

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/XX/XX

實驗十一

暫存器定址指令

姓名：王嘉羽 學號：00957116

班級：資工 3B

E-mail：vayne20011125@gmail.com

2022/12/05

● 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

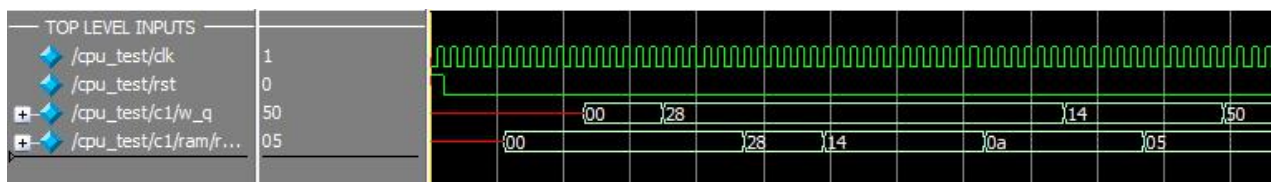
請建一個 MPLAB 專案，打入下方給的組合語言 code，BUILD 並生成 HEX 檔，再將 HEX 轉成 Program_Rom，模擬結果請參考下方的圖

```
#include <pl6Lf1826.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

✧ hw_1205.sv

```
# hw_1205.sv x
design > # hw_1205.sv
1 timescale 1ns/10ps
2 module hw_1205(
3     input clk,
4     input reset,
5     //output logic [7:0] port_b_out
6     output logic [7:0] w_q
7 );
8     //logic [7:0] w_q;
9     logic [10:0] pc_next, pc_q, mar_q;
10    logic load_pc, load_mar, load_ir_q, load_w, load_port_b; //load
11    logic [13:0] rom_out, ir_q;
12    logic reset_ir_q;
13    logic ram_en;
14    logic [2:0] ps,ns;
15    logic [3:0] op;
16    logic [7:0] alu_q, mux_out, ram_out, databus, RAM_mux, bcf_mux, bsf_mux, port_b_out;
17    logic [1:0] sel_RAM_mux;
18    logic sel_alu,sel_pc,sel_bus; //選擇器
19    logic [2:0] sel_bit;
20
21    //-----sel_pc-----
22    always_comb //下一指令
23    begin
24        if(sel_pc) pc_next = ir_q[10:0];
25        else pc_next = pc_q + 1;
26    end
27
28    always_ff @(posedge clk) //存load值，再讀取
29    begin
30        if(reset)
31            pc_q <= #1 0;
32        else if(load_pc)
33            pc_q <= #1 pc_next;
34    end
35
36    //-----mar-----
37    always_ff @(posedge clk)
38    begin
39        if(load_mar)
40            mar_q <= #1 pc_q;
41    end
42
43    //-----ROM-----
44    Program_Rom rom(rom_out,mar_q);
45
46    //-----IR-----
47    always_ff @(posedge clk)
48    begin
49        if(reset_ir_q)
50            ir_q <= #1 0;
51        else if(load_ir_q)
52            ir_q <= #1 rom_out;
53    end
54
55    //-----load_w-----
56    always_ff @(posedge clk)
57    begin
58        if(reset)
59            w_q <= #1 0;
60        else if(load_w)
61            w_q <= #1 alu_q;
62    end
63
64    single_port_ram_128x8 ram(databus,ir_q[6:0],ram_en,clk,ram_out);
65
66    //-----sel_alu-----
67    always_comb
68    begin
69        if(sel_alu == 0) mux_out = ir_q[7:0];
70        else mux_out = RAM_mux[7:0];
71    end
72
73    //-----sel_bus-----
74    always_comb
75    begin
76        if(sel_bus == 0) databus = alu_q;
77        else databus = w_q;
78    end
79
80    //-----port_b-----
81    always_ff @(posedge clk)
82    begin
83        if(reset) port_b_out <= 0;
84        else if(load_port_b) port_b_out <= databus;
85    end
86
87    //-----ram_mux-----
88    always_comb
89    begin
90        case(sel_RAM_mux)
91            0: RAM_mux = ram_out;
92            1: RAM_mux = bcf_mux;
93            2: RAM_mux = bsf_mux;
94        endcase
95    end
96
97    //-----BCF_mux-----
98    always_comb
99    begin
```

```

98     case(sel_bit)
99         3'b000: bcf_mux = ram_out & 8'b1111 1110;
100         3'b001: bcf_mux = ram_out & 8'b1111 1101;
101         3'b010: bcf_mux = ram_out & 8'b1111 1011;
102         3'b011: bcf_mux = ram_out & 8'b1111 0111;
103         3'b100: bcf_mux = ram_out & 8'b1110 1111;
104         3'b101: bcf_mux = ram_out & 8'b1101 1111;
105         3'b110: bcf_mux = ram_out & 8'b1011 1111;
106         3'b111: bcf_mux = ram_out & 8'b0111 1111;
107     endcase
108 end
109
110 //-----BSF_mux-----
111 always_comb
112 begin
113     case(sel_bit)
114         3'b000: bsf_mux = ram_out | 8'b0000 0001;
115         3'b001: bsf_mux = ram_out | 8'b0000 0010;
116         3'b010: bsf_mux = ram_out | 8'b0000 0100;
117         3'b011: bsf_mux = ram_out | 8'b0000 1000;
118         3'b100: bsf_mux = ram_out | 8'b0001 0000;
119         3'b101: bsf_mux = ram_out | 8'b0010 0000;
120         3'b110: bsf_mux = ram_out | 8'b0100 0000;
121         3'b111: bsf_mux = ram_out | 8'b1000 0000;
122     endcase
123 end
124
125 //-----controller-----
126 //指令指令
127 assign MOVLM = (ir_q[13:8] == 5'b110000);
128 assign ADDLM = (ir_q[13:8] == 5'b111110);
129 assign SUBLM = (ir_q[13:8] == 5'b111100);
130 assign ANDLM = (ir_q[13:8] == 5'b111001);
131 assign IORLM = (ir_q[13:8] == 5'b111000);
132 assign XORLM = (ir_q[13:8] == 5'b111010);
133
134 assign d = ir_q[7];
135
136 assign ADDMF = (ir_q[13:8] == 6'b000111);
137 assign ANDMF = (ir_q[13:8] == 6'b000101);
138 assign CLRf = (ir_q[13:7] == 7'b00000011);
139 assign CLRw = (ir_q[13:2] == 12'b0000001000000);
140 assign CMF = (ir_q[13:8] == 6'b001001);
141 assign DECF = (ir_q[13:8] == 6'b000011);
142 assign GOTO = (ir_q[13:11] == 3'b101);
143
144 assign IMCF = (ir_q[13:8] == 6'b001010);
145 assign IORMF = (ir_q[13:8] == 6'b000100);
146 assign MOVF = (ir_q[13:8] == 6'b001000);
147 assign MOVMF = (ir_q[13:7] == 7'b0000001);
148 assign SUBMF = (ir_q[13:8] == 6'b000010);
149 assign XORMF = (ir_q[13:8] == 6'b000110);
150
151 assign BCF = (ir_q[13:10] == 4'b0100);
152 assign BSF = (ir_q[13:10] == 4'b0101);
153 assign BTFSC = (ir_q[13:10] == 4'b0110);
154 assign BTFSS = (ir_q[13:10] == 4'b0111);
155 assign DECFSZ = (ir_q[13:8] == 6'b001011);
156 assign INCFSZ = (ir_q[13:8] == 6'b001111);
157
158 assign sel_bit = ir_q[9:7];
159 assign btfsc_skip_bit = (ram_out[ir_q[9:7]] == 0);
160 assign btfss_skip_bit = (ram_out[ir_q[9:7]] == 1);
161 assign btfsc_btfss_skip_bit = (BTFSC & btfsc_skip_bit) | (BTFSS & btfss_skip_bit);
162 assign aluout_zero = (alu_q == 0);
163
164 assign addr_port_b = (ir_q[6:0] == 7'h0d);
165 assign ASRF = (ir_q[13:8] == 6'b110111);
166 assign LSLF = (ir_q[13:8] == 6'b110101);
167 assign LSRF = (ir_q[13:8] == 6'b110110);
168 assign RLF = (ir_q[13:8] == 6'b000101);
169 assign RRF = (ir_q[13:8] == 6'b000100);
170 assign SWAPF = (ir_q[13:8] == 6'b000110);
171
172 //-----alu----- 用op决定计算结果
173 always_comb
174 begin
175     if(reset)
176         alu_q <= #1 0;
177     else
178         begin
179             case(op)
180                 0: alu_q = mux_out + w_q;
181                 1: alu_q = mux_out - w_q;
182                 2: alu_q = mux_out & w_q;
183                 3: alu_q = mux_out | w_q;
184                 4: alu_q = mux_out ^ w_q;
185                 5: alu_q = mux_out;
186                 6: alu_q = mux_out + 1;
187                 7: alu_q = mux_out - 1;
188                 8: alu_q = 0;
189                 9: alu_q = ~mux_out;
190                 4'hA: alu_q = [mux_out[7],mux_out[7:1]]; //左移 高位 mux_out[7]
191                 4'hB: alu_q = [mux_out[6:0],1'b0]; //左移 左移8
192                 4'hC: alu_q = [1'b0,mux_out[7:1]]; //左移 左移0
193                 4'hD: alu_q = [mux_out[6:0],mux_out[7]]; //左移8
194                 4'hE: alu_q = [mux_out[0],mux_out[7:1]]; //左移0
195                 4'hF: alu_q = [mux_out[3:0],mux_out[7:4]];

```

```

195         default: alu_q = mux_out + w_q;
196     endcase
197 end
198
199
200
201 //-----fsm----- 有限状态机
202 parameter T0 = 0;
203 parameter T1 = 1;
204 parameter T2 = 2;
205 parameter T3 = 3;
206 parameter T4 = 4;
207 parameter T5 = 5;
208 parameter T6 = 6;
209
210 always_ff @(posedge clk)
211 begin
212     if(reset) ps <= #1 0;
213     else ps <= #1 ns;
214 end
215
216 always_comb
217 begin
218     //初始化
219     op = 0;
220     load_mar = 0;
221     load_pc = 0;
222     reset_lr_q = 0;
223     load_lr_q = 0;
224     load_w = 0;
225     sel_pc = 0;
226     sel_alu = 0;
227     sel_bus = 0;
228     ram_en = 0;
229     sel_RAM_mux = 0;
230     load_port_b = 0;
231     ns = 0;
232     case(ps)
233         T0: //初始化lr_q
234             begin
235                 reset_lr_q = 1;
236                 ns = T1;
237             end
238         T1:
239             begin
240                 load_mar = 1; //load mar
241                 ns = T2;
242             end
243         T2:
244             begin
245                 load_pc = 1; //load pc
246                 ns = T3;
247             end
248         T3:
249             begin
250                 //load lr_q
251                 load_lr_q = 1;
252                 ns = T4;
253             end
254         T4: //load w
255             begin
256                 if(MOVLW) op = 5;
257                 else if(ADDLW) op = 0;
258                 else if(SUBLW) op = 1;
259                 else if(ANDLW) op = 2;
260                 else if(IORLW) op = 3;
261                 else if(XORLW) op = 4;
262
263                 else if(ADDF) op = 0;
264                 else if(AMDF) op = 2;
265                 else if(CLRF) op = 8;
266                 else if(CLRW) op = 8;
267                 else if(CMPF) op = 9;
268                 else if(DECFF) op = 7;
269
270                 else if(INCF) op = 6;
271                 else if(IORWF) op = 3;
272                 else if(MOVF) op = 5;
273                 else if(SUBWF) op = 1;
274                 else if(XORWF) op = 4;
275
276                 else if(BCF || BSF) op = 5;
277                 else if(DECFSZ) op = 7;
278                 else if(INCFSZ) op = 6;
279                 else if(ASRF) op = 4'hA;
280                 else if(LSLF) op = 4'hB;
281                 else if(LSRF) op = 4'hC;
282                 else if(RLF) op = 4'hD;
283                 else if(RRF) op = 4'hE;
284                 else if(SHAPF) op = 4'hF;
285                 else op = 0;
286
287                 if(MOVLW || ADDLW || SUBLW || ANDLW || IORLW || XORLW)
288                     load_w = 1;
289                 else if(GOTO)
290

```



```

292         begin
293             sel_pc = 1;
294             load_pc = 1;
295         end
296     else if(ADDF || ANDF || IMCF || LORF || MOVF || SUBWF || XORWF)
297         begin
298             sel_alu = 1;
299             if(d==0) load_w = 1;
300             else ram_en = 1;
301         end
302     else if(CLRF) ram_en = 1;
303     else if(CLRW) load_w = 1;
304     else if(COMF || DECF)
305         begin
306             sel_alu = 1;
307             ram_en = 1;
308         end
309     else if(MOVSF)
310         begin
311             sel_bus = 1;
312             if(addr_port_b == -1) load_port_b = 1;
313             else if(addr_port_b == 0) ram_en = 1;
314         end
315     else if(BCF || BSF)
316         begin
317             sel_alu = 1;
318             if(BCF) sel_RAM_mux = 1; //BCF = 1,BSF = 2
319             else sel_RAM_mux = 2;
320             ram_en = 1;
321         end
322     else if(BTFSC || BTFSS)
323         begin
324             if(btfsc_btfss_skip_bit) load_pc = 1;
325         end
326     else if(DECFSZ || INCFSZ)
327         begin
328             sel_alu = 1;
329             if(d == 0) load_w = 1;
330             else ram_en = 1;
331
332             if(aluout_zero) load_pc = 1;
333         end
334     else if(ASRF || LSLF || LSRF || RLF || RRF || SWAPF)
335         begin
336             sel_alu = 1;
337             if(d == 0) load_w = 1;
338             else if(d == 1) ram_en = 1;
339         end
340     ns = TS;
341 end
342
343 TS: //2R30
344     begin
345         ns = T6;
346     end
347 T6:
348     begin
349         ns = T1;
350     end
351 endcase
352 end
353 endmodule

```

✧ Program_Rom.sv

```

design > Program_Rom.sv
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'h01A5;
11             10'h1 : data = 14'h0103;
12             10'h2 : data = 14'h3028;
13             10'h3 : data = 14'h00A5;
14             10'h4 : data = 14'h37A5;
15             10'h5 : data = 14'h0D25;
16             10'h6 : data = 14'h36A5;
17             10'h7 : data = 14'h3525;
18             10'h8 : data = 14'h0CA5;
19             10'h9 : data = 14'h0E25;
20             10'ha : data = 14'h280A;
21             10'hb : data = 14'h3400;
22             10'hc : data = 14'h3400;
23             default: data = 14'h0;
24         endcase
25     end
26
27     assign Rom_data_out = data;
28
29 endmodule

```

✧ single_port_ram_128x8.sv

```

design > 单 single_port_ram_128x8.sv
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.sv

```

simulation > tb > 单 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_1205 hw_1205_1(
9          .reset(reset), //()內的變數為tb的變數，"."後面為hw_1205.sv的變數，將2者對應起來
10         .clk(clk),
11         .w_q(w_q)
12     );
13
14     always #5 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         #5000 $stop;
20     end
21 endmodule

```

✧ compile.do

```

simulation > modelsim > 单 compile.do
1  #vlib work
2
3
4
5  # -----
6  vlog ../tb/testbench.sv
7  vlog ../../design/hw_1205.sv
8  vlog ../../design/Program_Rom.sv
9  vlog ../../design/single_port_ram_128x8.sv

```

✧ sim.do

```

simulation > modelsim > 单 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

```

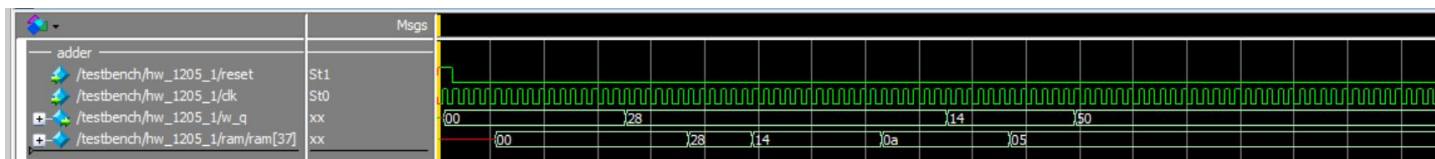
✧ wave.do

```

simulation > modelsim > wave.do
1  onerror {resume}
2  quietly WaveActivateNextPane {} 0
3
4  #add wave -noupdate -divider {TOP LEVEL INPUTS}
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_1205_1/reset
14  add wave -noupdate -format logic /testbench/hw_1205_1/clk
15  add wave -noupdate -format Literal -radix Hexadecimal /testbench/hw_1205_1/w_q
16  add wave -noupdate -format Literal -radix Hexadecimal /testbench/hw_1205_1/ram/ram\[37\]

```

● 模擬結果與結果說明：



和作業要求一致~一開始不一致我還以為是我組合語言打錯了，結果我發現我 `ram[37]` 和 `w_q` 放反了，上下的順序反了，然後調一下就對了！

● 結論與心得：

隨著程式碼變長，每次遇到問題就越來越難找了，因為不知道是這禮拜造成的錯，還是以前就有錯了。最近助教把前幾次的作業改好，我發現有一個地方我在檢查結果時沒有檢查到，然後助教發現錯了，我就去檢查我的 code，不檢查還好，一檢查發現..挖....錯一大堆，有一串 opcode 我都打錯...然後當初還可以正常運行...好好笑...幸好有即時發現，如果在段考才發現，真的會不知道該怎麼辦才好..

有了這個經驗，我現在寫 code 會寫完就檢查一次，不會等結果有錯才檢查(以前都是 ce 或結果錯誤才檢查....)，也因為這樣這次作業很快就完成了!沒有奇奇怪怪的錯誤!

這次有錯在一個地方，就是變數要初始化，我以前覺得初始化沒有很重要，反正會賦值給他，初值就不重要了，不過這好像是個很不好的習慣，不管寫甚麼語言、不管那個值會不會用到，都要養成初始化的好習慣才行!