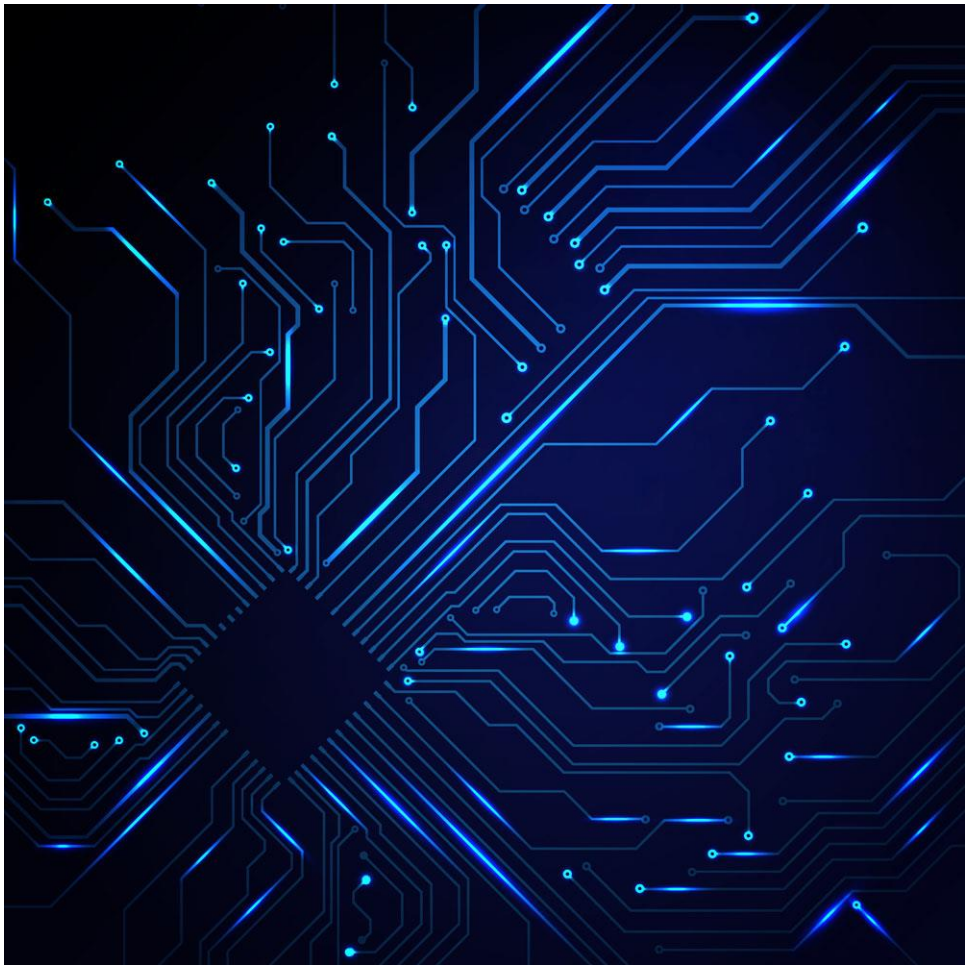


2022/12/19

實驗十三

跳躍指令

姓名：張銀軒　　學號：00957050

班級：資工 3A

E-mail：00957050@mail.ntou.edu.tw

2022/12/19

注意

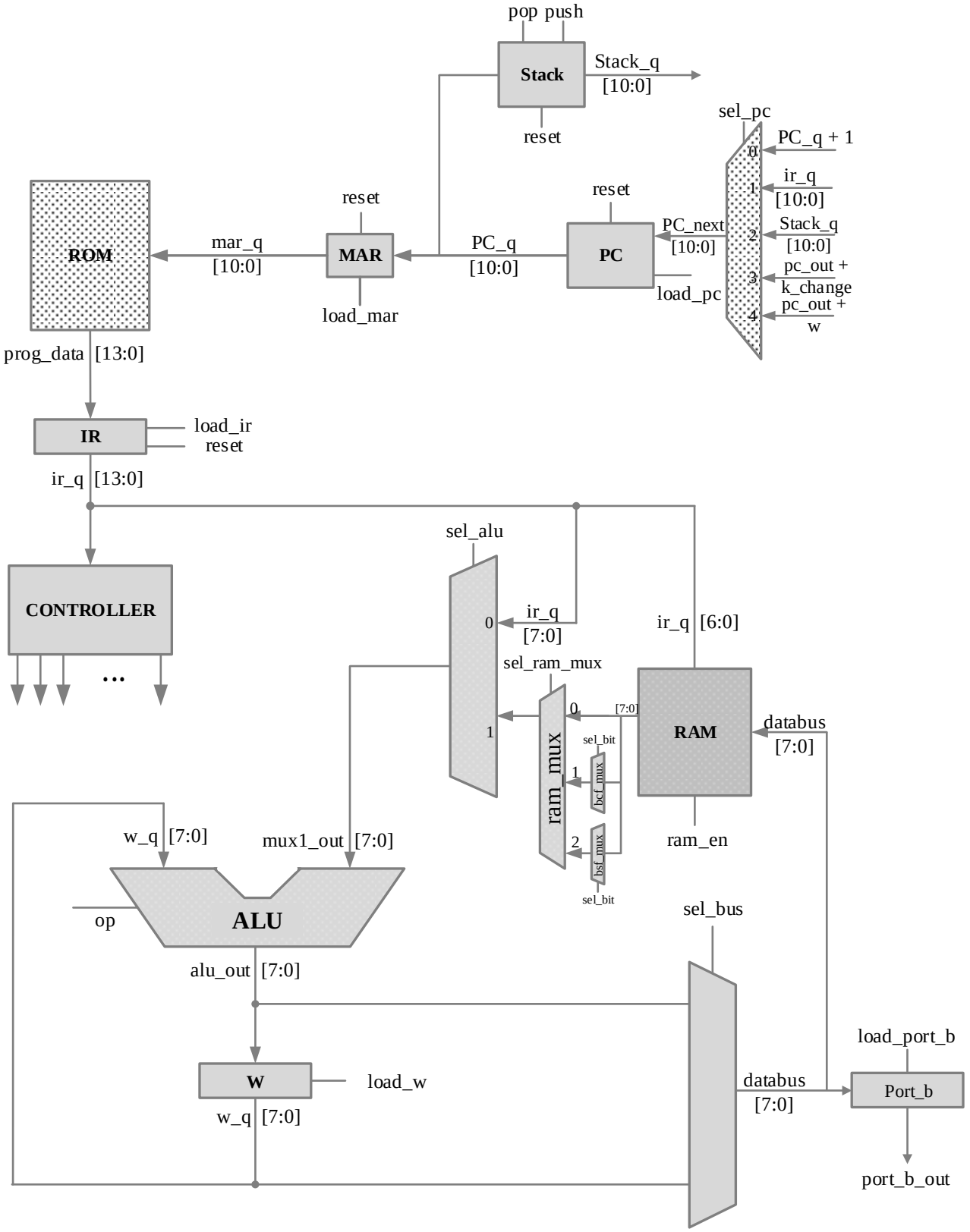
1. 繳交時一律轉 PDF 檔
2. 一人繳交一份
3. 檔名：學號_HW?.pdf
   檔名請按照作業檔名格式進行填寫
   未依照格式不予批改

● 實驗說明：

用 MPLAB 設計一個 Rom，使 0x21 和 0x22 兩個位址的 16 進制分別表示時鐘的時及分，即 0x22(分)的 16 進制會由 0 數到 59 後歸零，每當 0x22(分)歸零 0x21(時)就會加 1，每當做到 23:59 後會全部歸零

請交 MPLAB 專案及程式碼截圖，存時跟分的暫存器請分別設定為 0x21 跟 0x22
請用 BRA 的指令代替 goto 指令去做使用，使用方式一樣

● 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

C:\...\Program_Rom.asm

```
#include    <p16Lf1826.inc>

temp1       equ 0x24    ; 36
temp2       equ 0x23    ; 35
second      equ 0x22    ; 34
minute      equ 0x21    ; 33

;_________Program start
        org 0x00

start1      movlw   .23
            movwf temp2
            clrf minute

start2      movlw   .59
            movwf   temp1
            clrf    second
            clrw
loop        movlw .1
            addwf   second,1
            decfsz temp1,1
            bra loop
            addwf   minute,1
            decfsz  temp2,1
            bra     start2
            bra     start1
            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'h3018;
                10'h1 : data = 14'h00A3;
                10'h2 : data = 14'h01A1;
                10'h3 : data = 14'h303B;
                10'h4 : data = 14'h00A4;
                10'h5 : data = 14'h01A2;
                10'h6 : data = 14'h0103;
                10'h7 : data = 14'h3001;
                10'h8 : data = 14'h07A2;
                10'h9 : data = 14'h0BA4;
                10'ha : data = 14'h33FC;
                10'hb : data = 14'h07A1;
                10'hc : data = 14'h0BA3;
                10'hd : data = 14'h33F5;
                10'he : data = 14'h33F1;
                10'hf : data = 14'h3400;
                10'h10 : data = 14'h3400;
                default: data = 14'h0;
            endcase
        end

     assign Rom_data_out = data;

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 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, k_change;
    logic load_pc, load_mar, load_ir, reset_ir, load_w, ram_en, sel_alu, d, sel_bus;
    logic load_port_b;
    logic [2:0] 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;

    //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} - 1;
    assign k_change = {ir_q[8], ir_q[8], ir_q[8:0]}-1 ;

    //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
        else begin
            pc_next = pc_q + 1;
```

```
51          end
52      end
53
54      //pc
55      always_ff @( posedge clk ) begin
56          if(reset)
57              pc_q <= 0;
58          else if(load_pc)
59              pc_q <= pc_next;
60      end
61
62      //mar
63      always_ff @( posedge clk ) begin
64          if(load_mar)
65              mar_q <= pc_q;
66      end
67
68      //ROM
69      Program_Rom ROM_1(
70          .Rom_addr_in(mar_q),
71          .Rom_data_out(rom_q)
72      );
73
74      //IR
75      always_ff @( posedge clk ) begin
76          if(reset_ir)
77              ir_q <= 0;
78          else if(load_ir)
79              ir_q <= rom_q;
80      end
81
82      //RAM
83      single_port_ram_128x8 single_port_ram_128x8_1(
84          .data(databus),
85          .addr(ir_q[6:0]),
86          .ram_en(ram_en),
87          .clk(clk),
88          .ram_out(ram_out)
89      );
90
91      assign sel_bit = ir_q[9:7];
92
93      //BCF mux
94      always_comb begin
95          case (sel_bit)
96              3'b000: bcf_mux = ram_out & 8'b1111_1110;
97              3'b001: bcf_mux = ram_out & 8'b1111_1101;
98              3'b010: bcf_mux = ram_out & 8'b1111_1011;
99              3'b011: bcf_mux = ram_out & 8'b1111_0111;
100             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);
    assign  ANDWF = (ir_q[13:8]==6'h05);
    assign   CLRF = (ir_q[13:8]==6'h01 && d==1);
```

```
151     assign   CLRW = (ir_q[13:4]==10'h010 && ir_q[3:2]==2'h0);
152     assign   COMF = (ir_q[13:8]==6'h09);
153     assign   DECF = (ir_q[13:8]==6'h03);
154     assign   GOTO = (ir_q[13:11]==3'b101);
155
156     assign   INCF = (ir_q[13:8]==6'h0A);
157     assign  IORWF = (ir_q[13:8]==6'h04);
158     assign   MOVF = (ir_q[13:8]==6'h08);
159     assign  MOVWF = (ir_q[13:8]==6'h00 && ir_q[7]==1'b1);
160     assign  SUBWF = (ir_q[13:8]==6'h02);
161     assign  XORWF = (ir_q[13:8]==6'h06);
162
163     assign    BCF = (ir_q[13:12]==2'b01 && ir_q[11:10]==2'b00);
164     assign    BSF = (ir_q[13:12]==2'b01 && ir_q[11:10]==2'b01);
165     assign  BTFSC = (ir_q[13:12]==2'b01 && ir_q[11:10]==2'b10);
166     assign  BTFSS = (ir_q[13:12]==2'b01 && ir_q[11:10]==2'b11);
167     assign DECFSZ = (ir_q[13:8]==6'h0B);
168     assign INCFSZ = (ir_q[13:8]==6'h0F);
169
170     assign btfsc_skip_bit = (ram_out[ir_q[9:7]]==0);
171     assign btfss_skip_bit = (ram_out[ir_q[9:7]]==1);
172     assign btfsc_btfss_skip_bit = (BTFSC&btfsc_skip_bit) |
173                                   (BTFSS&btfss_skip_bit);
174     assign   ASRF = (ir_q[13:8]==6'h37);        // arithmetic right shift
175     assign   LSLF = (ir_q[13:8]==6'h35);        // logical left shift
176     assign   LSRF = (ir_q[13:8]==6'h36);        // logical right shift
177     assign    RLF = (ir_q[13:8]==6'h0D);        // rotate left f
178     assign    RRF = (ir_q[13:8]==6'h0C);        // rotate right f
179     assign   SWAP = (ir_q[13:8]==6'h0E); //{m7, m6,...m4, m3,...m0} => {m3,...m0, m7, m6,...m4}
180
181     assign   CALL = (ir_q[13:12]==2'b10 && ir_q[11]==0);
182     assign RETURN = (ir_q == 8);
183
184     assign    BRA = (ir_q[13:12]==2'b11 && ir_q[11:9]==3'b001);// relative branch
185     assign    BRW = (ir_q[13:0]==14'h000B);      // relative branch with W
186     assign    NOP = (ir_q[13:0]==14'h0000);      // no operation
187
188     //ALU
189     ALU ALU_1(
190         .op(op),
191         .w_q(w_q),
192         .mux1_out(mux1_out),
193         .alu_q(alu_q)
194     );
195     assign aluout_zero = (alu_q == 0);
196
197     //register
198     always_ff @( posedge clk ) begin
199         if(load_w)
200             w_q <= alu_q;
```

```
201     end
202
203     //mux 2 (select data into RAM)
204     always_comb begin
205         if(sel_bus) begin
206             databus = w_q;
207         end
208         else begin
209             databus = alu_q;
210         end
211     end
212
213     //Port_b
214     always_ff @( posedge clk ) begin
215         if(reset) begin
216             port_b_out <= 0;
217         end
218         else if(load_port_b) begin
219             port_b_out <= databus;
220         end
221     end
222
223     logic [3:0] ten,unit;
224     always_comb begin
225         ten = port_b_out / 10;
226         unit = port_b_out - ten * 10;
227     end
228
229     assign addr_port_b = (ir_q[6:0] == 7'h0D);
230
231     //controller
232     parameter T0 = 0;
233     parameter T1 = 1;
234     parameter T2 = 2;
235     parameter T3 = 3;
236     parameter T4 = 4;
237     parameter T5 = 5;
238     parameter T6 = 6;
239
240     always_ff @( posedge clk ) begin
241         if(reset) ps <= 0;
242         else ps <= ns;
243     end
244
245     always_comb begin
246         sel_alu = 0;
247         sel_pc = 0;
248         load_mar = 0;
249         load_pc = 0;
250         reset_ir = 0;
```

```
        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;
                sel_pc = 0;
                load_pc = 1;
                ns = T2;
            end
            T2: begin
                ns = T3;
            end
            T3: begin
                load_ir = 1;
                ns = T4;
            end
            T4: begin
                load_mar = 1;
                sel_pc = 2'b00;
                load_pc = 1;
                if(GOTO) begin//to skip the following instruction in T4
                    load_mar = 1;
                end
                else if(MOVLW) begin
                    sel_alu = 0;
                    op = 5;
                    load_w = 1;
                end
                else if(ADDLW) begin
                    sel_alu = 0;
                    op = 0;
                    load_w = 1;
                end
                else if(IORLW) begin
                    sel_alu = 0;
                    op = 3;
                    load_w = 1;
                end
```

```
301                 else if(ANDLW) begin
302                     sel_alu = 0;
303                     op = 2;
304                     load_w = 1;
305                 end
306                 else if(SUBLW) begin
307                     sel_alu = 0;
308                     op = 1;
309                     load_w = 1;
310                 end
311                 else if(XORLW) begin
312                     sel_alu = 0;
313                     op = 4;
314                     load_w = 1;
315                 end
316 
317                 else if(ADDWF) begin
318                     op = 0;
319                     sel_alu = 1;
320                     if(d) begin
321                         ram_en = 1;
322                     end
323                     else begin
324                         load_w = 1;
325                     end
326                 end
327                 else if(ANDWF) begin
328                     op = 2;
329                     sel_alu = 1;
330                     if(d) begin
331                         ram_en = 1;
332                     end
333                     else begin
334                         load_w = 1;
335                     end
336                 end
337                 else if(CLRF) begin
338                     op = 8;
339                     ram_en = 1;
340                 end
341                 else if(CLRW) begin
342                     op = 8;
343                     load_w = 1;
344                 end
345                 else if(COMF) begin
346                     op = 9;
347                     sel_alu = 1;
348                     ram_en = 1;
349                 end
350                 else if(DECF) begin
351                     op = 7;
352                     sel_alu = 1;
353                     ram_en = 1;
354                 end
355 
356                 else if(INCF) begin
357                     op = 6;
358                     sel_alu = 1;
359                     if(d) begin
360                         ram_en = 1;
361                         sel_bus = 0;
362                     end
363                     else begin
364                         load_w = 1;
365                     end
366                 end
367                 else if(IORWF) begin
368                     op = 3;
369                     sel_alu = 1;
370                     if(d) begin
371                         ram_en = 1;
372                         sel_bus = 0;
373                     end
374                     else begin
375                         load_w = 1;
376                     end
377                 end
378                 else if(MOVF) begin
379                     op = 5;
380                     sel_alu = 1;
381                     if(d) begin
382                         ram_en = 1;
383                         sel_bus = 0;
384                     end
385                     else begin
386                         load_w = 1;
387                     end
388                 end
389                 else if(MOVWF) begin
390                     sel_bus = 1;
391                     if(addr_port_b)begin
392                         load_port_b = 1;
393                     end
394                     else begin
395                         ram_en = 1;
396                     end
397                 end
398                 else if(SUBWF) begin
399                     op = 1;
400                     sel_alu = 1;
```

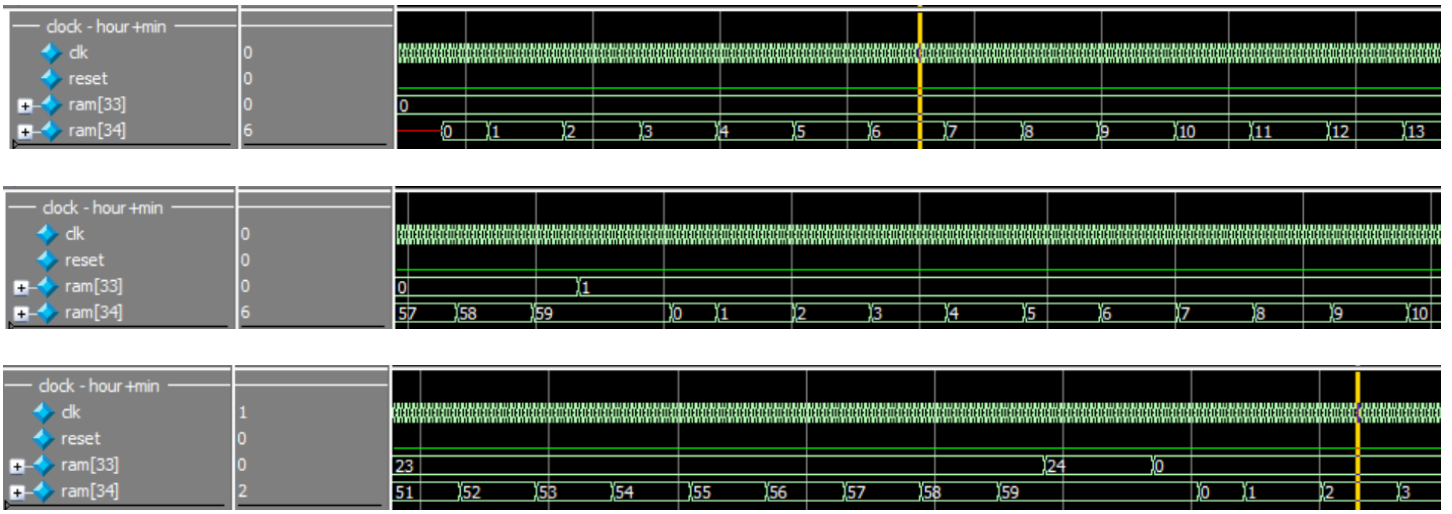
```
                if(d) begin
                    ram_en = 1;
                    sel_bus = 0;
                end
                else begin
                    load_w = 1;
                end
            end
            else if(XORWF) begin
                op = 4;
                sel_alu = 1;
                if(d) begin
                    ram_en = 1;
                    sel_bus = 0;
                end
                else begin
                    load_w = 1;
                end
            end
            else if(BCF)begin
                sel_alu = 1;
                sel_ram_mux = 1;
                op = 5;
                sel_bus = 0;
                ram_en = 1;
            end
            else if(BSF)begin
                sel_alu = 1;
                sel_ram_mux = 2;
                op = 5;
                sel_bus = 0;
                ram_en = 1;
            end
```

```

        else if(ASRF) begin
            sel_alu = 1;
            sel_ram_mux = 0;
            op = 4'hA;
            if(d)begin
                sel_bus = 0;
                ram_en = 1;
            end
            else begin
                load_w = 1;
            end
        end
        else if(LSLF) begin
            sel_alu = 1;
            sel_ram_mux = 0;
            op = 4'hB;
            if(d)begin
                sel_bus = 0;
                ram_en = 1;
            end
            else begin
                load_w = 1;
            end
        end
        else if(LSRF) begin
            sel_alu = 1;
            sel_ram_mux = 0;
            op = 4'hC;
            if(d)begin
                sel_bus = 0;
                ram_en = 1;
            end
```

```
                else begin
                    load_w = 1;
                end
            end
            else if(RLF) begin
                sel_alu = 1;
                sel_ram_mux = 0;
                op = 4'hD;
                if(d)begin
                    sel_bus = 0;
                    ram_en = 1;
                end
                else begin
                    load_w = 1;
                end
            end
            else if(RRF) begin
                sel_alu = 1;
                sel_ram_mux = 0;
                op = 4'hE;
                if(d)begin
                    sel_bus = 0;
                    ram_en = 1;
                end
                else begin
                    load_w = 1;
                end
            end
            else 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

            else if(CALL) begin
                push = 1;
            end
            ns = T5;
        end
        T5: begin
            if(GOTO) begin
                sel_pc = 2'b01;
                load_pc = 1;
            end
            else if(CALL)begin
                sel_pc = 2'b01;
                load_pc = 1;
            end
            else if(RETURN) begin
                pop = 1;
                sel_pc = 2'b10;
                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;
```

```
534                 end
535                 ns = T6;
536             end
537             T6: begin
538                 load_ir = 1;
539                 if(GOTO || CALL || RETURN || BRA || BRW) begin
540                     reset_ir = 1;
541                 end
542                 else if(DECFSZ)begin
543                     op = 7;
544                     sel_alu = 1;
545                     if(aluout_zero)begin
546                         reset_ir = 1;
547                     end
548                     if(d)begin
549                         sel_bus = 0;
550                         ram_en = 1;
551                     end
552                     else begin
553                         load_w = 1;
554                     end
555                 end
556                 else if(INCFSZ) begin
557                     op = 6;
558                     sel_alu = 1;
559                     if(aluout_zero)begin
560                         reset_ir = 1;
561                     end
562                     if(d)begin
563                         sel_bus = 0;
564                         ram_en = 1;
565                     end
566                     else begin
567                                 load_w = 1;
568                             end
569                         end
570                         else if(BTFSC || BTFSS)begin
571                             if( btfsc_btfss_skip_bit )begin
572                                 reset_ir = 1;
573                             end
574                         end
575                         ns = T4;
576                     end
577             endcase
578         end
579     endmodule
```

● 模擬結果與結果說明：



ram[33]: 小時，ram[34]: 分鐘，但是因為是一個一個指令執行的關係，59 分之後下一個狀態只能先把小時+1 再把分鐘歸 0。23:59 之後，因為我在組合語言裡面是先讓小時+1，再去做判斷，所以會有短暫的時間出現 24:59 以及 00:59，接著就能會到正常的 00:00

● 結論與心得：

這次的作業跟上次作業的分鐘和秒很像，只需要修改數字的範圍，就能實現，不過過程中還是發生組合語言寫錯，修改過後就沒問題了。