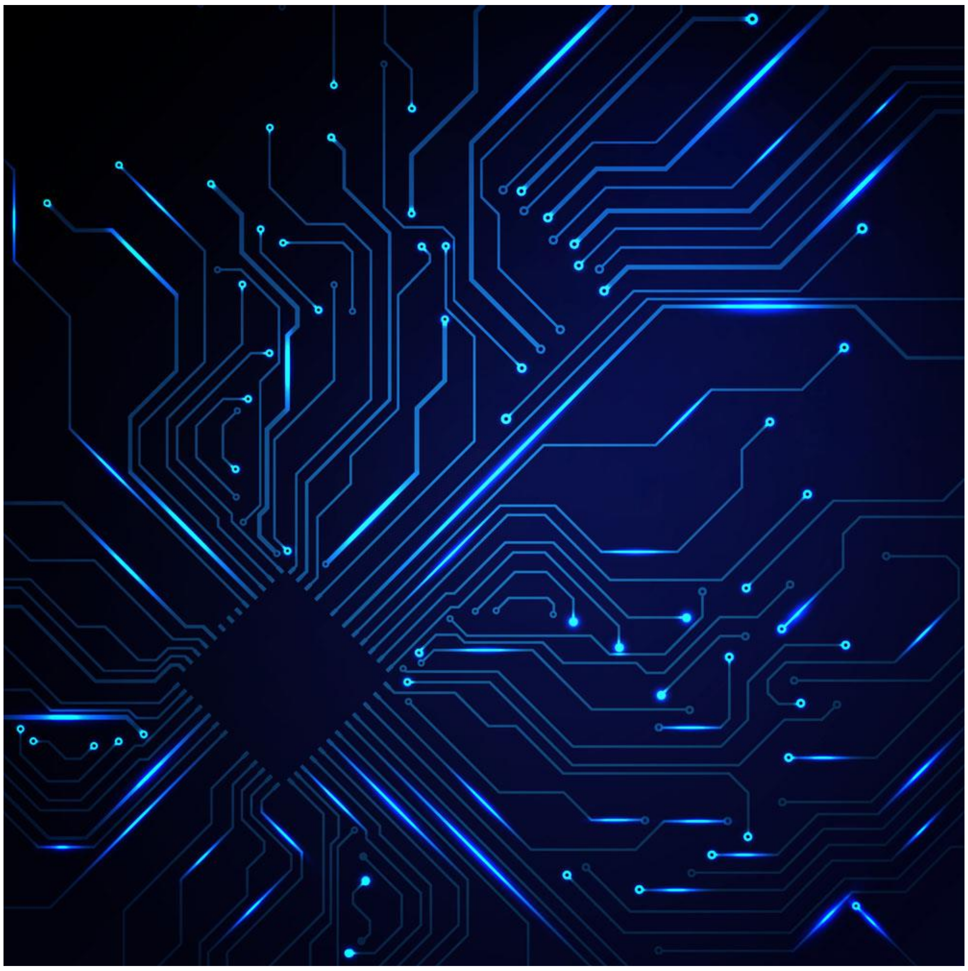


2022/11/28

實驗十
條件跳躍指令

姓名：張銀軒　　　學號：00957050

班級：資工 3A

E-mail：00957050@mail.ntou.edu.tw

注意

1. 繳交時一律轉 PDF 檔
2. 繳交期限為
   上完課後
   當週五晚上 12 點前
3. 一人繳交一份
4. 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

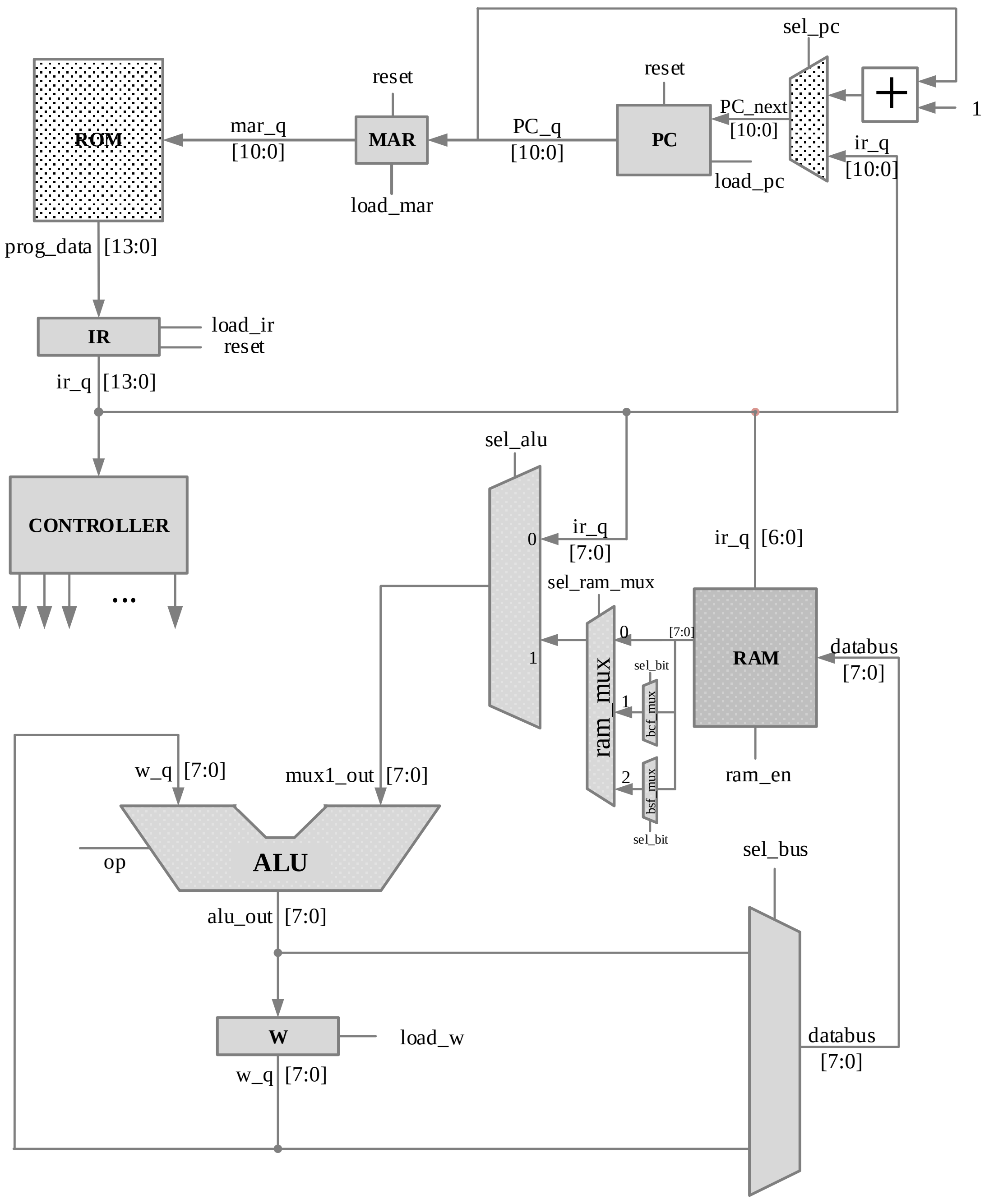
● 實驗說明：

1. 如圖所示，設計一個架構實現條件跳躍指令
2. 輸入：clk, reset
3. 輸出：w_q[7:0]

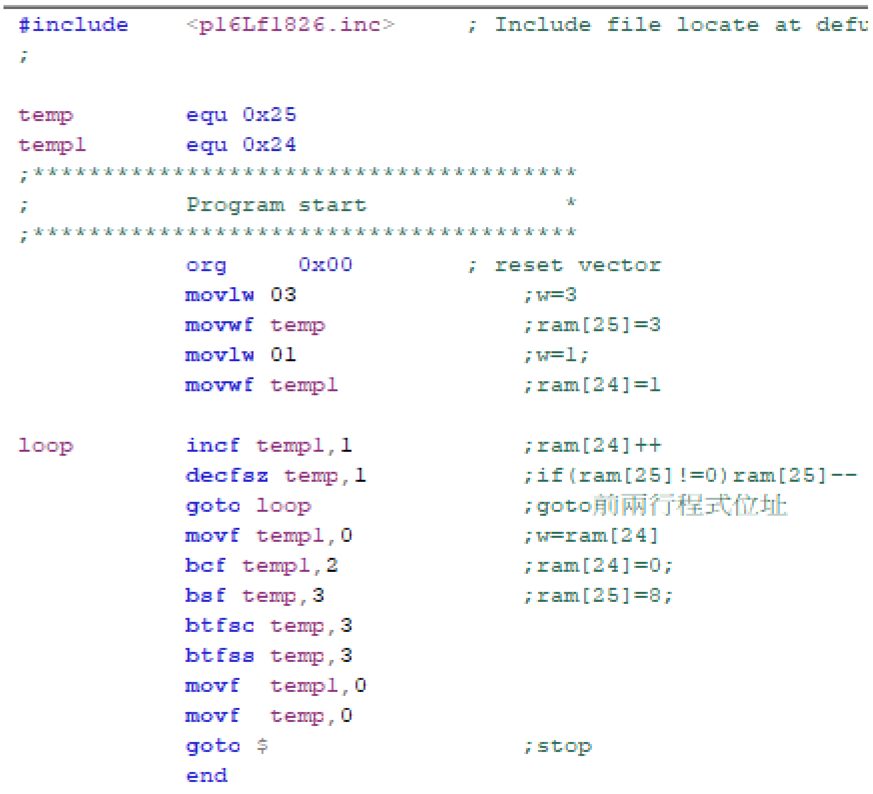
   請務必按照規定的 input 及 output 來做

   請建一個 MPLAB 專案，打入下方給的組合語言 code，BUILD 並生成 HEX 檔，再將 HEX 轉成 Program_Rom，模擬結果請參考下方的圖

● 系統硬體架構方塊圖（接線圖）：

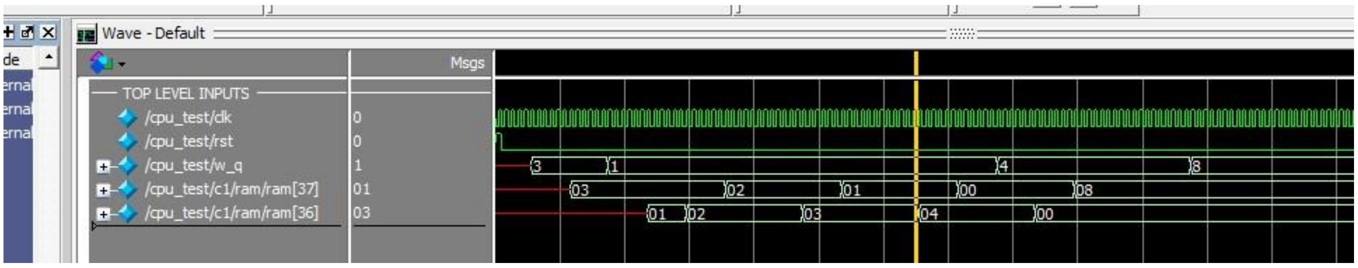


架構圖



```
#include      <p16Lf1826.inc>      ; Include file locate at defu
;

temp          equ 0x25
temp1         equ 0x24
;***************************************
;             Program start            *
;***************************************
              org     0x00         ; reset vector
              movlw 03                 ;w=3
              movwf temp               ;ram[25]=3
              movlw 01                 ;w=1;
              movwf temp1              ;ram[24]=1

loop          incf temp1,1             ;ram[24]++
              decfsz temp,1            ;if(ram[25]!=0)ram[25]--
              goto loop                ;goto前兩行程式位址
              movf temp1,0             ;w=ram[24]
              bcf temp1,2              ;ram[24]=0;
              bsf temp,3               ;ram[25]=8;
              btfsc temp,3
              btfss temp,3
              movf  temp1,0
              movf  temp,0
              goto $                   ;stop
              end
```

組合語言

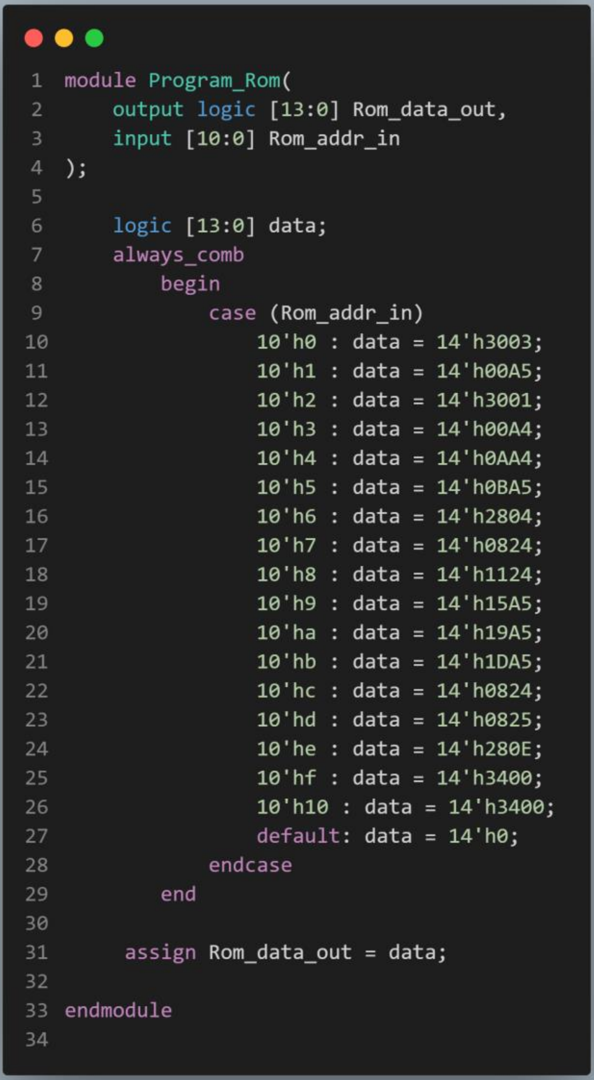


```
Wave - Default
                              Msgs
TOP LEVEL INPUTS
/cpu_test/clk                 0
/cpu_test/rst                 0
/cpu_test/w_q                 1      3  1  4  8
/cpu_test/c1/ram/ram[37]      01     03  02  01  00  08
/cpu_test/c1/ram/ram[36]      03     01  02  03  04  00
```

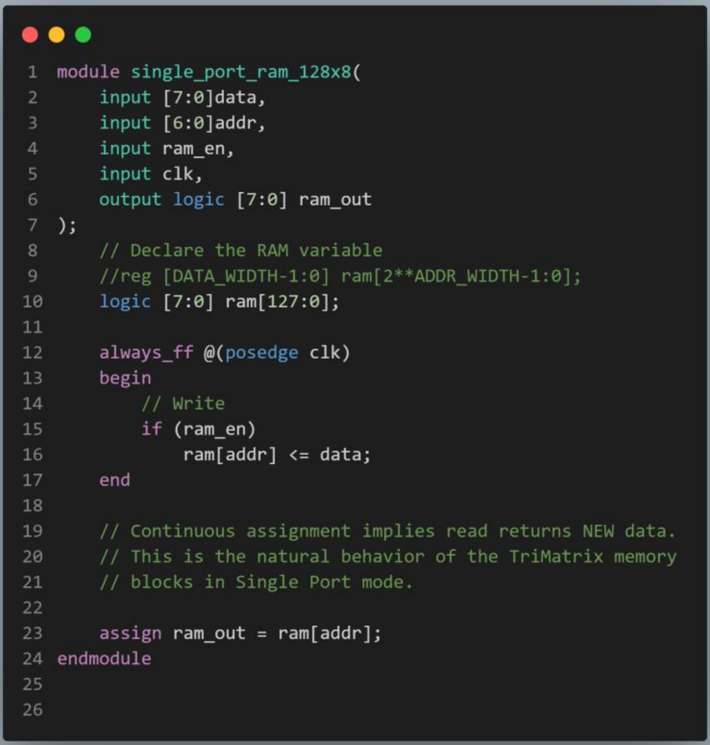
模擬結果

● 系統架構程式碼、測試資料程式碼與程式碼說明

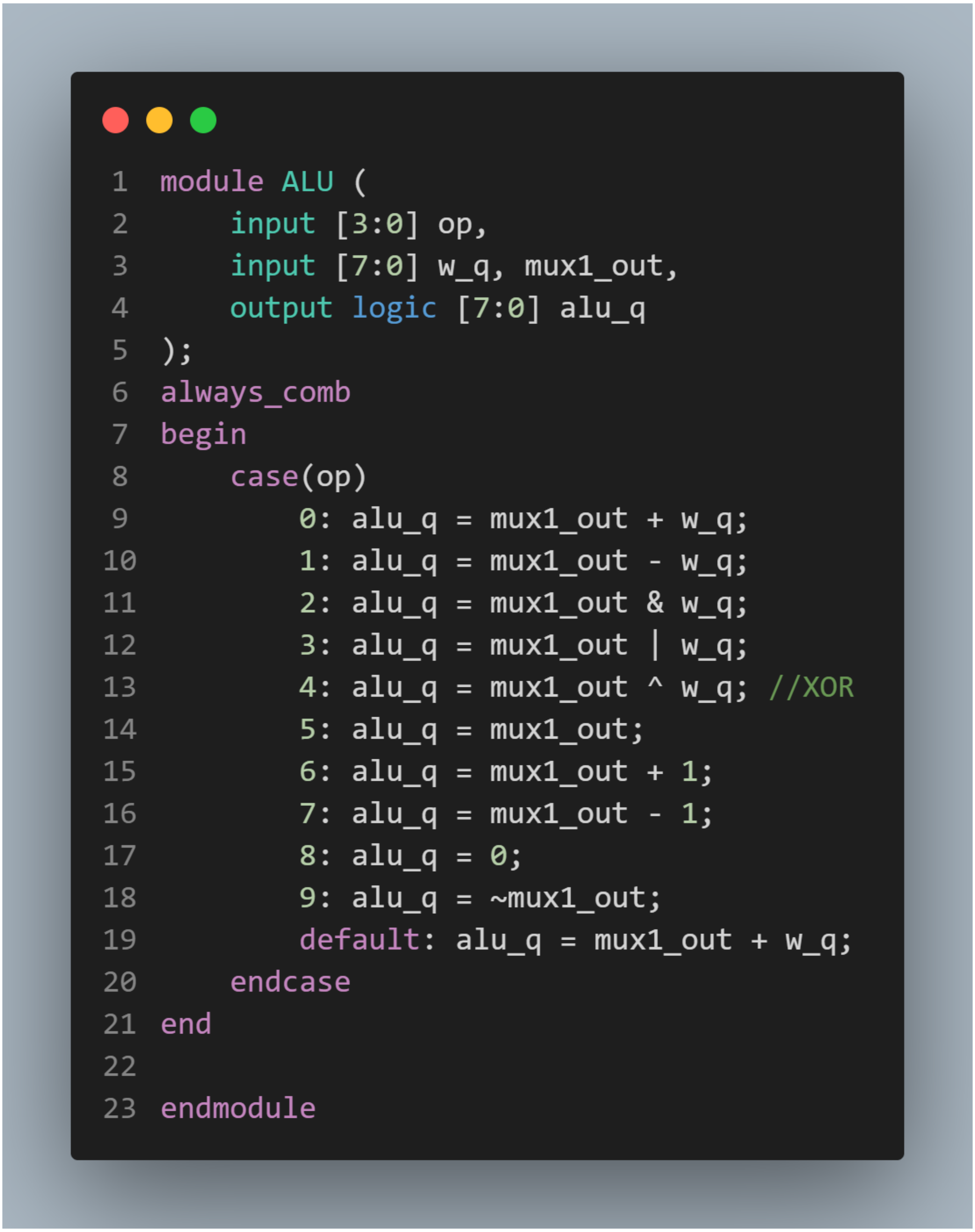
截圖請善用 win+shift+S



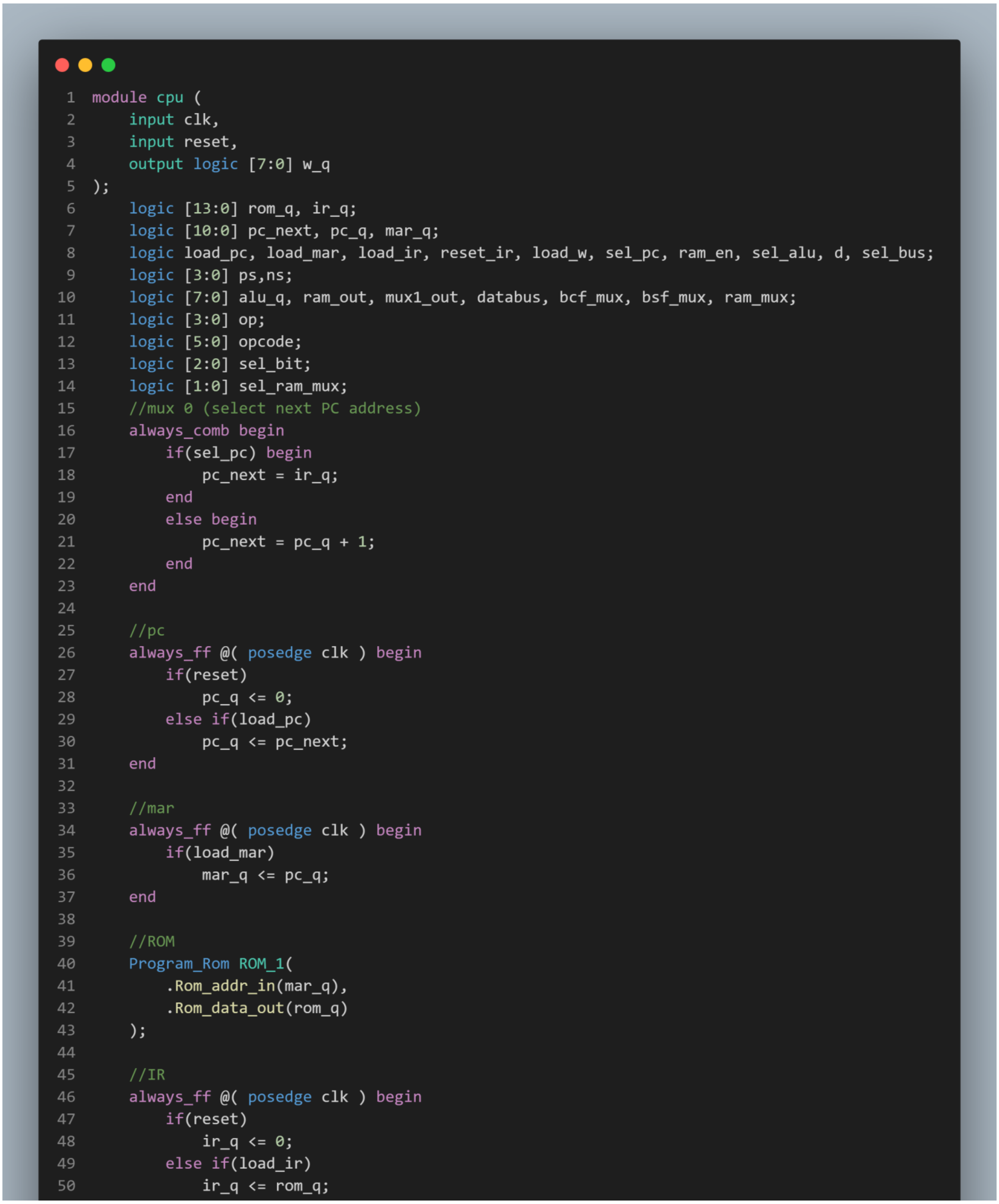
```
module Program_Rom(
    output logic [13:0] Rom_data_out,
    input [10:0] Rom_addr_in
);

    logic [13:0] data;
    always_comb
        begin
            case (Rom_addr_in)
                10'h0 : data = 14'h3003;
                10'h1 : data = 14'h00A5;
                10'h2 : data = 14'h3001;
                10'h3 : data = 14'h00A4;
                10'h4 : data = 14'h0AA4;
                10'h5 : data = 14'h0BA5;
                10'h6 : data = 14'h2804;
                10'h7 : data = 14'h0824;
                10'h8 : data = 14'h1124;
                10'h9 : data = 14'h15A5;
                10'ha : data = 14'h19A5;
                10'hb : data = 14'h1DA5;
                10'hc : data = 14'h0824;
                10'hd : data = 14'h0825;
                10'he : data = 14'h280E;
                10'hf : data = 14'h3400;
                10'h10 : data = 14'h3400;
                default: data = 14'h0;
            endcase
        end

     assign Rom_data_out = data;

endmodule
```



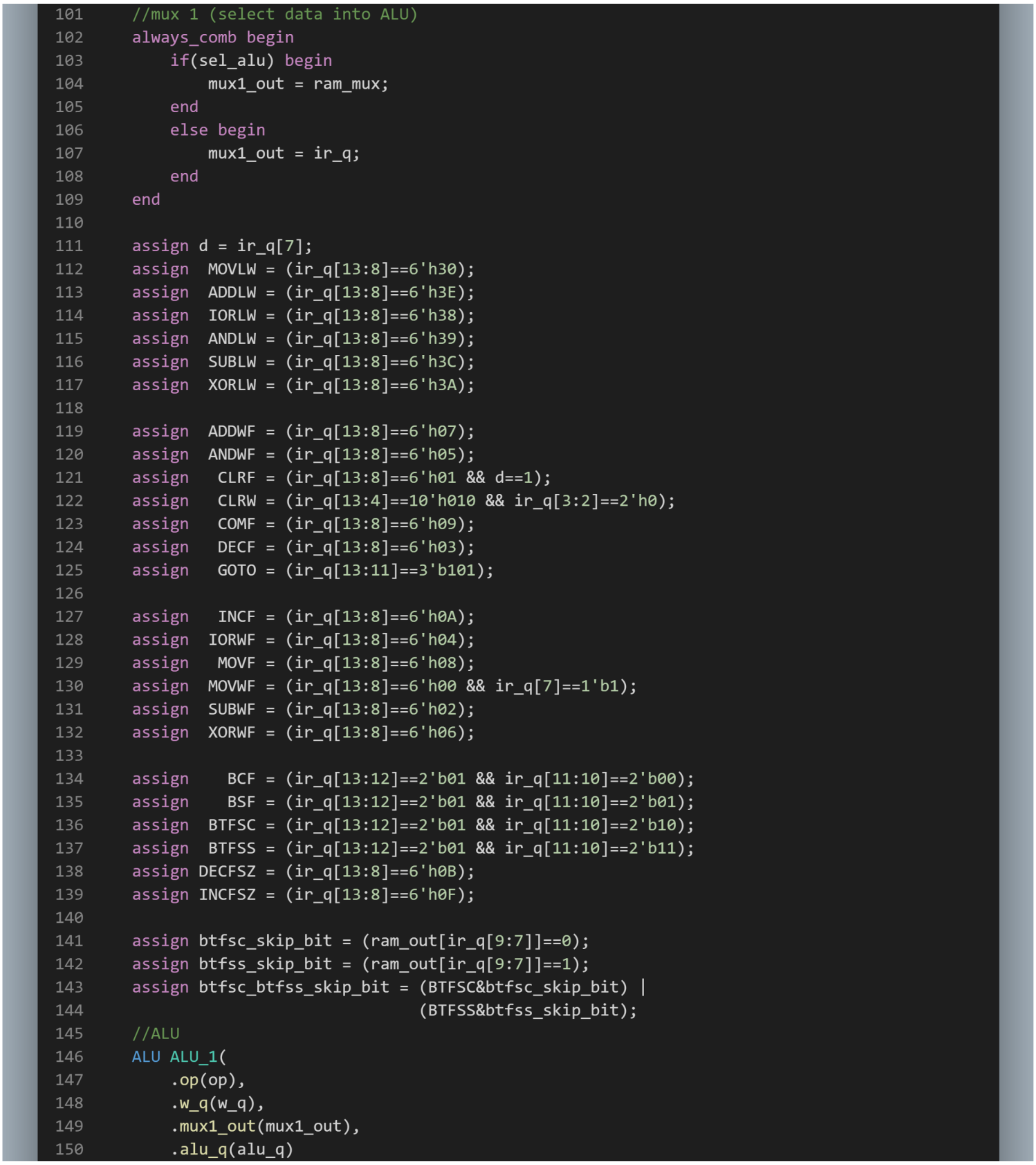
```
module single_port_ram_128x8(
    input [7:0]data,
    input [6:0]addr,
    input ram_en,
    input clk,
    output logic [7:0] ram_out
);
    // Declare the RAM variable
    //reg [DATA_WIDTH-1:0] ram[2**ADDR_WIDTH-1:0];
    logic [7:0] ram[127:0];

    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
```



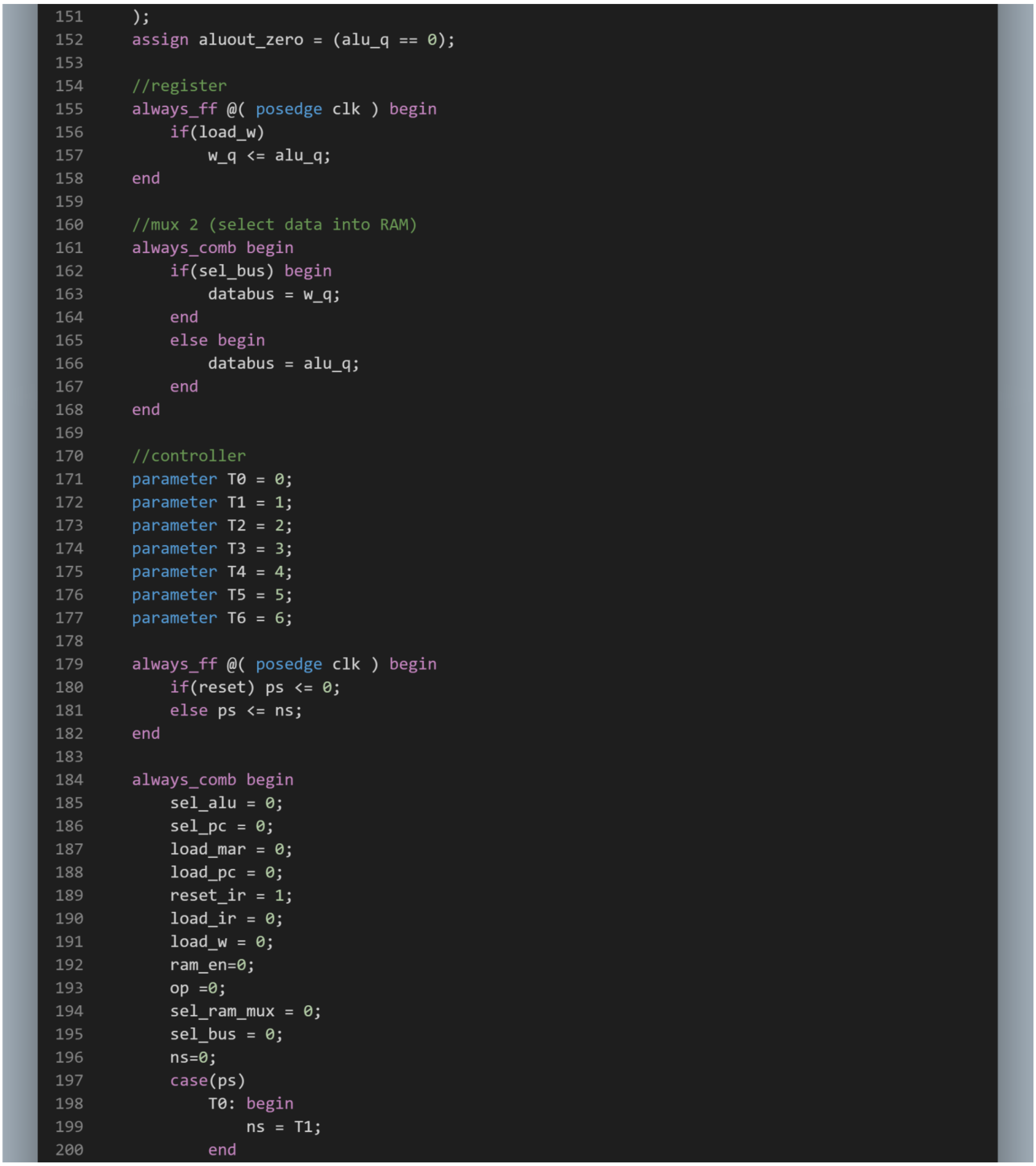
```systemverilog
module ALU (
    input [3:0] op,
    input [7:0] w_q, mux1_out,
    output logic [7:0] alu_q
);
always_comb
begin
    case(op)
        0: alu_q = mux1_out + w_q;
        1: alu_q = mux1_out - w_q;
        2: alu_q = mux1_out & w_q;
        3: alu_q = mux1_out | w_q;
        4: alu_q = mux1_out ^ w_q; //XOR
        5: alu_q = mux1_out;
        6: alu_q = mux1_out + 1;
        7: alu_q = mux1_out - 1;
        8: alu_q = 0;
        9: alu_q = ~mux1_out;
        default: alu_q = mux1_out + w_q;
    endcase
end

endmodule
```



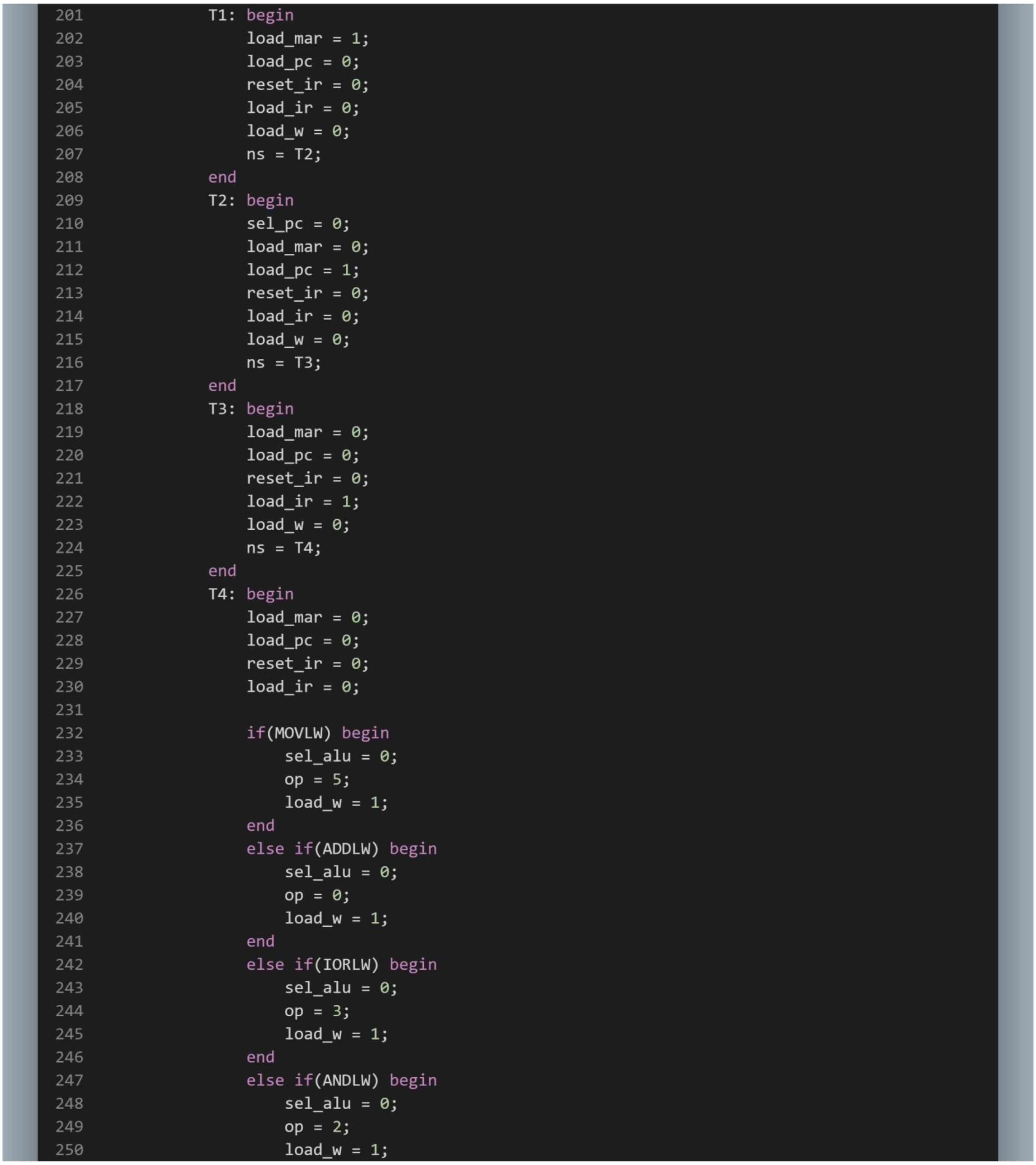
```
module cpu (
    input clk,
    input reset,
    output logic [7:0] w_q
);
    logic [13:0] rom_q, ir_q;
    logic [10:0] pc_next, pc_q, mar_q;
    logic load_pc, load_mar, load_ir, reset_ir, load_w, sel_pc, ram_en, sel_alu, d, sel_bus;
    logic [3:0] ps,ns;
    logic [7:0] alu_q, ram_out, mux1_out, databus, bcf_mux, bsf_mux, ram_mux;
    logic [3:0] op;
    logic [5:0] opcode;
    logic [2:0] sel_bit;
    logic [1:0] sel_ram_mux;
    //mux 0 (select next PC address)
    always_comb begin
        if(sel_pc) 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_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;
            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

```



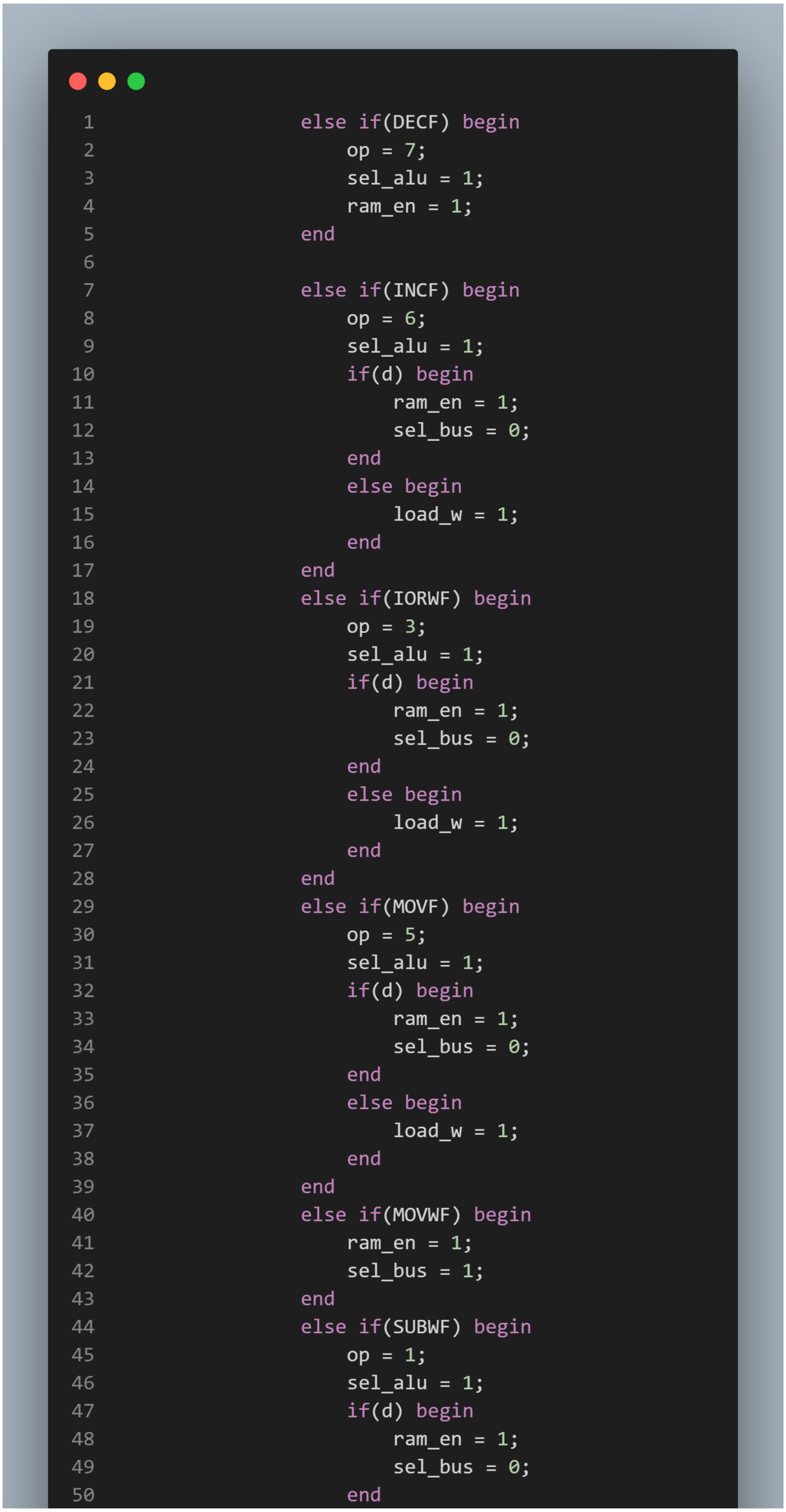
```
    //mux 1 (select data into ALU)
    always_comb begin
        if(sel_alu) begin
            mux1_out = ram_mux;
        end
        else begin
            mux1_out = ir_q;
        end
    end

    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);
    //ALU
    ALU ALU_1(
        .op(op),
        .w_q(w_q),
        .mux1_out(mux1_out),
        .alu_q(alu_q)
```



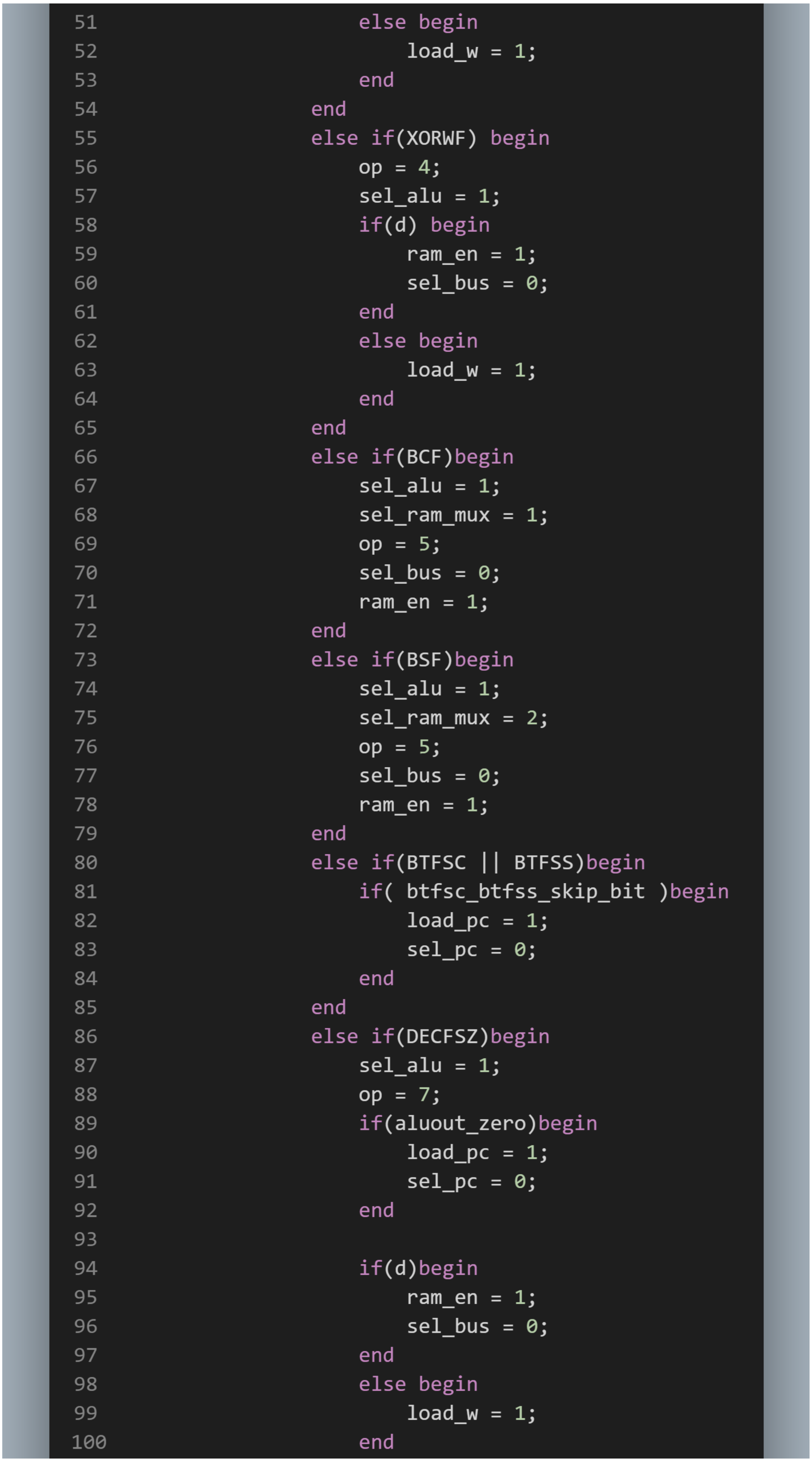
```
    );
    assign aluout_zero = (alu_q == 0);

    //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

    //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 = 1;
        load_ir = 0;
        load_w = 0;
        ram_en=0;
        op =0;
        sel_ram_mux = 0;
        sel_bus = 0;
        ns=0;
        case(ps)
            T0: begin
                ns = T1;
            end
```



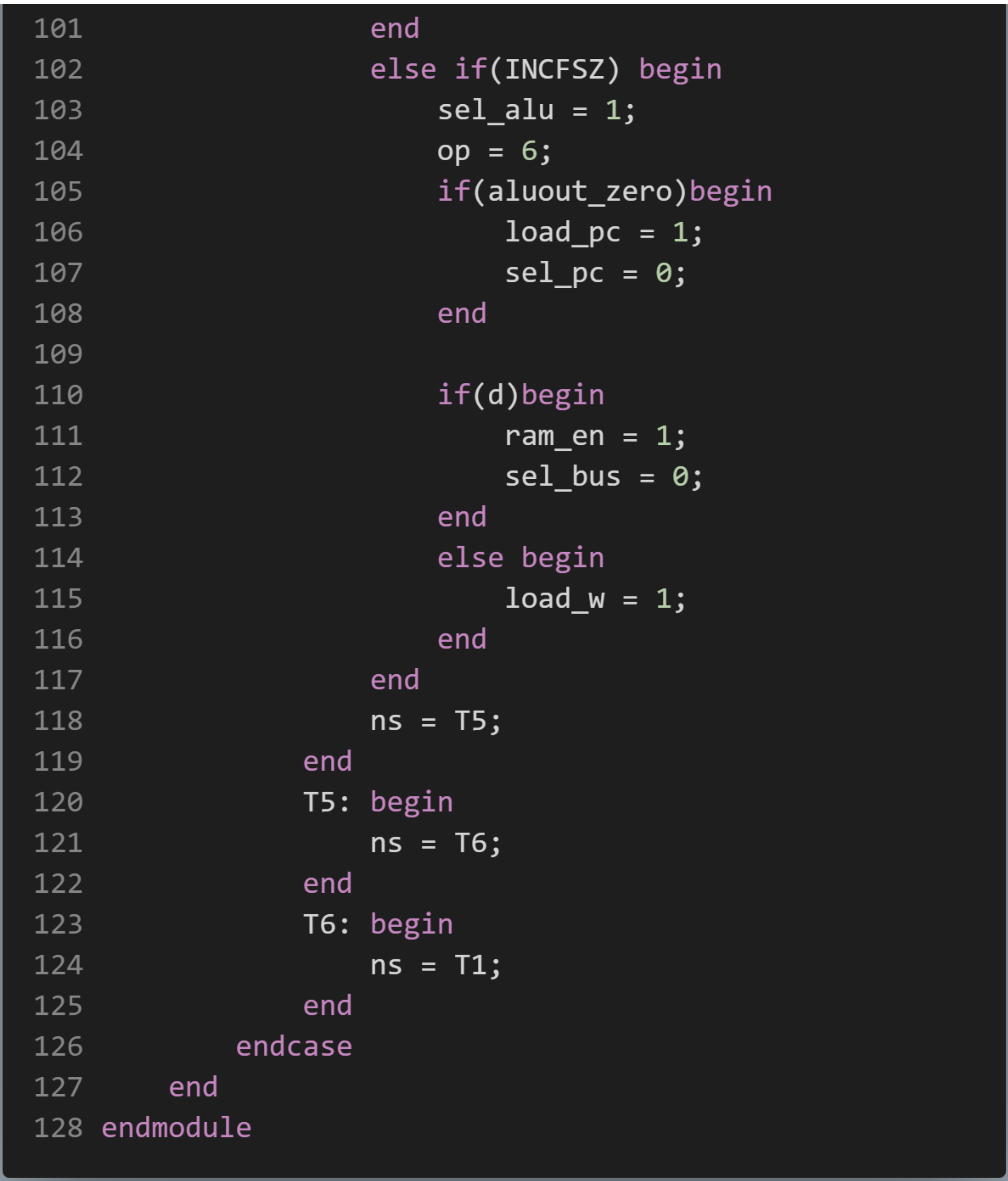
```
            T1: begin
                load_mar = 1;
                load_pc = 0;
                reset_ir = 0;
                load_ir = 0;
                load_w = 0;
                ns = T2;
            end
            T2: begin
                sel_pc = 0;
                load_mar = 0;
                load_pc = 1;
                reset_ir = 0;
                load_ir = 0;
                load_w = 0;
                ns = T3;
            end
            T3: begin
                load_mar = 0;
                load_pc = 0;
                reset_ir = 0;
                load_ir = 1;
                load_w = 0;
                ns = T4;
            end
            T4: begin
                load_mar = 0;
                load_pc = 0;
                reset_ir = 0;
                load_ir = 0;

                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
                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(GOTO) begin
                    sel_pc = 1;
                    load_pc = 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
                        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
```



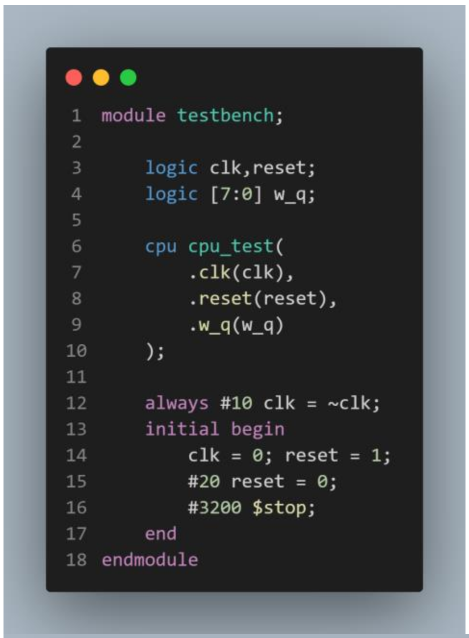
```
                else if(DECF) begin
                    op = 7;
                    sel_alu = 1;
                    ram_en = 1;
                end

                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
                    ram_en = 1;
                    sel_bus = 1;
                end
                else if(SUBWF) begin
                    op = 1;
                    sel_alu = 1;
                    if(d) begin
                        ram_en = 1;
                        sel_bus = 0;
                    end
```



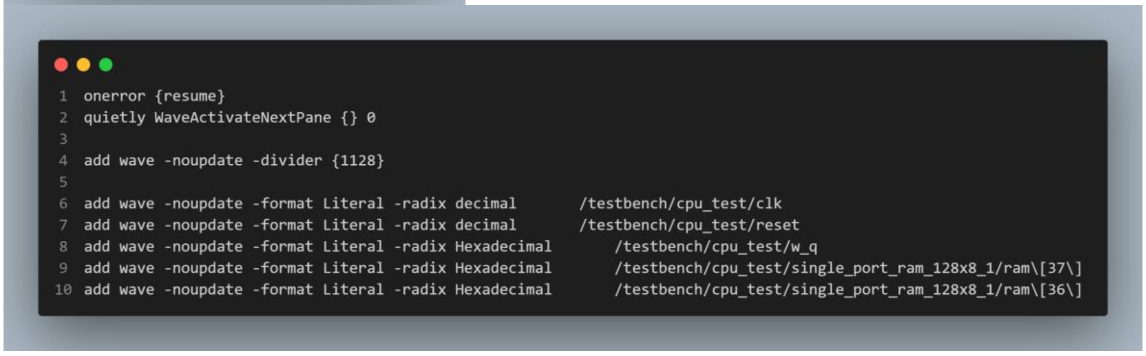
```
                        else begin
                            load_w = 1;
                        end
                    end
                    else if(XORWF) begin
                        op = 4;
                        sel_alu = 1;
                        if(d) begin
                            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(BTFSC || BTFSS)begin
                        if( btfsc_btfss_skip_bit )begin
                            load_pc = 1;
                            sel_pc = 0;
                        end
                    end
                    else if(DECFSZ)begin
                        sel_alu = 1;
                        op = 7;
                        if(aluout_zero)begin
                            load_pc = 1;
                            sel_pc = 0;
                        end

                        if(d)begin
                            ram_en = 1;
                            sel_bus = 0;
                        end
                        else begin
                            load_w = 1;
                        end
```



```
                end
                else if(INCFSZ) begin
                    sel_alu = 1;
                    op = 6;
                    if(aluout_zero)begin
                        load_pc = 1;
                        sel_pc = 0;
                    end

                    if(d)begin
                        ram_en = 1;
                        sel_bus = 0;
                    end
                    else begin
                        load_w = 1;
                    end
                end
                ns = T5;
            end
            T5: begin
                ns = T6;
            end
            T6: begin
                ns = T1;
            end
        endcase
    end
endmodule
```

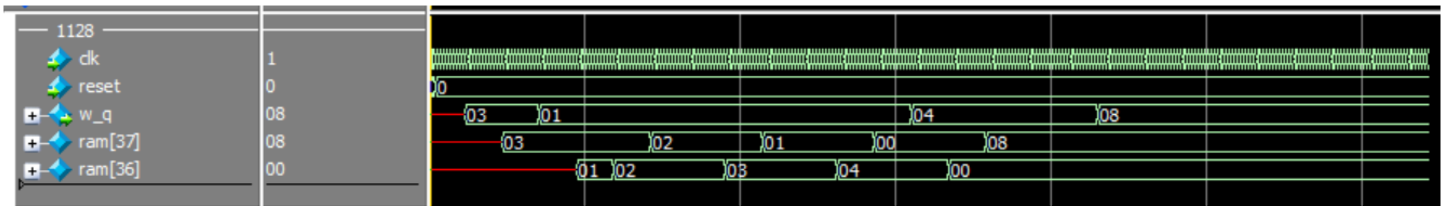


```
module testbench;

    logic clk,reset;
    logic [7:0] w_q;

    cpu cpu_test(
        .clk(clk),
        .reset(reset),
        .w_q(w_q)
    );

    always #10 clk = ~clk;
    initial begin
        clk = 0; reset = 1;
        #20 reset = 0;
        #3200 $stop;
    end
endmodule
```



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

add wave -noupdate -divider {1128}

add wave -noupdate -format Literal -radix decimal       /testbench/cpu_test/clk
add wave -noupdate -format Literal -radix decimal       /testbench/cpu_test/reset
add wave -noupdate -format Literal -radix Hexadecimal       /testbench/cpu_test/w_q
add wave -noupdate -format Literal -radix Hexadecimal       /testbench/cpu_test/single_port_ram_128x8_1/ram\[37\]
add wave -noupdate -format Literal -radix Hexadecimal       /testbench/cpu_test/single_port_ram_128x8_1/ram\[36\]
```

- **模擬結果與結果說明：**



能夠成功執行從組合語言轉譯過來的 program_rom 指令(movlw 03=>w=3......)

- **結論與心得：**

這次的作業增加了很多指令，硬體架構也有蠻大的變化，幸好能夠以舊的程式碼進行修改，讓我不至於從頭去思考整個架構是如何運作，只需要修改部分的程式碼就好，雖然執行的過程中發現結果不對，但是在助教以及同學的協助下，發現到有腳位沒有給預設值，以及一個硬體架構忘記修改，最終也成功執行出正確的結果，感謝助教的耐心指導。