

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

2022/12/20

實驗十五

PIPELINE

姓名：張銀軒 學號：00957050

班級：資工 3A

E-mail：00957050@mail.ntou.edu.tw

2022/12/20

● 實驗說明：

將 PIC MCU 改成 pipeline 的架構後執行以下測試檔。

PIPELINE 測試程式 1:

```
#include <pl6Lf1826.inc> ; Include file 1
;

temp equ 0x25
;*****
; Program start *
;*****

org 0x00 ; reset vector

loop clrw
      clrf temp
      movlw .1
      addlw .2
      sublw .3
      movlw .10
      movwf temp
      incf temp,1
      subwf temp,0
      bcf temp,3
      btfsc temp,3
      btfsc temp,1
      brw
      nop
      lslf temp,1
      lsrf temp,0
      goto loop

      end
```

PIPELINE 測試程式 2:

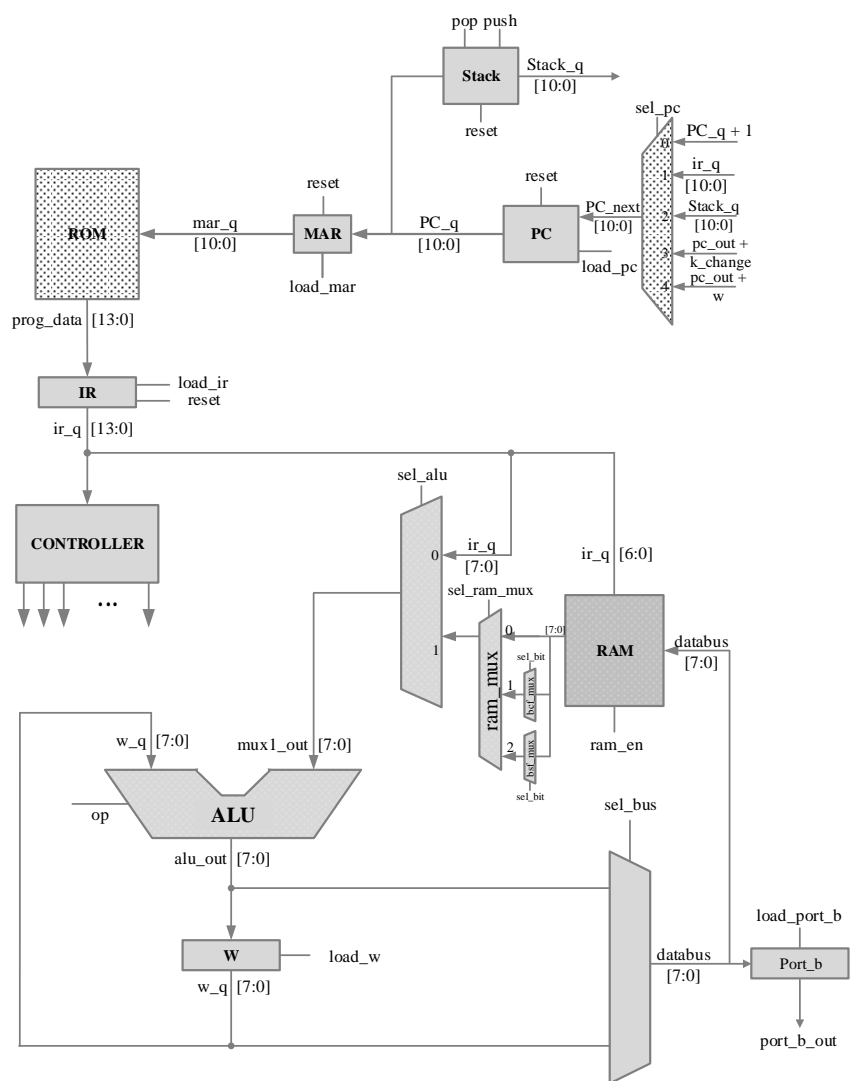
```
#include <pl6Lf1826.inc> ; Include file 1
;

temp equ 0x25
;*****
; Program start *
;*****

org 0x00 ; reset vector

movlw .1
movlw .2
movlw .3
movlw .4
movlw .5
call first
movlw .6
movlw .7
goto $
first movlw .8
      movlw .9
      return
      end
```

● 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

```

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         4'h0: alu_q = mux1_out[7:0] + w_q;
10        4'h1: alu_q = mux1_out[7:0] - w_q;
11        4'h2: alu_q = mux1_out[7:0] & w_q;
12        4'h3: alu_q = mux1_out[7:0] | w_q;
13        4'h4: alu_q = mux1_out[7:0] ^ w_q; //XOR
14        4'h5: alu_q = mux1_out[7:0];
15        4'h6: alu_q = mux1_out[7:0] + 1;
16        4'h7: alu_q = mux1_out[7:0] - 1;
17        4'h8: alu_q = 0;
18        4'h9: alu_q = ~mux1_out[7:0];
19        4'hA: alu_q = { mux1_out[7],mux1_out[7:1] }; //ASRF arithmetic right shift
20        4'hB: alu_q = { mux1_out[6:0], 1'b0 }; //LSLF logical left shift
21        4'hC: alu_q = { 1'b0, mux1_out[7:1] }; //LSRF logical right shift
22        4'hD: alu_q = { mux1_out[6:0], mux1_out[7] }; //RLF rotate left f
23        4'hE: alu_q = { mux1_out[0], mux1_out[7:1] }; //RRF rotate right f
24        4'hF: alu_q = { mux1_out[3:0], mux1_out[7:4] }; //{m7, m6,...m4, m3,...m0} => {m3,...m0, m7, m6,...m4}
25        default: alu_q = mux1_out[7:0] + w_q;
26    endcase
27 end
28
29 endmodule

```

```

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
25 endmodule

```

測試程式 1

```

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'h0103;
11             10'h1 : data = 14'h01A5;
12             10'h2 : data = 14'h3001;
13             10'h3 : data = 14'h3E02;
14             10'h4 : data = 14'h3C03;
15             10'h5 : data = 14'h300A;
16             10'h6 : data = 14'h00A5;
17             10'h7 : data = 14'h0AA5;
18             10'h8 : data = 14'h0225;
19             10'h9 : data = 14'h11A5;
20             10'ha : data = 14'h19A5;
21             10'hb : data = 14'h18A5;
22             10'hc : data = 14'h000B;
23             10'hd : data = 14'h0000;
24             10'he : data = 14'h35A5;
25             10'hf : data = 14'h3625;
26             10'h10 : data = 14'h2800;
27             10'h11 : data = 14'h3400;
28             10'h12 : data = 14'h3400;
29             default: data = 14'h0;
30         endcase
31     end
32
33     assign Rom_data_out = data;
34
35 endmodule
36

```

測試程式 2

```

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'h3001;
11             10'h1 : data = 14'h3002;
12             10'h2 : data = 14'h3003;
13             10'h3 : data = 14'h3004;
14             10'h4 : data = 14'h3005;
15             10'h5 : data = 14'h2009;
16             10'h6 : data = 14'h3006;
17             10'h7 : data = 14'h3007;
18             10'h8 : data = 14'h2808;
19             10'h9 : data = 14'h3008;
20             10'ha : data = 14'h3009;
21             10'hb : data = 14'h0008;
22             10'hc : data = 14'h3400;
23             10'hd : data = 14'h3400;
24             default: data = 14'h0;
25         endcase
26     end
27
28     assign Rom_data_out = data;
29
30 endmodule
31

```

```

1 module Stack (
2     output logic [10:0] stack_out,
3     input [10:0] stack_in,
4     input push,
5     input pop,
6     input reset,
7     input clk
8 );
9     logic [3:0] stk_ptr;
10    logic [10:0] stack [15:0];
11    logic [3:0] stk_index;
12
13    assign stk_index = stk_ptr + 1;
14    assign stack_out = stack[stk_ptr];
15
16    always_ff @( posedge clk ) begin
17        if(reset) begin
18            stk_ptr <= 4'b1111;
19        end
20        else if(push) begin
21            stack[stk_index] <= stack_in;
22            stk_ptr <= stk_ptr + 1;
23        end
24        else if(pop) begin
25            stk_ptr <= stk_ptr - 1 ;
26        end
27    end
28
29 endmodule

```

```

1 module cpu (
2     input clk,
3     input reset,
4     output logic [7:0] port_b_out
5 );
6     logic [13:0] rom_q, ir_q;
7     logic [10:0] pc_next, pc_q, mar_q, k_change;
8     logic load_pc, load_mar, load_ir, reset_ir, load_w, ram_en, sel_alu, d, sel_bus;
9     logic load_port_b;
10    logic [2:0] sel_pc;
11    logic [3:0] ps,ns;
12    logic [7:0] w_q, alu_q, ram_out, mux1_out, bcf_mux, bsf_mux, ram_mux;
13    logic [7:0] databus;
14    logic [3:0] op;
15    logic [5:0] opcode;
16    logic [2:0] sel_bit;
17    logic [1:0] sel_ram_mux;
18    //for Stack
19    logic pop, push;
20    logic [10:0] stack_out;
21
22    //Stack
23    Stack Stack_1(
24        .stack_out(stack_out),
25        .stack_in(pc_q),
26        .push(push),
27        .pop(pop),
28        .reset(reset),
29        .clk(clk)
30    );
31
32    assign w_change = {3'b0, w_q} - 1;
33    assign k_change = {ir_q[8], ir_q[8], ir_q[8:0]}-1 ;
34
35    //mux 0 (select next PC address)
36    always_comb begin
37        if(sel_pc == 4) begin
38            pc_next = pc_q + w_change;
39        end
40        else if(sel_pc == 3) begin
41            pc_next = pc_q + k_change;
42        end
43        else if(sel_pc == 2) begin
44            pc_next = stack_out;
45        end
46        else if(sel_pc == 1) begin
47            pc_next = ir_q;
48        end
49        else begin
50            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;

```

```

101         3'b101: bcf_mux = ram_out & 8'b1101_1111;
102         3'b110: bcf_mux = ram_out & 8'b1011_1111;
103         3'b111: bcf_mux = ram_out & 8'b0111_1111;
104     endcase
105 end
106
107 //BSF mux
108 always_comb begin
109     case (sel_bit)
110         3'b000: bsf_mux = ram_out | 8'b0000_0001;
111         3'b001: bsf_mux = ram_out | 8'b0000_0010;
112         3'b010: bsf_mux = ram_out | 8'b0000_0100;
113         3'b011: bsf_mux = ram_out | 8'b0000_1000;
114         3'b100: bsf_mux = ram_out | 8'b0001_0000;
115         3'b101: bsf_mux = ram_out | 8'b0010_0000;
116         3'b110: bsf_mux = ram_out | 8'b0100_0000;
117         3'b111: bsf_mux = ram_out | 8'b1000_0000;
118     endcase
119 end
120
121 //ram mux (select data into mux1)
122 always_comb begin
123     case (sel_ram_mux)
124         0: ram_mux = ram_out;
125         1: ram_mux = bcf_mux;
126         2: ram_mux = bsf_mux;
127     endcase
128 end
129
130 //mux 1 (select data into ALU)
131 always_comb begin
132     if(sel_alu) begin
133         mux1_out = ram_mux;
134     end
135     else begin
136         mux1_out = ir_q;
137     end
138 end
139
140 assign d = ir_q[7];
141 assign MOVLW = (ir_q[13:8]==6'h30);
142 assign ADDLW = (ir_q[13:8]==6'h3E);
143 assign IORLW = (ir_q[13:8]==6'h38);
144 assign ANDLW = (ir_q[13:8]==6'h39);
145 assign SUBLW = (ir_q[13:8]==6'h3C);
146 assign XORLW = (ir_q[13:8]==6'h3A);
147
148 assign ADDWF = (ir_q[13:8]==6'h07);
149 assign ANDWF = (ir_q[13:8]==6'h05);
150 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;

```

```

251     load_ir = 0;
252     load_w = 0;
253     ram_en = 0;
254     op = 0;
255     sel_ram_mux = 0;
256     sel_bus = 0;
257     load_port_b = 0;
258     ns=0;
259     //for Stack
260     push = 0;
261     pop = 0;
262     case(ps)
263         T0: begin
264             ns = T1;
265         end
266         T1: begin
267             load_mar = 1;
268             sel_pc = 0;
269             load_pc = 1;
270             ns = T2;
271         end
272         T2: begin
273             ns = T3;
274         end
275         T3: begin
276             load_ir = 1;
277             ns = T4;
278         end
279         T4: begin
280             load_mar = 1;
281             sel_pc = 2'b00;
282             load_pc = 1;
283             if(GOTO) begin//to skip the following instruction in T4
284                 load_mar = 1;
285             end
286             else if(MOVLW) begin
287                 sel_alu = 0;
288                 op = 5;
289                 load_w = 1;
290             end
291             else if(ADDLW) begin
292                 sel_alu = 0;
293                 op = 0;
294                 load_w = 1;
295             end
296             else if(IORLW) begin
297                 sel_alu = 0;
298                 op = 3;
299                 load_w = 1;
300             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

```

```

else if(IORWF) begin
    op = 3;
    sel_alu = 1;
    if(d) begin
        ram_en = 1;
        sel_bus = 0;
    end
    else begin
        load_w = 1;
    end
end
else if(MOVF) begin
    op = 5;
    sel_alu = 1;
    if(d) begin
        ram_en = 1;
        sel_bus = 0;
    end
    else begin
        load_w = 1;
    end
end
else if(MOVWF) begin
    sel_bus = 1;
    if(addr_port_b)begin
        load_port_b = 1;
    end
    else begin
        ram_en = 1;
    end
end
else if(SUBWF) begin
    op = 1;
    sel_alu = 1;

```

```

401         if(d) begin
402             ram_en = 1;
403             sel_bus = 0;
404         end
405         else begin
406             load_w = 1;
407         end
408     end
409     else if(XORWF) begin
410         op = 4;
411         sel_alu = 1;
412         if(d) begin
413             ram_en = 1;
414             sel_bus = 0;
415         end
416         else begin
417             load_w = 1;
418         end
419     end
420     else if(BCF)begin
421         sel_alu = 1;
422         sel_ram_mux = 1;
423         op = 5;
424         sel_bus = 0;
425         ram_en = 1;
426     end
427     else if(BSF)begin
428         sel_alu = 1;
429         sel_ram_mux = 2;
430         op = 5;
431         sel_bus = 0;
432         ram_en = 1;
433     end

```

```

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

```

```

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

```

```

501             ram_en = 1;
502         end
503         else begin
504             load_w = 1;
505         end
506     end
507
508     else if(CALL) begin
509         push = 1;
510     end
511     ns = T5;
512 end
513 T5: begin
514     if(GOTO) begin
515         sel_pc = 2'b01;
516         load_pc = 1;
517     end
518     else if(CALL)begin
519         sel_pc = 2'b01;
520         load_pc = 1;
521     end
522     else if(RETURN) begin
523         pop = 1;
524         sel_pc = 2'b10;
525         load_pc = 1;
526     end
527     else if(BRA) begin
528         load_pc = 1;
529         sel_pc = 3;
530     end
531     else if(BRW) begin
532         load_pc = 1;
533         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

```

```

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

```

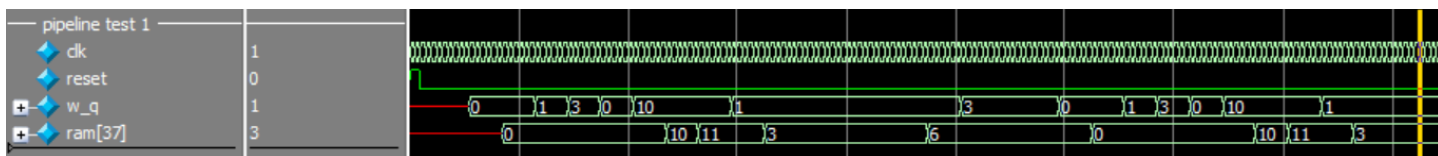
```

1 onerror {resume}
2 quietly WaveActivateNextPane {} 0
3
4 add wave -noupdate -divider {pipeline test 2}
5
6 add wave -noupdate -format Literal -radix Unsigned      /testbench/clk
7 add wave -noupdate -format Logic -radix decimal        /testbench/reset
8 add wave -noupdate -format Literal -radix Unsigned     /testbench/cpu_test/w_q

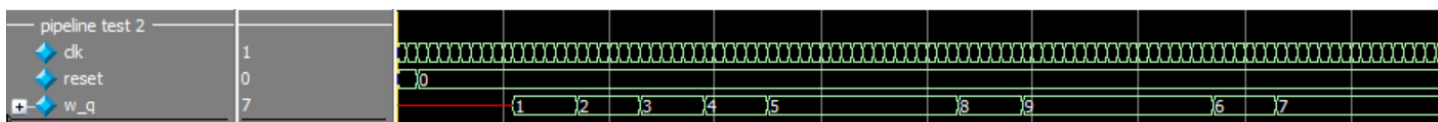
```

● 模擬結果與結果說明：

測試程式 1



測試程式 2



● 結論與心得：



本次作業是上課練習的其中一個部分，上課練習測試程式 3 時因為沒有發現指令判斷錯誤，所以結果一直是錯的，好在助教很耐心的教我一步一步確認是從哪邊發生錯誤，最後終於成功發現之前的 return 判斷錯誤，修改過後就正常了，十分感激助教的耐心與鼓勵。進行測試程式 2 時很快地就產生正確的結果，但是到了測試程式 1 就又發生問題了。一開始以為是 brw 跳到錯誤的位址，但是經過計算之後發現測試程式 1 中的 brw 指令剛好不會跳去其他位址，於是矛頭就指向 lsif 和 lsrf，發現 lsif 都沒有被 enable，經過一連串的尋找才發現程式裡面 sel_pc 的 bit 數不足，overflow 造成 brw 指令多跳了一個指令，修改過後就能正常行。在完成的過程中，重新去回顧之前做的所有的指令，了解每一個指令的流程及資料的變化，讓我對整學期的教學內容有更深刻的理解，謝謝老師的用心教學和助教的詳細講解！