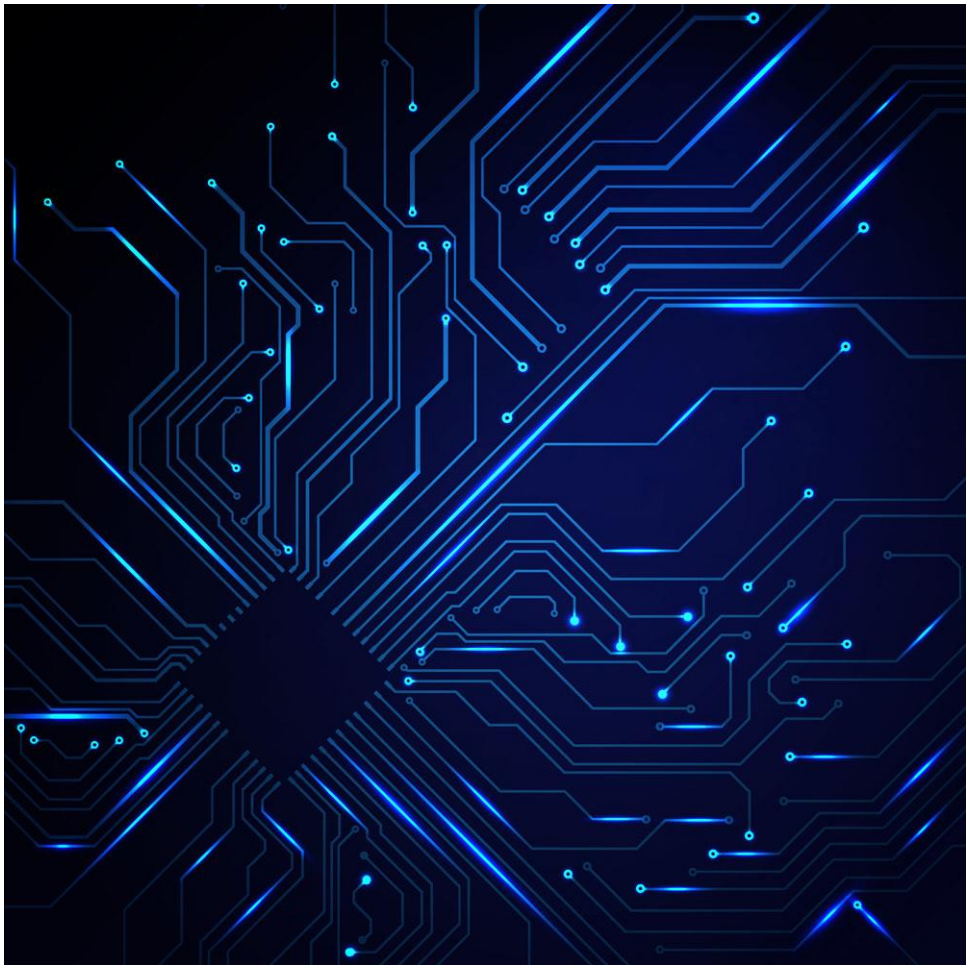


2022/12/12

實驗十二

跳躍指令

姓名：張銀軒　　學號：00957050

班級：資工 3A

E-mail：00957050@mail.ntou.edu.tw

2022/12/12

注意

1. 繳交時一律轉 PDF 檔
2. 繳交期限為
   上完課後
   本周日晚上 11:59 前
3. 一人繳交一份
4. 檔名：學號_HW?.pdf
   檔名請按照作業檔名格式進行填寫
   未依照格式不予批改

一、

● 實驗說明：

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

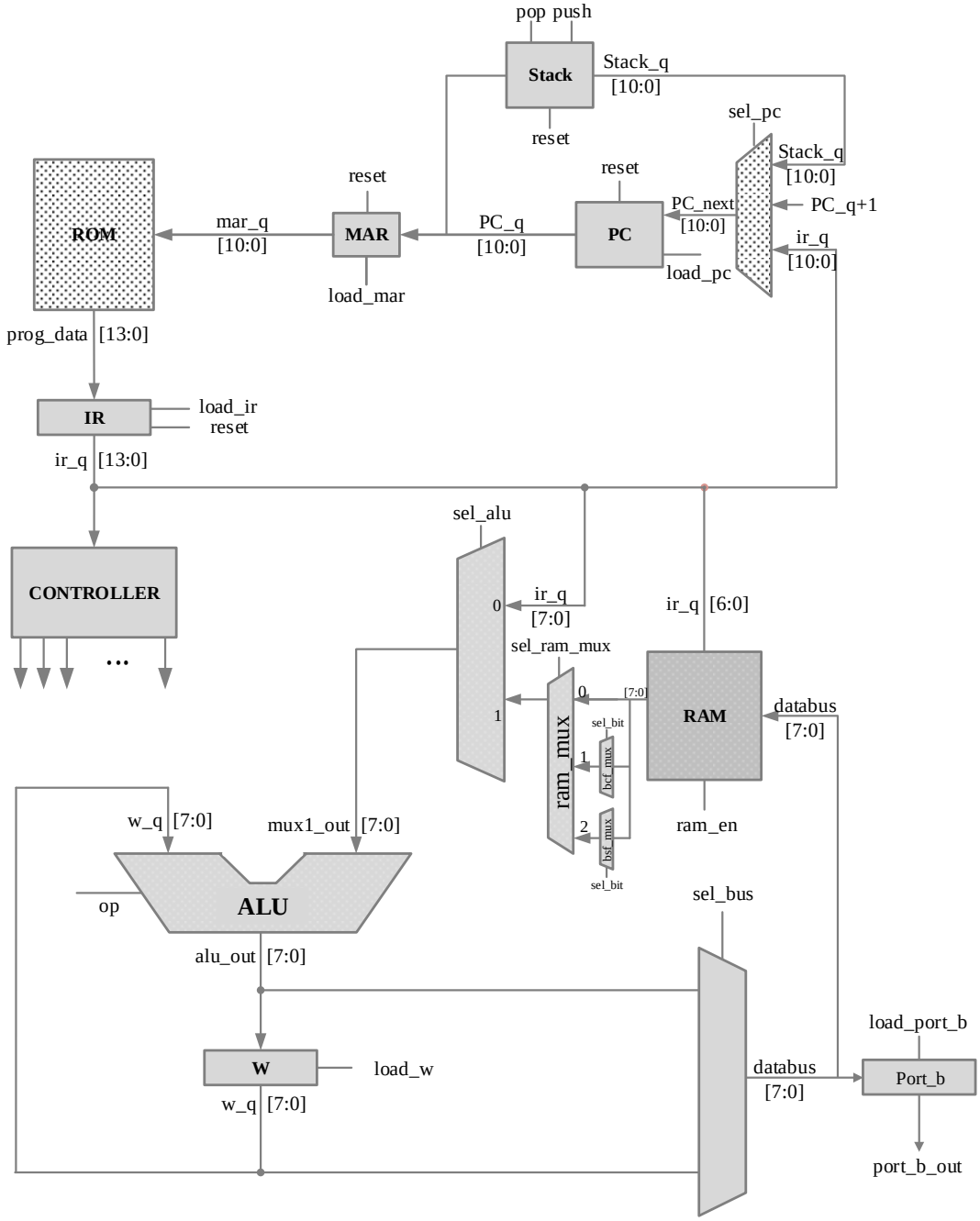
請用 BRA 或 BRW 指令代替 GOTO 指令

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

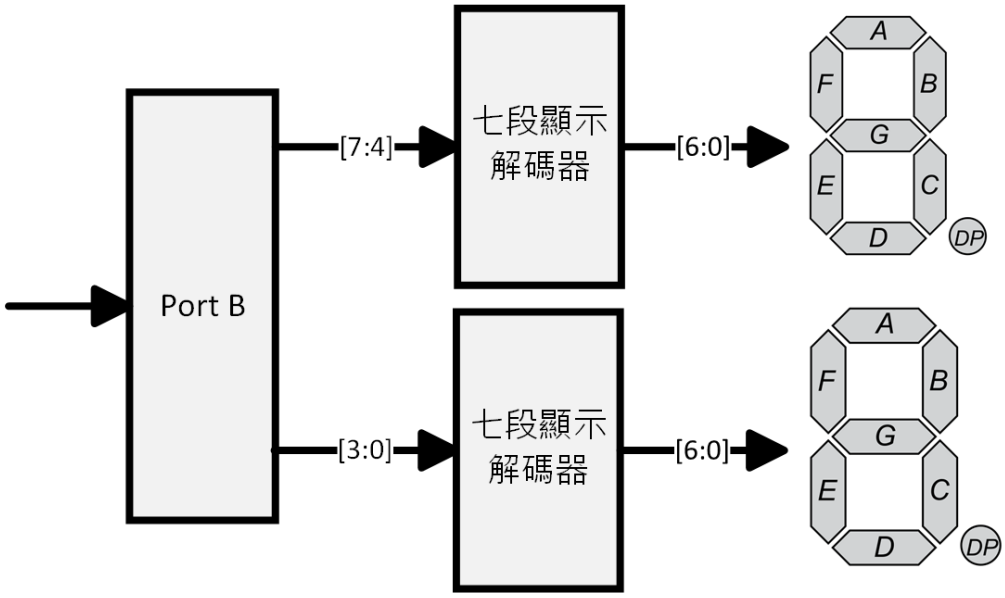
模擬用的組合語言不用加 delay 方便波形觀測

請交 MPLAB 中組合語言截圖、程式碼截圖與波形圖

● 系統硬體架構方塊圖（接線圖）：



圖一、架構圖



圖二、七段顯示器接法

● 系統架構程式碼、測試資料程式碼與程式碼說明

截圖請善用 win+shift+S

C:\...\Program_Rom.asm

```
temp        equ 0x25
templ       equ 0x24
countl      equ h'20'
count2      equ h'21'
count3      equ h'22'

;Program start
            org     0x00    ;reset vector

start       movlw   .60     ; w = 15
            movwf   templ   ; ram[36] = w = 15
            clrf    temp    ; ram[37] <= 0
            clrw            ; w <= 0
loopl       movf    temp,0  ; w = ram[37] = 1
            movwf   PORTB   ; PORTB = w = 1
            movlw   1       ; w <= 1
            addwf   temp,1  ; ram[37] = w+ram[37] = 1+0 = 1
            decfsz  templ,1 ; if(ram[36]-1==0) -> line 23
            bra loopl
            bra start
            end
```

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

    logic [13:0] data;
    always_comb
        begin
            case (Rom_addr_in)
                10'h0 : data = 14'h303C;
                10'h1 : data = 14'h00A4;
                10'h2 : data = 14'h01A5;
                10'h3 : data = 14'h0103;
                10'h4 : data = 14'h0825;
                10'h5 : data = 14'h008D;
                10'h6 : data = 14'h3001;
                10'h7 : data = 14'h07A5;
                10'h8 : data = 14'h0BA4;
                10'h9 : data = 14'h33FA;
                10'ha : data = 14'h33F5;
                10'hb : data = 14'h3400;
                10'hc : data = 14'h3400;
                default: data = 14'h0;
            endcase
        end

     assign Rom_data_out = data;

endmodule
```

```
module sevenSegmentDecoder(
  output logic [6:0] signal,
  input [3:0]num
);
  always_comb
  begin
    case(num)
       4'd0 : signal =7'b100_0000;
       4'd1 : signal =7'b111_1001;
       4'd2 : signal =7'b010_0100;
       4'd3 : signal =7'b011_0000;
       4'd4 : signal =7'b001_1001;
       4'd5 : signal =7'b001_0010;
       4'd6 : signal =7'b000_0010;
       4'd7 : signal =7'b111_1000;
       4'd8 : signal =7'b000_0000;
       4'd9 : signal =7'b001_0000;
      4'd10 : signal =7'b000_1000;
      4'd11 : signal =7'b000_0011;
      4'd12 : signal =7'b100_0110;
      4'd13 : signal =7'b010_0001;
      4'd14 : signal =7'b000_0110;
      4'd15 : signal =7'b000_1110;
  endcase
  end
endmodule
```

```
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 [3:0] stk_index;

    assign stk_index = stk_ptr + 1;
    assign stack_out = stack[stk_ptr];

    always_ff @( posedge clk ) begin
        if(reset) begin
            stk_ptr <= 4'b1111;
        end
        else if(push) begin
            stack[stk_index] <= stack_in;
            stk_ptr <= stk_ptr + 1;
        end
        else if(pop) begin
            stk_ptr <= stk_ptr - 1 ;
        end
    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;
        #110000 $stop;
    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, ram_en, sel_alu, d, sel_bus;
    logic load_port_b;
    logic [1:0] sel_pc;
    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;
    //for Stack
    logic pop, push;
    logic [10:0] stack_out;
    //for seven seg
    logic [6:0] seg0,seg1;

    //Stack
    Stack Stack_1(
        .stack_out(stack_out),
        .stack_in(pc_q),
        .push(push),
        .pop(pop),
        .reset(reset),
        .clk(clk)
    );

    assign w_change = {3'b0, w_q};
    assign k_change = {ir_q[8], ir_q[8], ir_q[8:0]};

    //mux 0 (select next PC address)
    always_comb begin
        if(sel_pc == 4) begin
            pc_next = pc_q + w_change;
        end
        else if(sel_pc == 3) begin
            pc_next = pc_q + k_change;
        end
        else if(sel_pc == 2) begin
            pc_next = stack_out;
        end
        else if(sel_pc == 1) begin
            pc_next = ir_q;
        end
```

```
51          else begin
52              pc_next = pc_q + 1;
53          end
54      end
55
56      //pc
57      always_ff @( posedge clk ) begin
58          if(reset)
59              pc_q <= 0;
60          else if(load_pc)
61              pc_q <= pc_next;
62      end
63
64      //mar
65      always_ff @( posedge clk ) begin
66          if(load_mar)
67              mar_q <= pc_q;
68      end
69
70      //ROM
71      Program_Rom ROM_1(
72          .Rom_addr_in(mar_q),
73          .Rom_data_out(rom_q)
74      );
75
76      //IR
77      always_ff @( posedge clk ) begin
78          if(reset)
79              ir_q <= 0;
80          else if(load_ir)
81              ir_q <= rom_q;
82      end
83
84      //RAM
85      single_port_ram_128x8 single_port_ram_128x8_1(
86          .data(databus),
87          .addr(ir_q[6:0]),
88          .ram_en(ram_en),
89          .clk(clk),
90          .ram_out(ram_out)
91      );
92
93      assign sel_bit = ir_q[9:7];
94
95      //BCF mux
96      always_comb begin
97          case (sel_bit)
98              3'b000: bcf_mux = ram_out & 8'b1111_1110;
99              3'b001: bcf_mux = ram_out & 8'b1111_1101;
100             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
    end

    //mux 1 (select data into ALU)
    always_comb begin
        if(sel_alu) begin
            mux1_out = ram_mux;
        end
        else begin
            mux1_out = ir_q;
        end
    end

    assign d = ir_q[7];
    assign  MOVLW = (ir_q[13:8]==6'h30);
    assign  ADDLW = (ir_q[13:8]==6'h3E);
    assign  IORLW = (ir_q[13:8]==6'h38);
    assign  ANDLW = (ir_q[13:8]==6'h39);
    assign  SUBLW = (ir_q[13:8]==6'h3C);
    assign  XORLW = (ir_q[13:8]==6'h3A);

    assign  ADDWF = (ir_q[13:8]==6'h07);
```

```
151     assign  ANDWF = (ir_q[13:8]==6'h05);
152     assign   CLRF = (ir_q[13:8]==6'h01 && d==1);
153     assign   CLRW = (ir_q[13:4]==10'h010 && ir_q[3:2]==2'h0);
154     assign   COMF = (ir_q[13:8]==6'h09);
155     assign   DECF = (ir_q[13:8]==6'h03);
156     assign   GOTO = (ir_q[13:11]==3'b101);
157
158     assign   INCF = (ir_q[13:8]==6'h0A);
159     assign  IORWF = (ir_q[13:8]==6'h04);
160     assign   MOVF = (ir_q[13:8]==6'h08);
161     assign  MOVWF = (ir_q[13:8]==6'h00 && ir_q[7]==1'b1);
162     assign  SUBWF = (ir_q[13:8]==6'h02);
163     assign  XORWF = (ir_q[13:8]==6'h06);
164
165     assign    BCF = (ir_q[13:12]==2'b01 && ir_q[11:10]==2'b00);
166     assign    BSF = (ir_q[13:12]==2'b01 && ir_q[11:10]==2'b01);
167     assign  BTFSC = (ir_q[13:12]==2'b01 && ir_q[11:10]==2'b10);
168     assign  BTFSS = (ir_q[13:12]==2'b01 && ir_q[11:10]==2'b11);
169     assign DECFSZ = (ir_q[13:8]==6'h0B);
170     assign INCFSZ = (ir_q[13:8]==6'h0F);
171
172     assign btfsc_skip_bit = (ram_out[ir_q[9:7]]==0);
173     assign btfss_skip_bit = (ram_out[ir_q[9:7]]==1);
174     assign btfsc_btfss_skip_bit = (BTFSC&btfsc_skip_bit) |
175                                   (BTFSS&btfss_skip_bit);
176     assign   ASRF = (ir_q[13:8]==6'h37);        // arithmetic right shift
177     assign   LSLF = (ir_q[13:8]==6'h35);        // logical left shift
178     assign   LSRF = (ir_q[13:8]==6'h36);        // logical right shift
179     assign    RLF = (ir_q[13:8]==6'h0D);        // rotate left f
180     assign    RRF = (ir_q[13:8]==6'h0C);        // rotate right f
181     assign   SWAP = (ir_q[13:8]==6'h0E); //{m7, m6,...m4, m3,...m0} => {m3,...m0, m7, m6,...m4}
182
183     assign   CALL = (ir_q[13:12]==2'b10 && ir_q[11]==0);
184     assign RETURN = (ir_q == 0);
185
186     assign    BRA = (ir_q[13:12]==2'b11 && ir_q[11:9]==3'b001);// relative branch
187     assign    BRW = (ir_q[13:0]==6'h000B);      // relative branch with W
188     assign    NOP = (ir_q[13:0]==6'h000B);      // no operation
189
190     //ALU
191     ALU ALU_1(
192         .op(op),
193         .w_q(w_q),
194         .mux1_out(mux1_out),
195         .alu_q(alu_q)
196     );
197     assign aluout_zero = (alu_q == 0);
198
199     //register
200     always_ff @( posedge clk ) begin
```

```
201         if(load_w)
202             w_q <= alu_q;
203     end
204 
205     //mux 2 (select data into RAM)
206     always_comb begin
207         if(sel_bus) begin
208             databus = w_q;
209         end
210         else begin
211             databus = alu_q;
212         end
213     end
214 
215     //Port_b
216     always_ff @( posedge clk ) begin
217         if(reset) begin
218             port_b_out <= 0;
219         end
220         else if(load_port_b) begin
221             port_b_out <= databus;
222         end
223     end
224 
225     logic [3:0] ten,unit;
226     always_comb begin
227         ten = port_b_out / 10;
228         unit = port_b_out - ten * 10;
229     end
230     //seven seg 0
231     sevenSegmentDecoder sevenSegmentDecoder_0(
232         .signal(seg0),
233         .num(ten)
234     );
235 
236     //seven seg 1
237     sevenSegmentDecoder sevenSegmentDecoder_1(
238         .signal(seg1),
239         .num(unit)
240     );
241 
242     assign addr_port_b = (ir_q[6:0] == 7'h0D);
243 
244     //controller
245     parameter T0 = 0;
246     parameter T1 = 1;
247     parameter T2 = 2;
248     parameter T3 = 3;
249     parameter T4 = 4;
250     parameter T5 = 5;
```

```
    parameter T6 = 6;

    always_ff @( posedge clk ) begin
        if(reset) ps <= 0;
        else ps <= ns;
    end

    always_comb begin
        sel_alu = 0;
        sel_pc = 0;
        load_mar = 0;
        load_pc = 0;
        reset_ir = 1;
        load_ir = 0;
        load_w = 0;
        ram_en = 0;
        op =0;
        sel_ram_mux = 0;
        sel_bus = 0;
        load_port_b = 0;
        ns=0;
        //for Stack
        push = 0;
        pop = 0;
        case(ps)
            T0: begin
                ns = T1;
            end
            T1: begin
                load_mar = 1;
                load_pc = 0;
                reset_ir = 0;
                load_ir = 0;
                load_w = 0;
                ns = T2;
            end
            T2: begin
                sel_pc = 0;
                load_mar = 0;
                load_pc = 1;
                reset_ir = 0;
                load_ir = 0;
                load_w = 0;
                ns = T3;
            end
            T3: begin
                load_mar = 0;
                load_pc = 0;
                reset_ir = 0;
                load_ir = 1;
```

```
301                 load_w = 0;
302                 ns = T4;
303             end
304             T4: begin
305                 load_mar = 0;
306                 load_pc = 0;
307                 reset_ir = 0;
308                 load_ir = 0;
309 
310                 if(MOVLW) begin
311                     sel_alu = 0;
312                     op = 5;
313                     load_w = 1;
314                 end
315                 else if(ADDLW) begin
316                     sel_alu = 0;
317                     op = 0;
318                     load_w = 1;
319                 end
320                 else if(IORLW) begin
321                     sel_alu = 0;
322                     op = 3;
323                     load_w = 1;
324                 end
325                 else if(ANDLW) begin
326                     sel_alu = 0;
327                     op = 2;
328                     load_w = 1;
329                 end
330                 else if(SUBLW) begin
331                     sel_alu = 0;
332                     op = 1;
333                     load_w = 1;
```

```
334                 end
335                 else if(XORLW) begin
336                     sel_alu = 0;
337                     op = 4;
338                     load_w = 1;
339                 end
340 
341 
342                 else if(GOTO) begin
343                     sel_pc = 1;
344                     load_pc = 1;
345                 end
346                 else if(ADDWF) begin
347                     op = 0;
348                     sel_alu = 1;
349                     if(d) begin
350                         ram_en = 1;
351                     end
352                     else begin
353                         load_w = 1;
354                     end
355                 end
356                 else if(ANDWF) begin
357                     op = 2;
358                     sel_alu = 1;
359                     if(d) begin
360                         ram_en = 1;
361                     end
362                     else begin
363                         load_w = 1;
364                     end
365                 end
366                 else if(CLRF) begin
```

```
367                     op = 8;
368                     ram_en = 1;
369                 end
370                 else if(CLRW) begin
371                     op = 8;
372                     load_w = 1;
373                 end
374                 else if(COMF) begin
375                     op = 9;
376                     sel_alu = 1;
377                     ram_en = 1;
378                 end
379                 else if(DECF) begin
380                     op = 7;
381                     sel_alu = 1;
382                     ram_en = 1;
383                 end
384 
385                 else if(INCF) begin
386                     op = 6;
387                     sel_alu = 1;
388                     if(d) begin
389                         ram_en = 1;
390                         sel_bus = 0;
391                     end
392                     else begin
393                         load_w = 1;
394                     end
395                 end
396                 else if(IORWF) begin
397                     op = 3;
398                     sel_alu = 1;
399                     if(d) begin
400                         ram_en = 1;
```

```
401                     sel_bus = 0;
402                 end
403                 else begin
404                     load_w = 1;
405                 end
406             end
407             else if(MOVF) begin
408                 op = 5;
409                 sel_alu = 1;
410                 if(d) begin
411                     ram_en = 1;
412                     sel_bus = 0;
413                 end
414                 else begin
415                     load_w = 1;
416                 end
417             end
418             else if(MOVWF) begin
419                 sel_bus = 1;
420                 if(addr_port_b)begin
421                     load_port_b = 1;
422                 end
423                 else begin
424                     ram_en = 1;
425                 end
426             end
427             else if(SUBWF) begin
428                 op = 1;
429                 sel_alu = 1;
430                 if(d) begin
431                     ram_en = 1;
432                     sel_bus = 0;
433                 end
```

```
434                 else begin
435                     load_w = 1;
436                 end
437             end
438             else if(XORWF) begin
439                 op = 4;
440                 sel_alu = 1;
441                 if(d) begin
442                     ram_en = 1;
443                     sel_bus = 0;
444                 end
445                 else begin
446                     load_w = 1;
447                 end
448             end
449             else if(BCF)begin
450                 sel_alu = 1;
451                 sel_ram_mux = 1;
452                 op = 5;
453                 sel_bus = 0;
454                 ram_en = 1;
455             end
456             else if(BSF)begin
457                 sel_alu = 1;
458                 sel_ram_mux = 2;
459                 op = 5;
460                 sel_bus = 0;
461                 ram_en = 1;
462             end
463             else if(BTFSC || BTFSS)begin
464                 if( btfsc_btfss_skip_bit )begin
465                     load_pc = 1;
466                     sel_pc = 0;
```

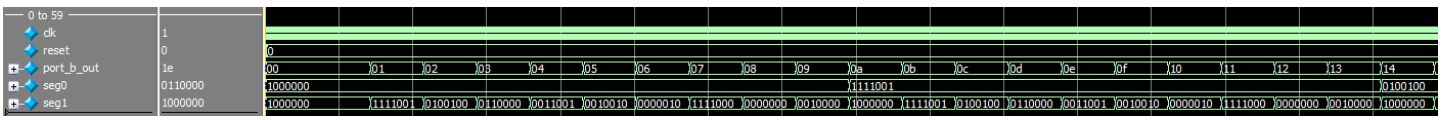
```
467                     end
468                 end
469                 else if(DECFSZ)begin
470                     sel_alu = 1;
471                     op = 7;
472                     if(aluout_zero)begin
473                         load_pc = 1;
474                         sel_pc = 0;
475                     end
476 
477                     if(d)begin
478                         ram_en = 1;
479                         sel_bus = 0;
480                     end
481                     else begin
482                         load_w = 1;
483                     end
484                 end
485                 else if(INCFSZ) begin
486                     sel_alu = 1;
487                     op = 6;
488                     if(aluout_zero)begin
489                         load_pc = 1;
490                         sel_pc = 0;
491                     end
492 
493                     if(d)begin
494                         ram_en = 1;
495                         sel_bus = 0;
496                     end
497                     else begin
498                         load_w = 1;
499                     end
500                 end
```

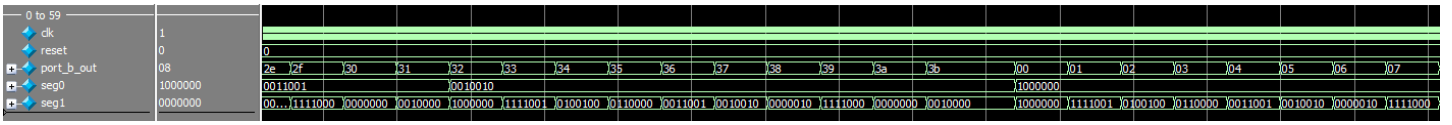
```
501
502                 else if(ASRF) begin
503                     sel_alu = 1;
504                     sel_ram_mux = 0;
505                     op = 4'hA;
506                     if(d)begin
507                         sel_bus = 0;
508                         ram_en = 1;
509                     end
510                     else begin
511                         load_w = 1;
512                     end
513                 end
514                 else if(LSLF) begin
515                     sel_alu = 1;
516                     sel_ram_mux = 0;
517                     op = 4'hB;
518                     if(d)begin
519                         sel_bus = 0;
520                         ram_en = 1;
521                     end
522                     else begin
523                         load_w = 1;
524                     end
525                 end
526                 else if(LSRF) begin
527                     sel_alu = 1;
528                     sel_ram_mux = 0;
529                     op = 4'hC;
530                     if(d)begin
531                         sel_bus = 0;
532                         ram_en = 1;
533                     end
```

```
534                     else begin
535                         load_w = 1;
536                     end
537                 end
538                 else if(RLF) begin
539                     sel_alu = 1;
540                     sel_ram_mux = 0;
541                     op = 4'hD;
542                     if(d)begin
543                         sel_bus = 0;
544                         ram_en = 1;
545                     end
546                     else begin
547                         load_w = 1;
548                     end
549                 end
550                 else if(RRF) begin
551                     sel_alu = 1;
552                     sel_ram_mux = 0;
553                     op = 4'hE;
554                     if(d)begin
555                         sel_bus = 0;
556                         ram_en = 1;
557                     end
558                     else begin
559                         load_w = 1;
560                     end
561                 end
562                 else if(SWAP) begin
563                     sel_alu = 1;
564                     sel_ram_mux = 0;
565                     op = 4'hF;
566                     if(d)begin
```

```
567                         sel_bus = 0;
568                         ram_en = 1;
569                     end
570                     else begin
571                         load_w = 1;
572                     end
573                 end
574
575                 else if(CALL) begin
576                     sel_pc = 1;
577                     load_pc = 1;
578                     push = 1;
579                 end
580                 else if(RETURN) begin
581                     sel_pc = 2;
582                     load_pc = 1;
583                     pop = 1;
584                 end
585                 else if(BRW) begin
586                     load_pc = 1;
587                     sel_pc = 4;
588                 end
589                 else if(BRA) begin
590                     load_pc = 1;
591                     sel_pc = 3;
592                 end
593                 ns = T5;
594             end
595             T5: begin
596                 ns = T6;
597             end
598             T6: begin
599                 ns = T1;
600             end
601         endcase
602     end
603 endmodule
```

● 模擬結果與結果說明：





1. 能從到 0 數到 59，接著再從 0 開始，一直循環

2. 能依照當下的數字輸出十位數字和個位數字的 7 段顯示器訊號

二、

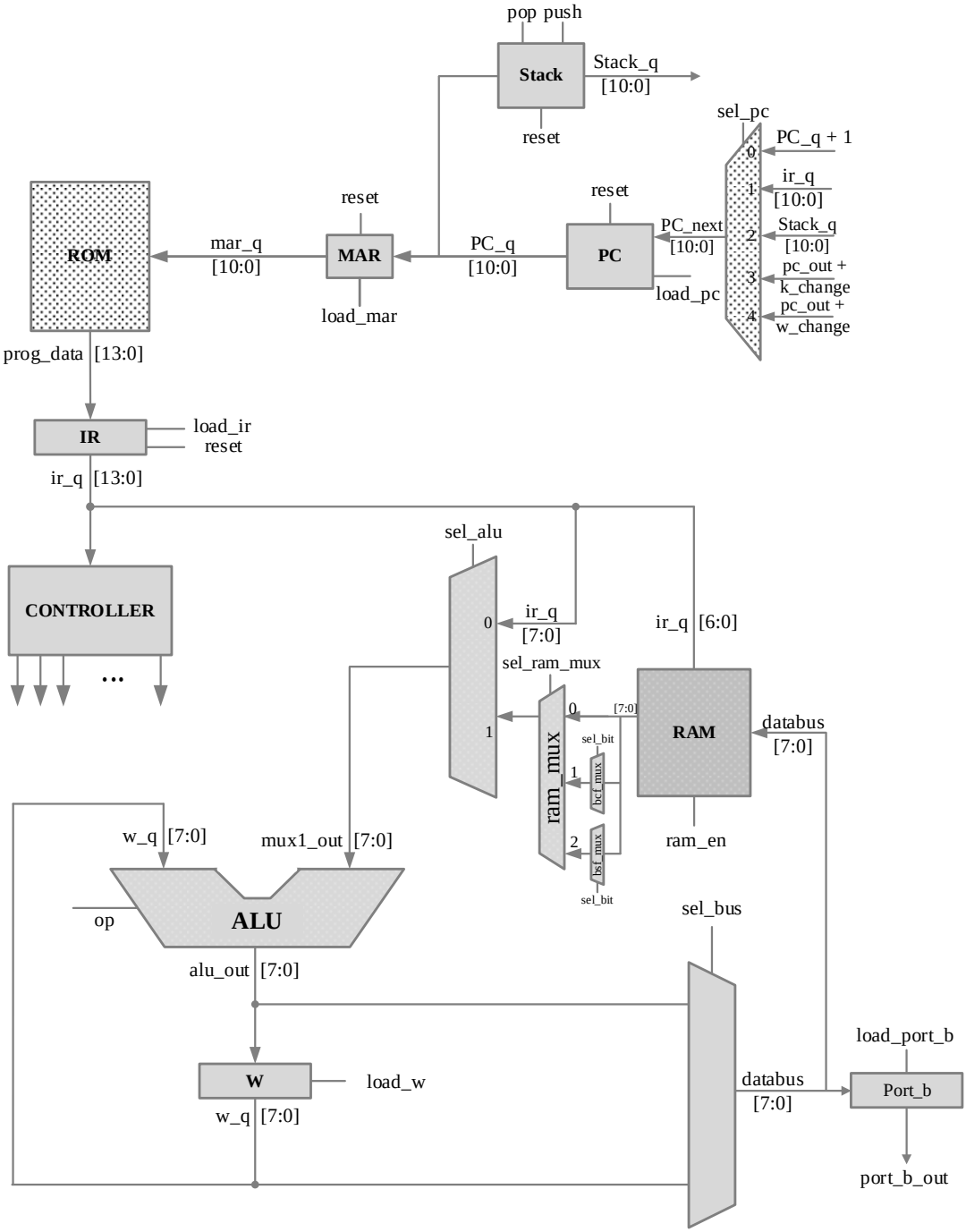
● 實驗說明：

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

模擬用的組合語言不用加 delay 方便波形觀測

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

● 系統硬體架構方塊圖（接線圖）：



架構圖

- **系統架構程式碼、測試資料程式碼與程式碼說明**
  截圖請善用 win+shift+S

- **模擬結果與結果說明：**

- 結論與心得：
  ✏️