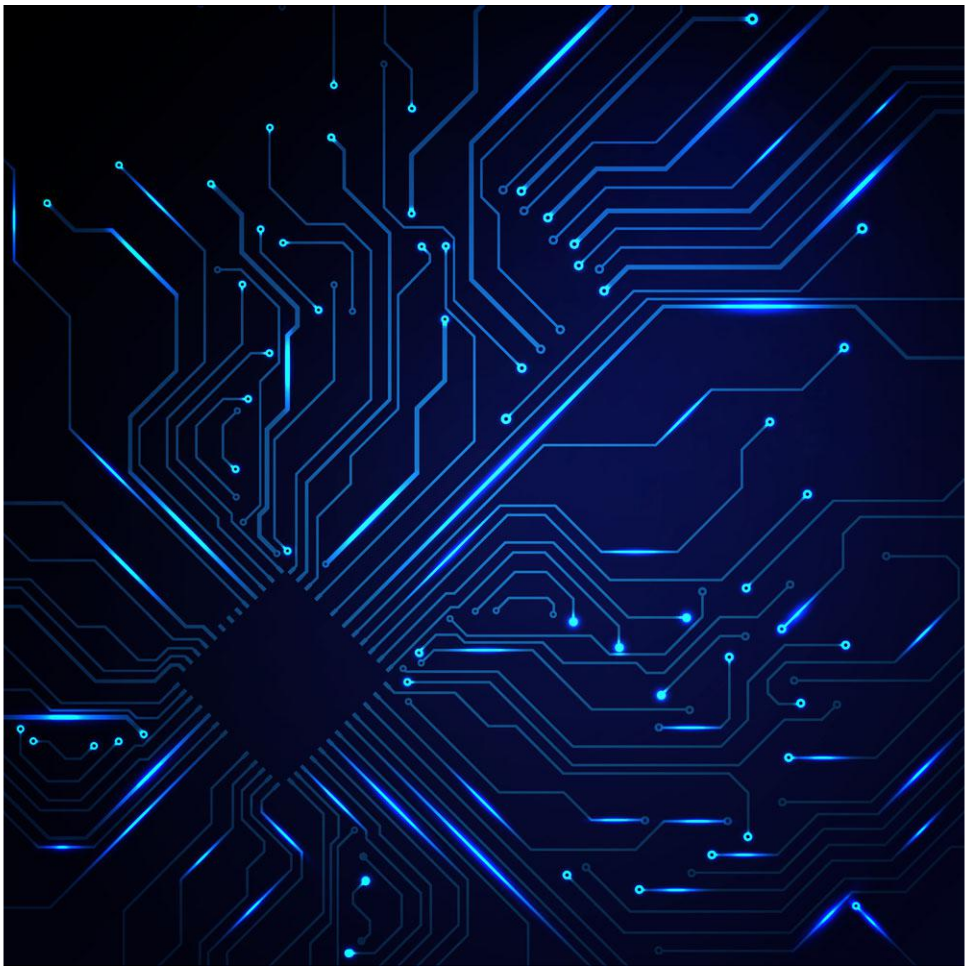


2022/12/24
實驗十三
跳躍指令

姓名：王嘉羽　　　學號：00957116

班級：資工 3B

E-mail：vayne20011125@gmail.com

2022/12/24

注意

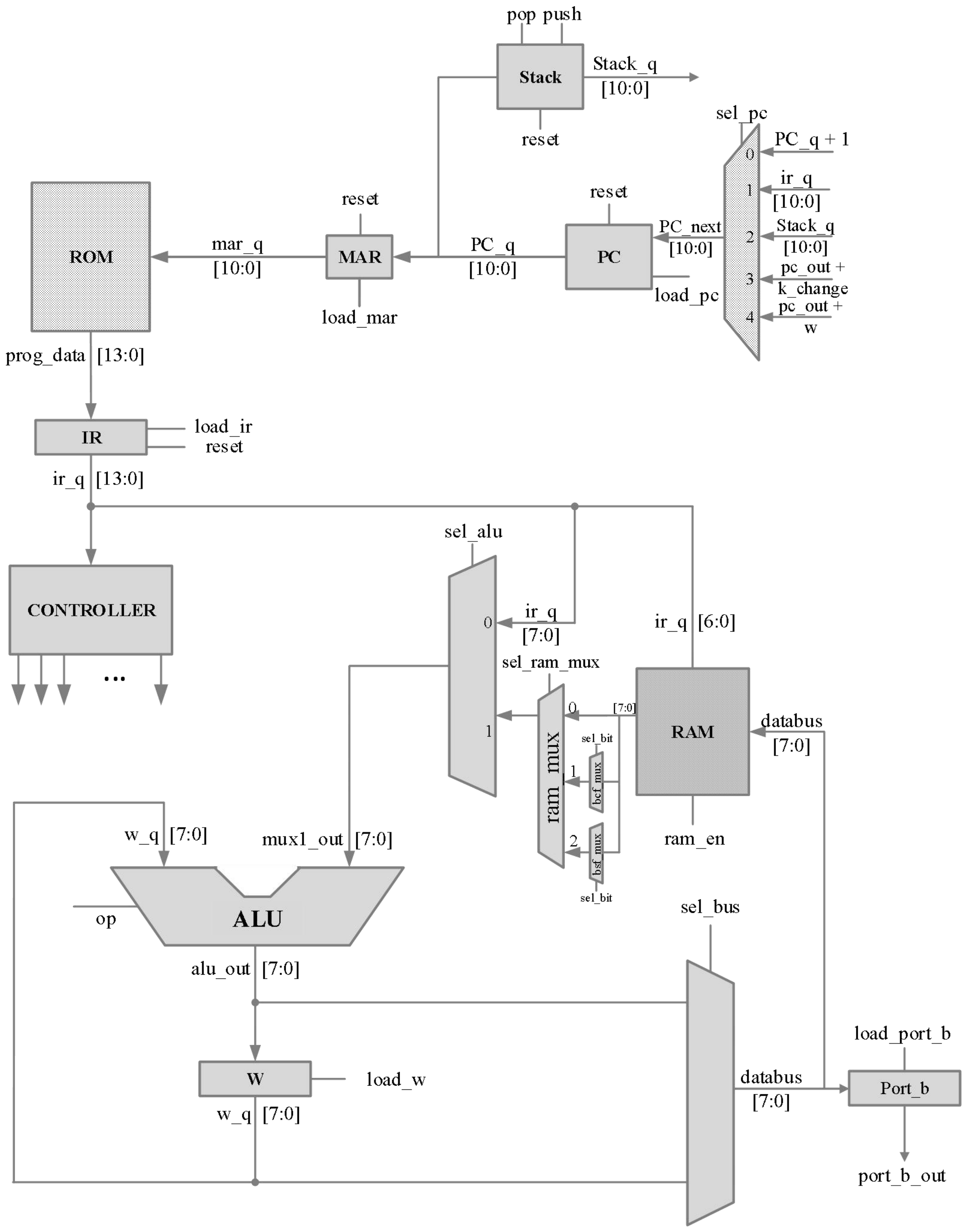
1. 繳交時一律轉 PDF 檔
2. 一人繳交一份
3. 檔名：學號_HW?.pdf
   檔名請按照作業檔名格式進行填寫
   未依照格式不予批改

● 實驗說明：

用 MPLAB 設計一個 Rom，使 0x21 和 0x22 兩個位址的 16 進制分別表示時鐘的時及分，即 0x22(分)的 16 進制會由 0 數到 59 後歸零，每當 0x22(分)歸零 0x21(時)就會加 1，每當做到 23:59 後會全部歸零

請交 MPLAB 專案及程式碼截圖，存時跟分的暫存器請分別設定為 0x21 跟 0x22
請用 BRA 的指令代替 goto 指令去做使用，使用方式一樣

● 系統硬體架構方塊圖（接線圖）：

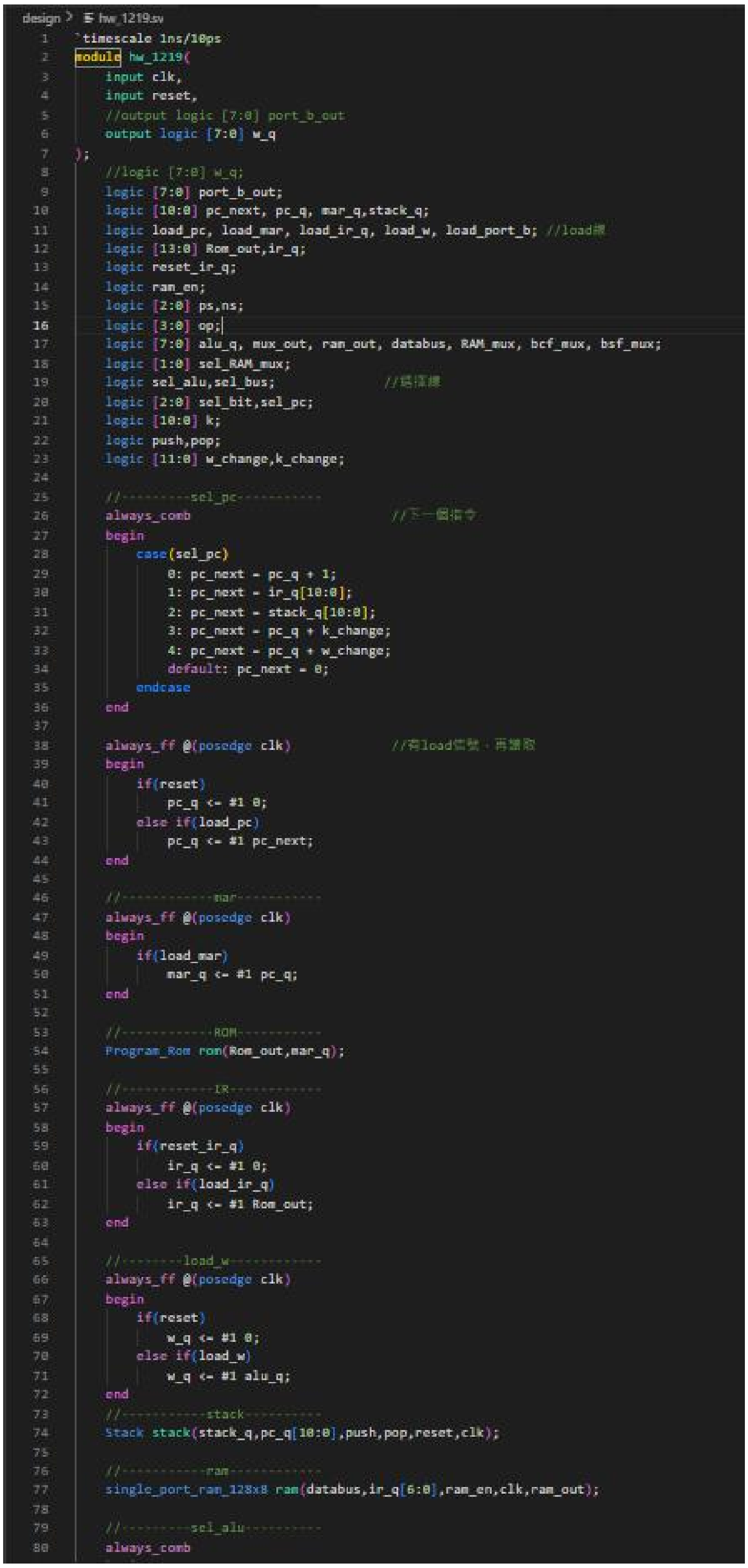


架構圖

- **系統架構程式碼、測試資料程式碼與程式碼說明**

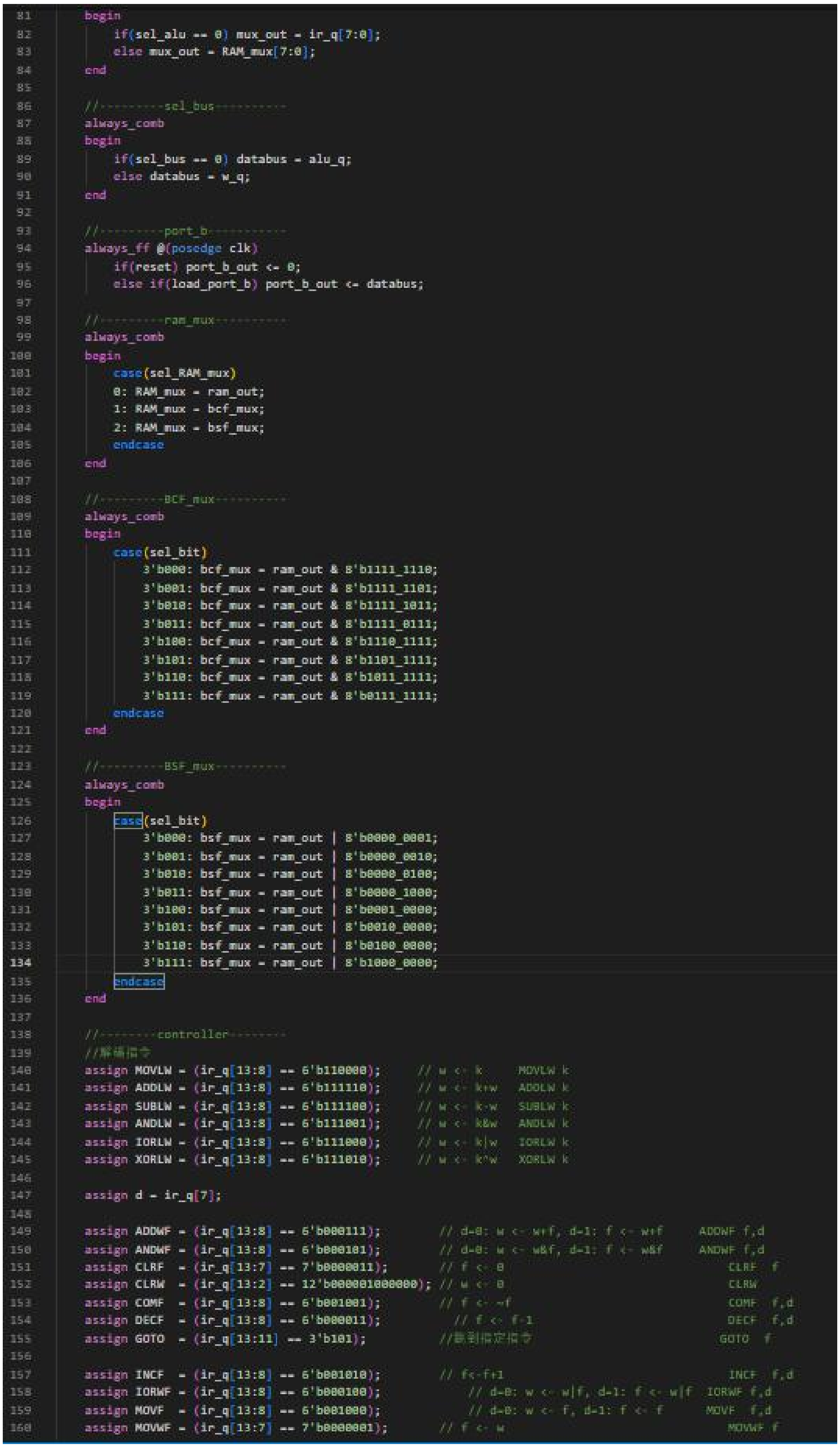
  截圖請善用 win+shift+S

✧ hw_1219.sv

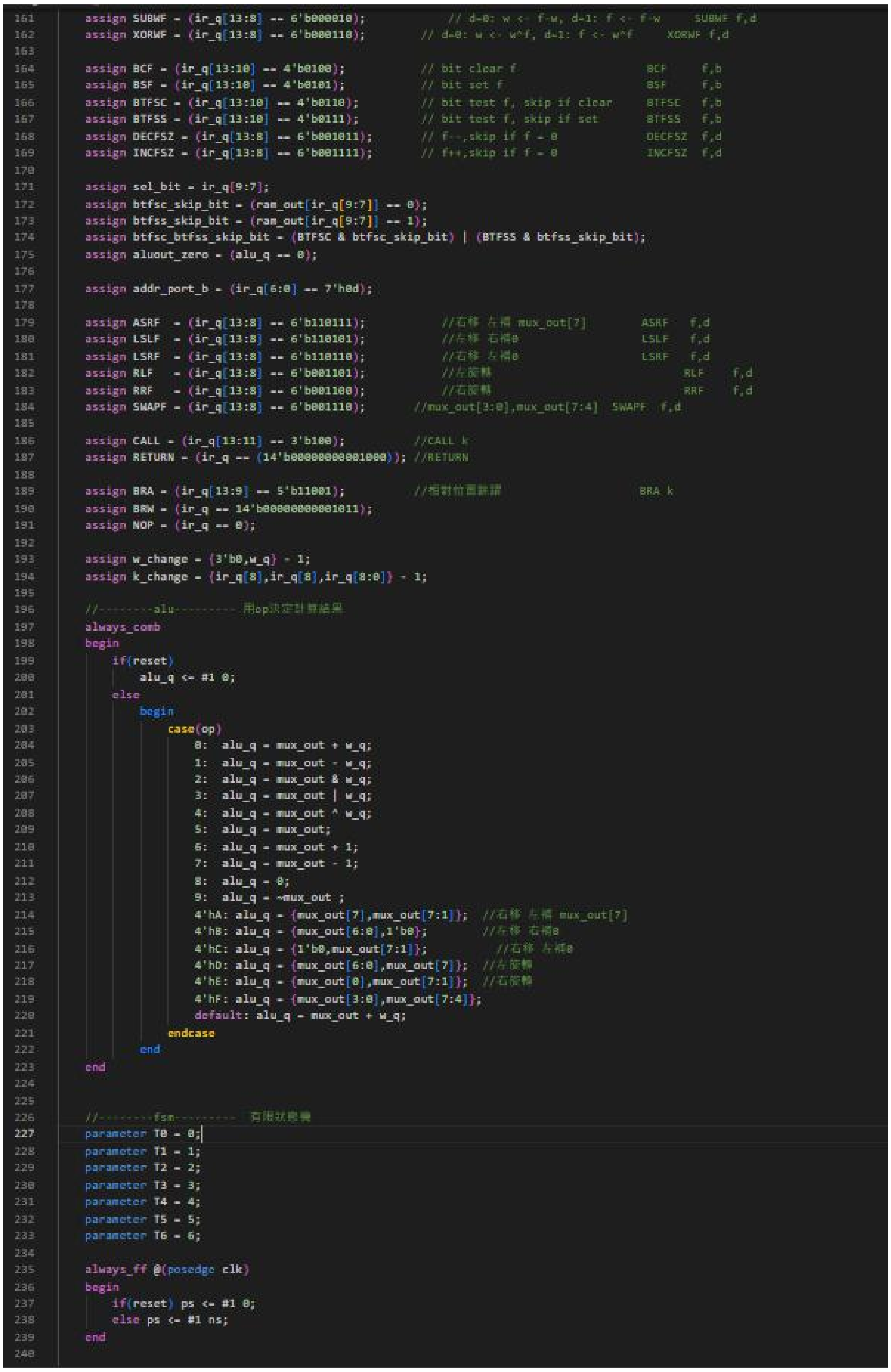


design > hw_1219.sv

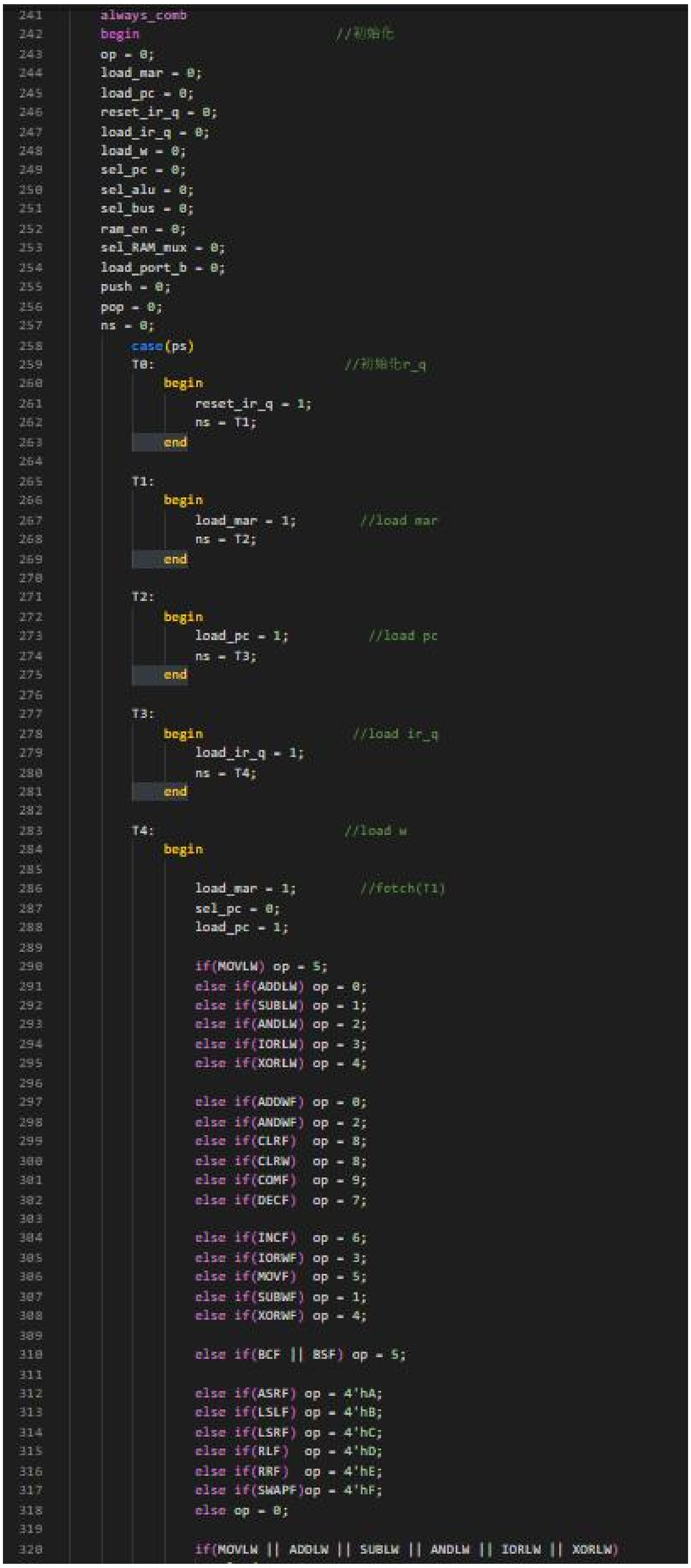
```
`timescale 1ns/10ps
module hw_1219(
    input clk,
    input reset,
    //output logic [7:0] port_b_out
    output logic [7:0] w_q
);
    //logic [7:0] w_q;
    logic [7:0] port_b_out;
    logic [10:0] pc_next, pc_q, mar_q,stack_q;
    logic load_pc, load_mar, load_ir_q, load_w, load_port_b; //load線
    logic [13:0] Rom_out,ir_q;
    logic reset_ir_q;
    logic ram_en;
    logic [2:0] ps,ns;
    logic [3:0] op;
    logic [7:0] alu_q, mux_out, ram_out, databus, RAM_mux, bcf_mux, bsf_mux;
    logic [1:0] sel_RAM_mux;
    logic sel_alu,sel_bus;             //選擇線
    logic [2:0] sel_bit,sel_pc;
    logic [10:0] k;
    logic push,pop;
    logic [11:0] w_change,k_change;

    //---------sel_pc-----------
    always_comb                        //下一個指令
    begin
        case(sel_pc)
            0: pc_next = pc_q + 1;
            1: pc_next = ir_q[10:0];
            2: pc_next = stack_q[10:0];
            3: pc_next = pc_q + k_change;
            4: pc_next = pc_q + w_change;
            default: pc_next = 0;
        endcase
    end

    always_ff @(posedge clk)           //有load信號，再讀取
    begin
        if(reset)
            pc_q <= #1 0;
        else if(load_pc)
            pc_q <= #1 pc_next;
    end

    //------------mar-----------
    always_ff @(posedge clk)
    begin
        if(load_mar)
            mar_q <= #1 pc_q;
    end

    //------------ROM-----------
    Program_Rom rom(Rom_out,mar_q);

    //------------IR------------
    always_ff @(posedge clk)
    begin
        if(reset_ir_q)
            ir_q <= #1 0;
        else if(load_ir_q)
            ir_q <= #1 Rom_out;
    end

    //--------load_w------------
    always_ff @(posedge clk)
    begin
        if(reset)
            w_q <= #1 0;
        else if(load_w)
            w_q <= #1 alu_q;
    end
    //----------stack----------
    Stack stack(stack_q,pc_q[10:0],push,pop,reset,clk);

    //-----------ram-----------
    single_port_ram_128x8 ram(databus,ir_q[6:0],ram_en,clk,ram_out);

    //---------sel_alu----------
    always_comb
```



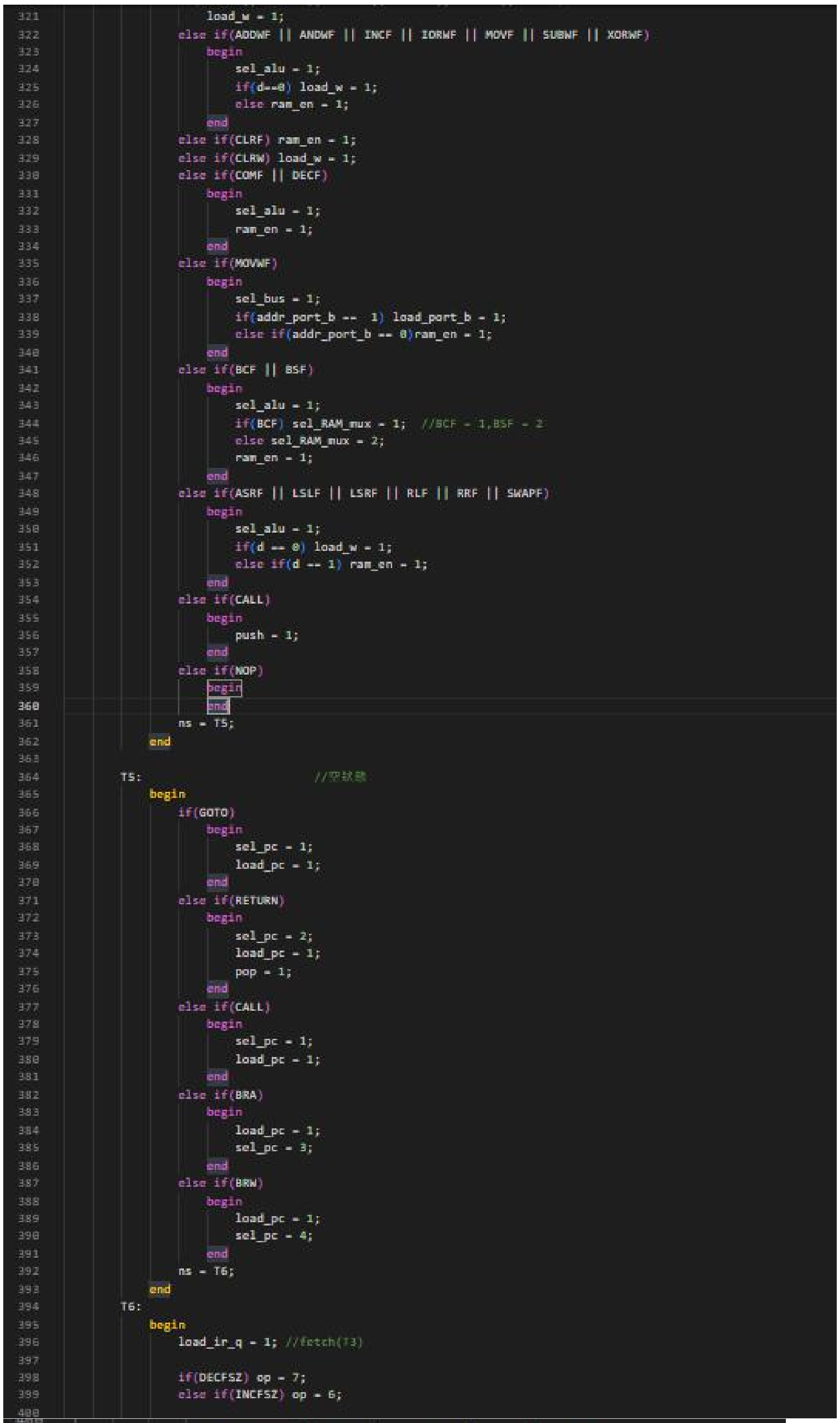
```
    begin
        if(sel_alu == 0) mux_out = ir_q[7:0];
        else mux_out = RAM_mux[7:0];
    end

    //---------sel_bus----------
    always_comb
    begin
        if(sel_bus == 0) databus = alu_q;
        else databus = w_q;
    end

    //---------port_b-----------
    always_ff @(posedge clk)
        if(reset) port_b_out <= 0;
        else if(load_port_b) port_b_out <= databus;

    //---------ram_mux----------
    always_comb
    begin
        case(sel_RAM_mux)
        0: RAM_mux = ram_out;
        1: RAM_mux = bcf_mux;
        2: RAM_mux = bsf_mux;
        endcase
    end

    //---------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

    //--------controller--------
    //解碼指令
    assign MOVLW = (ir_q[13:8] == 6'b110000);     // w <- k     MOVLW k
    assign ADDLW = (ir_q[13:8] == 6'b111110);     // w <- k+w   ADDLW k
    assign SUBLW = (ir_q[13:8] == 6'b111100);     // w <- k-w   SUBLW k
    assign ANDLW = (ir_q[13:8] == 6'b111001);     // w <- k&w   ANDLW k
    assign IORLW = (ir_q[13:8] == 6'b111000);     // w <- k|w   IORLW k
    assign XORLW = (ir_q[13:8] == 6'b111010);     // w <- k^w   XORLW k

    assign d = ir_q[7];

    assign ADDWF = (ir_q[13:8] == 6'b000111);        // d=0: w <- w+f, d=1: f <- w+f     ADDWF f,d
    assign ANDWF = (ir_q[13:8] == 6'b000101);        // d=0: w <- w&f, d=1: f <- w&f     ANDWF f,d
    assign CLRF  = (ir_q[13:7] == 7'b0000011);       // f <- 0                               CLRF  f
    assign CLRW  = (ir_q[13:2] == 12'b000001000000); // w <- 0                               CLRW
    assign COMF  = (ir_q[13:8] == 6'b001001);        // f <- ~f                              COMF  f,d
    assign DECF  = (ir_q[13:8] == 6'b000011);          // f <- f-1                           DECF  f,d
    assign GOTO  = (ir_q[13:11] == 3'b101);          //跳到指定指令                           GOTO  f

    assign INCF  = (ir_q[13:8] == 6'b001010);        // f<-f+1                               INCF  f,d
    assign IORWF = (ir_q[13:8] == 6'b000100);            // d=0: w <- w|f, d=1: f <- w|f  IORWF f,d
    assign MOVF  = (ir_q[13:8] == 6'b001000);            // d=0: w <- f, d=1: f <- f      MOVF  f,d
    assign MOVWF = (ir_q[13:7] == 7'b0000001);       // f <- w                               MOVWF f
```



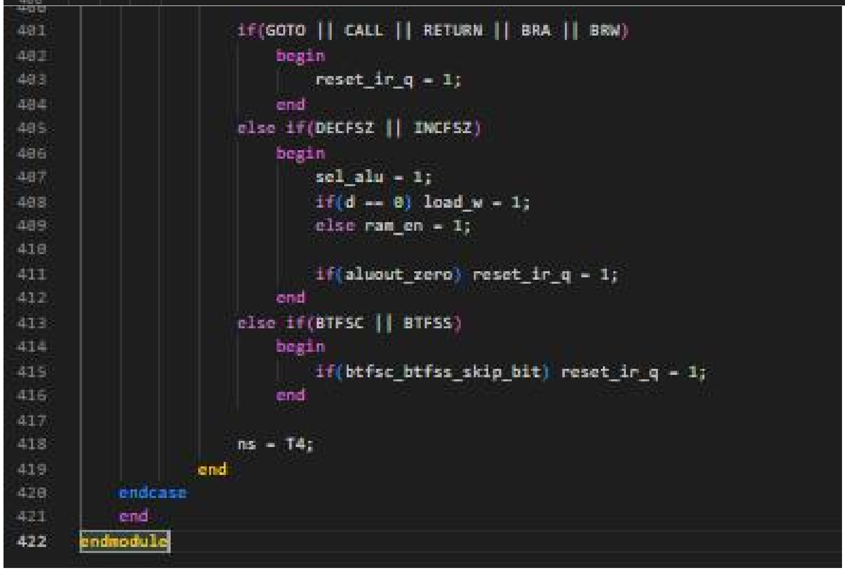
```
    assign SUBWF = (ir_q[13:8] == 6'b000010);            // d=0: w <- f-w, d=1: f <- f-w     SUBWF f,d
    assign XORWF = (ir_q[13:8] == 6'b000110);        // d=0: w <- w^f, d=1: f <- w^f     XORWF f,d

    assign BCF = (ir_q[13:10] == 4'b0100);           // bit clear f                    BCF     f,b
    assign BSF = (ir_q[13:10] == 4'b0101);           // bit set f                      BSF     f,b
    assign BTFSC = (ir_q[13:10] == 4'b0110);         // bit test f, skip if clear      BTFSC   f,b
    assign BTFSS = (ir_q[13:10] == 4'b0111);         // bit test f, skip if set        BTFSS   f,b
    assign DECFSZ = (ir_q[13:8] == 6'b001011);       // f--,skip if f = 0              DECFSZ  f,d
    assign INCFSZ = (ir_q[13:8] == 6'b001111);       // f++,skip if f = 0              INCFSZ  f,d

    assign sel_bit = ir_q[9:7];
    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 aluout_zero = (alu_q == 0);

    assign addr_port_b = (ir_q[6:0] == 7'h0d);

    assign ASRF  = (ir_q[13:8] == 6'b110111);           //右移 左補 mux_out[7]        ASRF   f,d
    assign LSLF  = (ir_q[13:8] == 6'b110101);           //左移 右補0                  LSLF   f,d
    assign LSRF  = (ir_q[13:8] == 6'b110110);           //右移 左補0                  LSRF   f,d
    assign RLF   = (ir_q[13:8] == 6'b001101);           //左旋轉                             RLF    f,d
    assign RRF   = (ir_q[13:8] == 6'b001100);           //右旋轉                             RRF    f,d
    assign SWAPF = (ir_q[13:8] == 6'b001110);       //mux_out[3:0],mux_out[7:4]  SWAPF  f,d

    assign CALL = (ir_q[13:11] == 3'b100);          //CALL k
    assign RETURN = (ir_q == (14'b00000000001000)); //RETURN

    assign BRA = (ir_q[13:9] == 5'b11001);          //相對位置跳躍                    BRA k
    assign BRW = (ir_q == 14'b00000000001011);
    assign NOP = (ir_q == 0);

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

    //--------alu---------- 用op決定計算結果
    always_comb
    begin
        if(reset)
            alu_q <= #1 0;
        else
            begin
                case(op)
                    0:  alu_q = mux_out + w_q;
                    1:  alu_q = mux_out - w_q;
                    2:  alu_q = mux_out & w_q;
                    3:  alu_q = mux_out | w_q;
                    4:  alu_q = mux_out ^ w_q;
                    5:  alu_q = mux_out;
                    6:  alu_q = mux_out + 1;
                    7:  alu_q = mux_out - 1;
                    8:  alu_q = 0;
                    9:  alu_q = ~mux_out ;
                    4'hA: alu_q = {mux_out[7],mux_out[7:1]};  //右移 左補 mux_out[7]
                    4'hB: alu_q = {mux_out[6:0],1'b0};        //左移 右補0
                    4'hC: alu_q = {1'b0,mux_out[7:1]};           //右移 左補0
                    4'hD: alu_q = {mux_out[6:0],mux_out[7]};  //左旋轉
                    4'hE: alu_q = {mux_out[0],mux_out[7:1]};  //右旋轉
                    4'hF: alu_q = {mux_out[3:0],mux_out[7:4]};
                    default: alu_q = mux_out + w_q;
                endcase
            end
    end


    //--------fsm----------  有限狀態機
    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 <= #1 0;
        else ps <= #1 ns;
    end

```



```
always_comb
begin                               //初始化
op = 0;
load_mar = 0;
load_pc = 0;
reset_ir_q = 0;
load_ir_q = 0;
load_w = 0;
sel_pc = 0;
sel_alu = 0;
sel_bus = 0;
ram_en = 0;
sel_RAM_mux = 0;
load_port_b = 0;
push = 0;
pop = 0;
ns = 0;
    case(ps)
    T0:                             //初始化ir_q
        begin
            reset_ir_q = 1;
            ns = T1;
        end

    T1:
        begin
            load_mar = 1;           //load mar
            ns = T2;
        end

    T2:
        begin
            load_pc = 1;            //load pc
            ns = T3;
        end

    T3:
        begin                       //load ir_q
            load_ir_q = 1;
            ns = T4;
        end

    T4:                             //load w
        begin

            load_mar = 1;           //fetch(T1)
            sel_pc = 0;
            load_pc = 1;

            if(MOVLW) op = 5;
            else if(ADDLW) op = 0;
            else if(SUBLW) op = 1;
            else if(ANDLW) op = 2;
            else if(IORLW) op = 3;
            else if(XORLW) op = 4;

            else if(ADDWF) op = 0;
            else if(ANDWF) op = 2;
            else if(CLRF)  op = 8;
            else if(CLRW)  op = 8;
            else if(COMF)  op = 9;
            else if(DECF)  op = 7;

            else if(INCF)  op = 6;
            else if(IORWF) op = 3;
            else if(MOVF)  op = 5;
            else if(SUBWF) op = 1;
            else if(XORWF) op = 4;

            else if(BCF || BSF) op = 5;

            else if(ASRF) op = 4'hA;
            else if(LSLF) op = 4'hB;
            else if(LSRF) op = 4'hC;
            else if(RLF)  op = 4'hD;
            else if(RRF)  op = 4'hE;
            else if(SWAPF)op = 4'hF;
            else op = 0;

            if(MOVLW || ADDLW || SUBLW || ANDLW || IORLW || XORLW)
```

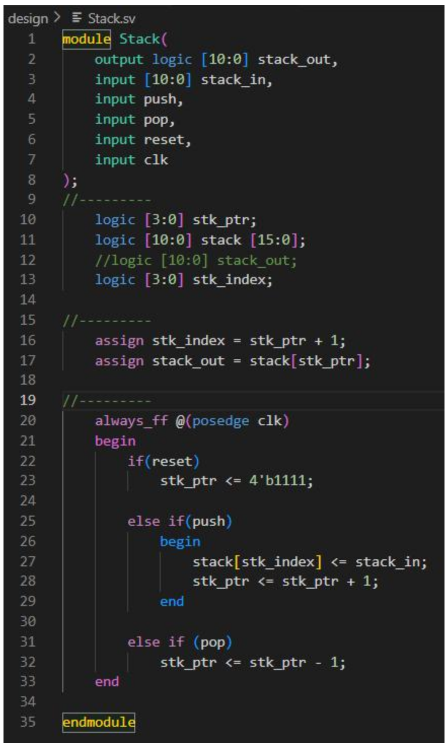


```
                        load_w = 1;
                    else if(ADDWF || ANDWF || INCF || IORWF || MOVF || SUBWF || XORWF)
                        begin
                            sel_alu = 1;
                            if(d==0) load_w = 1;
                            else ram_en = 1;
                        end
                    else if(CLRF) ram_en = 1;
                    else if(CLRW) load_w = 1;
                    else if(COMF || DECF)
                        begin
                            sel_alu = 1;
                            ram_en = 1;
                        end
                    else if(MOVWF)
                        begin
                            sel_bus = 1;
                            if(addr_port_b ==  1) load_port_b = 1;
                            else if(addr_port_b == 0)ram_en = 1;
                        end
                    else if(BCF || BSF)
                        begin
                            sel_alu = 1;
                            if(BCF) sel_RAM_mux = 1;  //BCF = 1,BSF = 2
                            else sel_RAM_mux = 2;
                            ram_en = 1;
                        end
                    else if(ASRF || LSLF || LSRF || RLF || RRF || SWAPF)
                        begin
                            sel_alu = 1;
                            if(d == 0) load_w = 1;
                            else if(d == 1) ram_en = 1;
                        end
                    else if(CALL)
                        begin
                            push = 1;
                        end
                    else if(NOP)
                        begin
                        end
                    ns = T5;
                end

            T5:                         //空狀態
                begin
                    if(GOTO)
                        begin
                            sel_pc = 1;
                            load_pc = 1;
                        end
                    else if(RETURN)
                        begin
                            sel_pc = 2;
                            load_pc = 1;
                            pop = 1;
                        end
                    else if(CALL)
                        begin
                            sel_pc = 1;
                            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_q = 1; //fetch(T3)

                    if(DECFSZ) op = 7;
                    else if(INCFSZ) op = 6;

```



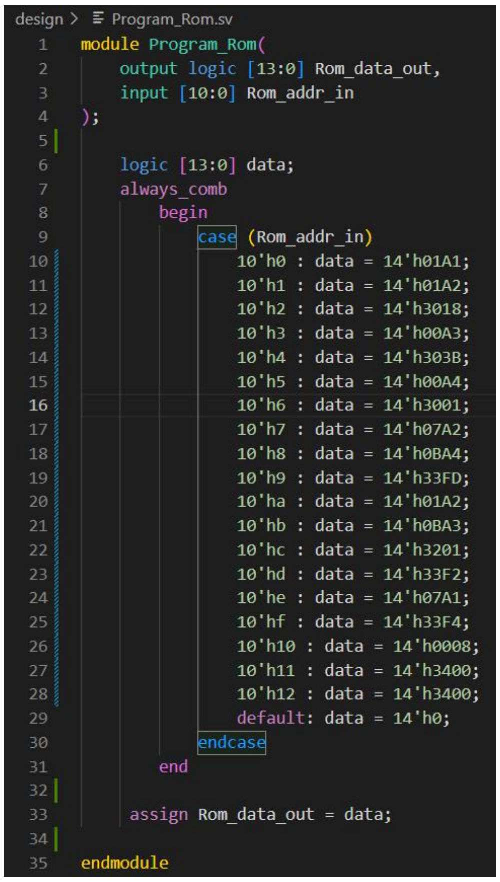
```
                    if(GOTO || CALL || RETURN || BRA || BRW)
                        begin
                            reset_ir_q = 1;
                        end
                    else if(DECFSZ || INCFSZ)
                        begin
                            sel_alu = 1;
                            if(d == 0) load_w = 1;
                            else ram_en = 1;

                            if(aluout_zero) reset_ir_q = 1;
                        end
                    else if(BTFSC || BTFSS)
                        begin
                            if(btfsc_btfss_skip_bit) reset_ir_q = 1;
                        end

                    ns = T4;
                end
    endcase
    end
endmodule
```

✧ Stack.sv



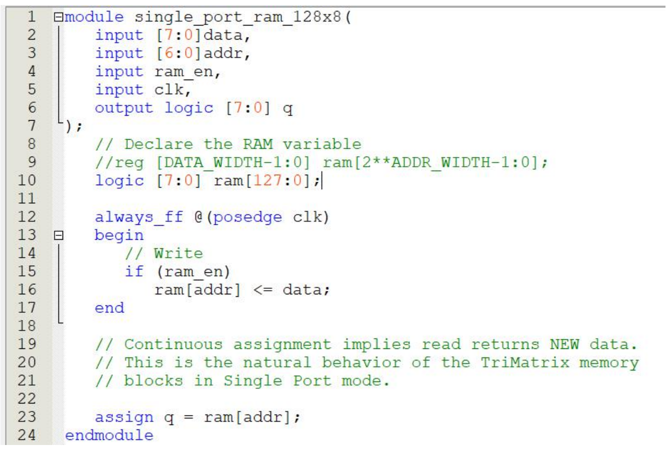
```systemverilog
module Stack(
    output logic [10:0] stack_out,
    input [10:0] stack_in,
    input push,
    input pop,
    input reset,
    input clk
);
//---------
    logic [3:0] stk_ptr;
    logic [10:0] stack [15:0];
    //logic [10:0] stack_out;
    logic [3:0] stk_index;

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

//---------
    always_ff @(posedge clk)
    begin
        if(reset)
            stk_ptr <= 4'b1111;

        else if(push)
            begin
                stack[stk_index] <= stack_in;
                stk_ptr <= stk_ptr + 1;
            end

        else if (pop)
            stk_ptr <= stk_ptr - 1;
    end

endmodule
```

✧ Program_Rom.sv



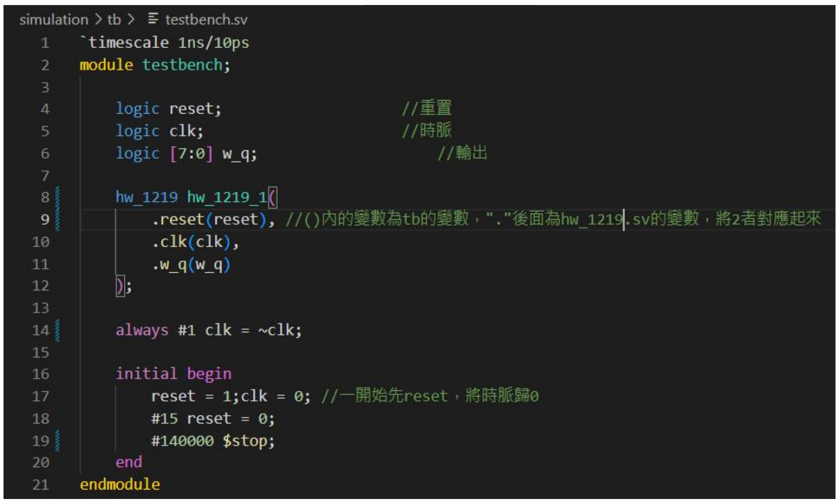
```systemverilog
module Program_Rom(
    output logic [13:0] Rom_data_out,
    input [10:0] Rom_addr_in
);

    logic [13:0] data;
    always_comb
        begin
            case (Rom_addr_in)
                10'h0 : data = 14'h01A1;
                10'h1 : data = 14'h01A2;
                10'h2 : data = 14'h3018;
                10'h3 : data = 14'h00A3;
                10'h4 : data = 14'h303B;
                10'h5 : data = 14'h00A4;
                10'h6 : data = 14'h3001;
                10'h7 : data = 14'h07A2;
                10'h8 : data = 14'h0BA4;
                10'h9 : data = 14'h33FD;
                10'ha : data = 14'h01A2;
                10'hb : data = 14'h0BA3;
                10'hc : data = 14'h3201;
                10'hd : data = 14'h33F2;
                10'he : data = 14'h07A1;
                10'hf : data = 14'h33F4;
                10'h10 : data = 14'h0008;
                10'h11 : data = 14'h3400;
                10'h12 : data = 14'h3400;
                default: data = 14'h0;
            endcase
        end

     assign Rom_data_out = data;

endmodule
```

✧ single_port_ram_128x8.sv



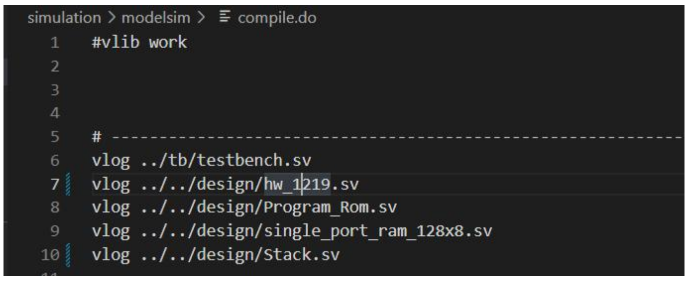
```
module single_port_ram_128x8(
    input [7:0]data,
    input [6:0]addr,
    input ram_en,
    input clk,
    output logic [7:0] q
);
    // Declare the RAM variable
    //reg [DATA_WIDTH-1:0] ram[2**ADDR_WIDTH-1:0];
    logic [7:0] ram[127:0];

    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 q = ram[addr];
endmodule
```

✧ testbench.sv



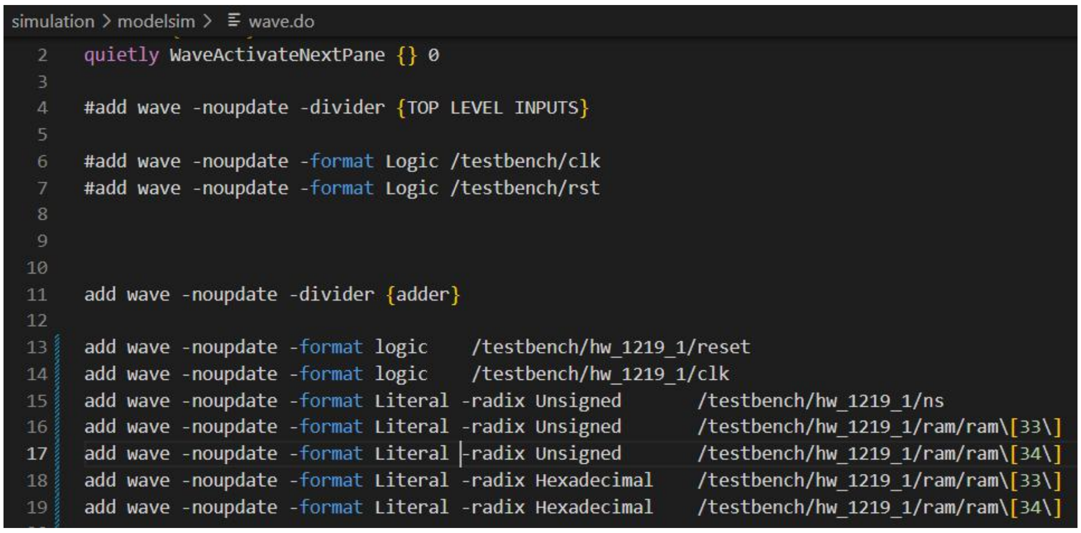
```
`timescale 1ns/10ps
module testbench;

    logic reset;                    //重置
    logic clk;                      //時脈
    logic [7:0] w_q;                    //輸出

    hw_1219 hw_1219_1(
        .reset(reset), //()內的變數為tb的變數，"."後面為hw_1219.sv的變數，將2者對應起來
        .clk(clk),
        .w_q(w_q)
    );

    always #1 clk = ~clk;

    initial begin
        reset = 1;clk = 0; //一開始先reset，將時脈歸0
        #15 reset = 0;
        #140000 $stop;
    end
endmodule
```

✧ compile.do



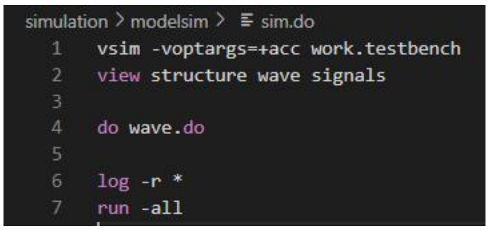
```
#vlib work



# -------------------------------------------------------------
vlog ../tb/testbench.sv
vlog ../../design/hw_1219.sv
vlog ../../design/Program_Rom.sv
vlog ../../design/single_port_ram_128x8.sv
vlog ../../design/Stack.sv
```

✧ wave.do



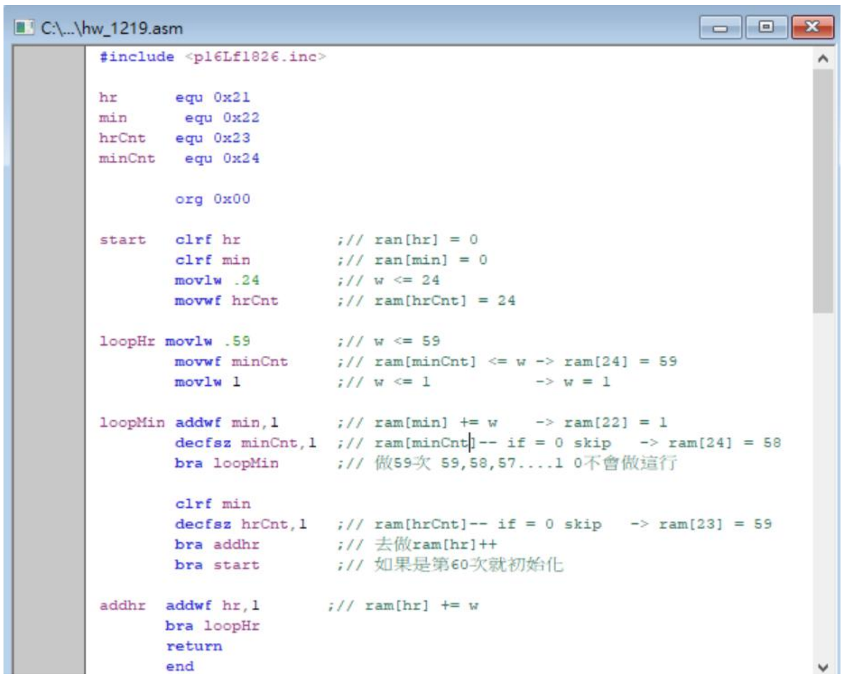
```
quietly WaveActivateNextPane {} 0

#add wave -noupdate -divider {TOP LEVEL INPUTS}

#add wave -noupdate -format Logic /testbench/clk
#add wave -noupdate -format Logic /testbench/rst



add wave -noupdate -divider {adder}

add wave -noupdate -format logic    /testbench/hw_1219_1/reset
add wave -noupdate -format logic    /testbench/hw_1219_1/clk
add wave -noupdate -format Literal -radix Unsigned      /testbench/hw_1219_1/ns
add wave -noupdate -format Literal -radix Unsigned      /testbench/hw_1219_1/ram/ram\[33\]
add wave -noupdate -format Literal -radix Unsigned      /testbench/hw_1219_1/ram/ram\[34\]
add wave -noupdate -format Literal -radix Hexadecimal   /testbench/hw_1219_1/ram/ram\[33\]
add wave -noupdate -format Literal -radix Hexadecimal   /testbench/hw_1219_1/ram/ram\[34\]
```

✧ sim.do


simulation > modelsim > sim.do

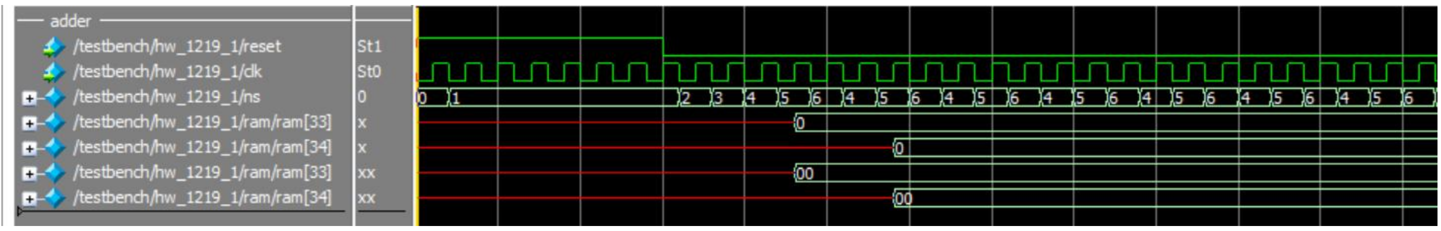
```
vsim -voptargs=+acc work.testbench
view structure wave signals

do wave.do

log -r *
run -all
```

✧ 組合語言程式碼

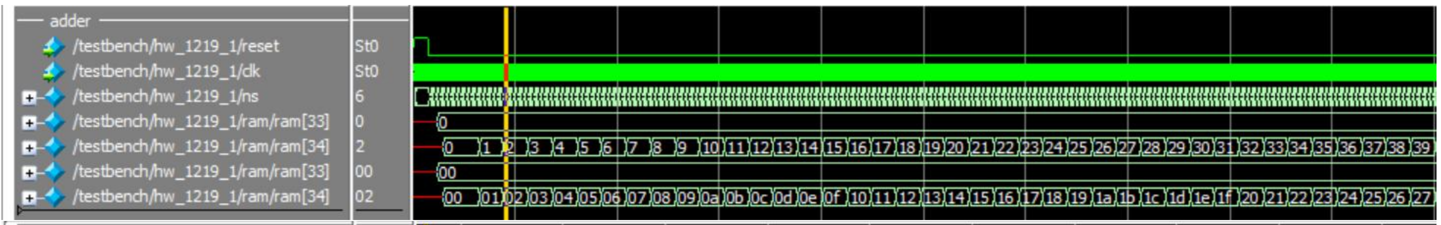

C:\...\hw_1219.asm

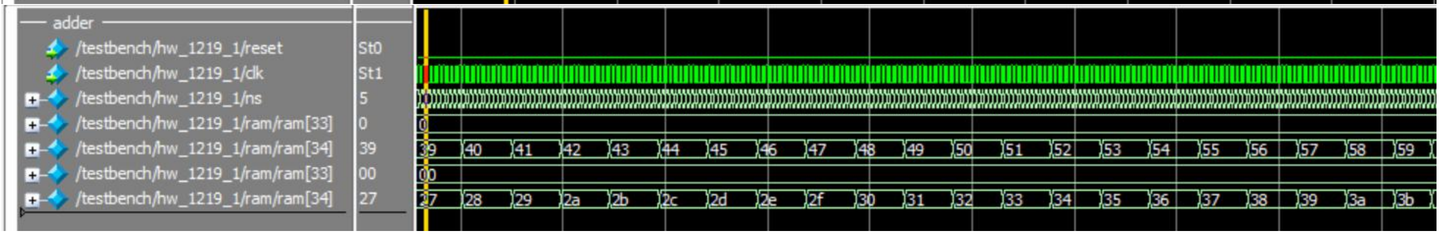
```
#include <p16Lf1826.inc>

hr       equ 0x21
min       equ 0x22
hrCnt    equ 0x23
minCnt    equ 0x24

         org 0x00

start    clrf hr            ;// ran[hr] = 0
         clrf min           ;// ran[min] = 0
         movlw .24          ;// w <= 24
         movwf hrCnt        ;// ram[hrCnt] = 24

loopHr movlw .59            ;// w <= 59
         movwf minCnt       ;// ram[minCnt] <= w -> ram[24] = 59
         movlw 1            ;// w <= 1            -> w = 1

loopMin addwf min,1         ;// ram[min] += w    -> ram[22] = 1
         decfsz minCnt,1   ;// ram[minCnt]-- if = 0 skip   -> ram[24] = 58
         bra loopMin        ;// 做59次 59,58,57....1 0不會做這行

         clrf min
         decfsz hrCnt,1     ;// ram[hrCnt]-- if = 0 skip   -> ram[23] = 59
         bra addhr          ;// 去做ram[hr]++
         bra start          ;// 如果是第60次就初始化

addhr   addwf hr,1        ;// ram[hr] += w
        bra loopHr
        return
        end
```

● 模擬結果與結果說明：

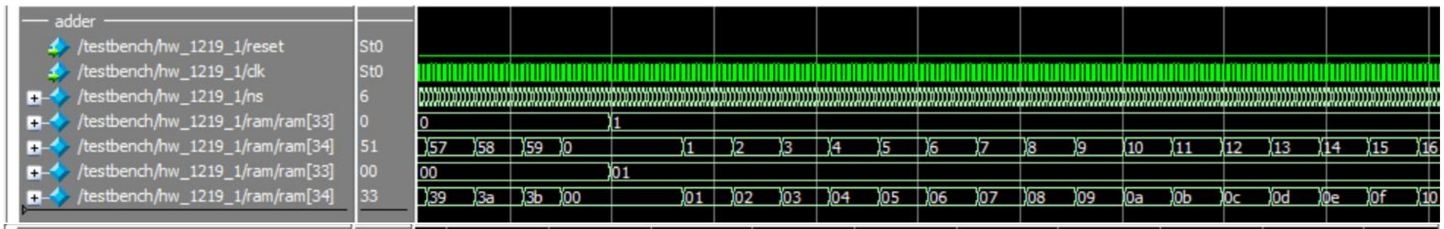


我是用 pipeline ns:0->1->2->3->4->5->6->4->5->6->4 ......

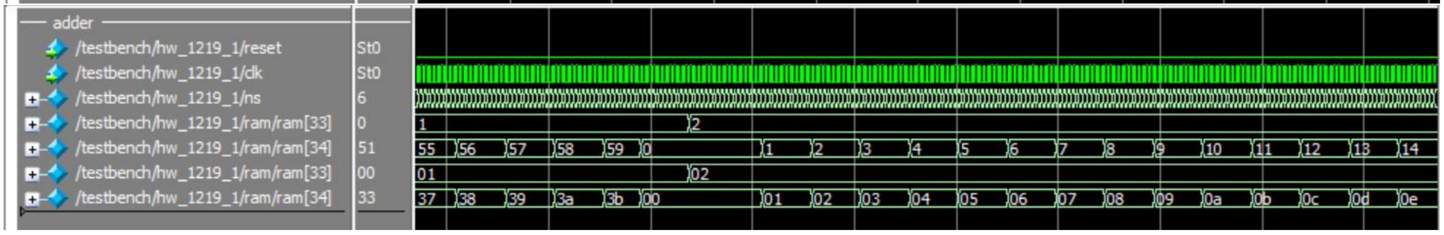




最下面 2 條是 16 進位，中間是 10 進位

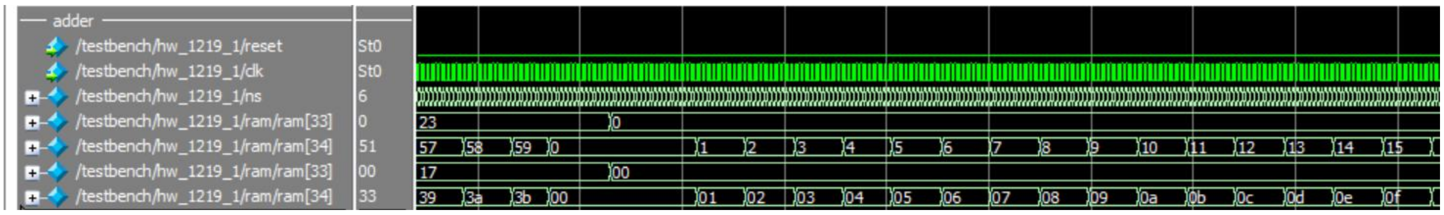


```
adder
/testbench/hw_1219_1/reset        St0
/testbench/hw_1219_1/clk          St0
/testbench/hw_1219_1/ns           6
/testbench/hw_1219_1/ram/ram[33]  0
/testbench/hw_1219_1/ram/ram[34]  51
/testbench/hw_1219_1/ram/ram[33]  00
/testbench/hw_1219_1/ram/ram[34]  33
```



```
adder
/testbench/hw_1219_1/reset        St0
/testbench/hw_1219_1/clk          St0
/testbench/hw_1219_1/ns           6
/testbench/hw_1219_1/ram/ram[33]  0
/testbench/hw_1219_1/ram/ram[34]  51
/testbench/hw_1219_1/ram/ram[33]  00
/testbench/hw_1219_1/ram/ram[34]  33
```

進位的瞬間



```
adder
/testbench/hw_1219_1/reset        St0
/testbench/hw_1219_1/clk          St0
/testbench/hw_1219_1/ns           6
/testbench/hw_1219_1/ram/ram[33]  0
/testbench/hw_1219_1/ram/ram[34]  51
/testbench/hw_1219_1/ram/ram[33]  00
/testbench/hw_1219_1/ram/ram[34]  33
```

歸 0 的瞬間

結論與心得：

- 這次教管線化，終於解決了我的疑惑，我之前一直不知道為甚麼 t5,t6 不做事還要留著，原來是為了要做管線化呀!管線化真的好聰明，他很有效的善用每一個 clk，也善用每一個邏輯閘，但是要好好設計出來真的很困難，要考慮很多事，如果沒有跳躍指令倒是還好，但是一有跳躍指令就要好好思考要怎麼樣才不會打結，設計出來的人真的太厲害了!
- 可能因為 16+2 的關係，這次一次出 3 份作業，希望老師可以開到第 18 週 qq hw14 要寫 4 題 c++轉組合語言，感覺好困難，但我會努力的!
- 下禮拜就要段考了，有鑑於我在把 code 改成管線化時，發現我 return 打 push = 1...哇!所以我決定在考試前好好的在檢查一次我的程式碼，以免考試的時候找不到 bug..畢竟不能帶 ppt 嘛，然後我好擔心考組合語言，所以我也把組合語言的用法當成註解打在指令旁邊了，像是 addlw f,d 之類的，還有他的意義也打下來了，希望考試順利><今天是平安夜~提前祝助教聖誕快樂! 但可能助教是很之後才看，那順便祝 2023 快樂!