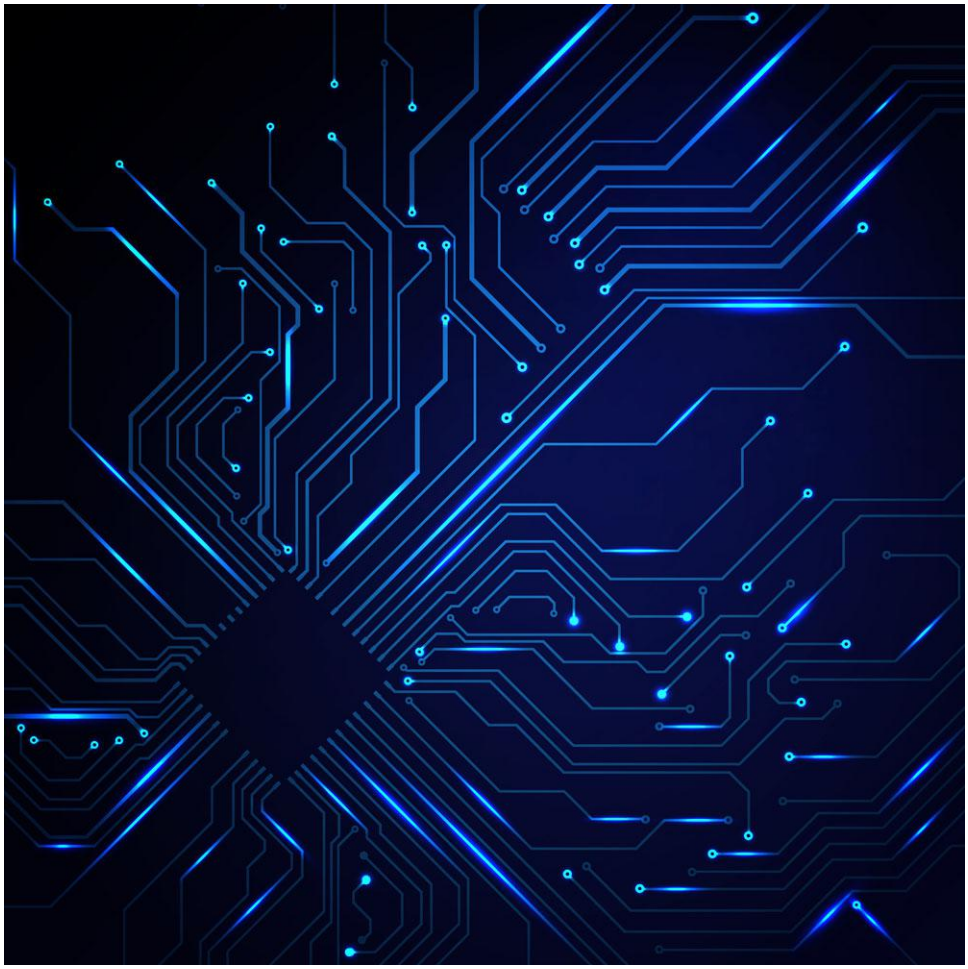


注意

1. 繳交時一律轉 PDF 檔
2. 繳交期限為
   上完課後
   當週五晚上 12 點前
3. 一人繳交一份
4. 檔名：學號_HW?.pdf
   檔名請按照作業檔名格式進行填寫
   未依照格式不予批改

2022/11/28

實驗十

條件跳躍指令

姓名：王嘉羽　　學號：00957116

班級：資工 3B

E-mail：vayne20011125@gmail.com

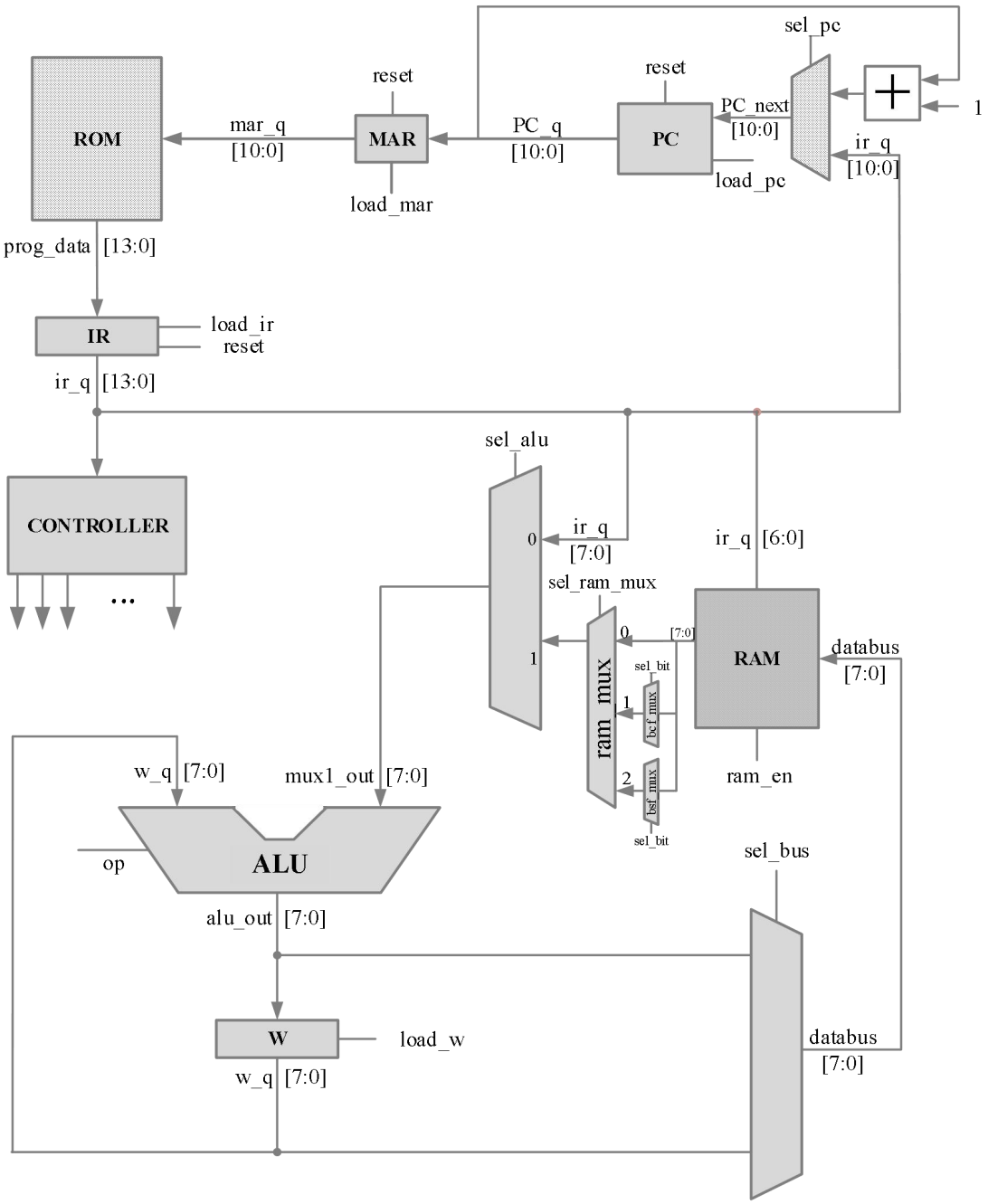
● 實驗說明：

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

請務必按照規定的 input 及 output 來做

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

● 系統硬體架構方塊圖（接線圖）：



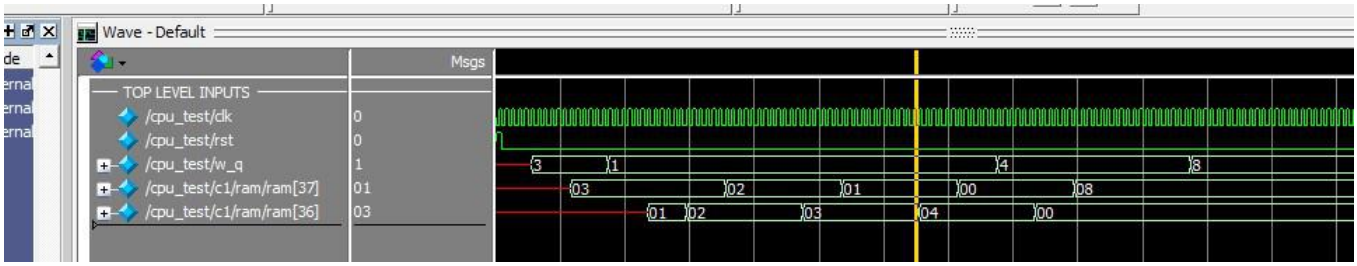
架構圖

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

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

loop         incf templ,1              ;ram[24]++
             decfsz temp,1             ;if(ram[25]!=0)ram[25]--
             goto loop                 ;goto前兩行程式位址
             movf templ,0              ;w=ram[24]
             bcf templ,2               ;ram[24]=0;
             bsf temp,3                ;ram[25]=8;
             btfsc temp,3
             btfss temp,3
             movf  templ,0
             movf  temp,0
             goto $                    ;stop
             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_1128.sv

design > hw_1128.sv

```
`timescale 1ns/10ps
module hw_1128(
    input clk,
    input reset,
    output 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號
    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, bs
    logic [1:0] sel_RAM_mux;
    logic sel_alu,sel_pc,sel_bus;              //選擇號
    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
    //-----------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

    //---------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);
    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[13:8] == 6'b000001);
    assign CLRW  = (ir_q[13:8] == 6'b000001);
    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[13:8] == 6'b000000);
    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);
    //--------alu--------- 用op决定計算結果
    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 ;
                    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
```

```
195     always_comb
196     begin                           //初始化
197     load_mar = 0;
198     load_pc = 0;
199     reset_ir_q = 0;
200     load_ir_q = 0;
201     load_w = 0;
202     sel_pc = 0;
203     sel_alu = 0;
204     sel_bus = 0;
205     ram_en = 0;
206     sel_RAM_mux = 0;
207     ns = 0;
208         case(ps)
209         T0:                         //初始化ir_q
210             begin
211                 reset_ir_q = 1;
212                 ns = T1;
213             end
214
215         T1:
216             begin
217                 load_mar = 1;        //load mar
218                 ns = T2;
219             end
220
221         T2:
222             begin
223                 load_pc = 1;          //load pc
224                 ns = T3;
225             end
226
227         T3:
228             begin                    //load ir_q
229                 load_ir_q = 1;
230                 ns = T4;
231             end
232
233         T4:                         //load w
234             begin
235                 if(MOVLW) op = 5;
236                 else if(ADDLW) op = 0;
237                 else if(SUBLW) op = 1;
238                 else if(ANDLW) op = 2;
239                 else if(IORLW) op = 3;
240                 else if(XORLW) op = 4;
241
242                 else if(ADDWF) op = 0;
243                 else if(ANDWF) op = 2;
244                 else if(CLRF)  op = 8;
245                 else if(CLRW)  op = 8;
246                 else if(COMF)  op = 9;
247                 else if(DECF)  op = 7;
248
249                 else if(INCF)  op = 6;
250                 else if(IORWF) op = 3;
251                 else if(MOVF)  op = 5;
252                 else if(SUBWF) op = 1;
253                 else if(XORWF) op = 4;
254
255                 else if(BCF || BSF) op = 5;
256                 else if(DECFSZ) op = 7;
257                 else if(INCFSZ) op = 6;
258                 else op = 10;
259
```

```
                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;
                        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
                    load_w = 1;

                ns = T5;
            end

        T5:                         //空状態
            begin
                ns = T6;
            end
        T6:
            begin
                ns = T1;
            end
    endcase
    end
endmodule
```

◆ 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'h3003;
                10'h1 : data = 14'h00A5;
                10'h2 : data = 14'h3001;
                10'h3 : data = 14'h00A4;
                10'h4 : data = 14'h0AA4;
                10'h5 : data = 14'h0BA5;
                10'h6 : data = 14'h2804;
                10'h7 : data = 14'h0824;
                10'h8 : data = 14'h1124;
                10'h9 : data = 14'h15A5;
                10'ha : data = 14'h19A5;
                10'hb : data = 14'h1DA5;
                10'hc : data = 14'h0824;
                10'hd : data = 14'h0825;
                10'he : data = 14'h280E;
                10'hf : data = 14'h3400;
                10'h10 : data = 14'h3400;
                default: data = 14'h0;
            endcase
        end
    assign Rom_data_out = data;
endmodule
```

◆ single_port_ram_128x8.sv

```
design/hw_1128.sv | design/Program_Rom.sv | Compilation Report - mcu | design/single_port_ram_128
```

```
1  module single_port_ram_128x8(
2      input [7:0]data,
3      input [6:0]addr,
4      input ram_en,
5      input clk,
6      output logic [7:0] q
7  );
8      // Declare the RAM variable
9      //reg [DATA_WIDTH-1:0] ram[2**ADDR_WIDTH-1:0];
10     logic [7:0] ram[127:0];
11
12     always_ff @(posedge clk)
13     begin
14        // Write
15        if (ram_en)
16           ram[addr] <= data;
17     end
18
19     // Continuous assignment implies read returns NEW data.
20     // This is the natural behavior of the TriMatrix memory
21     // blocks in Single Port mode.
22
23     assign q = ram[addr];
24 endmodule
```

- testbench.do

simulation > tb > testbench.sv

```
1  `timescale 1ns/10ps
2  module testbench;
3
4      logic reset;                    //重置
5      logic clk;                      //時脈
6      logic [7:0] w_q;                    //輸出
7
8      hw_1128 hw_1128_1(
9          .reset(reset), //()內的變數為tb的變數，"."後面為hw_1128.sv的變數，將2者對應起來
10         .clk(clk),
11         .w_q(w_q)
12     );
13
14     always #10 clk = ~clk;
15
16     initial begin
17         reset = 1;clk = 0; //一開始先reset，將時脈歸0
18         #15 reset = 0;
19         #4000 $stop;
20     end
21 endmodule
```

- sim.do

simulation > modelsim > sim.do

```
1  vsim -voptargs=+acc work.testbench
2  view structure wave signals
3
4  do wave.do
5
6  log -r *
7  run -all
```

- wave.do

simulation > modelsim > wave.do

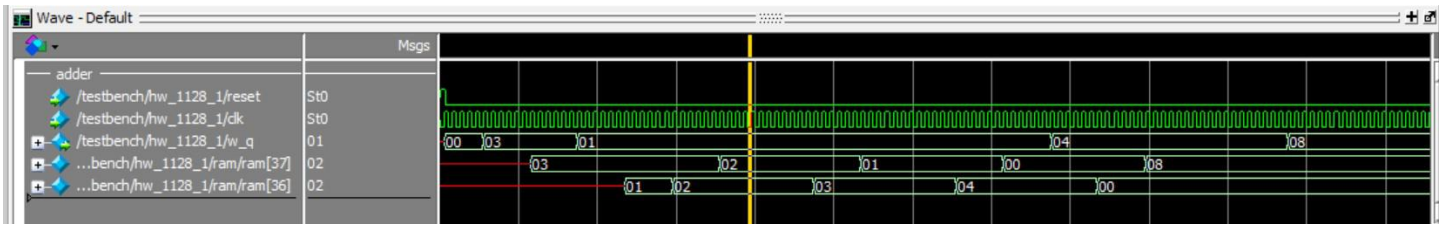
```
5
6  #add wave -noupdate -format Logic /testbench/clk
7  #add wave -noupdate -format Logic /testbench/rst
8
9
10
11 add wave -noupdate -divider {adder}
12
13 add wave -noupdate -format logic    /testbench/hw_1128_1/reset
14 add wave -noupdate -format logic    /testbench/hw_1128_1/clk
15 add wave -noupdate -format Literal -radix Hexadecimal   /testbench/hw_1128_1/w_q
16 add wave -noupdate -format Literal -radix Hexadecimal   /testbench/hw_1128_1/ram/ram\[37\]
17 add wave -noupdate -format Literal -radix Hexadecimal   /testbench/hw_1128_1/ram/ram\[36\]
```

- compile.do

simulation > modelsim > compile.do

```
1  #vlib work
2
3
4
5  # ----------------------------------------------------------------
6  vlog ../tb/testbench.sv
7  vlog ../../design/hw_1128.sv
8  vlog ../../design/Program_Rom.sv
9  vlog ../../design/single_port_ram_128x8.sv
10
```

模擬結果與結果說明：



和 ppt 結果相符合，助教上課也檢查過，應該是對的!

● 結論與心得：

- ✏️ 這次的主題是跳躍指令，越來越複雜了。然後寫 code 過程也非常不順利，一直有錯，然後都找不太到，找到一個後發現又有其他的錯，最一開始我是錯在 bcf 和 bsf 的 mux，要 and 或是 or 沒有寫好。然後還有各式各樣的小錯誤，都讓我崩潰不已。不過很感謝助教，他在旁邊非常有耐心的陪我找錯誤，讓我再最後順順利利的成功跑出結果了!
- ✏️ 這也讓我知道程式碼應該要好好寫，不應該求短而省略很多，但可能是軟體寫習慣了，都有一些奇怪的寫法....。可能下次要寫這種大型 code 的時候要好好的寫架構，不要亂寫，不然 debug 真的好痛苦 qq