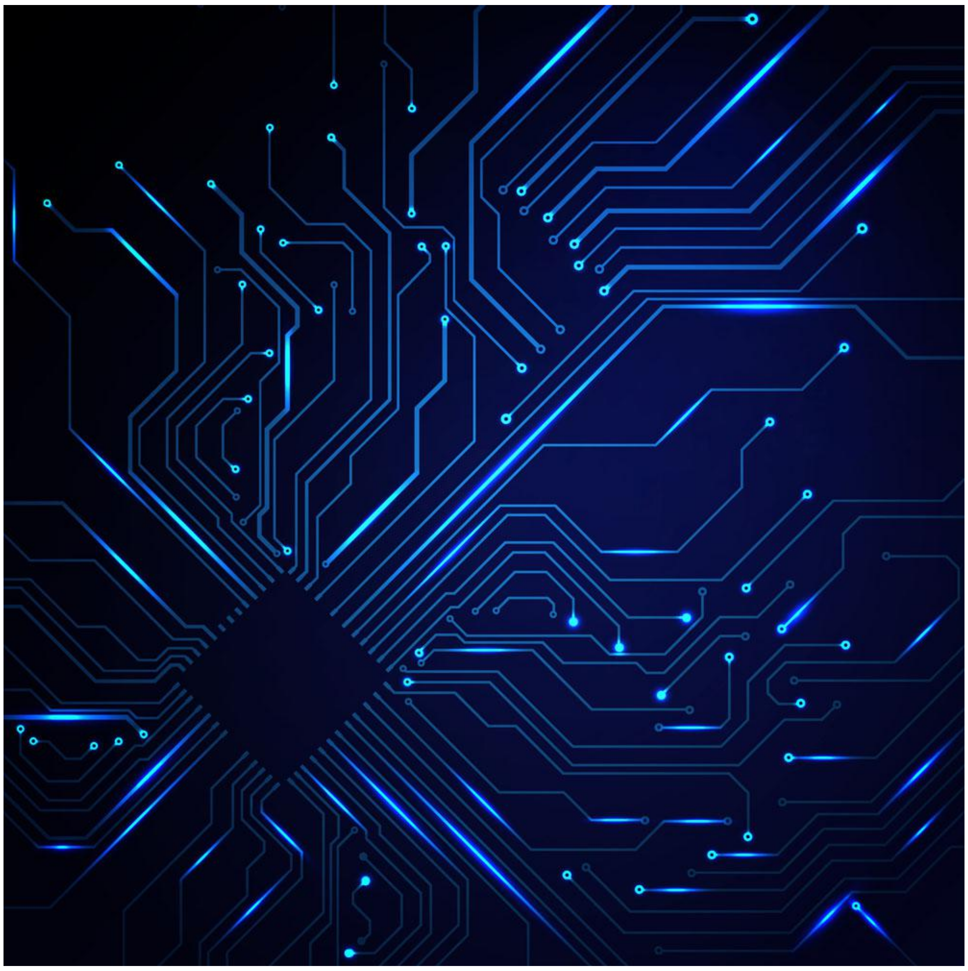


2022/01/02

實驗十四

進階組合語言

姓名：王嘉羽　　　學號：00957116

班級：資工 3B

E-mail：vayne20011125@gmail.com

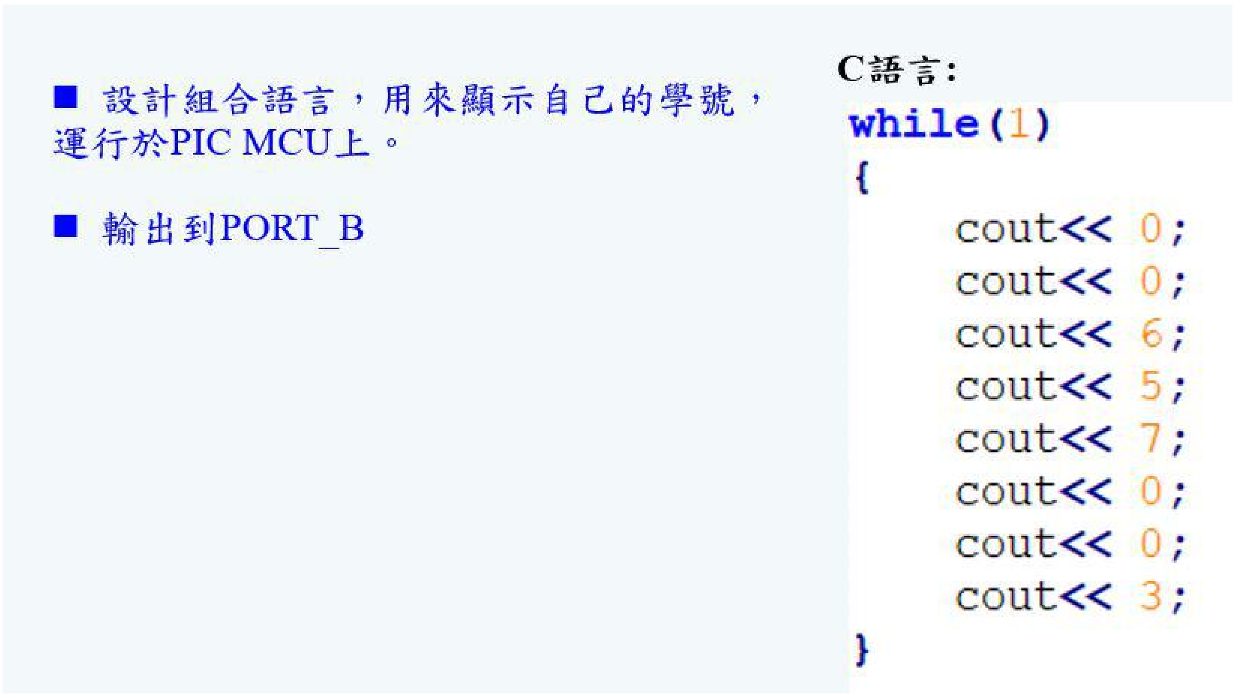
2022/01/02

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：

第一題:



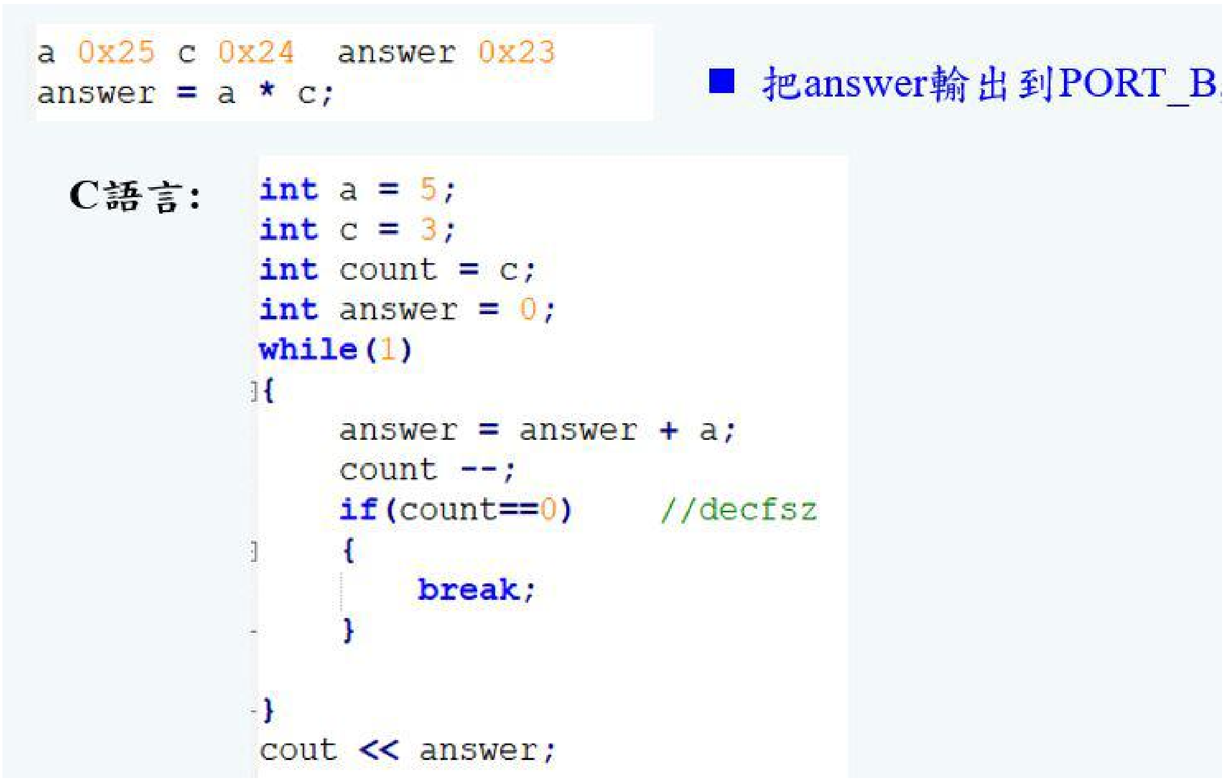
■ 設計組合語言，用來顯示自己的學號，運行於PIC MCU上。

■ 輸出到PORT_B

C語言:

```
while(1)
{
        cout<< 0;
        cout<< 0;
        cout<< 6;
        cout<< 5;
        cout<< 7;
        cout<< 0;
        cout<< 0;
        cout<< 3;
}
```

第二題:



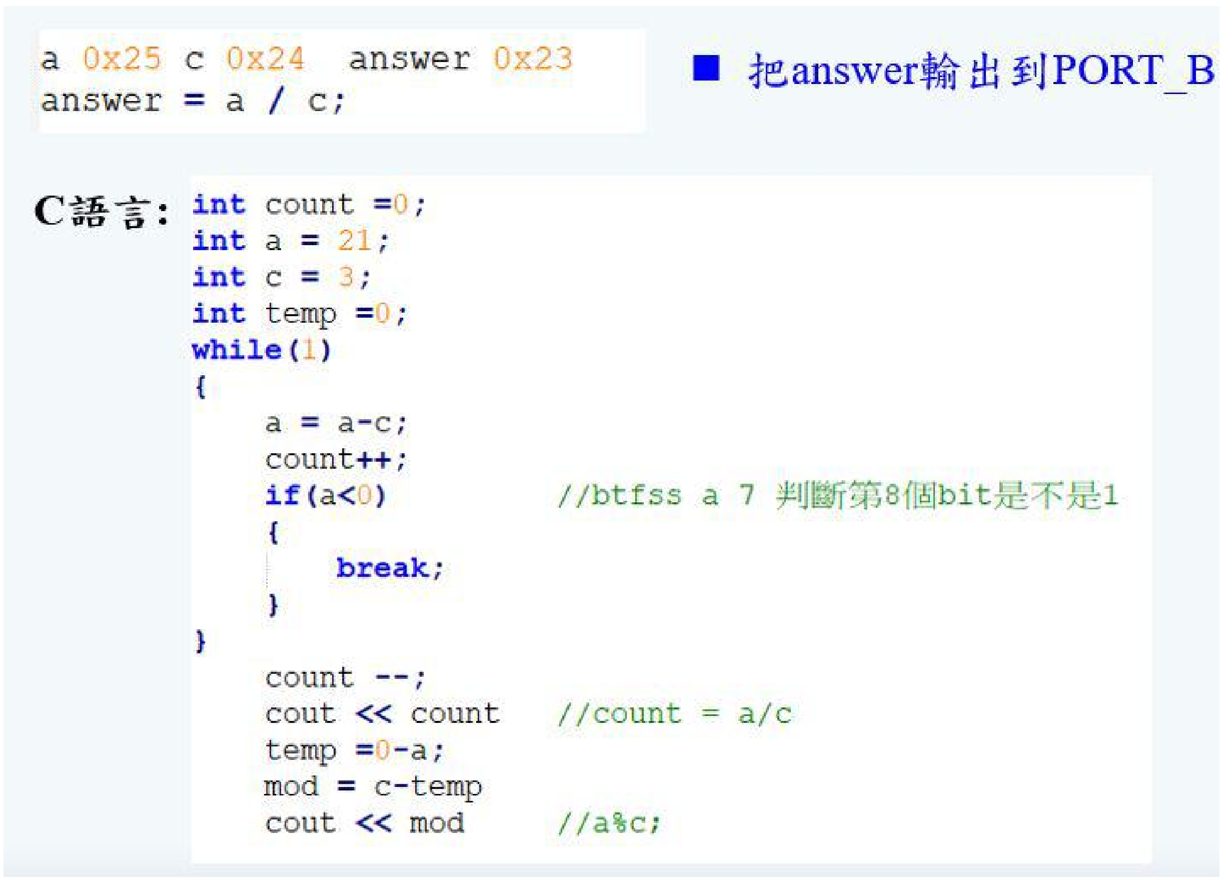
```
a 0x25 c 0x24  answer 0x23
answer = a * c;
```

■ 把answer輸出到PORT_B

C語言:

```
int a = 5;
int c = 3;
int count = c;
int answer = 0;
while(1)
{
    answer = answer + a;
    count --;
    if(count==0)     //decfsz
    {
        break;
    }

}
cout << answer;
```

第三題:



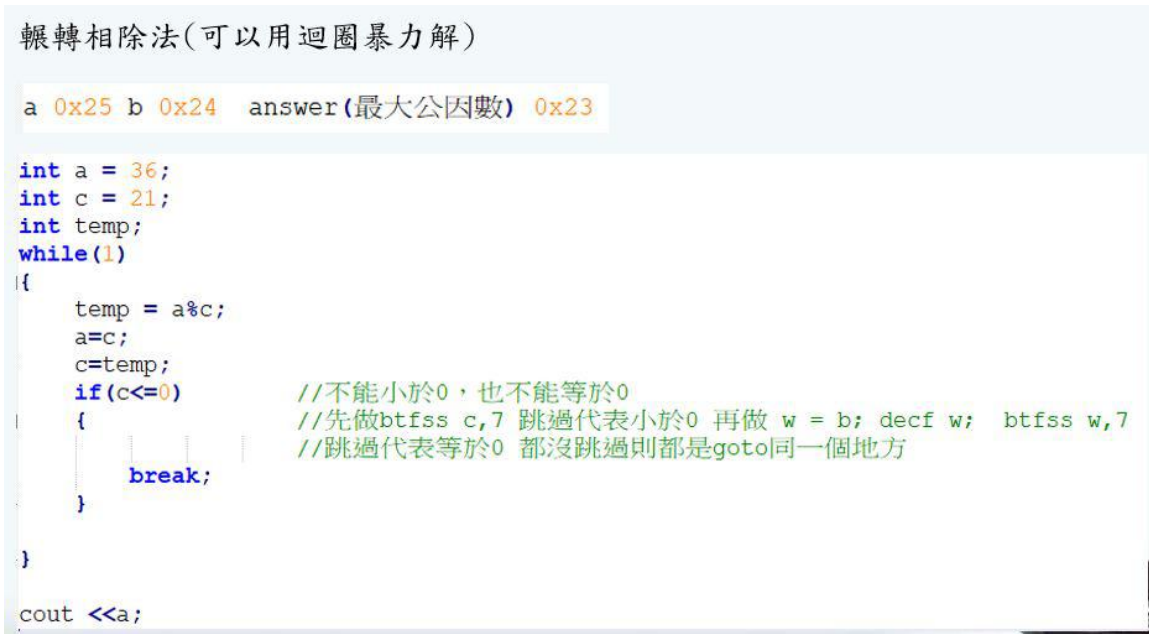
```
a 0x25 c 0x24  answer 0x23
answer = a / c;
```

■ 把answer輸出到PORT_B

C語言:

```
int count =0;
int a = 21;
int c = 3;
int temp =0;
while(1)
{
    a = a-c;
    count++;
    if(a<0)         //btfss a 7 判斷第8個bit是不是1
    {
        break;
    }
}
    count --;
    cout << count   //count = a/c
    temp =0-a;
    mod = c-temp
    cout << mod     //a%c;
```

第四題:

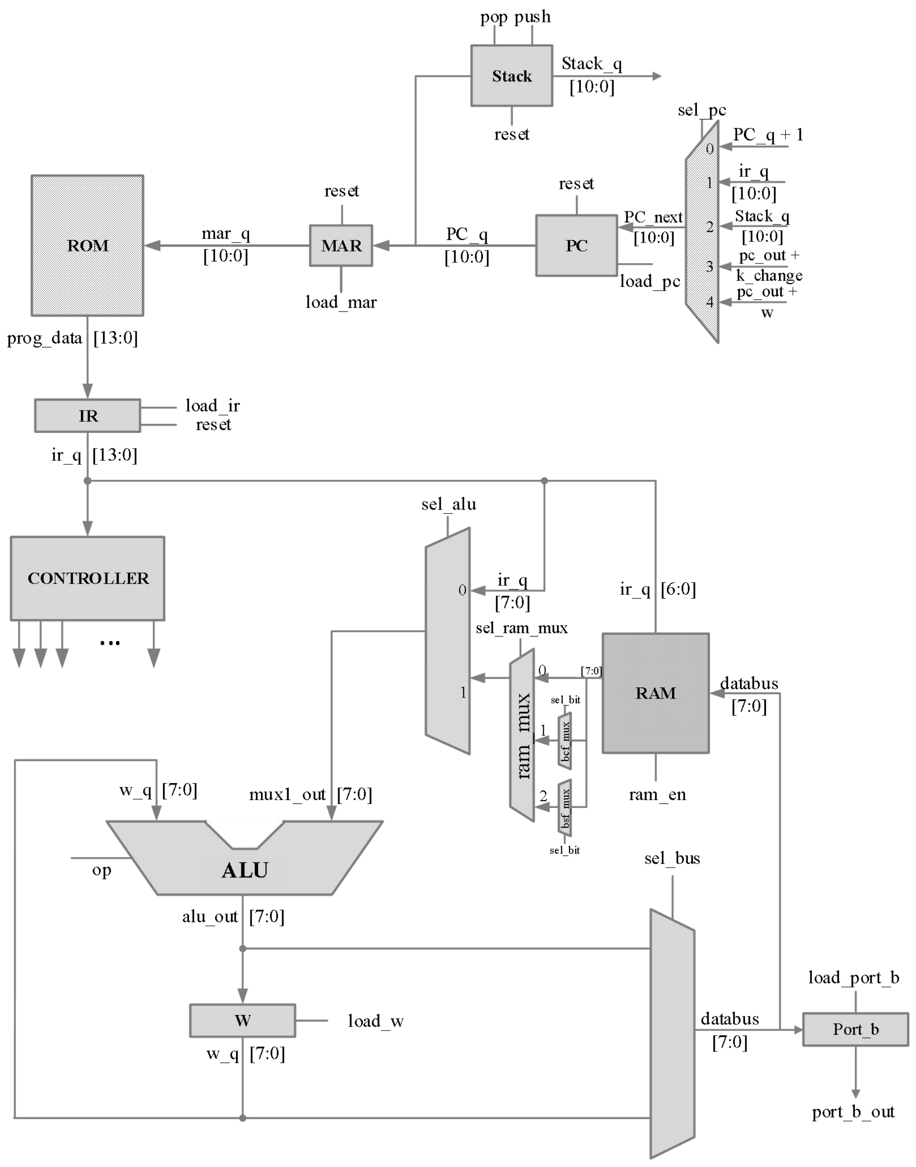


輾轉相除法(可以用迴圈暴力解)

```
a 0x25 b 0x24  answer(最大公因數) 0x23
```

```
int a = 36;
int c = 21;
int temp;
while(1)
{
    temp = a%c;
    a=c;
    c=temp;
    if(c<=0)         //不能小於0，也不能等於0
    {                //先做btfss c,7 跳過代表小於0 再做 w = b; decf w;  btfss w,7
                     //跳過代表等於0 都沒跳過則都是goto同一個地方
        break;
    }

}

cout <<a;
```

● 系統硬體架構方塊圖（接線圖）：

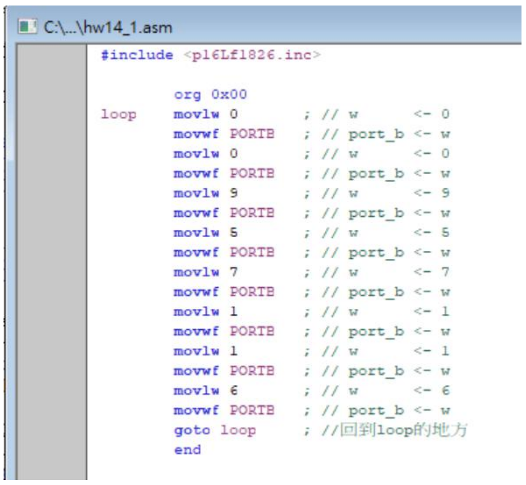


● 系統架構程式碼、測試資料程式碼與程式碼說明

截圖請善用 win+shift+S

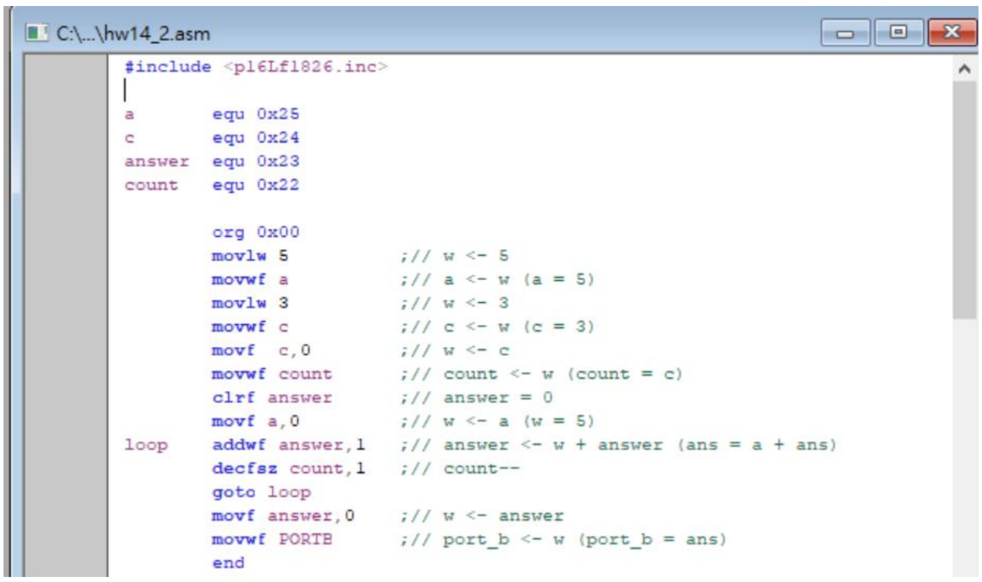
✧ 組語:

■ 第一題:


C:\...\hw14_1.asm

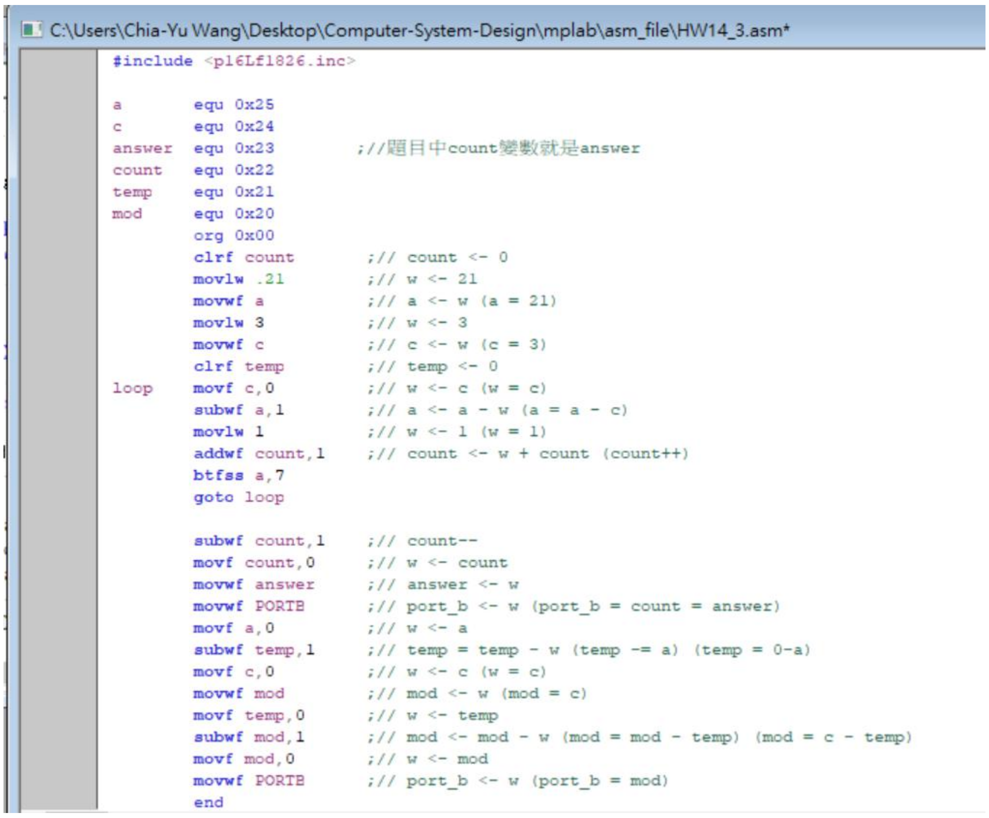
```
#include <p16Lf1826.inc>

        org 0x00
loop    movlw 0         ; // w      <- 0
        movwf PORTB     ; // port_b <- w
        movlw 0         ; // w      <- 0
        movwf PORTB     ; // port_b <- w
        movlw 9         ; // w      <- 9
        movwf PORTB     ; // port_b <- w
        movlw 5         ; // w      <- 5
        movwf PORTB     ; // port_b <- w
        movlw 7         ; // w      <- 7
        movwf PORTB     ; // port_b <- w
        movlw 1         ; // w      <- 1
        movwf PORTB     ; // port_b <- w
        movlw 1         ; // w      <- 1
        movwf PORTB     ; // port_b <- w
        movlw 6         ; // w      <- 6
        movwf PORTB     ; // port_b <- w
        goto loop       ; //回到loop的地方
        end
```

■ 第二題:


C:\...\hw14_2.asm

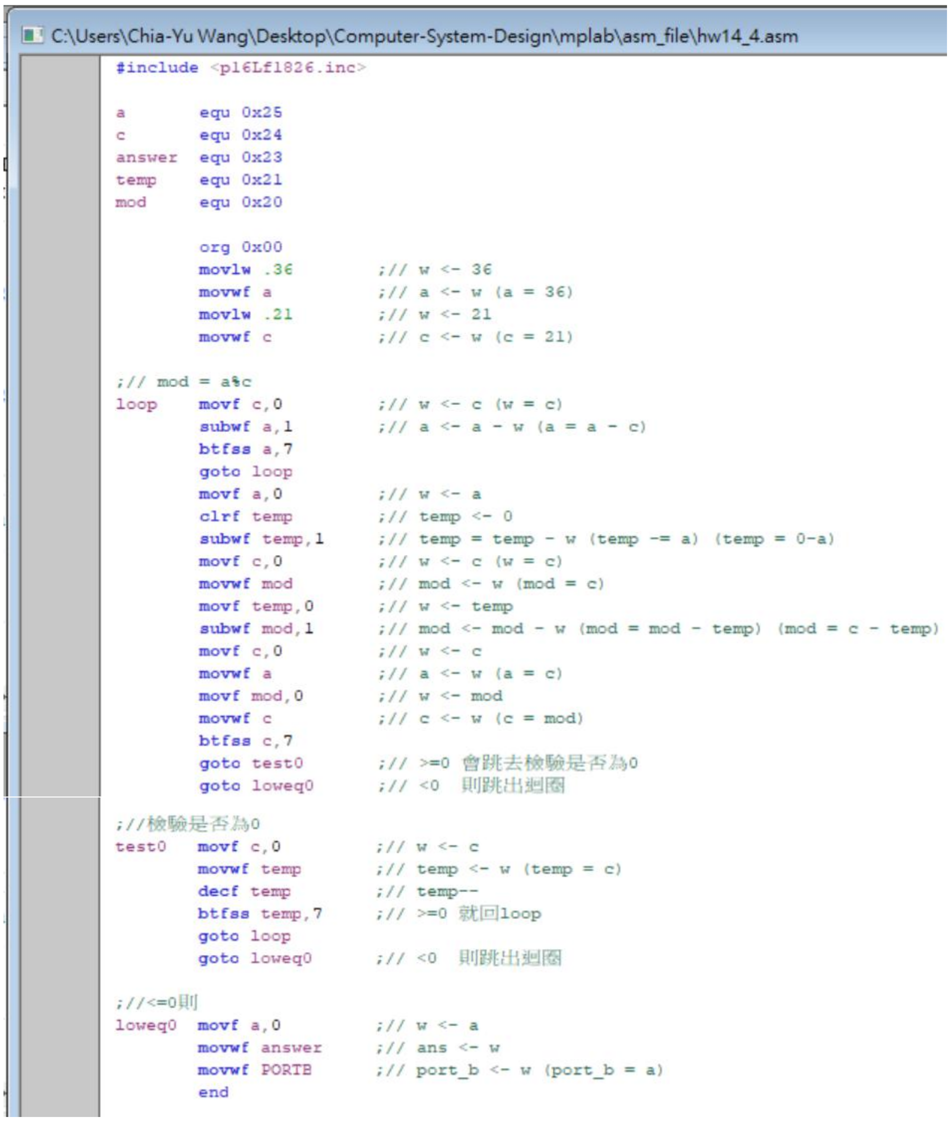
```
#include <p16Lf1826.inc>

a       equ 0x25
c       equ 0x24
answer  equ 0x23
count   equ 0x22

        org 0x00
        movlw 5             ;// w <- 5
        movwf a             ;// a <- w (a = 5)
        movlw 3             ;// w <- 3
        movwf c             ;// c <- w (c = 3)
        movf  c,0           ;// w <- c
        movwf count         ;// count <- w (count = c)
        clrf answer         ;// answer = 0
        movf a,0            ;// w <- a (w = 5)
loop    addwf answer,1      ;// answer <- w + answer (ans = a + ans)
        decfsz count,1      ;// count--
        goto loop
        movf answer,0       ;// w <- answer
        movwf PORTB         ;// port_b <- w (port_b = ans)
        end
```

■ 第三題:


C:\Users\Chia-Yu Wang\Desktop\Computer-System-Design\mplab\asm_file\HW14_3.asm*

```
#include <p16Lf1826.inc>

a       equ 0x25
c       equ 0x24
answer  equ 0x23          ;//題目中count變數就是answer
count   equ 0x22
temp    equ 0x21
mod     equ 0x20
        org 0x00
        clrf count          ;// count <- 0
        movlw .21           ;// w <- 21
        movwf a             ;// a <- w (a = 21)
        movlw 3             ;// w <- 3
        movwf c             ;// c <- w (c = 3)
        clrf temp           ;// temp <- 0
loop    movf c,0            ;// w <- c (w = c)
        subwf a,1           ;// a <- a - w (a = a - c)
        movlw 1             ;// w <- 1 (w = 1)
        addwf count,1       ;// count <- w + count (count++)
        btfss a,7
        goto loop

        subwf count,1       ;// count--
        movf count,0        ;// w <- count
        movwf answer        ;// answer <- w
        movwf PORTB         ;// port_b <- w (port_b = count = answer)
        movf a,0            ;// w <- a
        subwf temp,1        ;// temp = temp - w (temp -= a) (temp = 0-a)
        movf c,0            ;// w <- c (w = c)
        movwf mod           ;// mod <- w (mod = c)
        movf temp,0         ;// w <- temp
        subwf mod,1         ;// mod <- mod - w (mod = mod - temp) (mod = c - temp)
        movf mod,0          ;// w <- mod
        movwf PORTB         ;// port_b <- w (port_b = mod)
        end
```

■ 第四題:

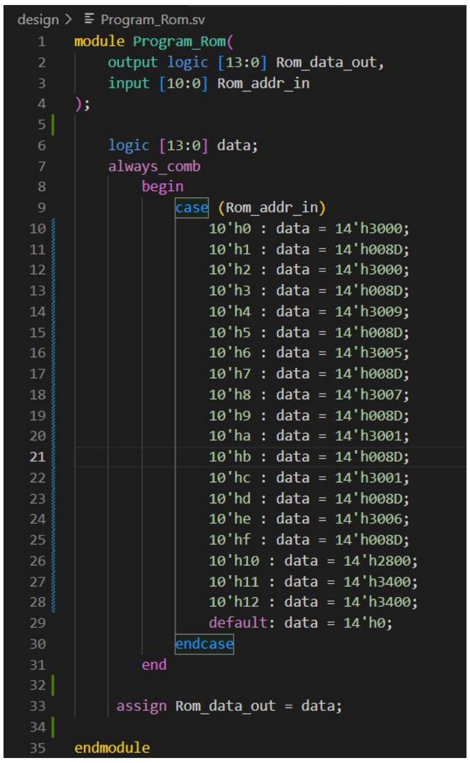


C:\Users\Chia-Yu Wang\Desktop\Computer-System-Design\mplab\asm_file\hw14_4.asm

```
#include <p16Lf1826.inc>

a       equ 0x25
c       equ 0x24
answer  equ 0x23
temp    equ 0x21
mod     equ 0x20

        org 0x00
        movlw .36          ;// w <- 36
        movwf a            ;// a <- w (a = 36)
        movlw .21          ;// w <- 21
        movwf c            ;// c <- w (c = 21)

;// mod = a%c
loop    movf c,0           ;// w <- c (w = c)
        subwf a,1          ;// a <- a - w (a = a - c)
        btfss a,7
        goto loop
        movf a,0           ;// w <- a
        clrf temp          ;// temp <- 0
        subwf temp,1       ;// temp = temp - w (temp -= a) (temp = 0-a)
        movf c,0           ;// w <- c (w = c)
        movwf mod          ;// mod <- w (mod = c)
        movf temp,0        ;// w <- temp
        subwf mod,1        ;// mod <- mod - w (mod = mod - temp) (mod = c - temp)
        movf c,0           ;// w <- c
        movwf a            ;// a <- w (a = c)
        movf mod,0         ;// w <- mod
        movwf c            ;// c <- w (c = mod)
        btfss c,7
        goto test0         ;// >=0 會跳去檢驗是否為0
        goto loweq0        ;// <0  則跳出迴圈

;//檢驗是否為0
test0   movf c,0           ;// w <- c
        movwf temp         ;// temp <- w (temp = c)
        decf temp          ;// temp--
        btfss temp,7       ;// >=0 就回loop
        goto loop
        goto loweq0        ;// <0  則跳出迴圈

;//<=0則
loweq0  movf a,0           ;// w <- a
        movwf answer       ;// ans <- w
        movwf PORTB        ;// port_b <- w (port_b = a)
        end
```

✧ Program_Rom.sv

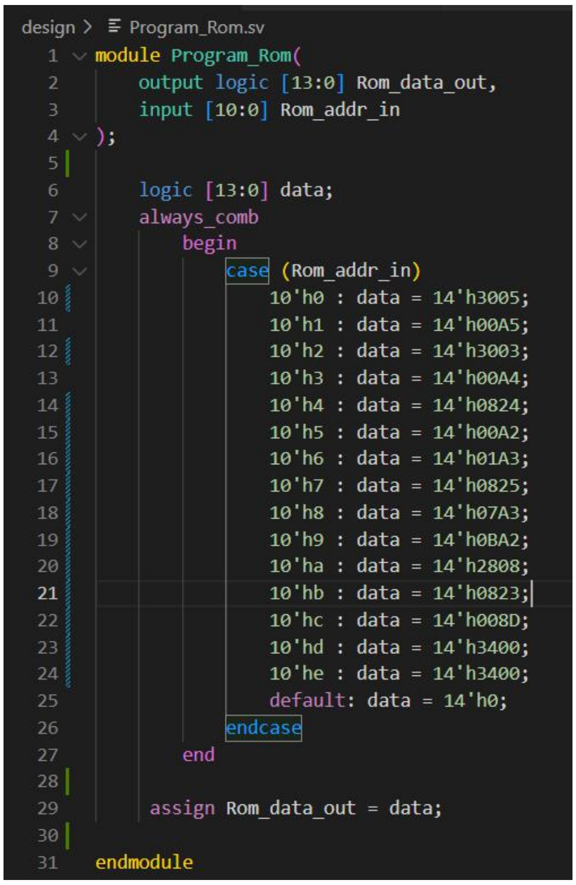
■ 第一題:



design > 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'h3000;
                10'h1 : data = 14'h008D;
                10'h2 : data = 14'h3000;
                10'h3 : data = 14'h008D;
                10'h4 : data = 14'h3009;
                10'h5 : data = 14'h008D;
                10'h6 : data = 14'h3005;
                10'h7 : data = 14'h008D;
                10'h8 : data = 14'h3007;
                10'h9 : data = 14'h008D;
                10'ha : data = 14'h3001;
                10'hb : data = 14'h008D;
                10'hc : data = 14'h3001;
                10'hd : data = 14'h008D;
                10'he : data = 14'h3006;
                10'hf : data = 14'h008D;
                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
```

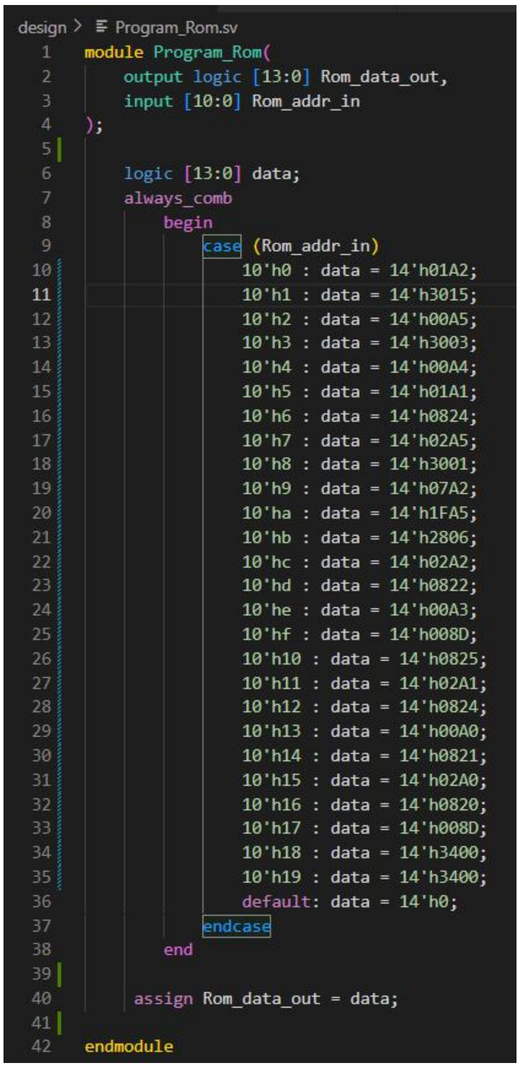
■ 第二題:



design > 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'h3005;
                10'h1 : data = 14'h00A5;
                10'h2 : data = 14'h3003;
                10'h3 : data = 14'h00A4;
                10'h4 : data = 14'h0824;
                10'h5 : data = 14'h00A2;
                10'h6 : data = 14'h01A3;
                10'h7 : data = 14'h0825;
                10'h8 : data = 14'h07A3;
                10'h9 : data = 14'h0BA2;
                10'ha : data = 14'h2808;
                10'hb : data = 14'h0823;
                10'hc : data = 14'h008D;
                10'hd : data = 14'h3400;
                10'he : data = 14'h3400;
                default: data = 14'h0;
            endcase
        end

     assign Rom_data_out = data;

endmodule
```

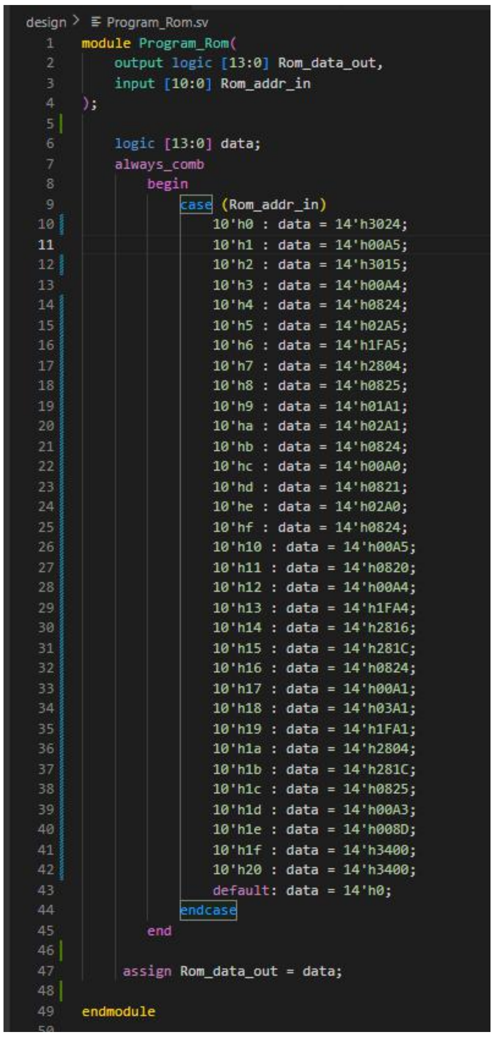
■ 第三題:



design > 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'h01A2;
                10'h1 : data = 14'h3015;
                10'h2 : data = 14'h00A5;
                10'h3 : data = 14'h3003;
                10'h4 : data = 14'h00A4;
                10'h5 : data = 14'h01A1;
                10'h6 : data = 14'h0824;
                10'h7 : data = 14'h02A5;
                10'h8 : data = 14'h3001;
                10'h9 : data = 14'h07A2;
                10'ha : data = 14'h1FA5;
                10'hb : data = 14'h2806;
                10'hc : data = 14'h02A2;
                10'hd : data = 14'h0822;
                10'he : data = 14'h00A3;
                10'hf : data = 14'h008D;
                10'h10 : data = 14'h0825;
                10'h11 : data = 14'h02A1;
                10'h12 : data = 14'h0824;
                10'h13 : data = 14'h00A0;
                10'h14 : data = 14'h0821;
                10'h15 : data = 14'h02A0;
                10'h16 : data = 14'h0820;
                10'h17 : data = 14'h008D;
                10'h18 : data = 14'h3400;
                10'h19 : data = 14'h3400;
                default: data = 14'h0;
            endcase
        end

     assign Rom_data_out = data;

endmodule
```

- 第四題:

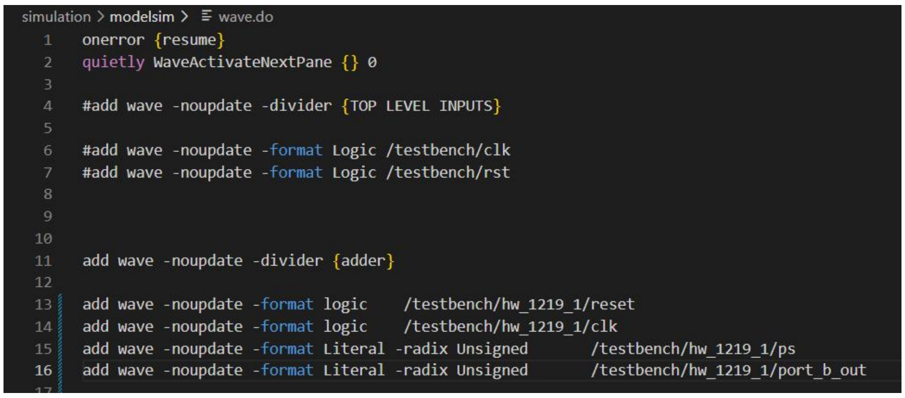


design > Program_Rom.sv

```
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'h3024;
                10'h1 : data = 14'h00A5;
                10'h2 : data = 14'h3015;
                10'h3 : data = 14'h00A4;
                10'h4 : data = 14'h0824;
                10'h5 : data = 14'h02A5;
                10'h6 : data = 14'h1FA5;
                10'h7 : data = 14'h2804;
                10'h8 : data = 14'h0825;
                10'h9 : data = 14'h01A1;
                10'ha : data = 14'h02A1;
                10'hb : data = 14'h0824;
                10'hc : data = 14'h00A0;
                10'hd : data = 14'h0821;
                10'he : data = 14'h02A0;
                10'hf : data = 14'h0824;
                10'h10 : data = 14'h00A5;
                10'h11 : data = 14'h0820;
                10'h12 : data = 14'h00A4;
                10'h13 : data = 14'h1FA4;
                10'h14 : data = 14'h2816;
                10'h15 : data = 14'h281C;
                10'h16 : data = 14'h0824;
                10'h17 : data = 14'h00A1;
                10'h18 : data = 14'h03A1;
                10'h19 : data = 14'h1FA1;
                10'h1a : data = 14'h2804;
                10'h1b : data = 14'h281C;
                10'h1c : data = 14'h0825;
                10'h1d : data = 14'h00A3;
                10'h1e : data = 14'h008D;
                10'h1f : data = 14'h3400;
                10'h20 : data = 14'h3400;
                default: data = 14'h0;
            endcase
        end

     assign Rom_data_out = data;

endmodule
```

✧ Wave.do

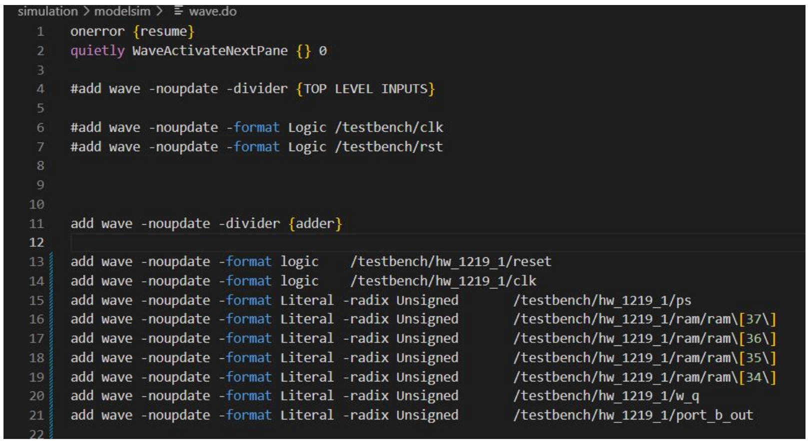
- 第一題:



simulation > modelsim > 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_1219_1/reset
add wave -noupdate -format logic    /testbench/hw_1219_1/clk
add wave -noupdate -format Literal -radix Unsigned        /testbench/hw_1219_1/ps
add wave -noupdate -format Literal -radix Unsigned        /testbench/hw_1219_1/port_b_out
```

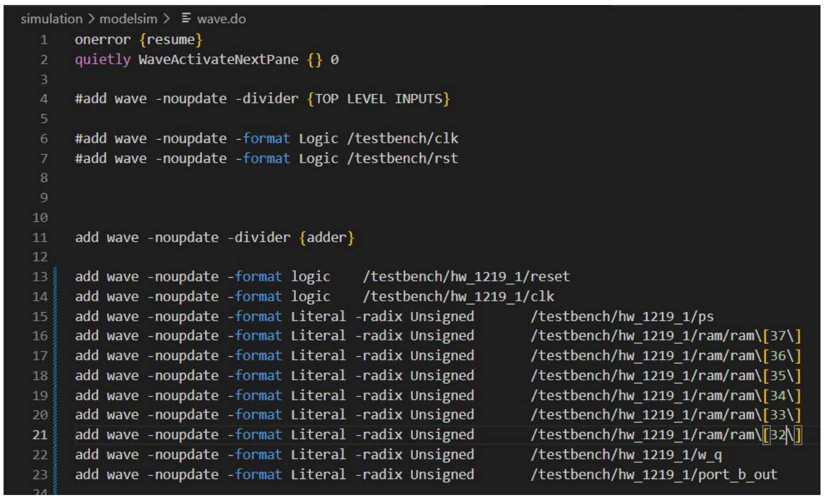
- 第二題:



simulation > modelsim > wave.do

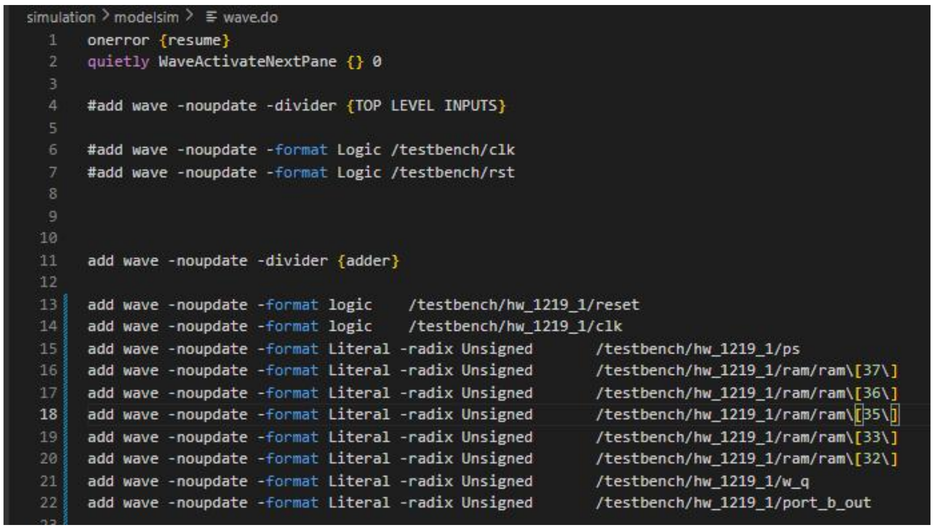
```
onerror {resume}
quietly WaveActivateNextPane {} 0

#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_1219_1/reset
add wave -noupdate -format logic    /testbench/hw_1219_1/clk
add wave -noupdate -format Literal -radix Unsigned        /testbench/hw_1219_1/ps
add wave -noupdate -format Literal -radix Unsigned        /testbench/hw_1219_1/ram/ram\[37\]
add wave -noupdate -format Literal -radix Unsigned        /testbench/hw_1219_1/ram/ram\[36\]
add wave -noupdate -format Literal -radix Unsigned        /testbench/hw_1219_1/ram/ram\[35\]
add wave -noupdate -format Literal -radix Unsigned        /testbench/hw_1219_1/ram/ram\[34\]
add wave -noupdate -format Literal -radix Unsigned        /testbench/hw_1219_1/w_q
add wave -noupdate -format Literal -radix Unsigned        /testbench/hw_1219_1/port_b_out
```

■ 第三題:


simulation > modelsim > wave.do

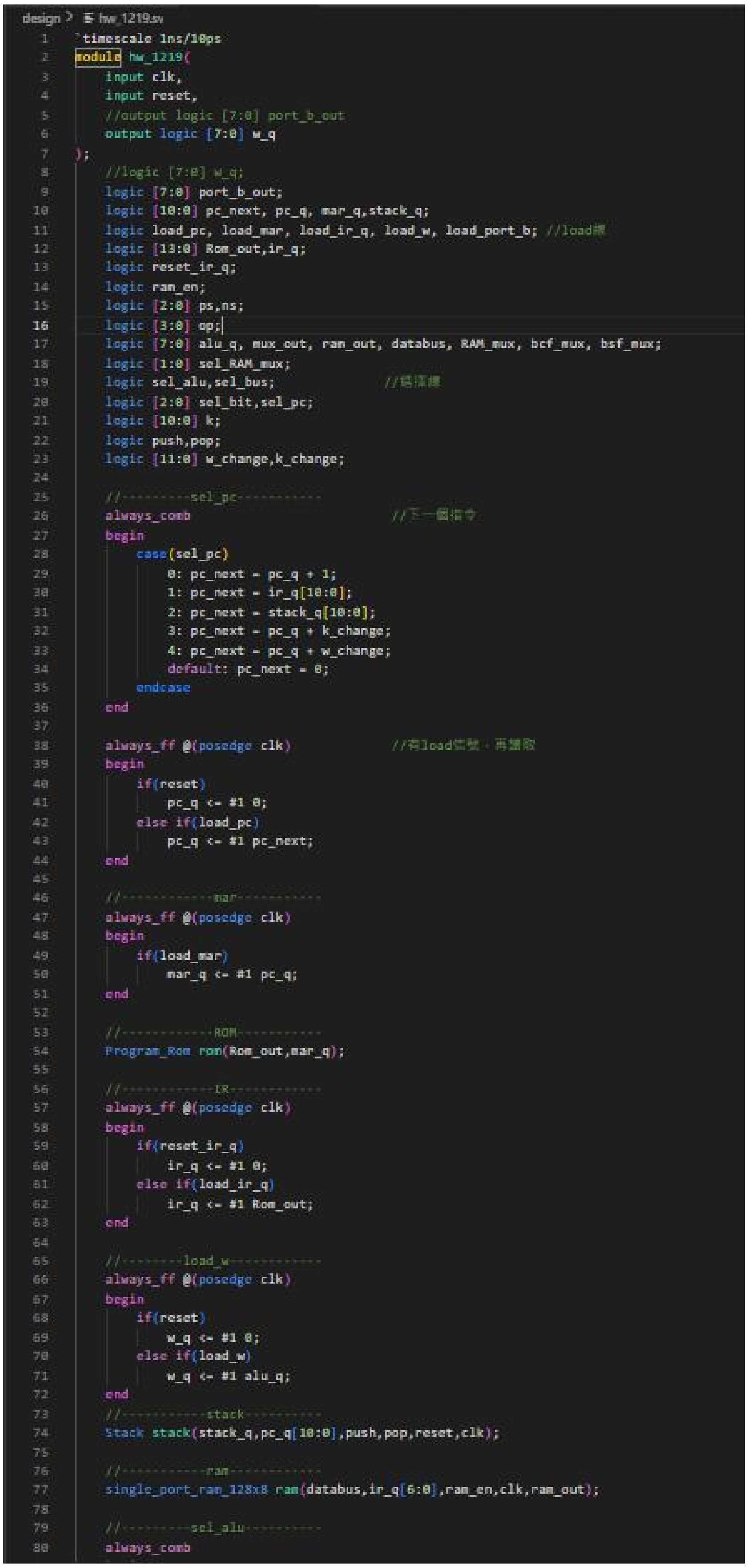
```
onerror {resume}
quietly WaveActivateNextPane {} 0

#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_1219_1/reset
add wave -noupdate -format logic    /testbench/hw_1219_1/clk
add wave -noupdate -format Literal -radix Unsigned       /testbench/hw_1219_1/ps
add wave -noupdate -format Literal -radix Unsigned       /testbench/hw_1219_1/ram/ram\[37\]
add wave -noupdate -format Literal -radix Unsigned       /testbench/hw_1219_1/ram/ram\[36\]
add wave -noupdate -format Literal -radix Unsigned       /testbench/hw_1219_1/ram/ram\[35\]
add wave -noupdate -format Literal -radix Unsigned       /testbench/hw_1219_1/ram/ram\[34\]
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\[32\]
add wave -noupdate -format Literal -radix Unsigned       /testbench/hw_1219_1/w_q
add wave -noupdate -format Literal -radix Unsigned       /testbench/hw_1219_1/port_b_out
```

■ 第四題:


simulation > modelsim > wave.do

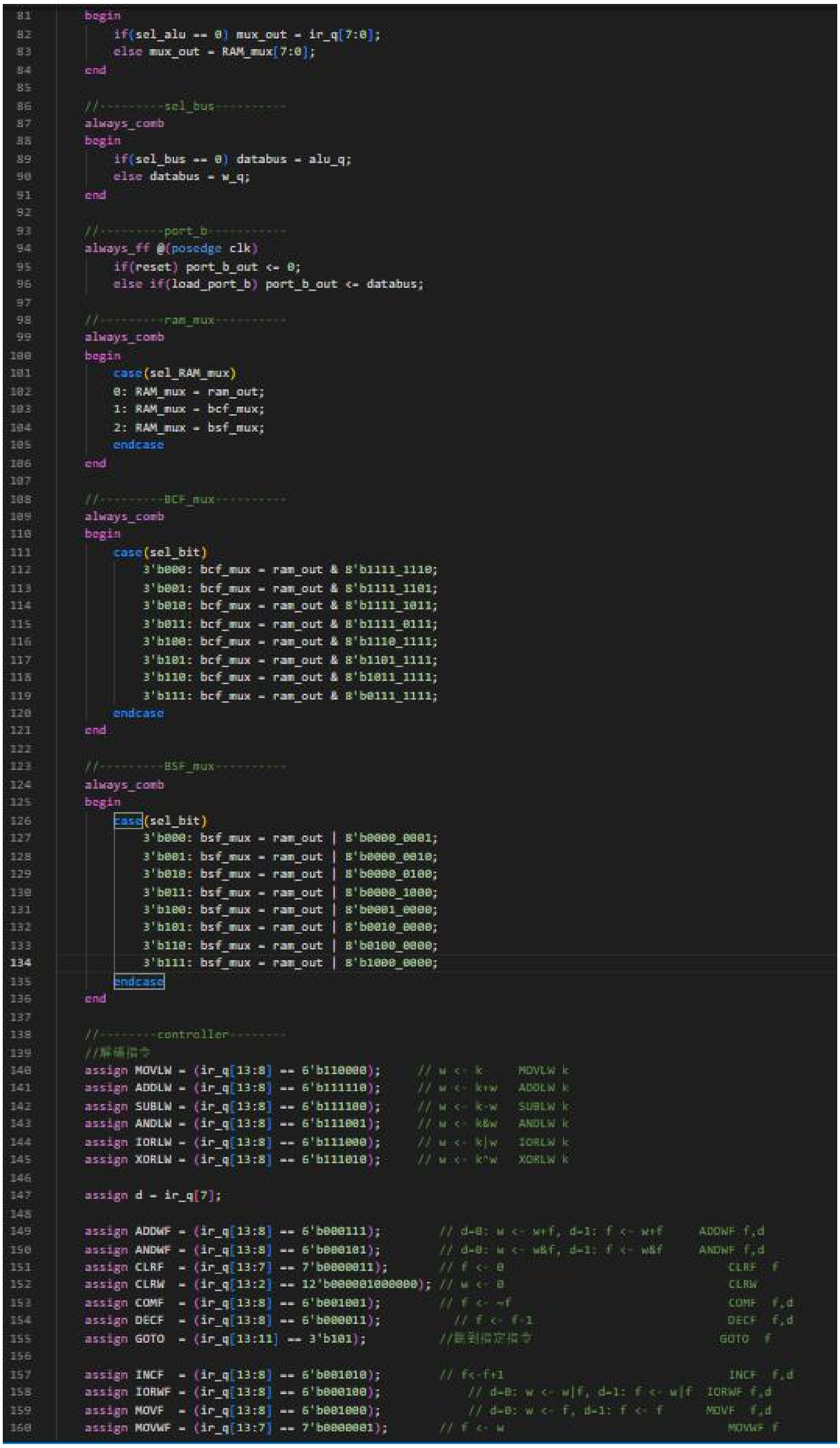
```
onerror {resume}
quietly WaveActivateNextPane {} 0

#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_1219_1/reset
add wave -noupdate -format logic    /testbench/hw_1219_1/clk
add wave -noupdate -format Literal -radix Unsigned       /testbench/hw_1219_1/ps
add wave -noupdate -format Literal -radix Unsigned       /testbench/hw_1219_1/ram/ram\[37\]
add wave -noupdate -format Literal -radix Unsigned       /testbench/hw_1219_1/ram/ram\[36\]
add wave -noupdate -format Literal -radix Unsigned       /testbench/hw_1219_1/ram/ram\[35\]
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\[32\]
add wave -noupdate -format Literal -radix Unsigned       /testbench/hw_1219_1/w_q
add wave -noupdate -format Literal -radix Unsigned       /testbench/hw_1219_1/port_b_out
```

✧ hw_1219.sv

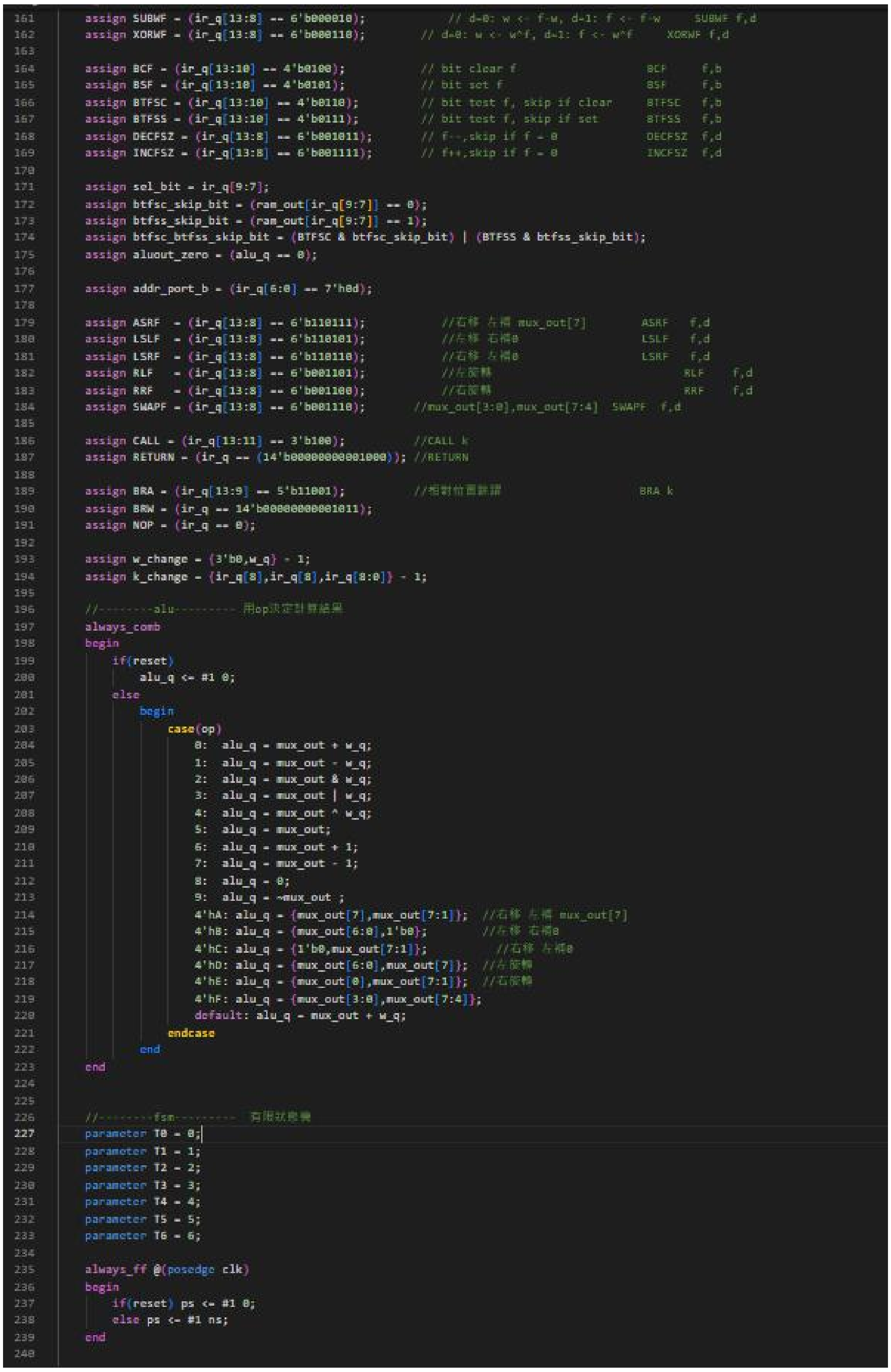


design > hw_1219.sv

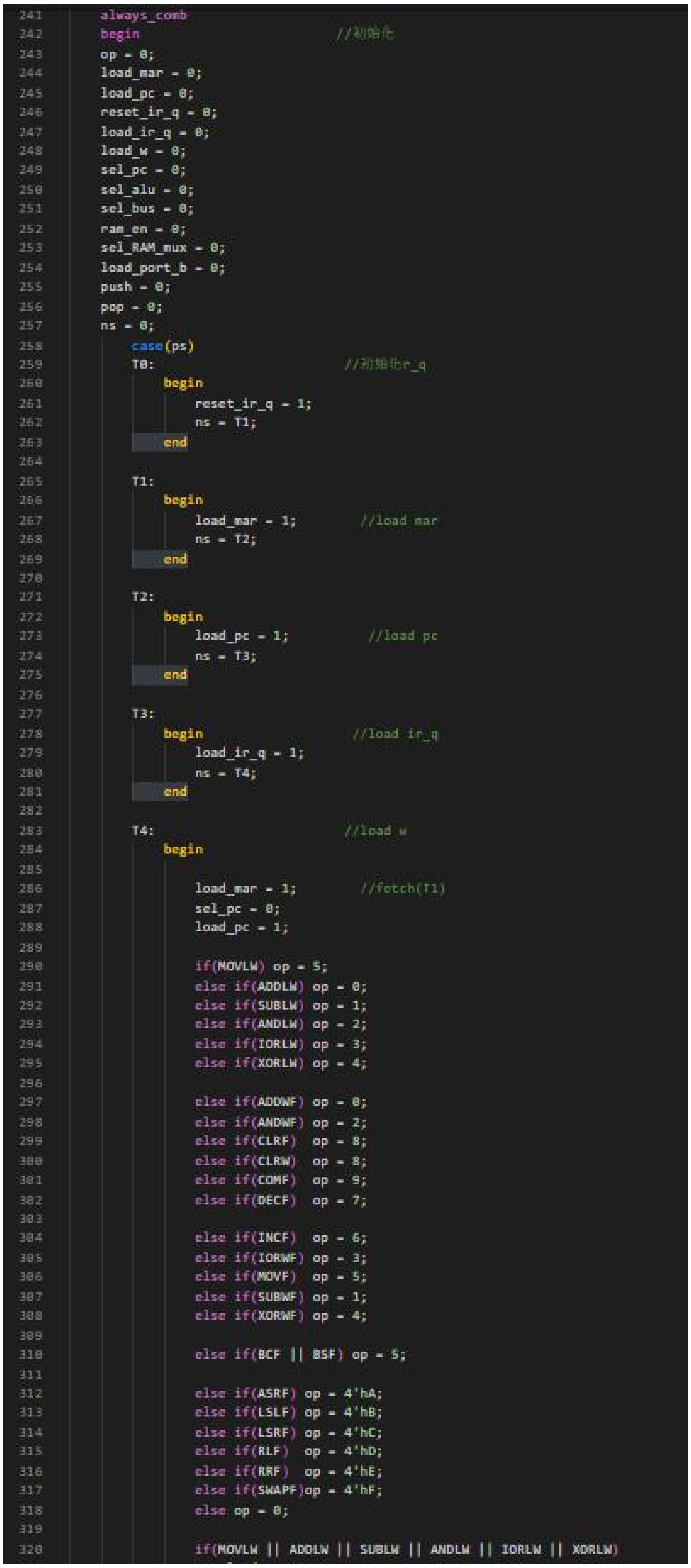
```
`timescale 1ns/10ps
module hw_1219(
    input clk,
    input reset,
    //output logic [7:0] port_b_out
    output logic [7:0] w_q
);
    //logic [7:0] w_q;
    logic [7:0] port_b_out;
    logic [10:0] pc_next, pc_q, mar_q,stack_q;
    logic load_pc, load_mar, load_ir_q, load_w, load_port_b; //load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
```



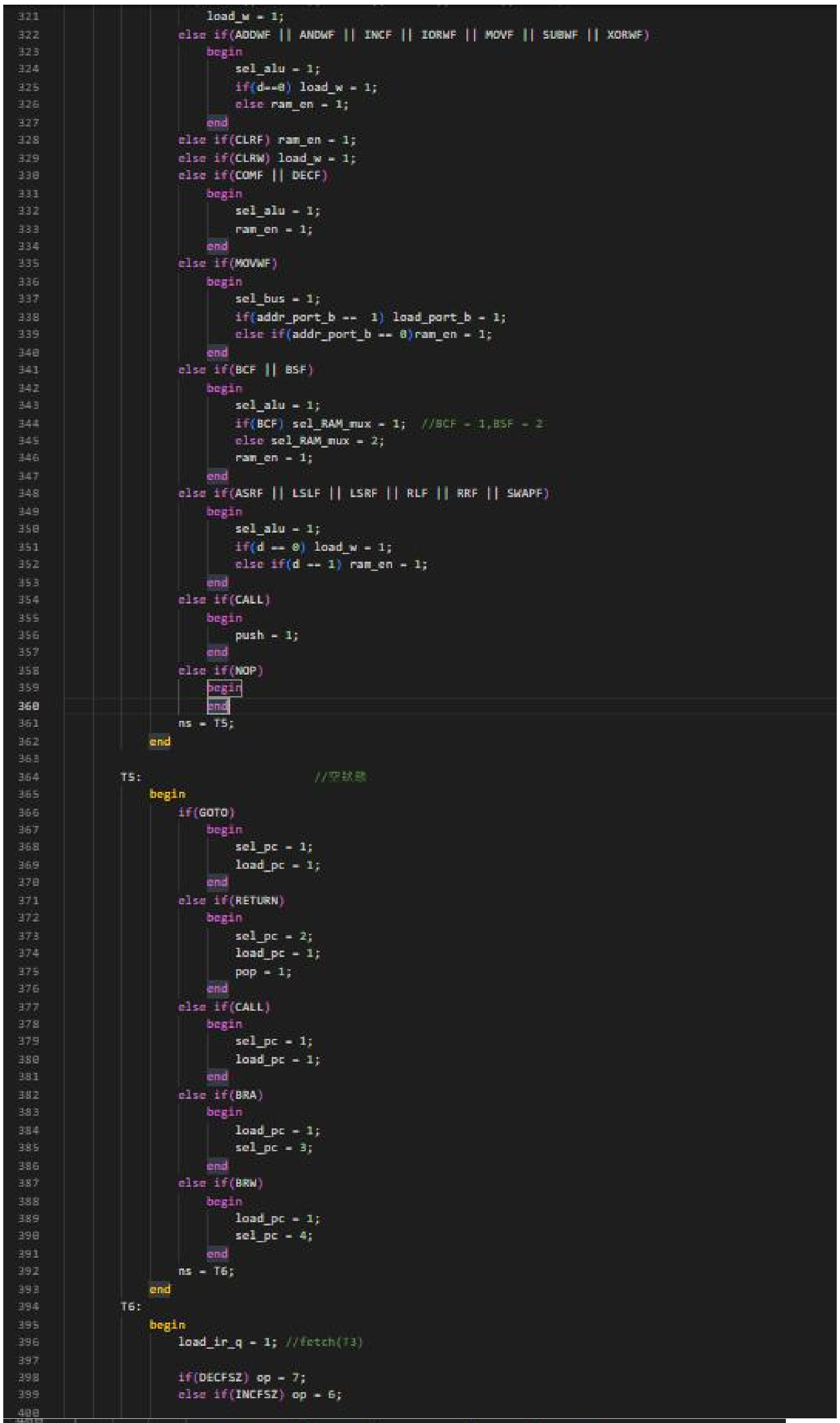
```
    begin
        if(sel_alu == 0) mux_out = ir_q[7:0];
        else mux_out = RAM_mux[7:0];
    end

    //---------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
```



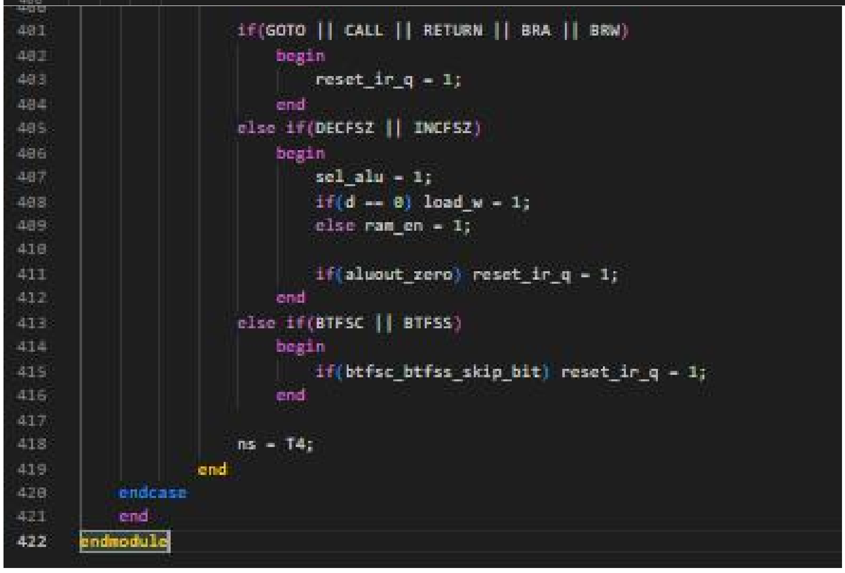
```
    assign SUBWF = (ir_q[13:8] == 6'b000010);            // d=0: w <- f-w, d=1: f <- f-w     SUBWF f,d
    assign XORWF = (ir_q[13:8] == 6'b000110);        // d=0: w <- w^f, d=1: f <- w^f     XORWF f,d

    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)
```

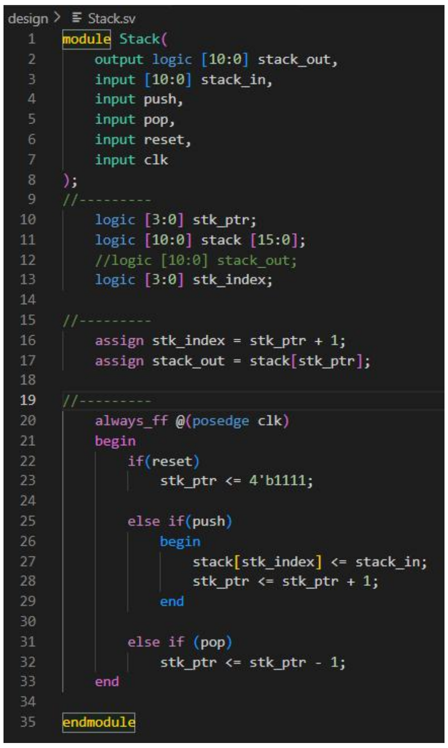


```verilog
                        load_w = 1;
                else if(ADDWF || ANDWF || INCF || IORWF || MOVF || SUBWF || XORWF)
                    begin
                        sel_alu = 1;
                        if(d==0) load_w = 1;
                        else ram_en = 1;
                    end
                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:                         //空状態
            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;

```



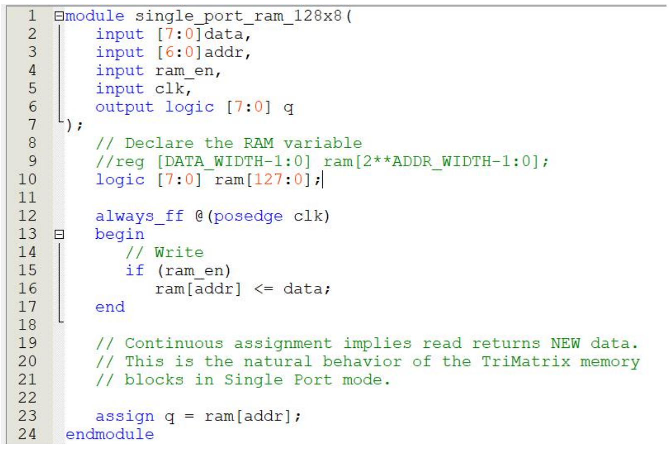
```verilog
                if(GOTO || CALL || RETURN || BRA || BRW)
                    begin
                        reset_ir_q = 1;
                    end
                else if(DECFSZ || INCFSZ)
                    begin
                        sel_alu = 1;
                        if(d == 0) load_w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



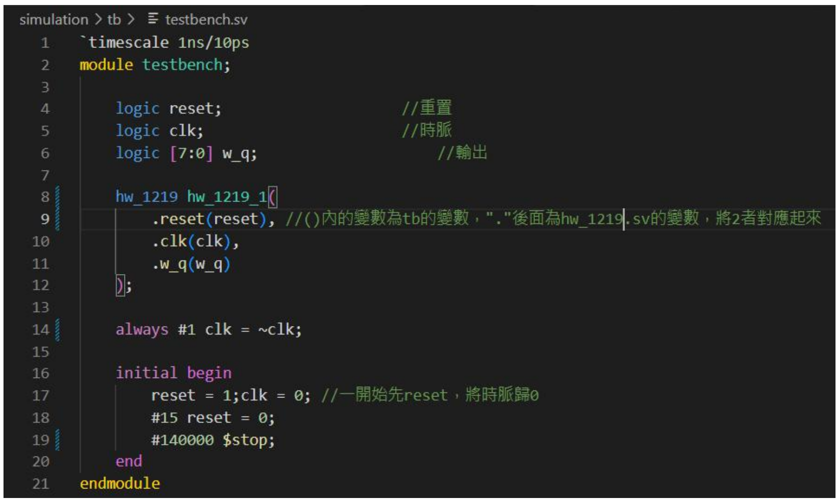
```
module Stack(
    output logic [10:0] stack_out,
    input [10:0] stack_in,
    input push,
    input pop,
    input reset,
    input clk
);
//---------
    logic [3:0] stk_ptr;
    logic [10:0] stack [15:0];
    //logic [10:0] stack_out;
    logic [3:0] stk_index;

//---------
    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
```

✧ single_port_ram_128x8.sv



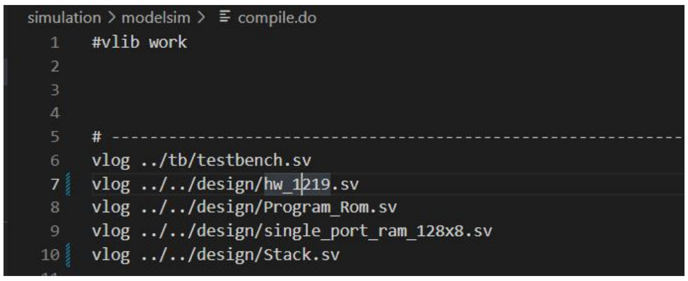
```
module single_port_ram_128x8(
    input [7:0]data,
    input [6:0]addr,
    input ram_en,
    input clk,
    output logic [7:0] q
);
    // Declare the RAM variable
    //reg [DATA_WIDTH-1:0] ram[2**ADDR_WIDTH-1:0];
    logic [7:0] ram[127:0];

    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



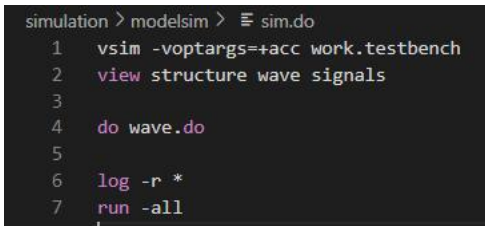
```
`timescale 1ns/10ps
module testbench;

    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


simulation > modelsim > 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
```

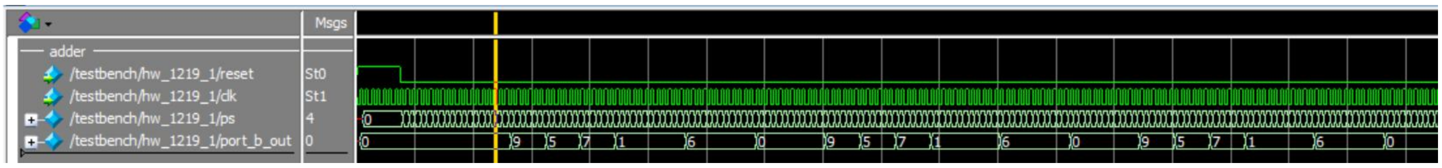
✧ sim.do


simulation > modelsim > sim.do

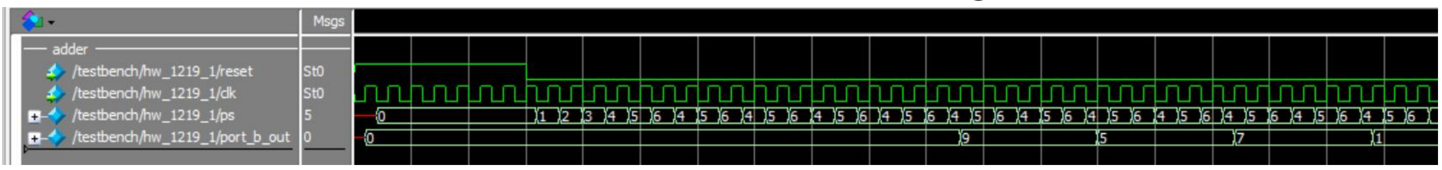
```
vsim -voptargs=+acc work.testbench
view structure wave signals

do wave.do

log -r *
run -all
```

● 模擬結果與結果說明：

✧ 第一題:

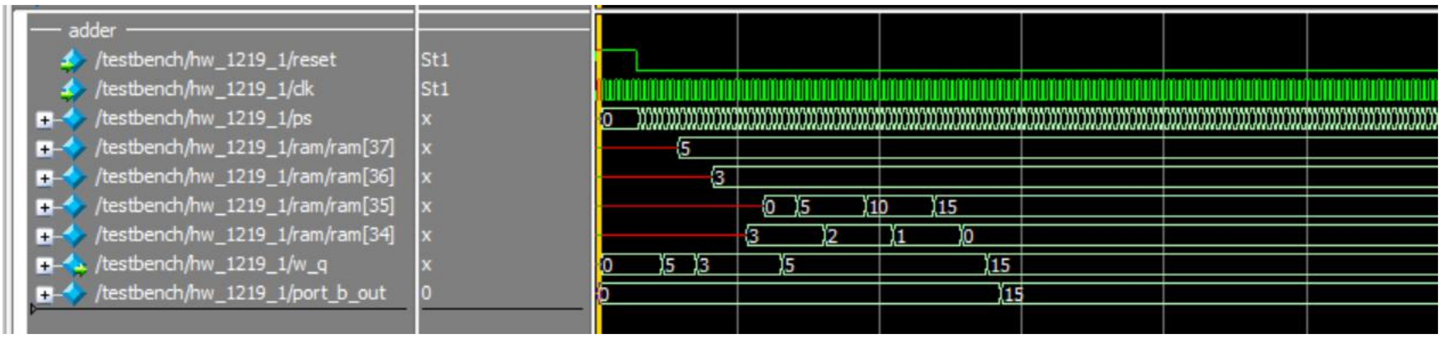

Msgs
adder
/testbench/hw_1219_1/reset St0
/testbench/hw_1219_1/clk St1
/testbench/hw_1219_1/ps 4
/testbench/hw_1219_1/port_b_out 0

(學號:00957116，6 會比較長是因為他去做下一次迴圈，所以會經過 goto 的指令)

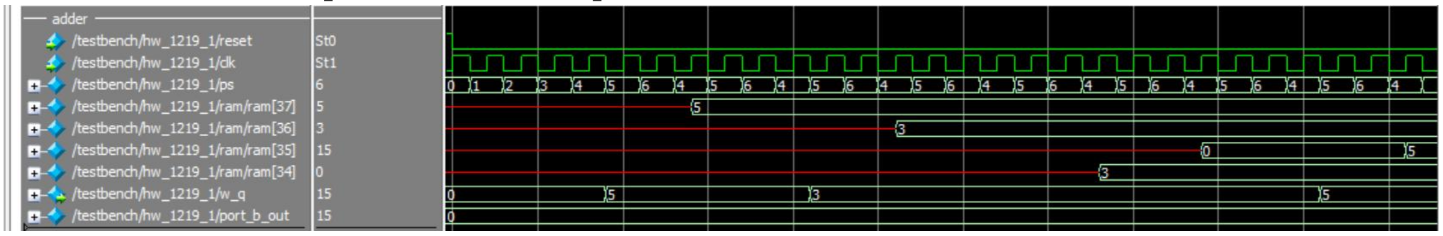

Msgs
adder
/testbench/hw_1219_1/reset St0
/testbench/hw_1219_1/clk St0
/testbench/hw_1219_1/ps 5
/testbench/hw_1219_1/port_b_out 0

(pipeline ps: 1->2->3->4->5->6->4->5->6->4............)

✧ 第二題:

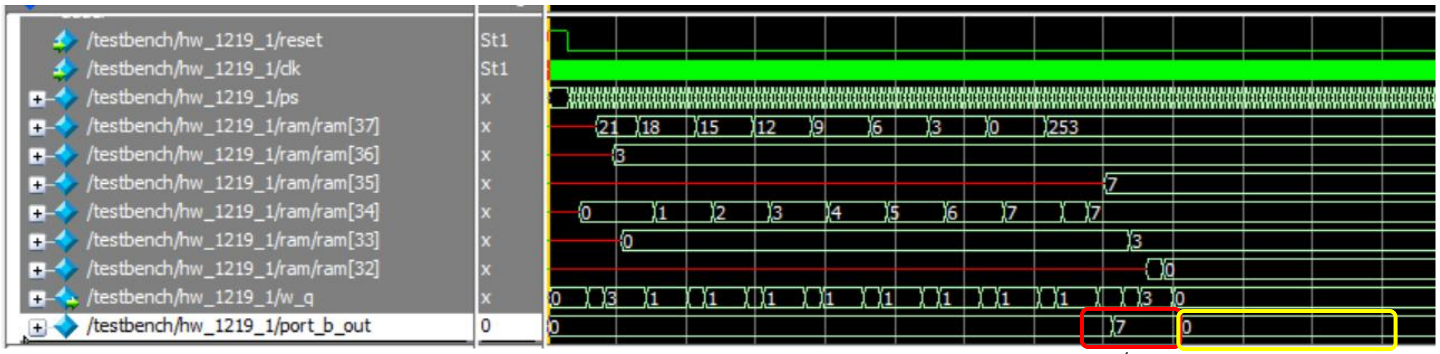

adder
/testbench/hw_1219_1/reset St1
/testbench/hw_1219_1/clk St1
/testbench/hw_1219_1/ps x
/testbench/hw_1219_1/ram/ram[37] x
/testbench/hw_1219_1/ram/ram[36] x
/testbench/hw_1219_1/ram/ram[35] x
/testbench/hw_1219_1/ram/ram[34] x
/testbench/hw_1219_1/w_q x
/testbench/hw_1219_1/port_b_out 0

ram[37] -> a
ram[36] -> c
ram[35] -> answer
ram[34] -> count
當答案算完後(離開 loop 時)，才將 ans 給 port_b


adder
/testbench/hw_1219_1/reset St0
/testbench/hw_1219_1/clk St1
/testbench/hw_1219_1/ps 6
/testbench/hw_1219_1/ram/ram[37] 5
/testbench/hw_1219_1/ram/ram[36] 3
/testbench/hw_1219_1/ram/ram[35] 15
/testbench/hw_1219_1/ram/ram[34] 0
/testbench/hw_1219_1/w_q 15
/testbench/hw_1219_1/port_b_out 15

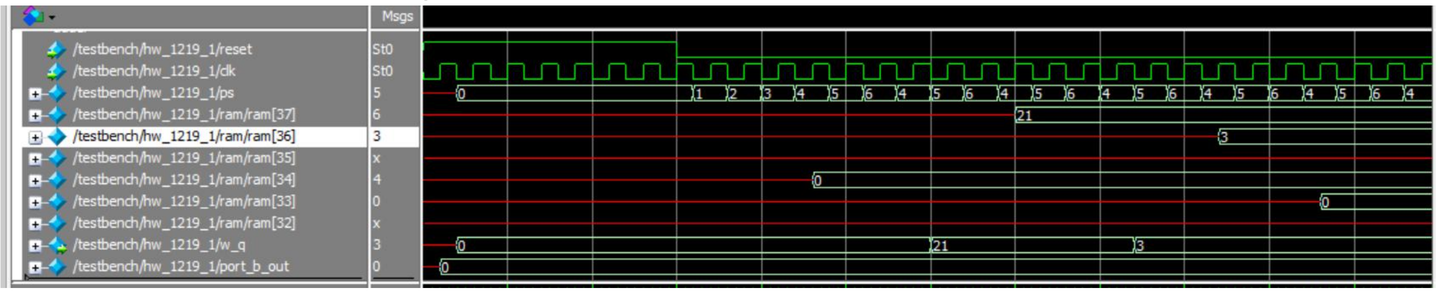
(pipeline ps: 1->2->3->4->5->6->4->5->6->4............)

✧ 第三題:


/testbench/hw_1219_1/reset St1
/testbench/hw_1219_1/clk St1
/testbench/hw_1219_1/ps x
/testbench/hw_1219_1/ram/ram[37] x
/testbench/hw_1219_1/ram/ram[36] x
/testbench/hw_1219_1/ram/ram[35] x
/testbench/hw_1219_1/ram/ram[34] x
/testbench/hw_1219_1/ram/ram[33] x
/testbench/hw_1219_1/ram/ram[32] x
/testbench/hw_1219_1/w_q x
/testbench/hw_1219_1/port_b_out 0

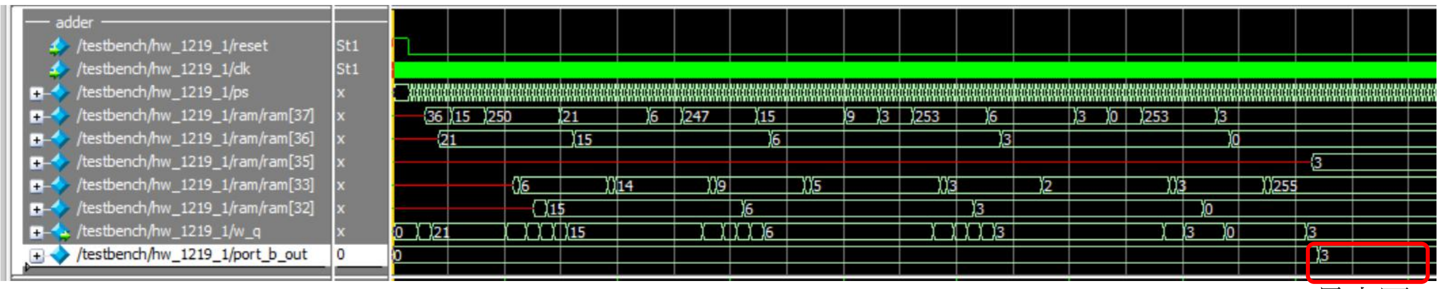
a/c　　a%c

在組合語言中對應到 ram 的變數:
ram[37] -> a (每次都-c，直到<0)
ram[36] -> c
ram[35] -> answer
ram[34] -> count (算-了幾次，就是答案)
ram[33] -> temp
ram[32] -> mod
Port_b 第一次的值為 a/c (同 answer，ram[35])
第二次為 a%c (同 mod，ram[32])


Msgs
/testbench/hw_1219_1/reset St0
/testbench/hw_1219_1/clk St0
/testbench/hw_1219_1/ps 5
/testbench/hw_1219_1/ram/ram[37] 6
/testbench/hw_1219_1/ram/ram[36] 3
/testbench/hw_1219_1/ram/ram[35] x
/testbench/hw_1219_1/ram/ram[34] 4
/testbench/hw_1219_1/ram/ram[33] 0
/testbench/hw_1219_1/ram/ram[32] x
/testbench/hw_1219_1/w_q 3
/testbench/hw_1219_1/port_b_out 0

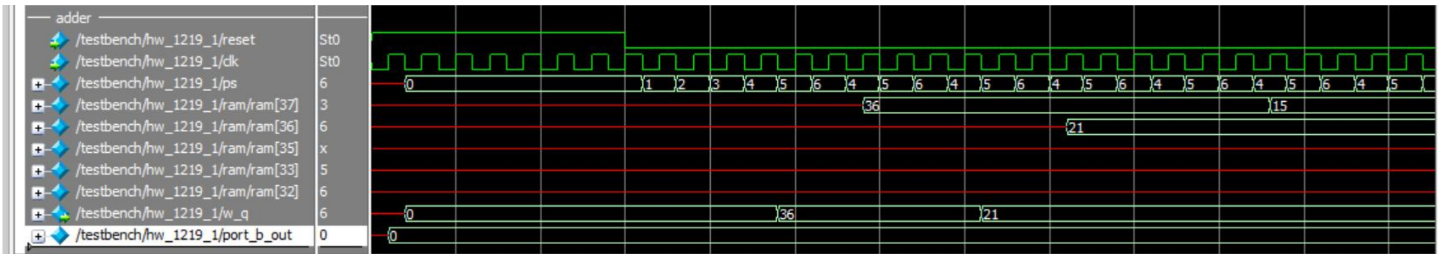
(pipeline ps: 1->2->3->4->5->6->4->5->6->4............)

✧ 第四題:


adder
/testbench/hw_1219_1/reset St1
/testbench/hw_1219_1/clk St1
/testbench/hw_1219_1/ps x
/testbench/hw_1219_1/ram/ram[37] x
/testbench/hw_1219_1/ram/ram[36] x
/testbench/hw_1219_1/ram/ram[35] x
/testbench/hw_1219_1/ram/ram[33] x
/testbench/hw_1219_1/ram/ram[32] x
/testbench/hw_1219_1/w_q x
/testbench/hw_1219_1/port_b_out 0

最大因

在組合語言中對應到 ram 的變數:
ram[37] -> a
ram[36] -> c
ram[35] -> answer
ram[33] -> temp
ram[32] -> mod
邏輯:
mod = a%c
a = c
c = mod
如果 c <=0 則 a 就是答案，把 a 輸入到 ans 和 port_b
如果 c > 0 則一直重複直到<=0



```
adder
/testbench/hw_1219_1/reset         St0
/testbench/hw_1219_1/clk           St0
/testbench/hw_1219_1/ps            6
/testbench/hw_1219_1/ram/ram[37]   3
/testbench/hw_1219_1/ram/ram[36]   6
/testbench/hw_1219_1/ram/ram[35]   x
/testbench/hw_1219_1/ram/ram[33]   5
/testbench/hw_1219_1/ram/ram[32]   6
/testbench/hw_1219_1/w_q           6
/testbench/hw_1219_1/port_b_out    0
```

(pipeline ps: 1->2->3->4->5->6->4->5->6->4............)

結論與心得：

- 終於寫完作業了...一開始看到 4 題 c 語言翻譯覺得很可怕，但從第一題這樣慢慢寫，循序漸進後，就還好了。因為最難的應該是第四題，前面就好像堆積木一樣，慢慢把他堆起來。所以到了最後就會發現都差不多，都是加一點加一點東西這樣。不過我遇到最大的問題是，為甚麼沒有變數=變數的指令呀?這樣如果要做 c = a - b 都必須要:

  w = a

  c = w

  w = b

  c = c - w

- 這樣子好麻煩歐 q，都要透過 w 來當中間的媒介。我想了想是不是因為我們做的 cpu 指令很短呀，現在是 sub f,d 如果要變數 = 變數的話就必須: sub f,f,d 之類的，但是我們的指令長度似乎沒有辦法那麼長，應該是因為這樣吧(?

- 這份作業寫完就正式 ending 掉這堂課了，考試我覺得還算理想吧，我交卷的時候助教:也太快了吧，那這樣表示我是第一個寫完的吧(x 我覺得這個考試算是有鑑別度的考試，因為現在的課程偏向”照著 ppt 打”只要這樣做然後不要粗心就會對，所以如果沒有給要哪個地方做甚麼，而是只給指令內容的話，那些照打的可能會很難適應，畢竟也沒有好好讀過自己的 code。我很幸運的在改成 pipeline 時，有再好好想過為啥有些要在 t4 做有些要在 t5 做，所以就很剛好的考試會寫，耶!

- 再 3 周就過年啦!提前祝老師、助教新年快樂~兔年發大財，我也要快點完成這學期的其他作業和報告了!