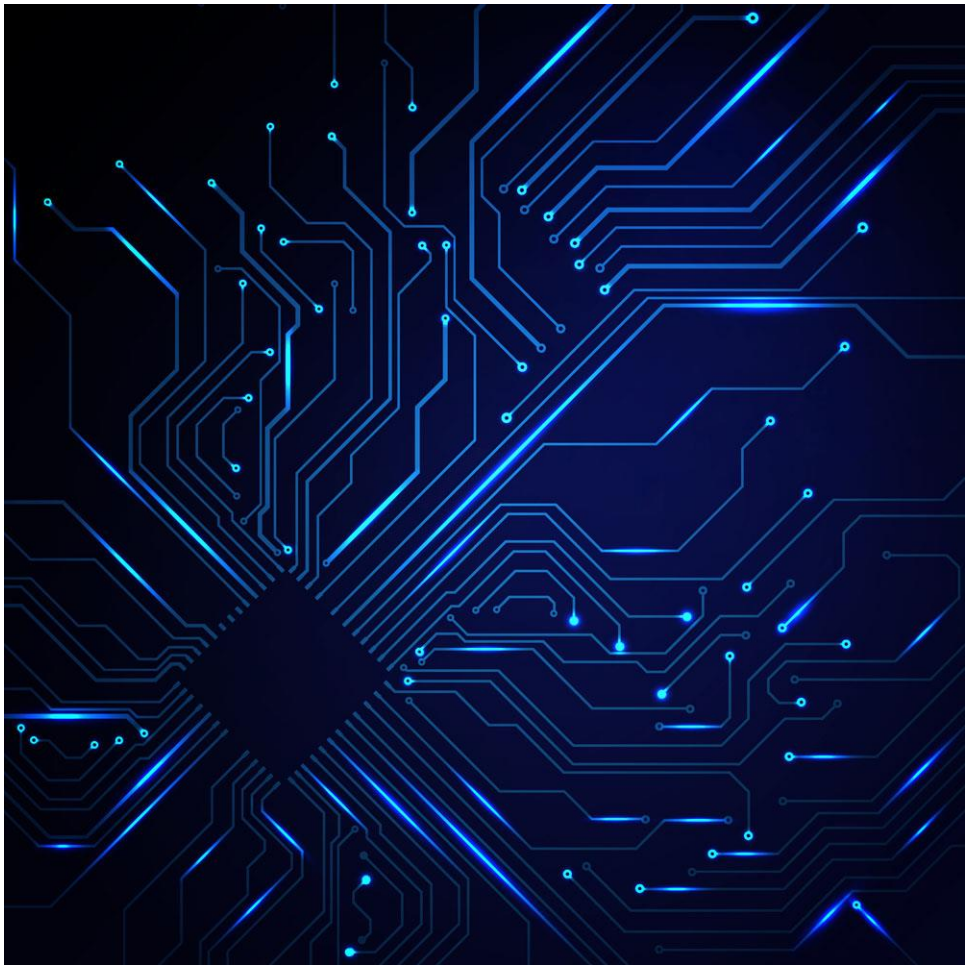


注意

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

2022/12/5

實驗十一

暫存器定址指令

姓名：張銀軒　　學號：00957050

班級：資工 3A

E-mail：00957050@mail.ntou.edu.tw

2022/12/5

● 實驗說明：

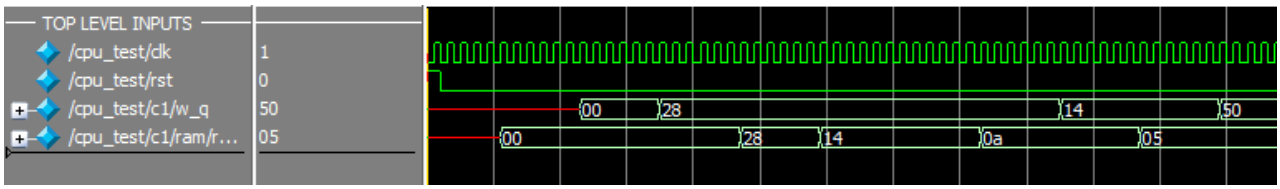
1. 如圖所示，設計一個架構實現條件跳躍指令
2. 輸入：clk, reset
3. 輸出：w_q[7:0]

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

```
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

```

```
module single_port_ram_128x8(
    input [7:0]data,
    input [6:0]addr,
    input ram_en,
    input clk,
    output logic [7:0] ram_out
);
    // 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 ram_out = ram[addr];
endmodule


```

```
module ALU (
    input [3:0] op,
    input [7:0] w_q, mux1_out,
    output logic [7:0] alu_q
);
always_comb
begin
    case(op)
        4'h0: alu_q = mux1_out[7:0] + w_q;
        4'h1: alu_q = mux1_out[7:0] - w_q;
        4'h2: alu_q = mux1_out[7:0] & w_q;
        4'h3: alu_q = mux1_out[7:0] | w_q;
        4'h4: alu_q = mux1_out[7:0] ^ w_q; //XOR
        4'h5: alu_q = mux1_out[7:0];
        4'h6: alu_q = mux1_out[7:0] + 1;
        4'h7: alu_q = mux1_out[7:0] - 1;
        4'h8: alu_q = 0;
        4'h9: alu_q = ~mux1_out[7:0];
        4'hA: alu_q = { mux1_out[7],mux1_out[7:1] };    //ASRF arithmetic right shift
        4'hB: alu_q = { mux1_out[6:0], 1'b0 };          //LSLF logical left shift
        4'hC: alu_q = { 1'b0, mux1_out[7:1] };          //LSRF logical right shift
        4'hD: alu_q = { mux1_out[6:0], mux1_out[7] };   //RLF rotate left f
        4'hE: alu_q = { mux1_out[0], mux1_out[7:1] };   //RRF rotate right f
        4'hF: alu_q = { mux1_out[3:0], mux1_out[7:4] }; //{m7, m6,...m4, m3,...m0} => {m3,...m0, m7, m6,...m4}
        default: alu_q = mux1_out[7:0] + w_q;
    endcase
end

endmodule
```

```
module cpu (
    input clk,
    input reset,
    output logic [7:0] port_b_out
);
    logic [13:0] rom_q, ir_q;
    logic [10:0] pc_next, pc_q, mar_q;
    logic load_pc, load_mar, load_ir, reset_ir, load_w, sel_pc, ram_en, sel_alu, d, sel_bus;
    logic load_port_b;
    logic [3:0] ps,ns;
    logic [7:0] w_q, alu_q, ram_out, mux1_out, bcf_mux, bsf_mux, ram_mux;
    logic [7:0] databus;
    logic [3:0] op;
    logic [5:0] opcode;
    logic [2:0] sel_bit;
    logic [1:0] sel_ram_mux;
    //mux 0 (select next PC address)
    always_comb begin
        if(sel_pc) begin
            pc_next = ir_q;
        end
        else begin
            pc_next = pc_q + 1;
        end
    end

    //pc
    always_ff @( posedge clk ) begin
        if(reset)
            pc_q <= 0;
        else if(load_pc)
            pc_q <= pc_next;
    end

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

    //ROM
    Program_Rom ROM_1(
        .Rom_addr_in(mar_q),
        .Rom_data_out(rom_q)
    );

    //IR
    always_ff @( posedge clk ) begin
        if(reset)
            ir_q <= 0;
```

```
        else if(load_ir)
            ir_q <= rom_q;
    end

    //RAM
    single_port_ram_128x8 single_port_ram_128x8_1(
        .data(databus),
        .addr(ir_q[6:0]),
        .ram_en(ram_en),
        .clk(clk),
        .ram_out(ram_out)
    );

    assign sel_bit = ir_q[9:7];

    //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

    //ram mux (select data into mux1)
    always_comb begin
        case (sel_ram_mux)
            0: ram_mux = ram_out;
            1: ram_mux = bcf_mux;
            2: ram_mux = bsf_mux;
        endcase
```

```
101     end
102
103     //mux 1 (select data into ALU)
104     always_comb begin
105         if(sel_alu) begin
106             mux1_out = ram_mux;
107         end
108         else begin
109             mux1_out = ir_q;
110         end
111     end
112
113     assign d = ir_q[7];
114     assign  MOVLW = (ir_q[13:8]==6'h30);
115     assign  ADDLW = (ir_q[13:8]==6'h3E);
116     assign  IORLW = (ir_q[13:8]==6'h38);
117     assign  ANDLW = (ir_q[13:8]==6'h39);
118     assign  SUBLW = (ir_q[13:8]==6'h3C);
119     assign  XORLW = (ir_q[13:8]==6'h3A);
120
121     assign  ADDWF = (ir_q[13:8]==6'h07);
122     assign  ANDWF = (ir_q[13:8]==6'h05);
123     assign   CLRF = (ir_q[13:8]==6'h01 && d==1);
124     assign   CLRW = (ir_q[13:4]==10'h010 && ir_q[3:2]==2'h0);
125     assign   COMF = (ir_q[13:8]==6'h09);
126     assign   DECF = (ir_q[13:8]==6'h03);
127     assign   GOTO = (ir_q[13:11]==3'b101);
128
129     assign   INCF = (ir_q[13:8]==6'h0A);
130     assign  IORWF = (ir_q[13:8]==6'h04);
131     assign   MOVF = (ir_q[13:8]==6'h08);
132     assign  MOVWF = (ir_q[13:8]==6'h00 && ir_q[7]==1'b1);
133     assign  SUBWF = (ir_q[13:8]==6'h02);
134     assign  XORWF = (ir_q[13:8]==6'h06);
135
136     assign    BCF = (ir_q[13:12]==2'b01 && ir_q[11:10]==2'b00);
137     assign    BSF = (ir_q[13:12]==2'b01 && ir_q[11:10]==2'b01);
138     assign  BTFSC = (ir_q[13:12]==2'b01 && ir_q[11:10]==2'b10);
139     assign  BTFSS = (ir_q[13:12]==2'b01 && ir_q[11:10]==2'b11);
140     assign DECFSZ = (ir_q[13:8]==6'h0B);
141     assign INCFSZ = (ir_q[13:8]==6'h0F);
142
143     assign btfsc_skip_bit = (ram_out[ir_q[9:7]]==0);
144     assign btfss_skip_bit = (ram_out[ir_q[9:7]]==1);
145     assign btfsc_btfss_skip_bit = (BTFSC&btfsc_skip_bit) |
146                                   (BTFSS&btfss_skip_bit);
147     assign   ASRF = (ir_q[13:8]==6'h37);
148     assign   LSLF = (ir_q[13:8]==6'h35);
149     assign   LSRF = (ir_q[13:8]==6'h36);
150     assign    RLF = (ir_q[13:8]==6'h0D);
```

```
151     assign    RRF = (ir_q[13:8]==6'h0C);
152     assign   SWAP = (ir_q[13:8]==6'h0E); //{m7, m6,...m4, m3,...m0} => {m3,...m0, m7, m6,...m4}
153     //ALU
154     ALU ALU_1(
155         .op(op),
156         .w_q(w_q),
157         .mux1_out(mux1_out),
158         .alu_q(alu_q)
159     );
160     assign aluout_zero = (alu_q == 0);
161
162     //register
163     always_ff @( posedge clk ) begin
164         if(load_w)
165             w_q <= alu_q;
166     end
167
168     //mux 2 (select data into RAM)
169     always_comb begin
170         if(sel_bus) begin
171             databus = w_q;
172         end
173         else begin
174             databus = alu_q;
175         end
176     end
177
178     //Port_b
179     always_ff @( posedge clk ) begin
180         if(reset) begin
181             port_b_out <= 0;
182         end
183         else if(load_port_b) begin
184             port_b_out <= databus;
185         end
186     end
187
188     assign addr_port_b = (ir_q[6:0] == 7'h0D);
189
190     //controller
191     parameter T0 = 0;
192     parameter T1 = 1;
193     parameter T2 = 2;
194     parameter T3 = 3;
195     parameter T4 = 4;
196     parameter T5 = 5;
197     parameter T6 = 6;
198
199     always_ff @( posedge clk ) begin
200         if(reset) ps <= 0;
```

```
201         else ps <= ns;
202     end
203 
204     always_comb begin
205         sel_alu = 0;
206         sel_pc = 0;
207         load_mar = 0;
208         load_pc = 0;
209         reset_ir = 1;
210         load_ir = 0;
211         load_w = 0;
212         ram_en = 0;
213         op =0;
214         sel_ram_mux = 0;
215         sel_bus = 0;
216         load_port_b = 0;
217         ns=0;
218         case(ps)
219             T0: begin
220                 ns = T1;
221             end
222             T1: begin
223                 load_mar = 1;
224                 load_pc = 0;
225                 reset_ir = 0;
226                 load_ir = 0;
227                 load_w = 0;
228                 ns = T2;
229             end
230             T2: begin
231                 sel_pc = 0;
232                 load_mar = 0;
233                 load_pc = 1;
234                 reset_ir = 0;
235                 load_ir = 0;
236                 load_w = 0;
237                 ns = T3;
238             end
239             T3: begin
240                 load_mar = 0;
241                 load_pc = 0;
242                 reset_ir = 0;
243                 load_ir = 1;
244                 load_w = 0;
245                 ns = T4;
246             end
247             T4: begin
248                 load_mar = 0;
249                 load_pc = 0;
250                 reset_ir = 0;
```

```
251                 load_ir = 0;
252
253                 if(MOVLW) begin
254                     sel_alu = 0;
255                     op = 5;
256                     load_w = 1;
257                 end
258                 else if(ADDLW) begin
259                     sel_alu = 0;
260                     op = 0;
261                     load_w = 1;
262                 end
263                 else if(IORLW) begin
264                     sel_alu = 0;
265                     op = 3;
266                     load_w = 1;
267                 end
268                 else if(ANDLW) begin
269                     sel_alu = 0;
270                     op = 2;
271                     load_w = 1;
272                 end
273                 else if(SUBLW) begin
274                     sel_alu = 0;
275                     op = 1;
276                     load_w = 1;
277                 end
278                 else if(XORLW) begin
279                     sel_alu = 0;
280                     op = 4;
281                     load_w = 1;
282                 end
283
284
285                 else if(GOTO) begin
286                     sel_pc = 1;
287                     load_pc = 1;
288                 end
289                 else if(ADDWF) begin
290                     op = 0;
291                     sel_alu = 1;
292                     if(d) begin
293                         ram_en = 1;
294                     end
295                     else begin
296                         load_w = 1;
297                     end
298                 end
299                 else if(ANDWF) begin
300                     op = 2;
```

```
301                     sel_alu = 1;
302                     if(d) begin
303                         ram_en = 1;
304                     end
305                     else begin
306                         load_w = 1;
307                     end
308                 end
309                 else if(CLRF) begin
310                     op = 8;
311                     ram_en = 1;
312                 end
313                 else if(CLRW) begin
314                     op = 8;
315                     load_w = 1;
316                 end
317                 else if(COMF) begin
318                     op = 9;
319                     sel_alu = 1;
320                     ram_en = 1;
321                 end
322                 else if(DECF) begin
323                     op = 7;
324                     sel_alu = 1;
325                     ram_en = 1;
326                 end
327 
328                 else if(INCF) begin
329                     op = 6;
330                     sel_alu = 1;
331                     if(d) begin
332                         ram_en = 1;
333                         sel_bus = 0;
```

```
334                     end
335                     else begin
336                         load_w = 1;
337                     end
338                 end
339                 else if(IORWF) begin
340                     op = 3;
341                     sel_alu = 1;
342                     if(d) begin
343                         ram_en = 1;
344                         sel_bus = 0;
345                     end
346                     else begin
347                         load_w = 1;
348                     end
349                 end
350                 else if(MOVF) begin
351                     op = 5;
352                     sel_alu = 1;
353                     if(d) begin
354                         ram_en = 1;
355                         sel_bus = 0;
356                     end
357                     else begin
358                         load_w = 1;
359                     end
360                 end
361                 else if(MOVWF) begin
362                     sel_bus = 1;
363                     if(addr_port_b)begin
364                         load_port_b = 1;
365                     end
366                     else begin
```

```
367                     ram_en = 1;
368                 end
369             end
370             else if(SUBWF) begin
371                 op = 1;
372                 sel_alu = 1;
373                 if(d) begin
374                     ram_en = 1;
375                     sel_bus = 0;
376                 end
377                 else begin
378                     load_w = 1;
379                 end
380             end
381             else if(XORWF) begin
382                 op = 4;
383                 sel_alu = 1;
384                 if(d) begin
385                     ram_en = 1;
386                     sel_bus = 0;
387                 end
388                 else begin
389                     load_w = 1;
390                 end
391             end
392             else if(BCF)begin
393                 sel_alu = 1;
394                 sel_ram_mux = 1;
395                 op = 5;
396                 sel_bus = 0;
397                 ram_en = 1;
398             end
399             else if(BSF)begin
400                 sel_alu = 1;
```

```
401                     sel_ram_mux = 2;
402                     op = 5;
403                     sel_bus = 0;
404                     ram_en = 1;
405                 end
406                 else if(BTFSC || BTFSS)begin
407                     if( btfsc_btfss_skip_bit )begin
408                         load_pc = 1;
409                         sel_pc = 0;
410                     end
411                 end
412                 else if(DECFSZ)begin
413                     sel_alu = 1;
414                     op = 7;
415                     if(aluout_zero)begin
416                         load_pc = 1;
417                         sel_pc = 0;
418                     end
419 
420                     if(d)begin
421                         ram_en = 1;
422                         sel_bus = 0;
423                     end
424                     else begin
425                         load_w = 1;
426                     end
427                 end
428                 else if(INCFSZ) begin
429                     sel_alu = 1;
430                     op = 6;
431                     if(aluout_zero)begin
432                         load_pc = 1;
433                         sel_pc = 0;
```

```
434                 end
435
436                 if(d)begin
437                     ram_en = 1;
438                     sel_bus = 0;
439                 end
440                 else begin
441                     load_w = 1;
442                 end
443             end
444
445             else if(ASRF) begin
446                 sel_alu = 1;
447                 sel_ram_mux = 0;
448                 op = 4'hA;
449                 if(d)begin
450                     sel_bus = 0;
451                     ram_en = 1;
452                 end
453                 else begin
454                     load_w = 1;
455                 end
456             end
457             if(LSLF) begin
458                 sel_alu = 1;
459                 sel_ram_mux = 0;
460                 op = 4'hB;
461                 if(d)begin
462                     sel_bus = 0;
463                     ram_en = 1;
464                 end
465                 else begin
466                     load_w = 1;
```

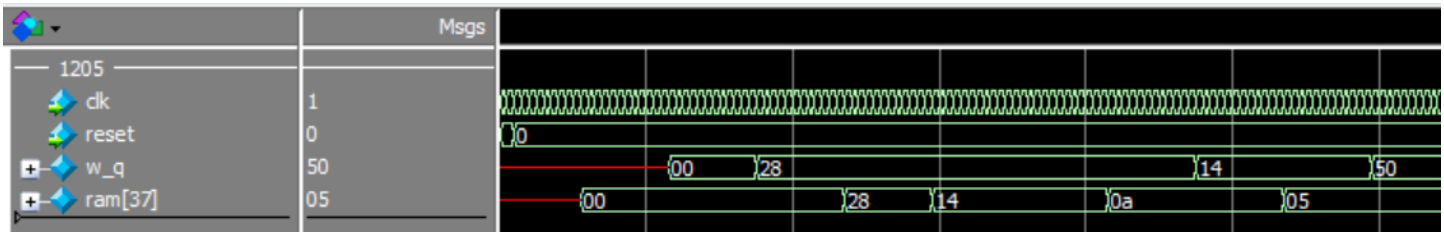
```
467                 end
468             end
469             if(LSRF) begin
470                 sel_alu = 1;
471                 sel_ram_mux = 0;
472                 op = 4'hC;
473                 if(d)begin
474                     sel_bus = 0;
475                     ram_en = 1;
476                 end
477                 else begin
478                     load_w = 1;
479                 end
480             end
481             if(RLF) begin
482                 sel_alu = 1;
483                 sel_ram_mux = 0;
484                 op = 4'hD;
485                 if(d)begin
486                     sel_bus = 0;
487                     ram_en = 1;
488                 end
489                 else begin
490                     load_w = 1;
491                 end
492             end
493             if(RRF) begin
494                 sel_alu = 1;
495                 sel_ram_mux = 0;
496                 op = 4'hE;
497                 if(d)begin
498                     sel_bus = 0;
499                     ram_en = 1;
500                 end
```

```
                    else begin
                        load_w = 1;
                    end
                end
                if(SWAP) begin
                    sel_alu = 1;
                    sel_ram_mux = 0;
                    op = 4'hF;
                    if(d)begin
                        sel_bus = 0;
                        ram_en = 1;
                    end
                    else begin
                        load_w = 1;
                    end
                end
                ns = T5;
            end
            T5: begin
                ns = T6;
            end
            T6: begin
                ns = T1;
            end
        endcase
    end
endmodule
```

```
module testbench;

    logic clk,reset;
    logic [7:0] port_b_out;

    cpu cpu_test(
        .clk(clk),
        .reset(reset),
        .port_b_out(port_b_out)
    );

    always #10 clk = ~clk;
    initial begin
        clk = 0; reset = 1;
        #20 reset = 0;
        #9000 $stop;
    end
endmodule
```

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

add wave -noupdate -divider {1205}

add wave -noupdate -format Literal -radix decimal       /testbench/cpu_test/clk
add wave -noupdate -format Literal -radix decimal       /testbench/cpu_test/reset
add wave -noupdate -format Literal -radix Hexadecimal   /testbench/cpu_test/w_q
add wave -noupdate -format Literal -radix Hexadecimal   /testbench/cpu_test/single_port_ram_128x8_1/ram\[37\]
```

● 模擬結果與結果說明：



能夠依照 Program_Rom 的指令，做出正確的模擬(3028=>MOVLW 28, 如上圖所示 w=28...)

● 結論與心得：

今天又增加了一些位移的指令，與此同時 ALU 也進行了擴充，有了前幾週的經驗，今天很快就找到自己的一個錯誤，但是還剩下一個錯誤，後來經過助教指導，才知道我的邏輯沒有寫錯，只是 wave 檔選擇的線有誤，修正過後就是正確的波形圖，十分感謝助教的指導，今天的錯誤我會謹記在心。