

2022/11/28

實驗十

條件跳躍指令

姓名：張銀軒 學號：00957050

班級：資工 3A

E-mail：00957050@mail.ntou.edu.tw

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

● 實驗說明：

1. 如圖所示，設計一個架構實現條件跳躍指令

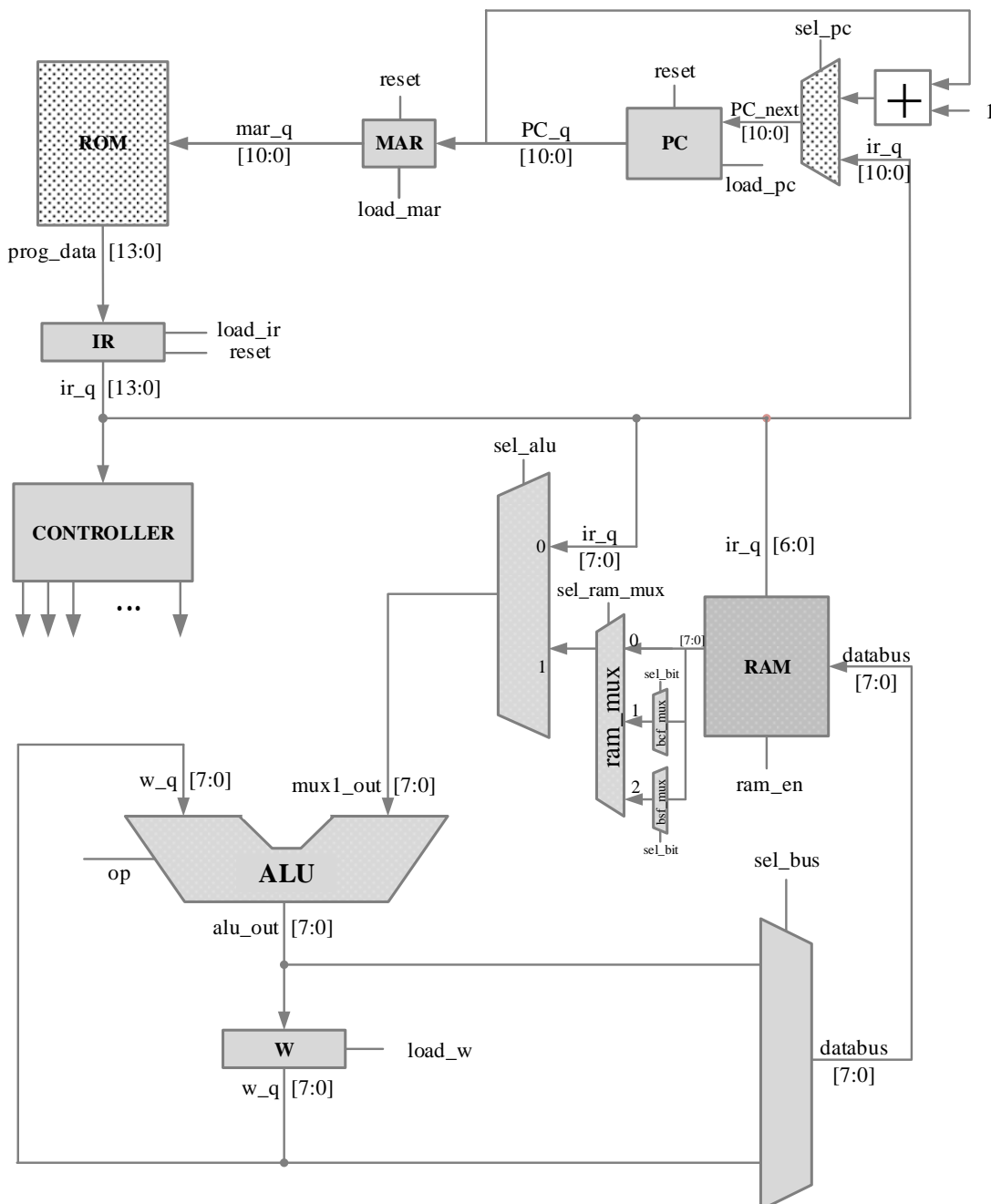
2. 輸入：clk, reset

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

```

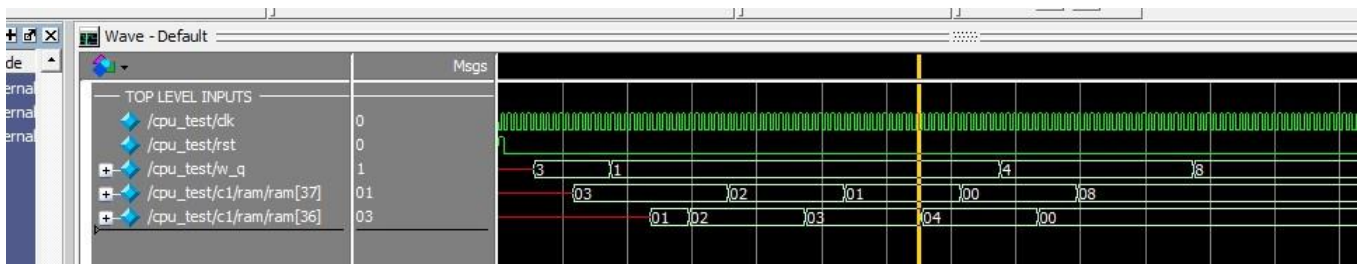
#include    <pl6Lf1826.inc>      ; Include file locate at defu
;

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

```

1  module Program_Rom(
2      output logic [13:0] Rom_data_out,
3      input [10:0] Rom_addr_in
4  );
5
6      logic [13:0] data;
7      always_comb
8          begin
9              case (Rom_addr_in)
10                 10'h0 : data = 14'h3003;
11                 10'h1 : data = 14'h00A5;
12                 10'h2 : data = 14'h3001;
13                 10'h3 : data = 14'h00A4;
14                 10'h4 : data = 14'h0AA4;
15                 10'h5 : data = 14'h0BA5;
16                 10'h6 : data = 14'h2804;
17                 10'h7 : data = 14'h0824;
18                 10'h8 : data = 14'h1124;
19                 10'h9 : data = 14'h15A5;
20                 10'ha : data = 14'h19A5;
21                 10'hb : data = 14'h1DA5;
22                 10'hc : data = 14'h0824;
23                 10'hd : data = 14'h0825;
24                 10'he : data = 14'h280E;
25                 10'hf : data = 14'h3400;
26                 10'h10 : data = 14'h3400;
27                 default: data = 14'h0;
28             endcase
29         end
30
31         assign Rom_data_out = data;
32
33 endmodule
34

```

```

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] ram_out
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 ram_out = ram[addr];
24 endmodule
25
26

```



```
1  module ALU (  
2      input [3:0] op,  
3      input [7:0] w_q, mux1_out,  
4      output logic [7:0] alu_q  
5  );  
6  always_comb  
7  begin  
8      case(op)  
9          0: alu_q = mux1_out + w_q;  
10         1: alu_q = mux1_out - w_q;  
11         2: alu_q = mux1_out & w_q;  
12         3: alu_q = mux1_out | w_q;  
13         4: alu_q = mux1_out ^ w_q; //XOR  
14         5: alu_q = mux1_out;  
15         6: alu_q = mux1_out + 1;  
16         7: alu_q = mux1_out - 1;  
17         8: alu_q = 0;  
18         9: alu_q = ~mux1_out;  
19         default: alu_q = mux1_out + w_q;  
20     endcase  
21 end  
22  
23 endmodule
```

```

1 module cpu (
2     input clk,
3     input reset,
4     output logic [7:0] w_q
5 );
6     logic [13:0] rom_q, ir_q;
7     logic [10:0] pc_next, pc_q, mar_q;
8     logic load_pc, load_mar, load_ir, reset_ir, load_w, sel_pc, ram_en, sel_alu, d, sel_bus;
9     logic [3:0] ps,ns;
10    logic [7:0] alu_q, ram_out, mux1_out, databus, bcf_mux, bsf_mux, ram_mux;
11    logic [3:0] op;
12    logic [5:0] opcode;
13    logic [2:0] sel_bit;
14    logic [1:0] sel_ram_mux;
15    //mux 0 (select next PC address)
16    always_comb begin
17        if(sel_pc) begin
18            pc_next = ir_q;
19        end
20        else begin
21            pc_next = pc_q + 1;
22        end
23    end
24
25    //pc
26    always_ff @(posedge clk) begin
27        if(reset)
28            pc_q <= 0;
29        else if(load_pc)
30            pc_q <= pc_next;
31    end
32
33    //mar
34    always_ff @(posedge clk) begin
35        if(load_mar)
36            mar_q <= pc_q;
37    end
38
39    //ROM
40    Program_Rom ROM_1(
41        .Rom_addr_in(mar_q),
42        .Rom_data_out(rom_q)
43    );
44
45    //IR
46    always_ff @(posedge clk) begin
47        if(reset)
48            ir_q <= 0;
49        else if(load_ir)
50            ir_q <= rom_q;

```

```

51     end
52
53     //RAM
54     single_port_ram_128x8 single_port_ram_128x8_1(
55         .data(databus),
56         .addr(ir_q[6:0]),
57         .ram_en(ram_en),
58         .clk(clk),
59         .ram_out(ram_out)
60     );
61
62     assign sel_bit = ir_q[9:7];
63
64     //BCF mux
65     always_comb begin
66         case (sel_bit)
67             3'b000: bcf_mux = ram_out & 8'b1111_1110;
68             3'b001: bcf_mux = ram_out & 8'b1111_1101;
69             3'b010: bcf_mux = ram_out & 8'b1111_1011;
70             3'b011: bcf_mux = ram_out & 8'b1111_0111;
71             3'b100: bcf_mux = ram_out & 8'b1110_1111;
72             3'b101: bcf_mux = ram_out & 8'b1101_1111;
73             3'b110: bcf_mux = ram_out & 8'b1011_1111;
74             3'b111: bcf_mux = ram_out & 8'b0111_1111;
75         endcase
76     end
77
78     //BSF mux
79     always_comb begin
80         case (sel_bit)
81             3'b000: bsf_mux = ram_out | 8'b0000_0001;
82             3'b001: bsf_mux = ram_out | 8'b0000_0010;
83             3'b010: bsf_mux = ram_out | 8'b0000_0100;
84             3'b011: bsf_mux = ram_out | 8'b0000_1000;
85             3'b100: bsf_mux = ram_out | 8'b0001_0000;
86             3'b101: bsf_mux = ram_out | 8'b0010_0000;
87             3'b110: bsf_mux = ram_out | 8'b0100_0000;
88             3'b111: bsf_mux = ram_out | 8'b1000_0000;
89         endcase
90     end
91
92     //ram mux (select data into mux1)
93     always_comb begin
94         case (sel_ram_mux)
95             0: ram_mux = ram_out;
96             1: ram_mux = bcf_mux;
97             2: ram_mux = bsf_mux;
98         endcase
99     end
100

```

```

101 //mux 1 (select data into ALU)
102 always_comb begin
103     if(sel_alu) begin
104         mux1_out = ram_mux;
105     end
106     else begin
107         mux1_out = ir_q;
108     end
109 end
110
111 assign d = ir_q[7];
112 assign MOVLW = (ir_q[13:8]==6'h30);
113 assign ADDLW = (ir_q[13:8]==6'h3E);
114 assign IORLW = (ir_q[13:8]==6'h38);
115 assign ANDLW = (ir_q[13:8]==6'h39);
116 assign SUBLW = (ir_q[13:8]==6'h3C);
117 assign XORLW = (ir_q[13:8]==6'h3A);
118
119 assign ADDWF = (ir_q[13:8]==6'h07);
120 assign ANDWF = (ir_q[13:8]==6'h05);
121 assign CLRF = (ir_q[13:8]==6'h01 && d==1);
122 assign CLRW = (ir_q[13:4]==10'h010 && ir_q[3:2]==2'h0);
123 assign COMF = (ir_q[13:8]==6'h09);
124 assign DECF = (ir_q[13:8]==6'h03);
125 assign GOTO = (ir_q[13:11]==3'b101);
126
127 assign INCF = (ir_q[13:8]==6'h0A);
128 assign IORWF = (ir_q[13:8]==6'h04);
129 assign MOVF = (ir_q[13:8]==6'h08);
130 assign MOVWF = (ir_q[13:8]==6'h00 && ir_q[7]==1'b1);
131 assign SUBWF = (ir_q[13:8]==6'h02);
132 assign XORWF = (ir_q[13:8]==6'h06);
133
134 assign BCF = (ir_q[13:12]==2'b01 && ir_q[11:10]==2'b00);
135 assign BSF = (ir_q[13:12]==2'b01 && ir_q[11:10]==2'b01);
136 assign BTFSC = (ir_q[13:12]==2'b01 && ir_q[11:10]==2'b10);
137 assign BTFSS = (ir_q[13:12]==2'b01 && ir_q[11:10]==2'b11);
138 assign DECFSZ = (ir_q[13:8]==6'h0B);
139 assign INCFSZ = (ir_q[13:8]==6'h0F);
140
141 assign btfsc_skip_bit = (ram_out[ir_q[9:7]]==0);
142 assign btfss_skip_bit = (ram_out[ir_q[9:7]]==1);
143 assign btfsc_btfss_skip_bit = (BTFSC&btfsc_skip_bit) |
144                                (BTFSS&btfss_skip_bit);
145 //ALU
146 ALU ALU_1(
147     .op(op),
148     .w_q(w_q),
149     .mux1_out(mux1_out),
150     .alu_q(alu_q)

```



```

151     );
152     assign aluout_zero = (alu_q == 0);
153
154     //register
155     always_ff @( posedge clk ) begin
156         if(load_w)
157             w_q <= alu_q;
158     end
159
160     //mux 2 (select data into RAM)
161     always_comb begin
162         if(sel_bus) begin
163             databus = w_q;
164         end
165         else begin
166             databus = alu_q;
167         end
168     end
169
170     //controller
171     parameter T0 = 0;
172     parameter T1 = 1;
173     parameter T2 = 2;
174     parameter T3 = 3;
175     parameter T4 = 4;
176     parameter T5 = 5;
177     parameter T6 = 6;
178
179     always_ff @( posedge clk ) begin
180         if(reset) ps <= 0;
181         else ps <= ns;
182     end
183
184     always_comb begin
185         sel_alu = 0;
186         sel_pc = 0;
187         load_mar = 0;
188         load_pc = 0;
189         reset_ir = 1;
190         load_ir = 0;
191         load_w = 0;
192         ram_en=0;
193         op =0;
194         sel_ram_mux = 0;
195         sel_bus = 0;
196         ns=0;
197         case(ps)
198             T0: begin
199                 ns = T1;
200             end

```

```

201         T1: begin
202             load_mar = 1;
203             load_pc = 0;
204             reset_ir = 0;
205             load_ir = 0;
206             load_w = 0;
207             ns = T2;
208         end
209         T2: begin
210             sel_pc = 0;
211             load_mar = 0;
212             load_pc = 1;
213             reset_ir = 0;
214             load_ir = 0;
215             load_w = 0;
216             ns = T3;
217         end
218         T3: begin
219             load_mar = 0;
220             load_pc = 0;
221             reset_ir = 0;
222             load_ir = 1;
223             load_w = 0;
224             ns = T4;
225         end
226         T4: begin
227             load_mar = 0;
228             load_pc = 0;
229             reset_ir = 0;
230             load_ir = 0;
231
232             if(MOVLW) begin
233                 sel_alu = 0;
234                 op = 5;
235                 load_w = 1;
236             end
237             else if(ADDLW) begin
238                 sel_alu = 0;
239                 op = 0;
240                 load_w = 1;
241             end
242             else if(IORLW) begin
243                 sel_alu = 0;
244                 op = 3;
245                 load_w = 1;
246             end
247             else if(ANDLW) begin
248                 sel_alu = 0;
249                 op = 2;
250                 load_w = 1;

```

```
251         end
252     else if(SUBLW) begin
253         sel_alu = 0;
254         op = 1;
255         load_w = 1;
256     end
257     else if(XORLW) begin
258         sel_alu = 0;
259         op = 4;
260         load_w = 1;
261     end
262
263
264     else if(GOTO) begin
265         sel_pc = 1;
266         load_pc = 1;
267     end
268     else if(ADDWF) begin
269         op = 0;
270         sel_alu = 1;
271         if(d) begin
272             ram_en = 1;
273         end
274         else begin
275             load_w = 1;
276         end
277     end
278     else if(ANDWF) begin
279         op = 2;
280         sel_alu = 1;
281         if(d) begin
282             ram_en = 1;
283         end
284         else begin
285             load_w = 1;
286         end
287     end
288     else if(CLRF) begin
289         op = 8;
290         ram_en = 1;
291     end
292     else if(CLRW) begin
293         op = 8;
294         load_w = 1;
295     end
296     else if(COMF) begin
297         op = 9;
298         sel_alu = 1;
299         ram_en = 1;
300     end
```



```
1         else if(DECF) begin
2             op = 7;
3             sel_alu = 1;
4             ram_en = 1;
5         end
6
7         else if(INCF) begin
8             op = 6;
9             sel_alu = 1;
10            if(d) begin
11                ram_en = 1;
12                sel_bus = 0;
13            end
14            else begin
15                load_w = 1;
16            end
17        end
18        else if(IORWF) begin
19            op = 3;
20            sel_alu = 1;
21            if(d) begin
22                ram_en = 1;
23                sel_bus = 0;
24            end
25            else begin
26                load_w = 1;
27            end
28        end
29        else if(MOVF) begin
30            op = 5;
31            sel_alu = 1;
32            if(d) begin
33                ram_en = 1;
34                sel_bus = 0;
35            end
36            else begin
37                load_w = 1;
38            end
39        end
40        else if(MOVWF) begin
41            ram_en = 1;
42            sel_bus = 1;
43        end
44        else if(SUBWF) begin
45            op = 1;
46            sel_alu = 1;
47            if(d) begin
48                ram_en = 1;
49                sel_bus = 0;
50            end
```

```

51         else begin
52             load_w = 1;
53         end
54     end
55     else if(XORWF) begin
56         op = 4;
57         sel_alu = 1;
58         if(d) begin
59             ram_en = 1;
60             sel_bus = 0;
61         end
62         else begin
63             load_w = 1;
64         end
65     end
66     else if(BCF)begin
67         sel_alu = 1;
68         sel_ram_mux = 1;
69         op = 5;
70         sel_bus = 0;
71         ram_en = 1;
72     end
73     else if(BSF)begin
74         sel_alu = 1;
75         sel_ram_mux = 2;
76         op = 5;
77         sel_bus = 0;
78         ram_en = 1;
79     end
80     else if(BTFSC || BTFSS)begin
81         if( btfsc_btfss_skip_bit )begin
82             load_pc = 1;
83             sel_pc = 0;
84         end
85     end
86     else if(DECFSZ)begin
87         sel_alu = 1;
88         op = 7;
89         if(aluout_zero)begin
90             load_pc = 1;
91             sel_pc = 0;
92         end
93     end
94     if(d)begin
95         ram_en = 1;
96         sel_bus = 0;
97     end
98     else begin
99         load_w = 1;
100    end

```

```
101         end
102         else if(INCFSZ) begin
103             sel_alu = 1;
104             op = 6;
105             if(aluout_zero)begin
106                 load_pc = 1;
107                 sel_pc = 0;
108             end
109
110             if(d)begin
111                 ram_en = 1;
112                 sel_bus = 0;
113             end
114             else begin
115                 load_w = 1;
116             end
117         end
118         ns = T5;
119     end
120     T5: begin
121         ns = T6;
122     end
123     T6: begin
124         ns = T1;
125     end
126 endcase
127 end
128 endmodule
```

```

1 module testbench;
2
3     logic clk,reset;
4     logic [7:0] w_q;
5
6     cpu cpu_test(
7         .clk(clk),
8         .reset(reset),
9         .w_q(w_q)
10    );
11
12    always #10 clk = ~clk;
13    initial begin
14        clk = 0; reset = 1;
15        #20 reset = 0;
16        #3200 $stop;
17    end
18 endmodule

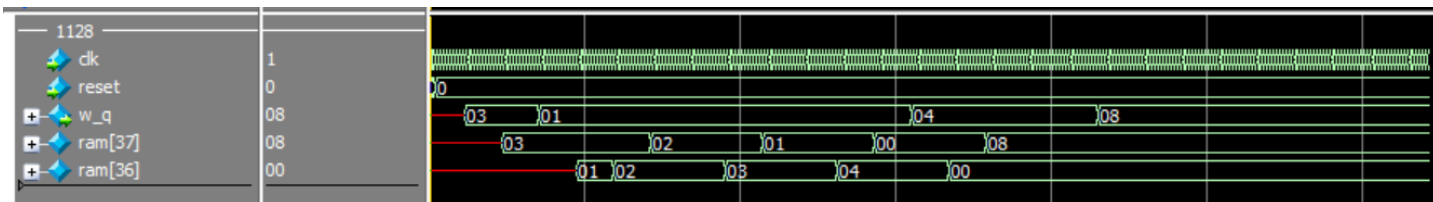
```

```

1 onerror {resume}
2 quietly WaveActivateNextPane {} 0
3
4 add wave -noupdate -divider {1128}
5
6 add wave -noupdate -format Literal -radix decimal    /testbench/cpu_test/clk
7 add wave -noupdate -format Literal -radix decimal    /testbench/cpu_test/reset
8 add wave -noupdate -format Literal -radix Hexadecimal /testbench/cpu_test/w_q
9 add wave -noupdate -format Literal -radix Hexadecimal /testbench/cpu_test/single_port_ram_128x8_1/ram\[37\]
10 add wave -noupdate -format Literal -radix Hexadecimal /testbench/cpu_test/single_port_ram_128x8_1/ram\[36\]

```

● 模擬結果與結果說明：



能夠成功執行從組合語言轉譯過來的 program_rom 指令(movlw 03=>w=3.....)

● 結論與心得：

這次的作業增加了很多指令，硬體架構也有蠻大的變化，幸好能夠以舊的程式碼進行修改，讓我不至於從頭去思考整個架構是如何運作，只需要修改部分的程式碼就好，雖然執行的過程中發現結果不對，但是在助教以及同學的協助下，發現到有腳位沒有給預設值，以及一個硬體架構忘記修改，最終也成功執行出正確的結果，感謝助教的耐心指導。