begin
                        load_w = 1;
                    end
                end
                else if(SWAP) begin
                    sel_alu = 1;
                    sel_ram_mux = 0;
                    op = 4'hF;
                    if(d)begin
                        sel_bus = 0;
```



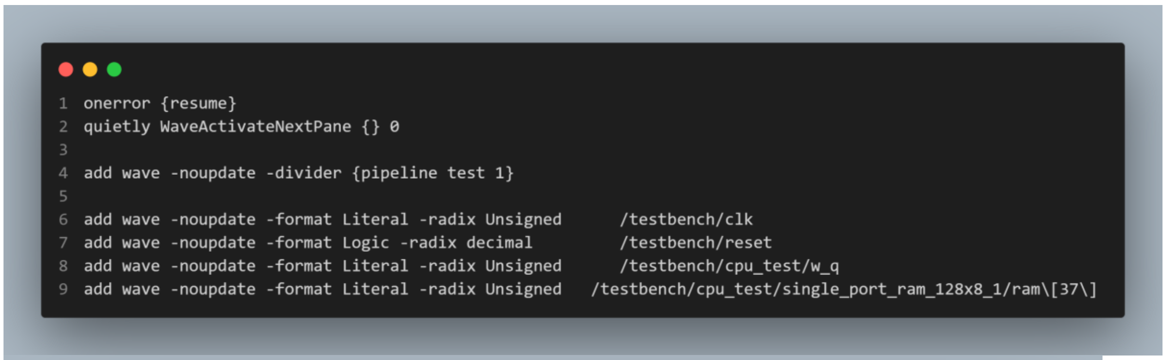
```
                        ram_en = 1;
                    end
                    else begin
                        load_w = 1;
                    end
                end

                else if(CALL) begin
                    push = 1;
                end
                ns = T5;
            end
            T5: begin
                if(GOTO) begin
                    sel_pc = 2'b01;
                    load_pc = 1;
                end
                else if(CALL)begin
                    sel_pc = 2'b01;
                    load_pc = 1;
                end
                else if(RETURN) begin
                    pop = 1;
                    sel_pc = 2'b10;
                    load_pc = 1;
                end
                else if(BRA) begin
                    load_pc = 1;
                    sel_pc = 3;
                end
                else if(BRW) begin
                    load_pc = 1;
                    sel_pc = 4;
```



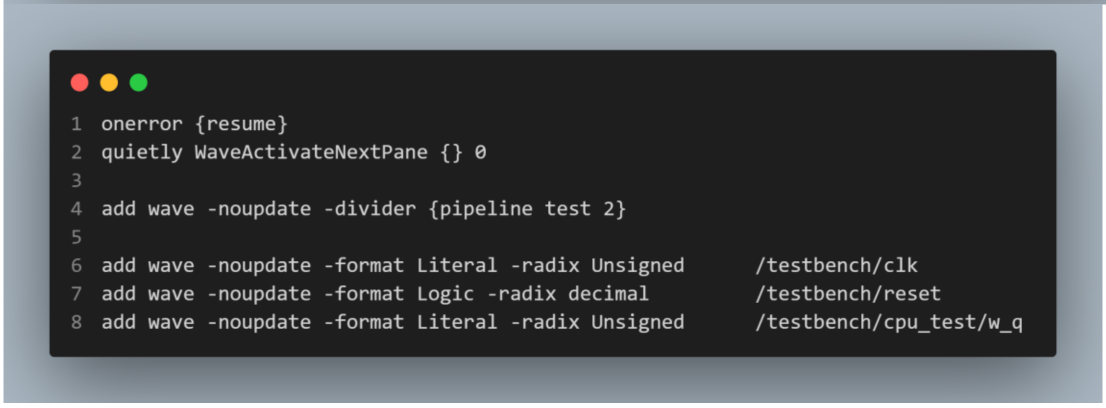
```
                end
                ns = T6;
            end
            T6: begin
                load_ir = 1;
                if(GOTO || CALL || RETURN || BRA || BRW) begin
                    reset_ir = 1;
                end
                else if(DECFSZ)begin
                    op = 7;
                    sel_alu = 1;
                    if(aluout_zero)begin
                        reset_ir = 1;
                    end
                    if(d)begin
                        sel_bus = 0;
                        ram_en = 1;
                    end
                    else begin
                        load_w = 1;
                    end
                end
                else if(INCFSZ) begin
                    op = 6;
                    sel_alu = 1;
                    if(aluout_zero)begin
                        reset_ir = 1;
                    end
                    if(d)begin
                        sel_bus = 0;
                        ram_en = 1;
                    end
                    else begin
```



```
                        load_w = 1;
                    end
                end
                else if(BTFSC || BTFSS)begin
                    if( btfsc_btfss_skip_bit )begin
                        reset_ir = 1;
                    end
                end
                ns = T4;
            end
        endcase
    end
endmodule
```



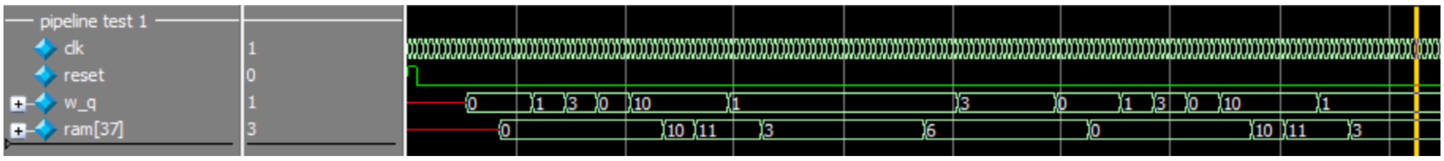
```
onerror {resume}
quietly WaveActivateNextPane {} 0

add wave -noupdate -divider {pipeline test 1}

add wave -noupdate -format Literal -radix Unsigned      /testbench/clk
add wave -noupdate -format Logic -radix decimal         /testbench/reset
add wave -noupdate -format Literal -radix Unsigned      /testbench/cpu_test/w_q
add wave -noupdate -format Literal -radix Unsigned   /testbench/cpu_test/single_port_ram_128x8_1/ram\[37\]
```



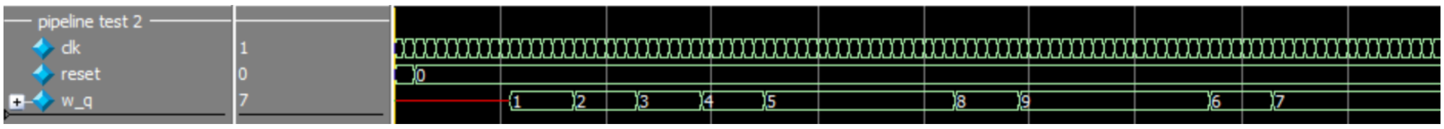
```
onerror {resume}
quietly WaveActivateNextPane {} 0

add wave -noupdate -divider {pipeline test 2}

add wave -noupdate -format Literal -radix Unsigned      /testbench/clk
add wave -noupdate -format Logic -radix decimal         /testbench/reset
add wave -noupdate -format Literal -radix Unsigned      /testbench/cpu_test/w_q
```

模擬結果與結果說明：

測試程式 1



測試程式 2



結論與心得：

本次作業是上課練習的其中一個部分，上課練習測試程式 3 時因為沒有發現指令判斷錯誤，所以結果一直是錯的，好在助教很耐心的教我一步一步確認是從哪邊發生錯誤，最後終於成功發現之前的 return 判斷錯誤，修改過後就正常了，十分感激助教的耐心與鼓勵。進行測試程式 2 時很快地就產生正確的結果，但是到了測試程式 1 就又發生問題了。一開始以為是 brw 跳到錯誤的位址，但是經過計算之後發現測試程式 1 中的 brw 指令剛好不會跳去其他位址，於是矛頭就指向 lslf 和 lsrf，發現 lslf 都沒有被 enable，經過一連串的尋找才發現程式裡面 sel_pc 的 bit 數不足，overflow 造成 brw 指令多跳了一個指令，修改過後就能正常行。在完成的過程中，重新去回顧之前做的所有的指令，了解每一個指令的流程及資料的變化，讓我對整學期的教學內容有更深刻的理解，謝謝老師的用心教學和助教的詳細講解!