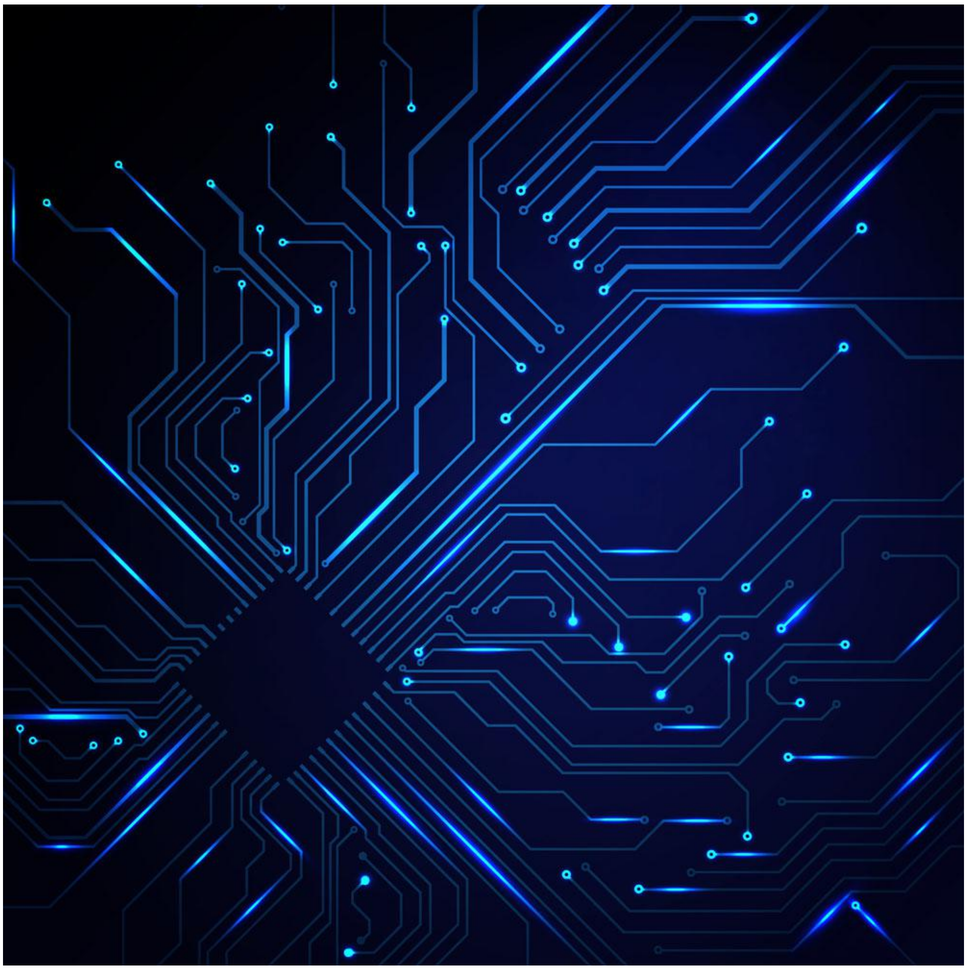


2022/12/12

實驗十二

跳躍指令

姓名：王嘉羽　　　學號：00957116

班級：資工 3B

E-mail：vayne20011125@gmail.com

2022/12/12

注意

1. 繳交時一律轉 PDF 檔
2. 繳交期限為
   上完課後
   本周日晚上 11:59 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

一、

● 實驗說明：

設計組合語言，用來顯示時鐘的「秒」，運行於 PIC MCU 上。由 00 數到 59 再歸 00，並且不斷重複，將結果輸出於 port_ B 上並以 16 進位顯示(10 進位不算分)

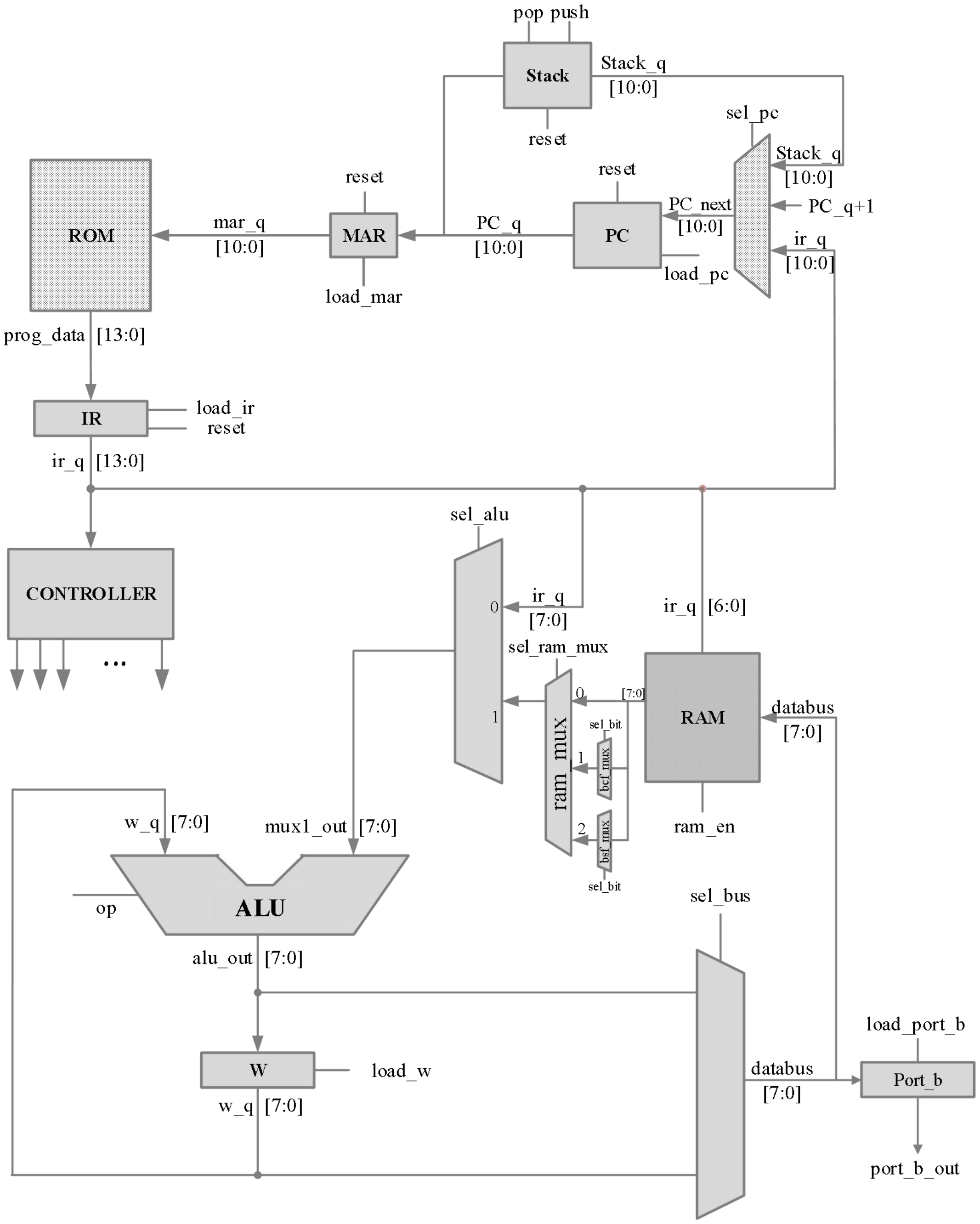
請用 BRA 或 BRW 指令代替 GOTO 指令

加分：在課堂實作或補強時將此架構燒錄到 DE0 上的結果給助教檢查，即可加分。兩個七段顯示器分別顯示時鐘的秒之高低位數。接法如圖二

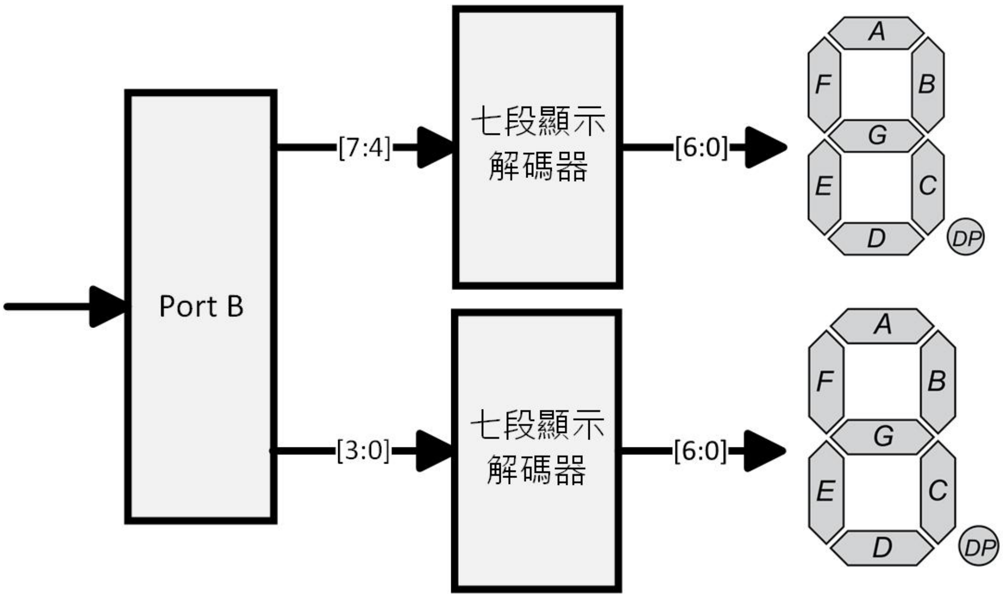
模擬用的組合語言不用加 delay 方便波形觀測

請交 MPLAB 中組合語言截圖、程式碼截圖與波形圖

● 系統硬體架構方塊圖（接線圖）：



圖一、架構圖

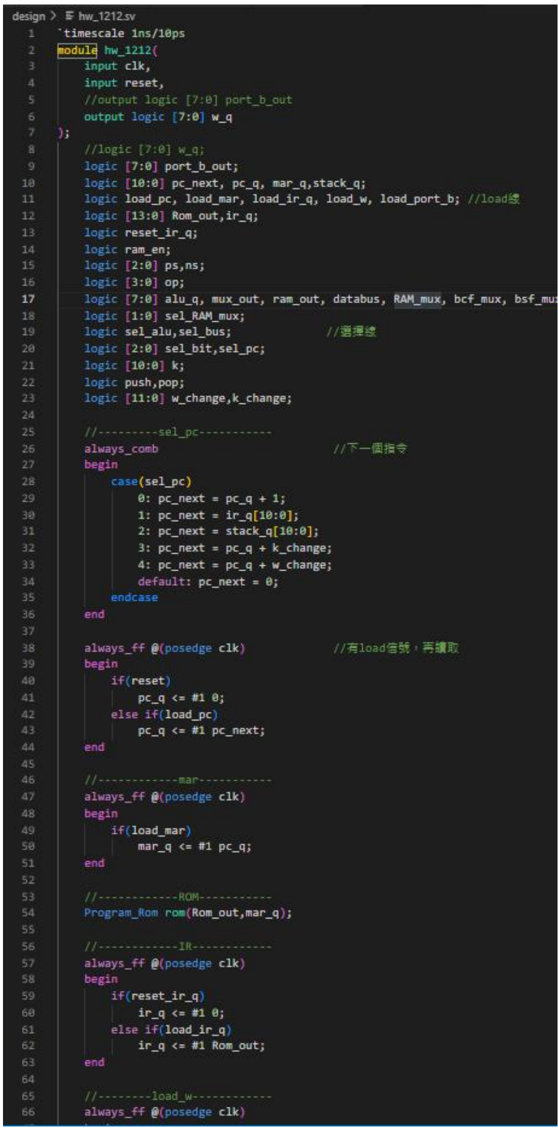


圖二、七段顯示器接法

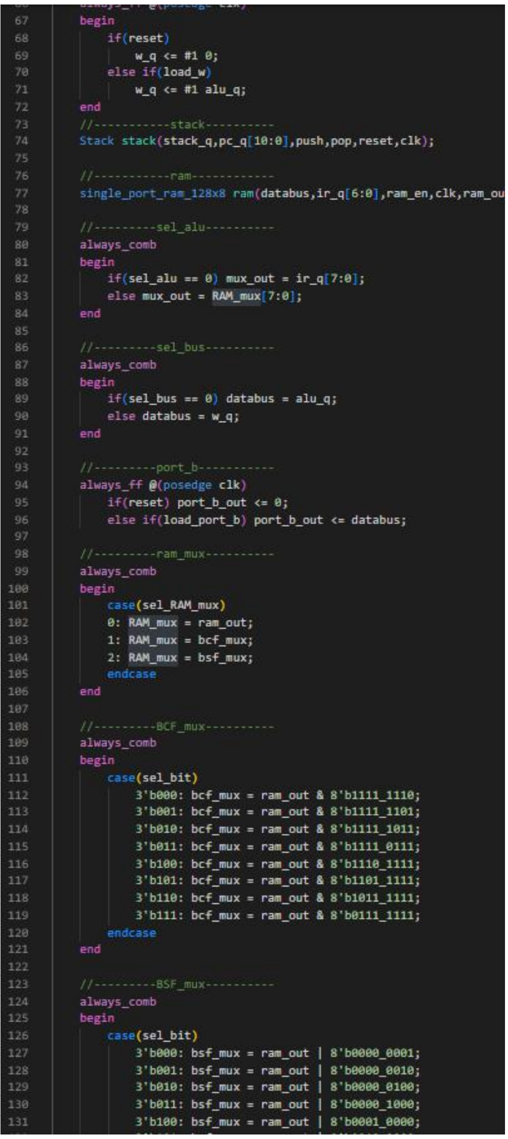
● 系統架構程式碼、測試資料程式碼與程式碼說明

截圖請善用 win+shift+S

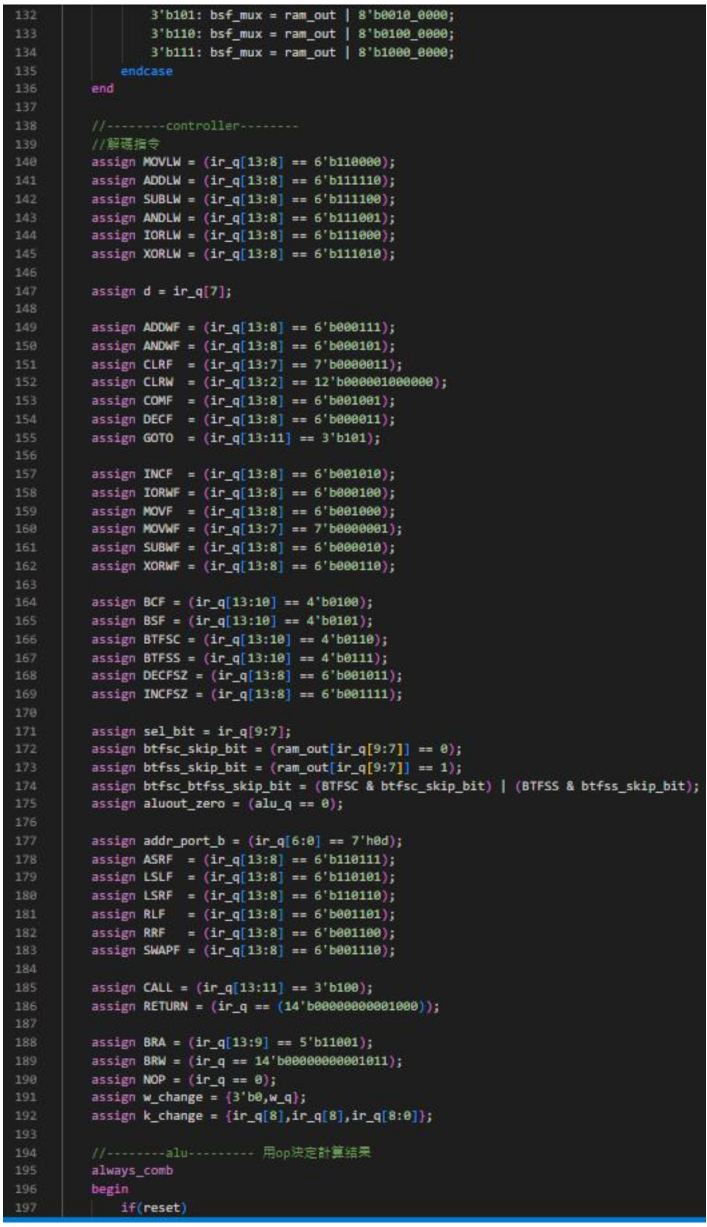
✧ hw_1212.sv



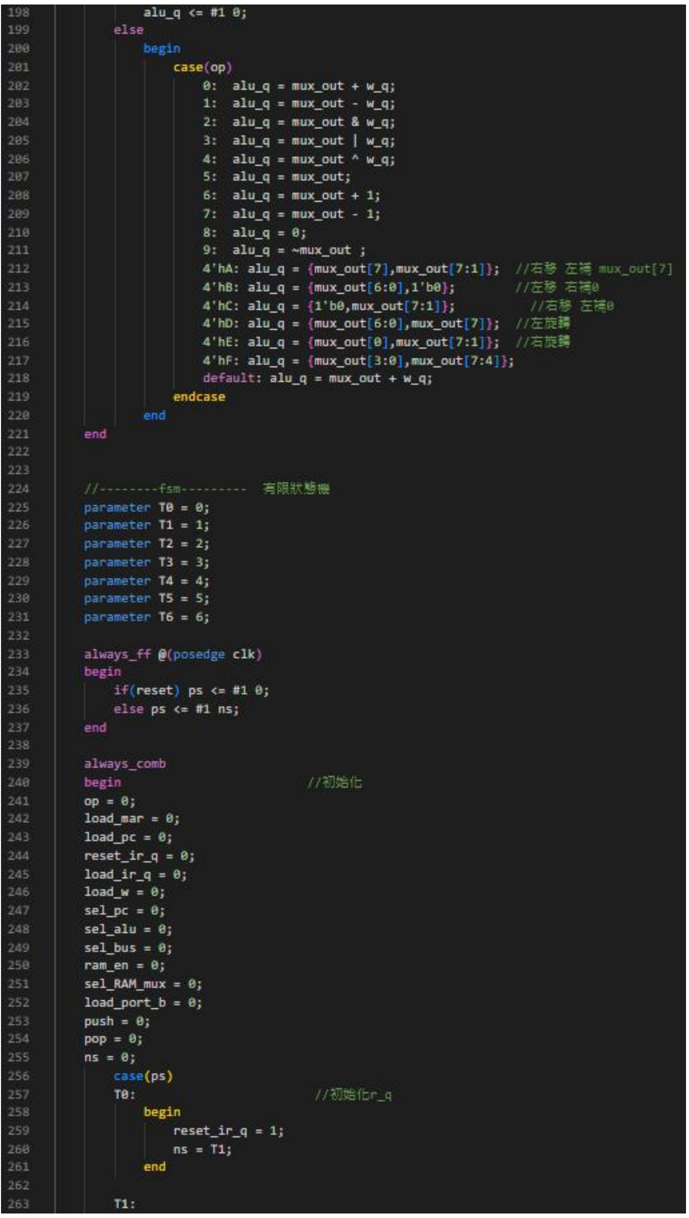
```
`timescale 1ns/10ps
module hw_1212(
    input clk,
    input reset,
    //output logic [7:0] port_b_out
    output logic [7:0] w_q
);
    //logic [7:0] w_q;
    logic [7:0] port_b_out;
    logic [10:0] pc_next, pc_q, mar_q,stack_q;
    logic load_pc, load_mar, load_ir_q, load_w, load_port_b; //load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_mu
    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)
```



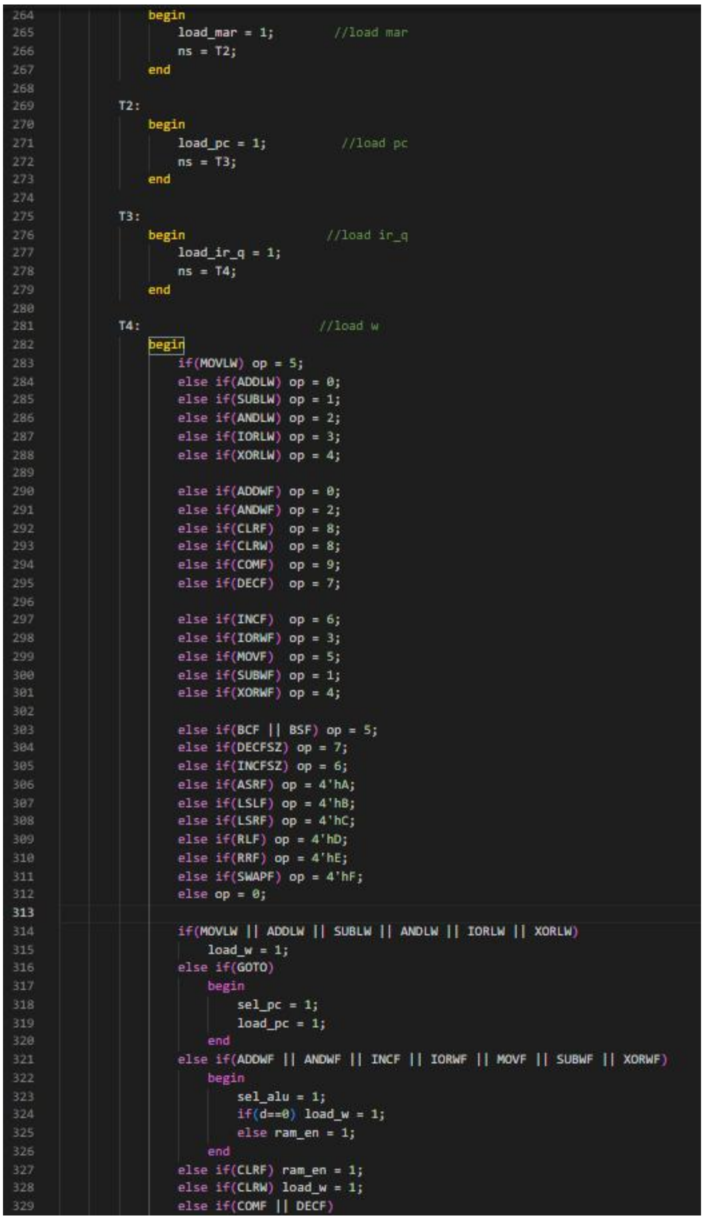
```
    begin
        if(reset)
            w_q <= #1 0;
        else if(load_w)
            w_q <= #1 alu_q;
    end
    //-----------stack----------
    Stack stack(stack_q,pc_q[10:0],push,pop,reset,clk);

    //-----------ram------------
    single_port_ram_128x8 ram(databus,ir_q[6:0],ram_en,clk,ram_ou

    //---------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;
```



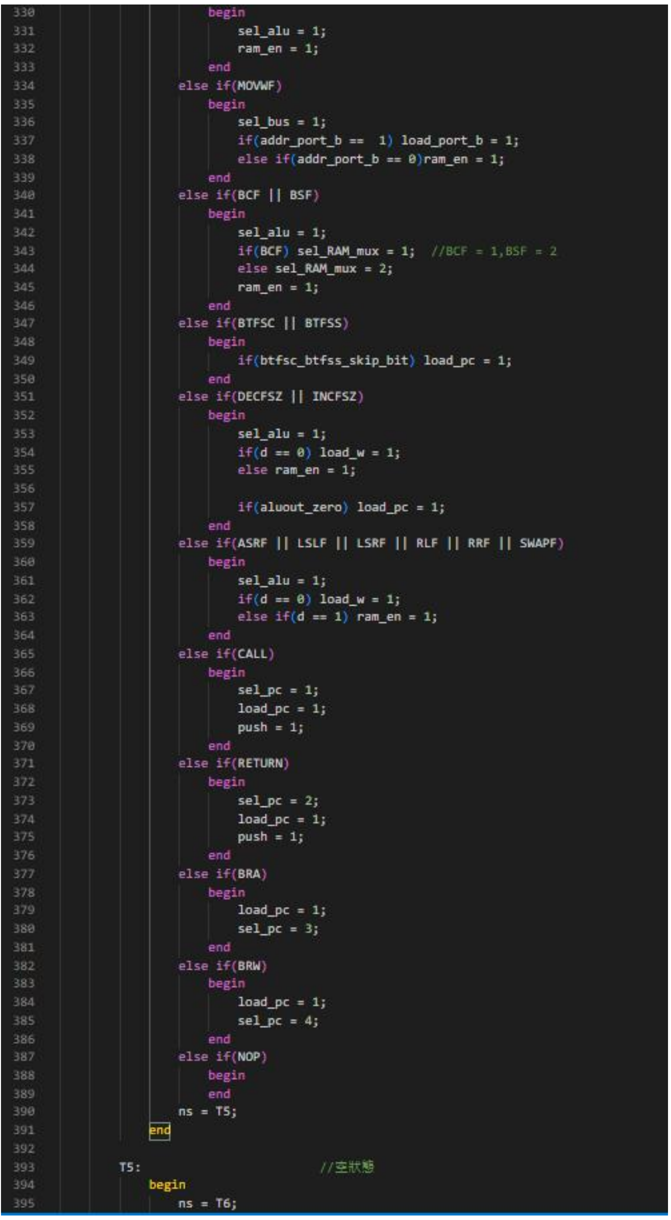
```verilog
            3'b101: bsf_mux = ram_out | 8'b0010_0000;
            3'b110: bsf_mux = ram_out | 8'b0100_0000;
            3'b111: bsf_mux = ram_out | 8'b1000_0000;
        endcase
    end

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

    assign d = ir_q[7];

    assign ADDWF = (ir_q[13:8] == 6'b000111);
    assign ANDWF = (ir_q[13:8] == 6'b000101);
    assign CLRF  = (ir_q[13:7] == 7'b0000011);
    assign CLRW  = (ir_q[13:2] == 12'b000001000000);
    assign COMF  = (ir_q[13:8] == 6'b001001);
    assign DECF  = (ir_q[13:8] == 6'b000011);
    assign GOTO  = (ir_q[13:11] == 3'b101);

    assign INCF  = (ir_q[13:8] == 6'b001010);
    assign IORWF = (ir_q[13:8] == 6'b000100);
    assign MOVF  = (ir_q[13:8] == 6'b001000);
    assign MOVWF = (ir_q[13:7] == 7'b0000001);
    assign SUBWF = (ir_q[13:8] == 6'b000010);
    assign XORWF = (ir_q[13:8] == 6'b000110);

    assign BCF = (ir_q[13:10] == 4'b0100);
    assign BSF = (ir_q[13:10] == 4'b0101);
    assign BTFSC = (ir_q[13:10] == 4'b0110);
    assign BTFSS = (ir_q[13:10] == 4'b0111);
    assign DECFSZ = (ir_q[13:8] == 6'b001011);
    assign INCFSZ = (ir_q[13:8] == 6'b001111);

    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);
    assign LSLF  = (ir_q[13:8] == 6'b110101);
    assign LSRF  = (ir_q[13:8] == 6'b110110);
    assign RLF   = (ir_q[13:8] == 6'b001101);
    assign RRF   = (ir_q[13:8] == 6'b001100);
    assign SWAPF = (ir_q[13:8] == 6'b001110);

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

    assign BRA = (ir_q[13:9] == 5'b11001);
    assign BRW = (ir_q == 14'b00000000001011);
    assign NOP = (ir_q == 0);
    assign w_change = {3'b0,w_q};
    assign k_change = {ir_q[8],ir_q[8],ir_q[8:0]};

    //--------alu--------- 用op決定計算結果
    always_comb
    begin
        if(reset)
```



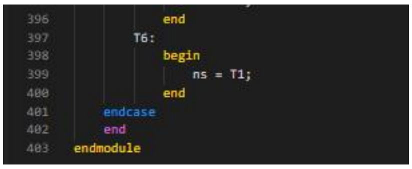
```verilog
            alu_q <= #1 0;
        else
            begin
                case(op)
                    0:  alu_q = mux_out + w_q;
                    1:  alu_q = mux_out - w_q;
                    2:  alu_q = mux_out & w_q;
                    3:  alu_q = mux_out | w_q;
                    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r_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
                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(DECFSZ) op = 7;
                else if(INCFSZ) op = 6;
                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(GOTO)
                    begin
                        sel_pc = 1;
                        load_pc = 1;
                    end
                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)
```

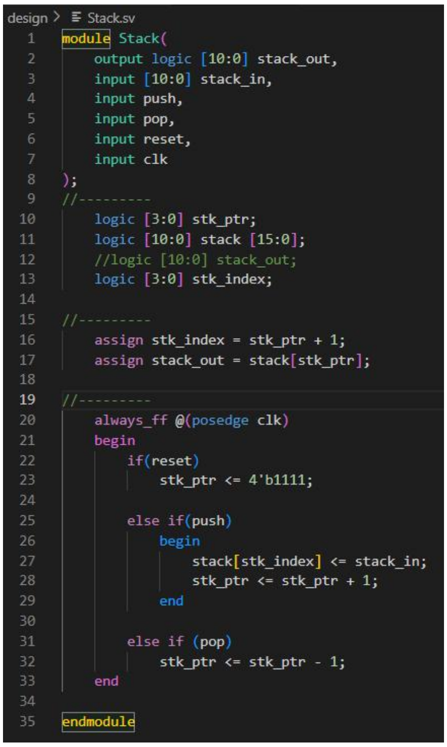


```
                    begin
                        sel_alu = 1;
                        ram_en = 1;
                    end
                else if(MOVWF)
                    begin
                        sel_bus = 1;
                        if(addr_port_b ==  1) load_port_b = 1;
                        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(BTFSC || BTFSS)
                    begin
                        if(btfsc_btfss_skip_bit) load_pc = 1;
                    end
                else if(DECFSZ || INCFSZ)
                    begin
                        sel_alu = 1;
                        if(d == 0) load_w = 1;
                        else ram_en = 1;

                        if(aluout_zero) load_pc = 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
                        sel_pc = 1;
                        load_pc = 1;
                        push = 1;
                    end
                else if(RETURN)
                    begin
                        sel_pc = 2;
                        load_pc = 1;
                        push = 1;
                    end
                else if(BRA)
                    begin
                        load_pc = 1;
                        sel_pc = 3;
                    end
                else if(BRW)
                    begin
                        load_pc = 1;
                        sel_pc = 4;
                    end
                else if(NOP)
                    begin
                    end
                ns = T5;
            end

        T5:                         //空狀態
            begin
                ns = T6;
```



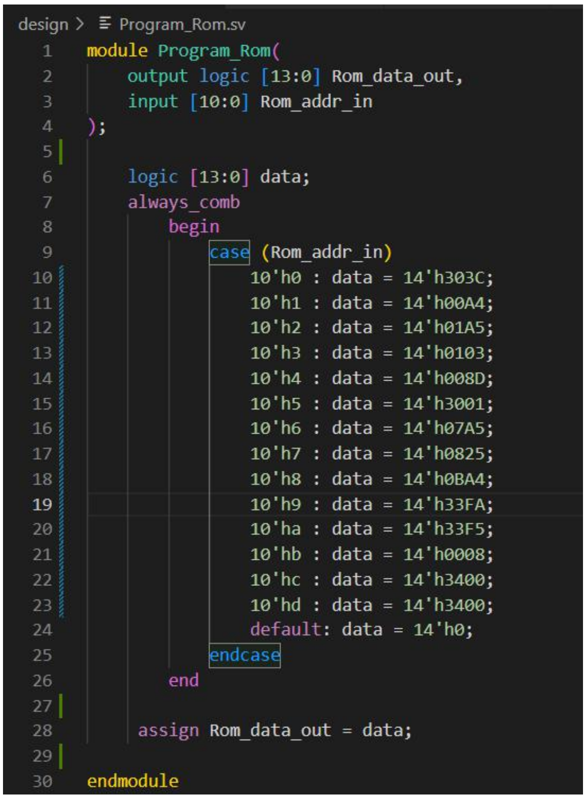
```
            end
        T6:
            begin
                ns = T1;
            end
    endcase
    end
endmodule
```

✧ Stack.sv



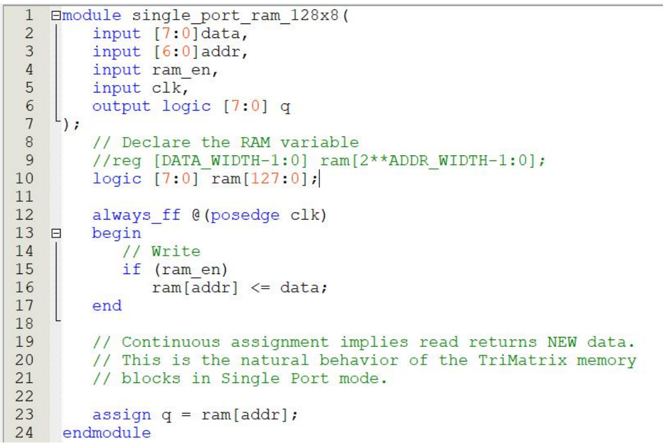
```
module Stack(
    output logic [10:0] stack_out,
    input [10:0] stack_in,
    input push,
    input pop,
    input reset,
    input clk
);
//---------
    logic [3:0] stk_ptr;
    logic [10:0] stack [15:0];
    //logic [10:0] stack_out;
    logic [3:0] stk_index;

//---------
    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



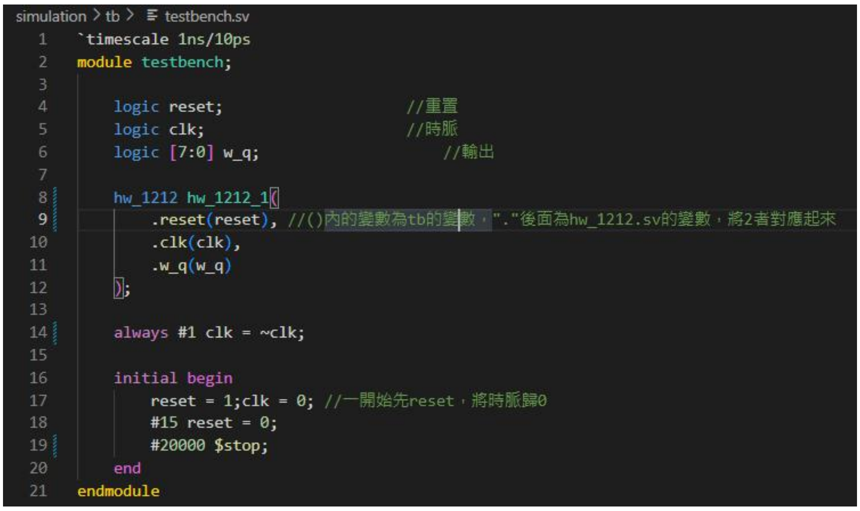
```
module Program_Rom(
    output logic [13:0] Rom_data_out,
    input [10:0] Rom_addr_in
);

    logic [13:0] data;
    always_comb
        begin
            case (Rom_addr_in)
                10'h0 : data = 14'h303C;
                10'h1 : data = 14'h00A4;
                10'h2 : data = 14'h01A5;
                10'h3 : data = 14'h0103;
                10'h4 : data = 14'h008D;
                10'h5 : data = 14'h3001;
                10'h6 : data = 14'h07A5;
                10'h7 : data = 14'h0825;
                10'h8 : data = 14'h0BA4;
                10'h9 : data = 14'h33FA;
                10'ha : data = 14'h33F5;
                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
```

✧ single_port_ram_128x8.sv



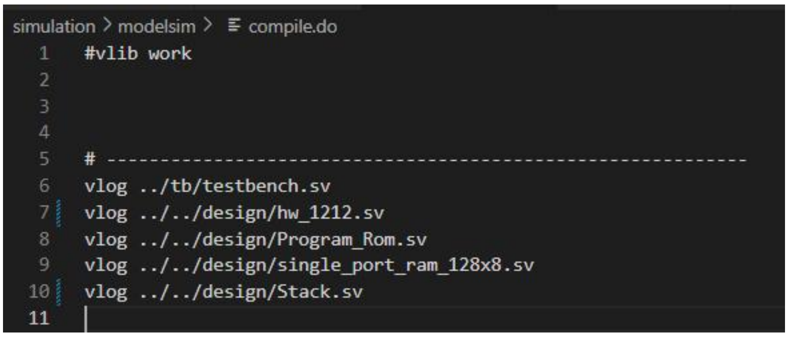
```
module single_port_ram_128x8(
   input [7:0]data,
   input [6:0]addr,
   input ram_en,
   input clk,
   output logic [7:0] q
);
   // Declare the RAM variable
   //reg [DATA_WIDTH-1:0] ram[2**ADDR_WIDTH-1:0];
   logic [7:0] ram[127:0];

   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



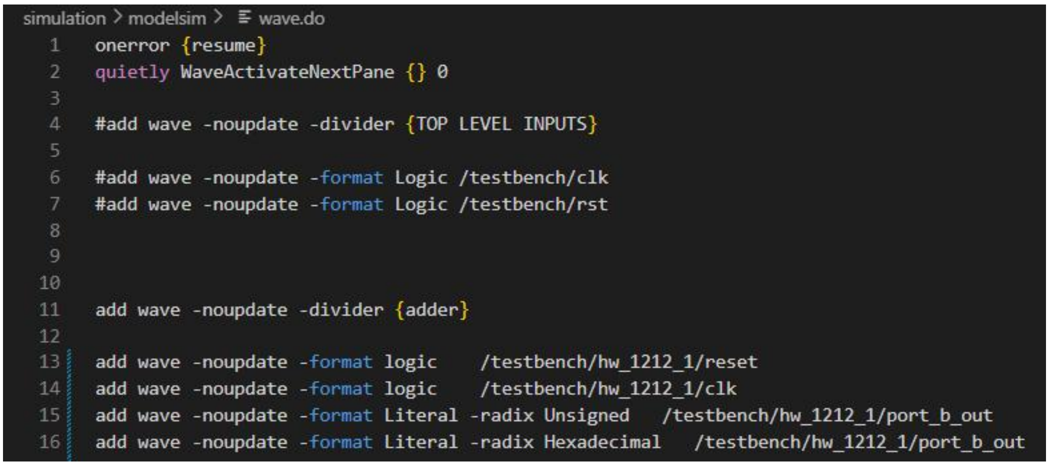
```
`timescale 1ns/10ps
module testbench;

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

    hw_1212 hw_1212_1(
        .reset(reset), //()內的變數為tb的變數，"."後面為hw_1212.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;
        #20000 $stop;
    end
endmodule
```

✧ compile.do



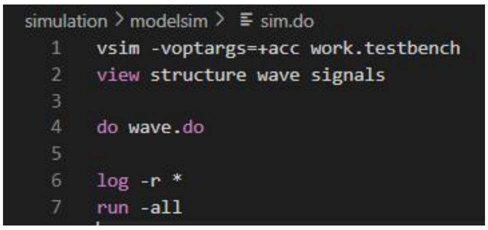
```
#vlib work



# ------------------------------------------------------------
vlog ../tb/testbench.sv
vlog ../../design/hw_1212.sv
vlog ../../design/Program_Rom.sv
vlog ../../design/single_port_ram_128x8.sv
vlog ../../design/Stack.sv

```

✧ wave.do



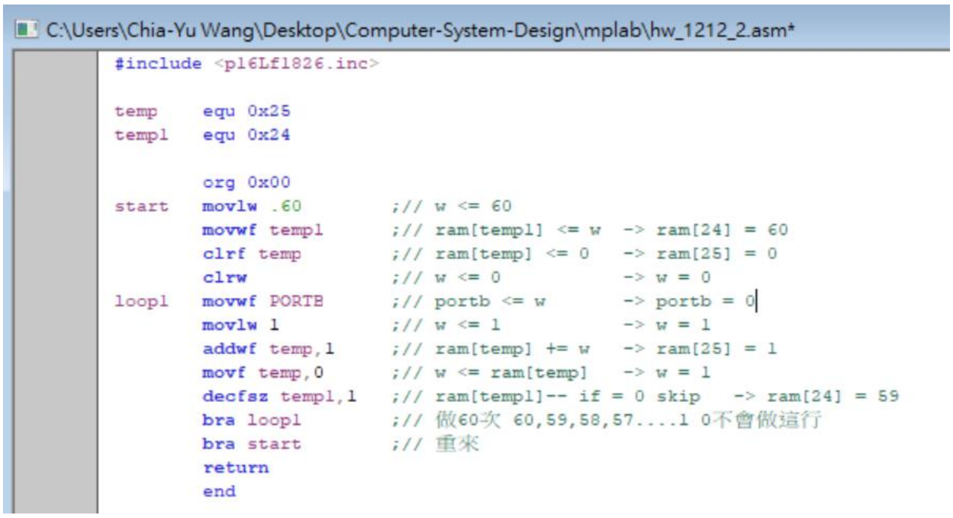
```
onerror {resume}
quietly WaveActivateNextPane {} 0

#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_1212_1/reset
add wave -noupdate -format logic    /testbench/hw_1212_1/clk
add wave -noupdate -format Literal -radix Unsigned   /testbench/hw_1212_1/port_b_out
add wave -noupdate -format Literal -radix Hexadecimal   /testbench/hw_1212_1/port_b_out
```

✧ sim.do



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

do wave.do

log -r *
run -all
```

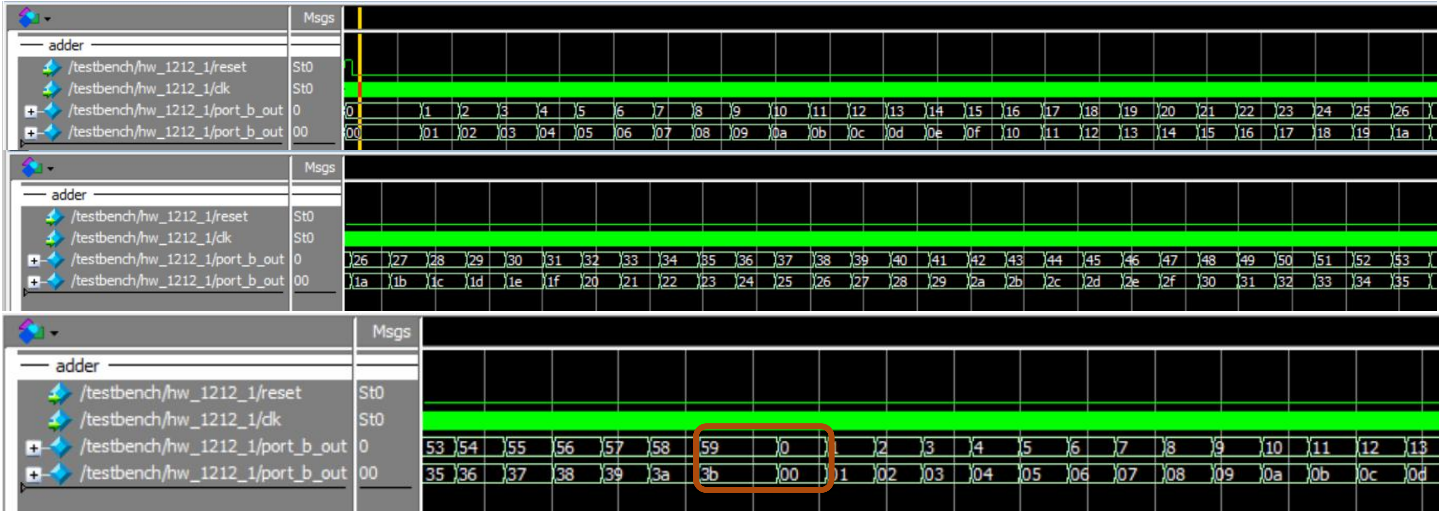
✧ 組合語言



C:\Users\Chia-Yu Wang\Desktop\Computer-System-Design\mplab\hw_1212_2.asm*

```
#include <p16Lf1826.inc>

temp    equ 0x25
templ   equ 0x24

        org 0x00
start   movlw .60           ;// w <= 60
        movwf templ         ;// ram[templ] <= w   -> ram[24] = 60
        clrf temp           ;// ram[temp] <= 0    -> ram[25] = 0
        clrw                ;// w <= 0            -> w = 0
loopl   movwf PORTB         ;// portb <= w        -> portb = 0
        movlw 1             ;// w <= 1            -> w = 1
        addwf temp,1        ;// ram[temp] += w    -> ram[25] = 1
        movf temp,0         ;// w <= ram[temp]    -> w = 1
        decfsz templ,1      ;// ram[templ]-- if = 0 skip   -> ram[24] = 59
        bra loopl           ;// 做60次 60,59,58,57....1 0不會做這行
        bra start           ;// 重來
        return
        end
```

● 模擬結果與結果說明：



Msgs
adder
/testbench/hw_1212_1/reset St0
/testbench/hw_1212_1/clk St0
/testbench/hw_1212_1/port_b_out 0
/testbench/hw_1212_1/port_b_out 00
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15 16 17 18 19 1a
26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53
1a 1b 1c 1d 1e 1f 20 21 22 23 24 25 26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35
53 54 55 56 57 58 59 0 1 2 3 4 5 6 7 8 9 10 11 12 13
35 36 37 38 39 3a 3b 00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d

我把 10 進位和 16 進位都印出來了~我上課有 demo 給助教看 應該是對的~

我的迴圈會把 w 給 portb 60 次，從 0 到 59，59 之後他會跑出迴圈然後作初始化，所以 59 比較長

二、

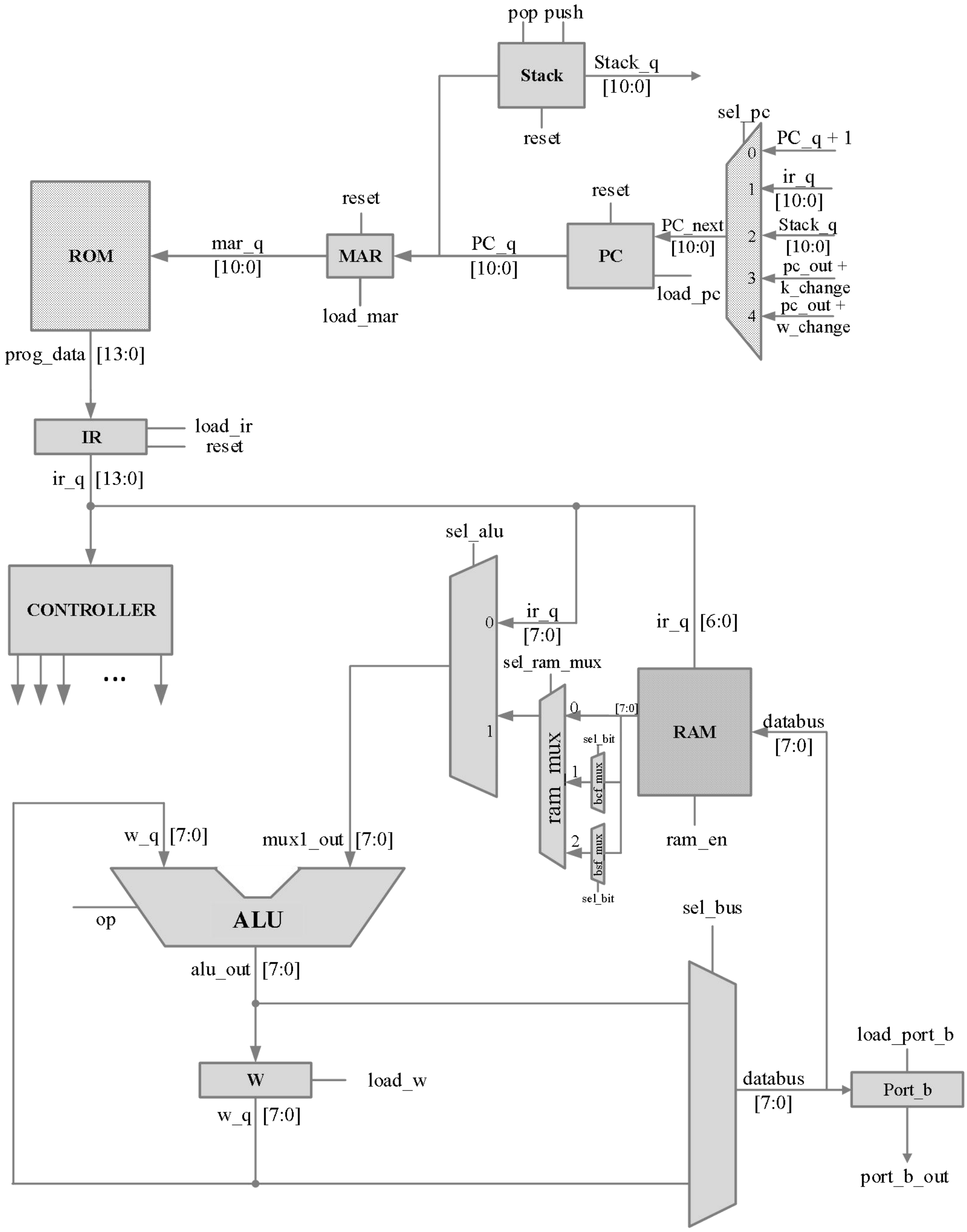
● 實驗說明：

用 MPLAB 設計一個 Rom，使 0x21 和 0x22 兩個位址的 16 進制(用 10 進位顯示不算分)分別表示時鐘的分及秒，即 0x22(秒)的 16 進制會由 1 數到 59 後歸零，每當 0x22(秒)歸零 0x21(分)就會加 1

模擬用的組合語言不用加 delay 方便波形觀測

請交 MPLAB 中組合語言截圖、程式碼截圖與波形圖，存分跟秒的暫存器請分別設定為 0x21 跟 0x22

● 系統硬體架構方塊圖（接線圖）：



架構圖

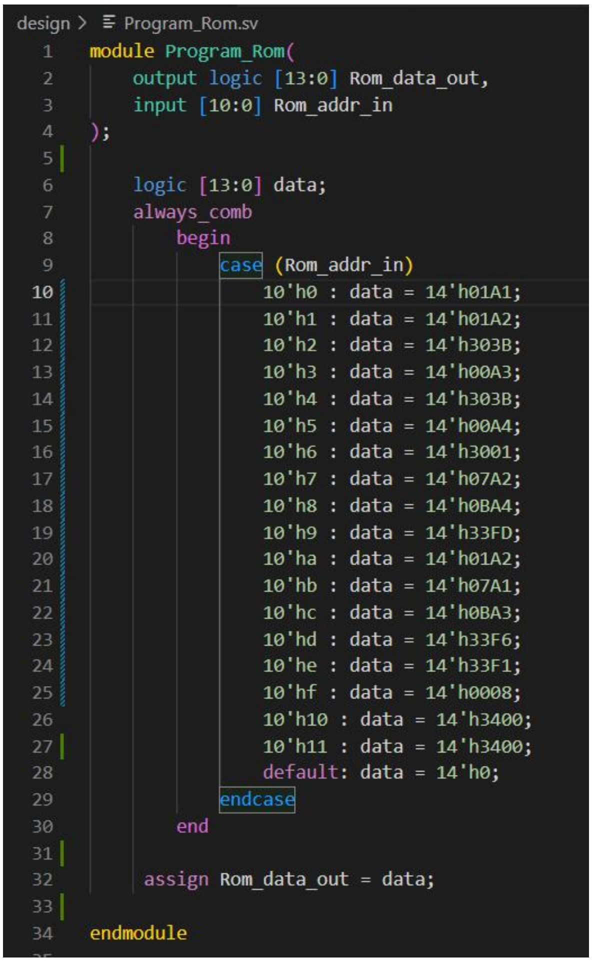
● 系統架構程式碼、測試資料程式碼與程式碼說明

截圖請善用 win+shift+S

只有 Program_Rom.sv、wave.do 和組合語言不一樣

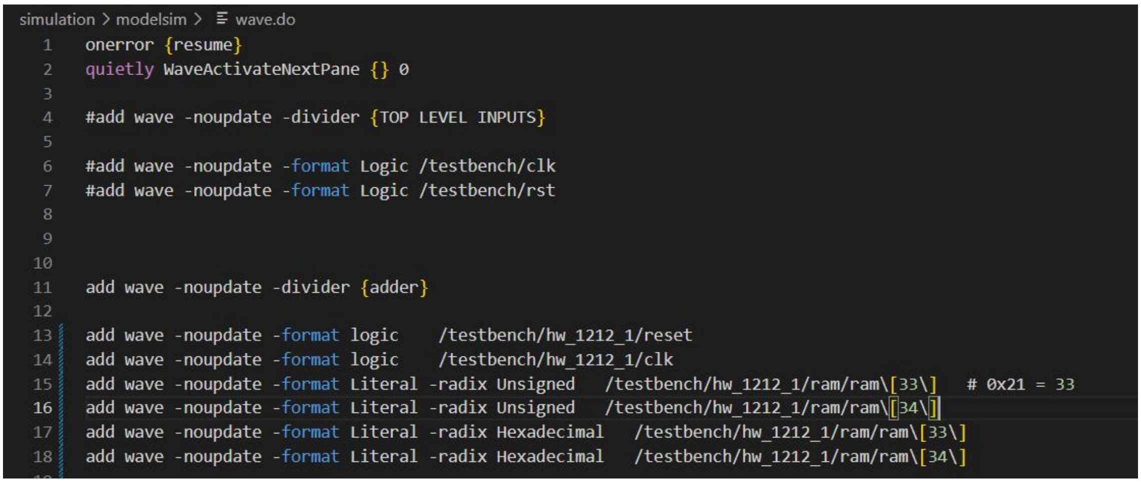
所以我就只給這 3 個~

✧ Program_Rom.sv



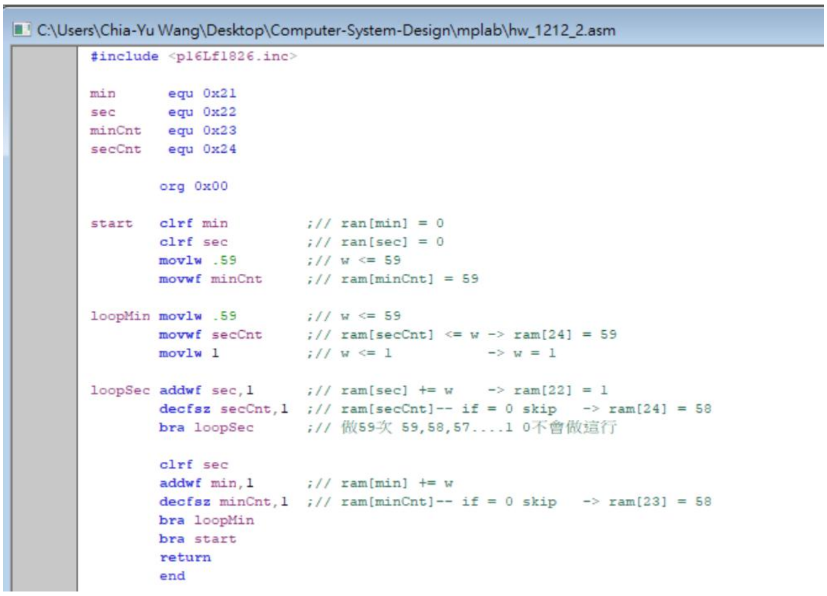
```
module Program_Rom(
    output logic [13:0] Rom_data_out,
    input [10:0] Rom_addr_in
);

    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'h303B;
                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'h07A1;
                10'hc : data = 14'h0BA3;
                10'hd : data = 14'h33F6;
                10'he : data = 14'h33F1;
                10'hf : data = 14'h0008;
                10'h10 : data = 14'h3400;
                10'h11 : data = 14'h3400;
                default: data = 14'h0;
            endcase
        end

     assign Rom_data_out = data;

endmodule
```

✧ wave.do



```
onerror {resume}
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_1212_1/reset
add wave -noupdate -format logic    /testbench/hw_1212_1/clk
add wave -noupdate -format Literal -radix Unsigned   /testbench/hw_1212_1/ram/ram\[33\]   # 0x21 = 33
add wave -noupdate -format Literal -radix Unsigned   /testbench/hw_1212_1/ram/ram\[34\]
add wave -noupdate -format Literal -radix Hexadecimal   /testbench/hw_1212_1/ram/ram\[33\]
add wave -noupdate -format Literal -radix Hexadecimal   /testbench/hw_1212_1/ram/ram\[34\]
```

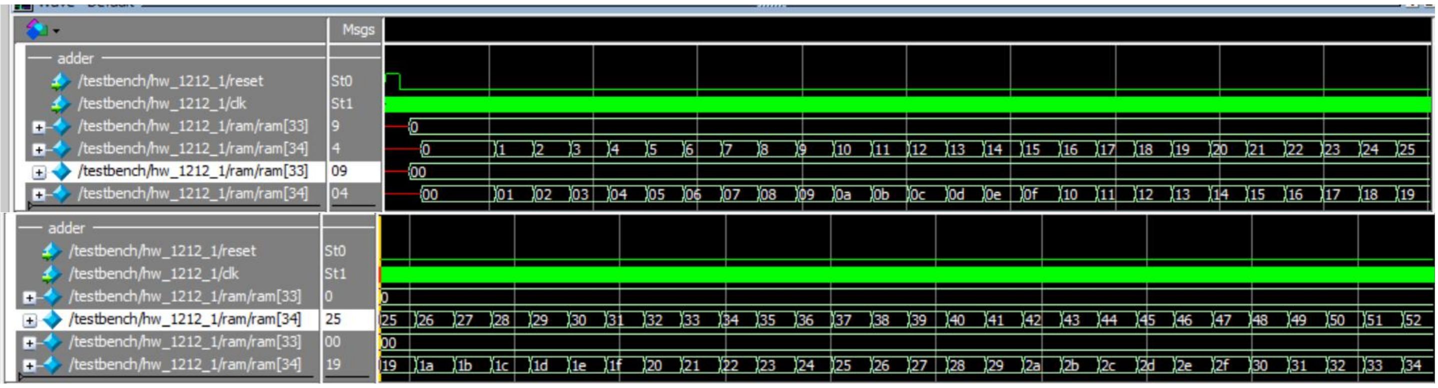
✧ 組合語言



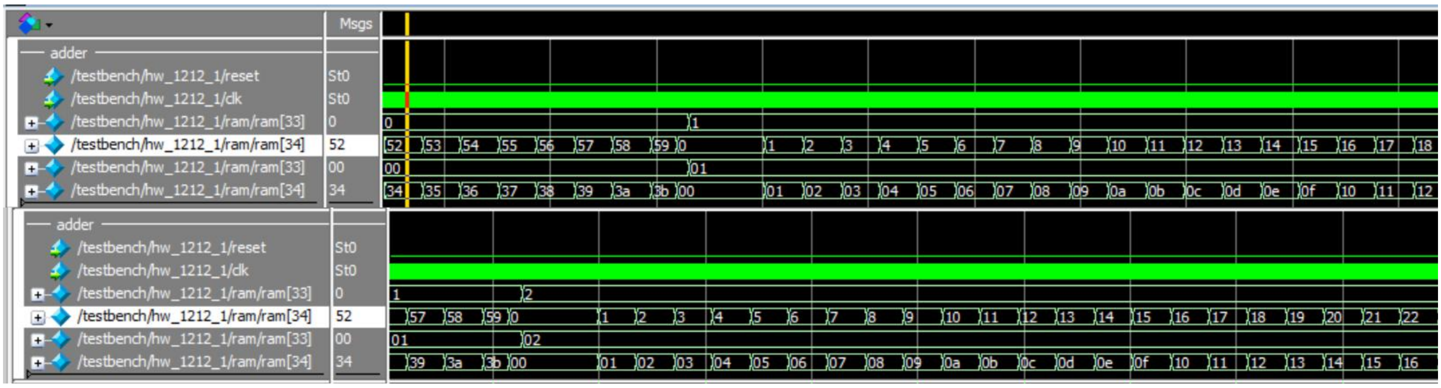
C:\Users\Chia-Yu Wang\Desktop\Computer-System-Design\mplab\hw_1212_2.asm

```
#include <p16Lf1826.inc>

min       equ 0x21
sec       equ 0x22
minCnt    equ 0x23
secCnt    equ 0x24

        org 0x00

start   clrf min          ;// ran[min] = 0
        clrf sec          ;// ran[sec] = 0
        movlw .59         ;// w <= 59
        movwf minCnt      ;// ram[minCnt] = 59

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

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

        clrf sec
        addwf min,1       ;// ram[min] += w
        decfsz minCnt,1   ;// ram[minCnt]-- if = 0 skip   -> ram[23] = 58
        bra loopMin
        bra start
        return
        end
```

● 模擬結果與結果說明：


Msgs
adder
/testbench/hw_1212_1/reset St0
/testbench/hw_1212_1/clk St1
/testbench/hw_1212_1/ram/ram[33] 9
/testbench/hw_1212_1/ram/ram[34] 4
/testbench/hw_1212_1/ram/ram[33] 09
/testbench/hw_1212_1/ram/ram[34] 04
adder
/testbench/hw_1212_1/reset St0
/testbench/hw_1212_1/clk St1
/testbench/hw_1212_1/ram/ram[33] 0
/testbench/hw_1212_1/ram/ram[34] 25
/testbench/hw_1212_1/ram/ram[33] 00
/testbench/hw_1212_1/ram/ram[34] 19

秒的計數


Msgs
adder
/testbench/hw_1212_1/reset St0
/testbench/hw_1212_1/clk St0
/testbench/hw_1212_1/ram/ram[33] 0
/testbench/hw_1212_1/ram/ram[34] 52
/testbench/hw_1212_1/ram/ram[33] 00
/testbench/hw_1212_1/ram/ram[34] 34
adder
/testbench/hw_1212_1/reset St0
/testbench/hw_1212_1/clk St0
/testbench/hw_1212_1/ram/ram[33] 0
/testbench/hw_1212_1/ram/ram[34] 52
/testbench/hw_1212_1/ram/ram[33] 00
/testbench/hw_1212_1/ram/ram[34] 34

分的計數

0 比較長是因為 sec 59 結束後我讓他立刻歸 0，然後 min+1，之後去做初始化，所以 0 會拉得比較長。

上面 2 條是十進位的，下面是 16 進位~

結論與心得：

- 今天的東西不難，但是一看到作業要寫組合語言就好害怕 qq 不過不過好好的去研究一下會發現蠻簡單的 ww，可能是因為之前計算機組織課有好好上過，加上寫好多個禮拜的 cpu 指令，所以比較有那個概念，至少比大二上第一次學要好多了，也可能是因為進步了所以才比較好上手，如果段考要考的話，希望可以提供每個指令在做甚麼，指令太多了，每次寫前都要一直翻 ptt，如果考試沒有 ppt 會很慘 qq。
- 今天學到最多的真的是組合語言怎麼寫，怎麼有技巧的寫，我一直想把他寫短，但是如果寫很短波形圖會很醜，因為雖然結果一樣，但是中間 clk 的數量有差，所以我最後還是為了好讀懂，增加了很多可以刪掉的指令 qq 像是第二題，我是先 sec 歸 0 在 min+1，我原本其實是先+1 然後等到初始化在歸 0，但這樣波型圖就會出現 min = 1 ,sec = 59，這種在切換 clk 的問題...雖然我覺得不影響結果，但是怕被扣分，我還是改了 qq