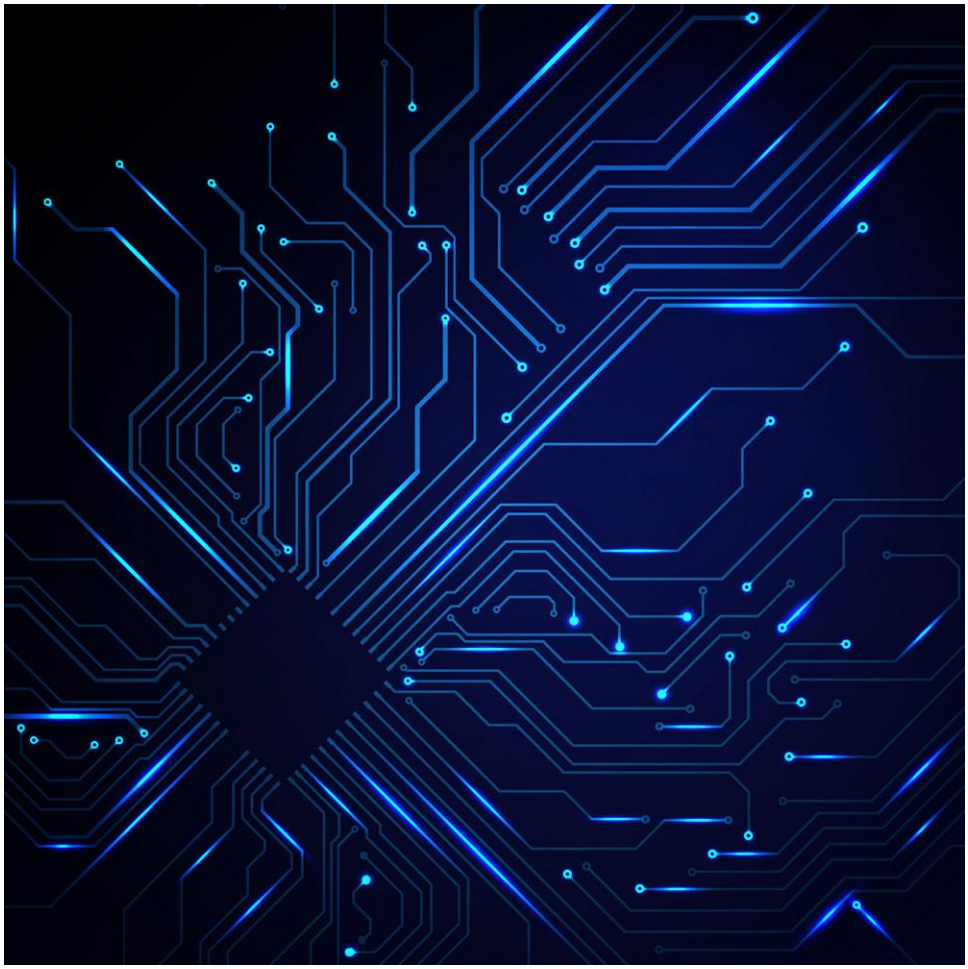


# 注意

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

2022/XX/XX

# 實驗十一

# 暫存器定址指令

姓名：王嘉羽　　學號：00957116

班級：資工 3B

E-mail：vayne20011125@gmail.com

2022/12/05

## ● 實驗說明：

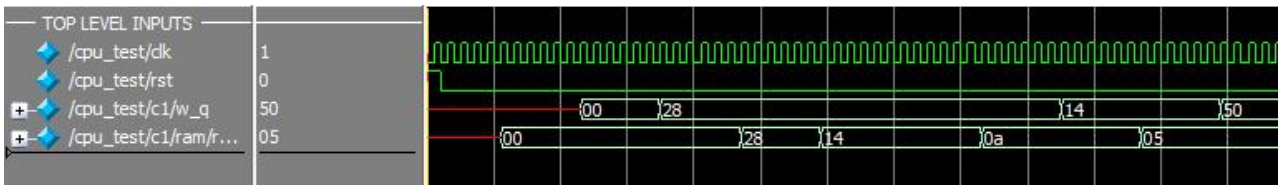
1. 如圖所示，設計一個架構實現條件跳躍指令
2. 輸入：clk, reset
3. 輸出：w_q[7:0]

請務必按照規定的 input 及 output 來做

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

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

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

            clrf    temp        ; //ram[37]<=0
            clrw                ; //w<=0
            movlw   28          ; //w<=0x28
            movwf   temp        ; //ram[37]<=0x28
            asrf    temp,1      ; //ram[37]<=0x14
            rlf     temp,0      ; //w<=0x28
            lsrf    temp,1      ; //ram[37]<=0x0a
            lslf    temp,0      ; //w<=0x14
            rrf     temp,1      ; //ram[37]<=0x05
            swapf   temp,0      ; //w<=0x50
            goto    $
            end
```

組合語言




模擬結果

## ● 系統架構程式碼、測試資料程式碼與程式碼說明

## 截圖請善用 win+shift+S

✧ hw_1205.sv

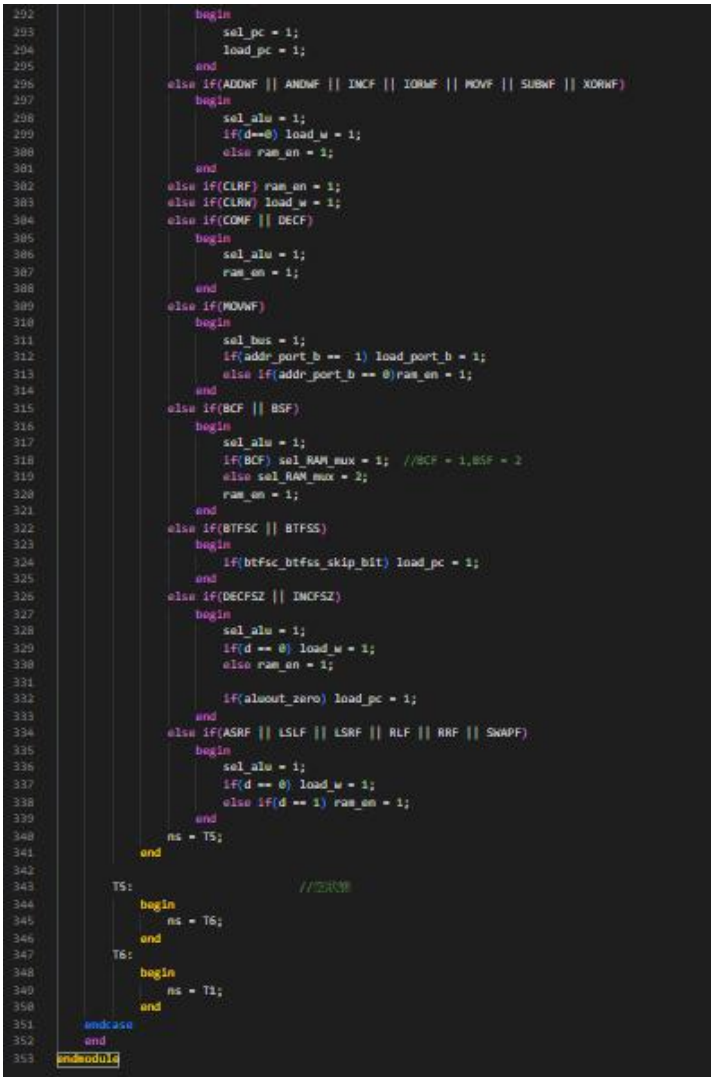
hw_1205.sv U

design > hw_1205.sv

```
`timescale 1ns/10ps
module hw_1205(
    input clk,
    input reset,
    //output logic [7:0] port_b_out
    output logic [7:0] w_q
);
    //logic [7:0] w_q;
    logic [10:0] pc_next, pc_q, mar_q;
    logic load_pc, load_mar, load_ir_q, load_w, load_port_b; //load訊
    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, port_b_out;
    logic [1:0] sel_RAM_mux;
    logic sel_alu,sel_pc,sel_bus;              //選擇線
    logic [2:0] sel_bit;

    //---------sel_pc----------
    always_comb                         //下一個指令
    begin
        if(sel_pc) pc_next = ir_q[10:0];
        else pc_next = pc_q + 1;
    end

    always_ff @(posedge clk)            //有load訊號，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
    //----------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--------
//解码指令
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);
//--------alu--------- 当op决定计算结果
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 左补 mux_out[7]
                4'hB: alu_q = {mux_out[6:0],1'b0};        //左移 右补0
                4'hC: alu_q = {1'b0,mux_out[7:1]};          //右移 左补0
                4'hD: alu_q = {mux_out[6:0],mux_out[7]};  //左旋转
                4'hE: alu_q = {mux_out[0],mux_out[7:1]};  //右旋转
                4'hF: alu_q = {mux_out[3:0],mux_out[7:4]};
```

```
                default: alu_q = mux_out + w_q;
            endcase
        end
    end


    //--------fsm---------  状态机编辑
    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;
    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
                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)
```



✧ Program_Rom.sv

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'h01A5;
                10'h1 : data = 14'h0103;
                10'h2 : data = 14'h3028;
                10'h3 : data = 14'h00A5;
                10'h4 : data = 14'h37A5;
                10'h5 : data = 14'h0D25;
                10'h6 : data = 14'h36A5;
                10'h7 : data = 14'h3525;
                10'h8 : data = 14'h0CA5;
                10'h9 : data = 14'h0E25;
                10'ha : data = 14'h280A;
                10'hb : data = 14'h3400;
                10'hc : data = 14'h3400;
                default: data = 14'h0;
            endcase
        end

     assign Rom_data_out = data;

endmodule
```

✧ single_port_ram_128x8.sv

design > single_port_ram_128x8.sv

```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

simulation > tb > testbench.sv

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

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

    hw_1205 hw_1205_1(
        .reset(reset), //()內的變數為tb的變數，"."後面為hw_1205.sv的變數，將2者對應起來
        .clk(clk),
        .w_q(w_q)
    );

    always #5 clk = ~clk;

    initial begin
        reset = 1;clk = 0; //一開始先reset，將時脈歸0
        #15 reset = 0;
        #5000 $stop;
    end
endmodule
```

✧ compile.do

simulation > modelsim > compile.do

```
#vlib work



# ----------------------------------------------------------
vlog ../tb/testbench.sv
vlog ../../design/hw_1205.sv
vlog ../../design/Program_Rom.sv
vlog ../../design/single_port_ram_128x8.sv
```

✧ sim.do

simulation > modelsim > sim.do

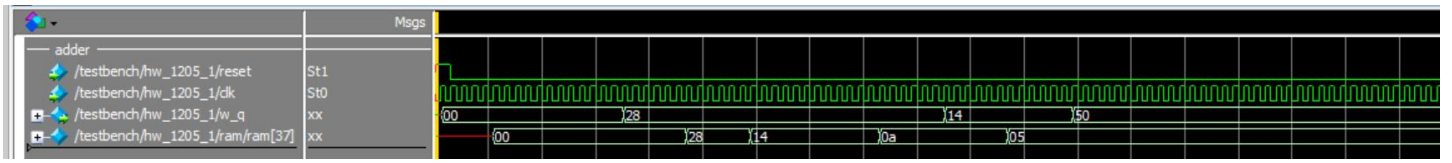
```
vsim -voptargs=+acc work.testbench
view structure wave signals

do wave.do

log -r *
run -all
```

✧ wave.do

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_1205_1/reset
add wave -noupdate -format logic    /testbench/hw_1205_1/clk
add wave -noupdate -format Literal -radix Hexadecimal   /testbench/hw_1205_1/w_q
add wave -noupdate -format Literal -radix Hexadecimal   /testbench/hw_1205_1/ram/ram\[37\]
```

## ● 模擬結果與結果說明：




和作業要求一致~一開始不一致我還以為是我組合語言打錯了，結果我發現我 ram[37]和 w_q 放反了，上下的順序反了，然後調一下就對了!

## ● 結論與心得：

- 隨著程式碼變長，每次遇到問題就越來越難找了，因為不知道是這禮拜造成的錯，還是以前就有錯了。最近助教把前幾次的作業改好，我發現有一個地方我在檢查結果時沒有檢查到，然後助教發現錯了，我就去檢查我的 code，不檢查還好，一檢查發現..挖....錯一大堆，有一串 opcode 我都打錯...然後當初還可以正常運行...好好笑...幸好有即時發現，如果在段考才發現，真的會不知道該怎麼辦才好..
- 有了這個經驗，我現在寫 code 會寫完就檢查一次，不會等結果有錯才檢查(以前都是 ce 或結果錯誤才檢查....)，也因為這樣這次作業很快就完成了!沒有奇奇怪怪的錯誤!
- 這次有錯在一個地方，就是變數要初始化，我以前覺得初始化沒有很重要，反正會賦值給他，初值就不重要了，不過這好像是個很不好的習慣，不管寫甚麼語言、不管那個值會不會用到，都要養成初始化的好習慣才行!