

# 數位系統設計作業

## HW11

學號: 01257027 | 姓名: 林承羿

實作想法：

首先，此項作業沒有跟任何數值輸出有關，要求只有依照指令的模式讓 LED 燈以固定模式閃爍，故以下照片不會出現任何跟數值計算有關的 code，本人為方便聚焦重點，只提供與老師不同的部分，及自己更改的部分。

## 第一題

程式碼

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_unsigned.all;
4
5  entity Control is
6      port(
7          Inst31_26:in std_logic_vector(5 downto 0);
8          SWImport:out std_logic;
9          Jump:out std_logic;
10         Halt:out std_logic;
11         RegImport:out std_logic;
12         RegOutport:out std_logic;
13         RegDst:out std_logic;
14         Branch:out std_logic;
15         MemRead:out std_logic;
16         MemtoReg:out std_logic;
17         ALUOp:out std_logic_vector(1 downto 0);
18         MemWrite:out std_logic;
19         ALUSrc:out std_logic;
20         RegWrite:out std_logic;
21         OUTLED:out std_logic -- LED輸出
22     );
23 end Control;
24 architecture Control of Control is
25     signal tmp:std_logic_vector(14 downto 0); -- 增加一個LED控制
26 begin
27
28     tmp <= "000000100100010" when inst31_26 = 0 else --R
29         "000000011110000" when inst31_26 = 35 else --lw
30         "000000010001000" when inst31_26 = 43 else --sw
31         "000000000000101" when inst31_26 = 4 else --beq
32         "000010000100011" when inst31_26 = 63 else --RegImport
33         "000001000000011" when inst31_26 = 62 else --RegOutportout
34         "000100000000011" when inst31_26 = 61 else --Halt
35         "001000000000011" when inst31_26 = 2 else --Jump
36         "010000000100011" when inst31_26 = 60 else --SWImport
37         "100000000000000" when inst31_26 = 26 else -- LED 控制 26
38         "000000000000000";
```

```

39
40     OUTLED <= tmp(14);           -- 控制LED
41     Swimport <= tmp(13);
42     Jump <= tmp(12);
43     Halt <= tmp(11);
44     RegImport <= tmp(10);
45     RegOutport <= tmp(9);
46
47     RegDst <= tmp(8);
48     ALUSrc <= tmp(7);
49     MemtoReg <= tmp(6);
50     RegWrite <= tmp(5);
51     MemRead <= tmp(4);
52     MemWrite <= tmp(3);
53     Branch <= tmp(2);
54     ALUOp <= tmp(1 downto 0);
55
56 end Control;

```

可以看到其中添加了 OUTLED，作為題目要求的控制信號。

```

1 library ieee;
2 use ieee.std_logic_1164.all;
3 use ieee.std_logic_unsigned.all;
4
5 entity Instruction_memory is
6     port(
7         Reset:in std_logic;
8         Read_address:in integer range 0 to 63;
9         Instruction:out std_logic_vector(31 downto 0)
10    );
11 end Instruction_memory;
12 architecture Instruction_memory of Instruction_memory is
13     type mem_array is array (0 to 63) of std_logic_vector( 31 downto 0 );
14     signal memory : mem_array := (
15         x"68000000",x"08000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",
16         x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",
17         x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",
18         x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",
19         x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",
20         x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",
21         x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",
22         x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000");
23 begin
24
25     Instruction <= memory(Read_address);
26
27     process(Reset)
28     begin
29         if Reset = '0' then
30             memory(0) <= x"68000000"; -- 令LED亮起
31             memory(1) <= x"08000000"; -- 跳回第一個指令(LED暗)
32
33             for i in 2 to 63 loop
34                 memory(i) <= (others => '0');
35             end loop;
36         end if;
37     end process;

```

更改指令，此任務只是讓 LED 閃爍，並不涉及任何來源、目的暫存器更改，不需要存儲任何暫存器，故只需要 31~26 的位置設定為 26(本人設定 OUTLED 的 opcode)即可，兩個指令分別是 LED 亮，LED 暗，除了亮以外做其他事情都是暗的，故可以讓跳回第一個指令同時執行 LED 暗的結果。

```

1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_unsigned.all;
4  use ieee.std_logic_arith.all;
5
6  entity MIPS is
7      port(
8          clk:in std_logic;
9          SWin:in std_logic_vector(9 downto 0);
10         Reset:in std_logic;
11         PCout:out std_logic_vector(5 downto 0);
12         DP:out std_logic_vector(3 downto 0);
13         Hex0,Hex1,Hex2,Hex3:out std_logic_vector(6 downto 0);
14         OUTLEDone:out std_logic
15     );
16 end MIPS;
17 architecture MIPS of MIPS is
18
19     component Registers is
20         port(
21             clk:in std_logic;
22             Reset:in std_logic;
23             RegWrite:in std_logic;
24             Read_register1:in integer range 0 to 31;
25             Read_register2:in integer range 0 to 31;
26             Write_register:in integer range 0 to 31;
27             Write_data:in std_logic_vector(31 downto 0);
28             Read_data1:out std_logic_vector(31 downto 0);
29             Read_data2:out std_logic_vector(31 downto 0)
30         );
31     end component;

```

```

32
33 ✓      component ALU is
34 ✓      port(
35          A:in std_logic_vector(31 downto 0);
36          B:in std_logic_vector(31 downto 0);
37          Operation:in std_logic_vector(3 downto 0);
38          ALUresult:out std_logic_vector(31 downto 0);
39          Zero:out std_logic
40
41      );
42  end component;
43
44 ✓      component Data_memory is
45 ✓      port(
46          clk:in std_logic;
47          Address:in integer range 0 to 63;
48          Write_data:in std_logic_vector(31 downto 0);
49          MemWrite:in std_logic;
50          MemRead:in std_logic;
51          Read_data:out std_logic_vector(31 downto 0)
52      );
53  end component;
54
55 ✓      component Instruction_memory is
56 ✓      port(
57          Reset:in std_logic;
58          Read_address:in integer range 0 to 63;
59          Instruction:out std_logic_vector(31 downto 0)
60      );
61  end component;
62
63 ✓      component Sign_extend is
64 ✓      port(
65          Inst15_0:in std_logic_vector(15 downto 0);
66          Extendout:out std_logic_vector(31 downto 0)
67      );
68  end component;

```



```

70 component Control is
71     port(
72         Inst31_26:in std_logic_vector(5 downto 0);
73         SWImport:out std_logic;
74         Jump:out std_logic;
75         Halt:out std_logic;
76         RegImport:out std_logic;
77         RegOutport:out std_logic;
78         RegDst:out std_logic;
79         Branch:out std_logic;
80         MemRead:out std_logic;
81         MemtoReg:out std_logic;
82         ALUOp:out std_logic_vector(1 downto 0);
83         MemWrite:out std_logic;
84         ALUSrc:out std_logic;
85         RegWrite:out std_logic;
86         OUTLED:out std_logic
87     );
88 end component;
89
90 component ALUControl is
91     port(
92         ALUOp:in std_logic_vector(1 downto 0);
93         Inst5_0:in std_logic_vector(5 downto 0);
94         Operation:out std_logic_vector(3 downto 0)
95     );
96 end component;
97
98 component Add_32bit is
99     port(
100         A:in std_logic_vector(31 downto 0);
101         B:in std_logic_vector(31 downto 0);
102         Result:out std_logic_vector(31 downto 0)
103     );
104 end component;

```

```

105
106     component digits_large is
107     port(
108         BIN:in integer range 0 to 99999999;
109         num7: out integer range 0 to 9;
110         num6: out integer range 0 to 9;
111         num5: out integer range 0 to 9;
112         num4: out integer range 0 to 9;
113         num3: out integer range 0 to 9;
114         num2: out integer range 0 to 9;
115         num1: out integer range 0 to 9;
116         num0: out integer range 0 to 9
117     );
118
119     end component;
120
121     component decoder_7seg is
122     PORT(
123         BCD:in std_logic_vector(3 downto 0);
124         HEX:out std_logic_vector(6 downto 0)
125     );
126     end component;
127
128
129     type INT_array is Array (0 to 3) of integer range 0 to 9;
130     type INT_array_large is Array (0 to 7) of integer range 0 to 9;
131     type LOGIC_array is Array (0 to 3) of std_logic_vector(3 downto 0);
132     signal num1,num2: INT_array;
133     signal num: INT_array_large;
134     signal P:std_logic_vector(19 downto 0);
135     signal show:LOGIC_array;
136     signal cnt:std_logic_vector(25 downto 0);

```

```

137
138     signal Instruction:std_logic_vector(31 downto 0);
139     signal Extendout:std_logic_vector(31 downto 0);
140     signal Read_data1:std_logic_vector(31 downto 0);
141     signal Read_data2:std_logic_vector(31 downto 0);
142
143     signal ALUResult:std_logic_vector(31 downto 0);
144     signal ALU_B:std_logic_vector(31 downto 0);
145     signal Operation:std_logic_vector(3 downto 0);
146     signal Zero:std_logic;
147
148
149     signal Read_mem_data:std_logic_vector(31 downto 0);
150     signal PC:std_logic_vector(31 downto 0);
151     signal SWImport,Jump,Halt,RegImport,RegOutport:std_logic;
152     signal Branch,RegDst,MemRead,MemtoReg,MemWrite,ALUSrc,RegWrite, OUTLED:std_logic;
153     signal ALUOp:std_logic_vector(1 downto 0);
154
155     signal Read_register1,Read_register2,Write_register:integer range 0 to 31;
156     signal Write_data:std_logic_vector(31 downto 0);
157
158     signal Add_32bit_A:std_logic_vector(31 downto 0);
159     signal Add_32bit_B:std_logic_vector(31 downto 0);
160     signal AddResult:std_logic_vector(31 downto 0);
161     signal Output:std_logic_vector(31 downto 0);
162
163     signal delay:integer range 0 to 50000000;
164
165 begin
166
167 --Program Counter
168 process(clk,reset)
169 begin
170     if reset = '0' then
171         PC <= (others => '0');
172         delay <= 0;
173         OUTLEDone <= '1';
174     elsif clk'event and clk = '1' then
175         if Branch = '1' and Zero = '1' then
176             PC <= AddResult;
177
178         elsif Jump = '1' and delay = 24999999 then --直到暫停的時間結束，PC指到下一個指令
179             --elsif Jump = '1' and delay = 1 then
180             --else
181                 delay <= 0;
182                 OUTLEDone <= '1'; -- 亮
183                 PC <= PC(31 downto 28)&Instruction(25 downto 0)&"00";
184             elsif Halt = '0' then
185                 --else
186                     delay <= delay + 1; -- 延遲所有過程，讓PC暫停
187                 end if;
188
189             if OUTLED = '1' and delay = 49999999 then --直到暫停的時間結束，PC指到下一個指令
190                 --if OUTLED = '1' and delay = 3 then
191                     delay <= 0;
192                     OUTLEDone <= '0'; -- 暗
193                     PC <= PC +4;
194                 end if;
195             end if;
196         end process;
197
198 --Instruction Memory
199
200 Inst_Memory_part:
201 Instruction_memory port map (Reset,conv_integer(PC(7 downto 2)),Instruction);

```



```

203
204
205 --Registers
206 Read_register1 <= conv_integer(Instruction(20 downto 16)) when RegOutport = '1' else
207 | | | | | conv_integer(Instruction(25 downto 21));
208
209 Read_register2 <= conv_integer(Instruction(20 downto 16));
210
211 Write_register <= conv_integer(Instruction(20 downto 16)) when RegImport = '1' else --MUX
212 | | | | | conv_integer(Instruction(20 downto 16)) when SWImport = '1' else --MUX SWIN
213 | | | | | conv_integer(Instruction(20 downto 16)) when RegDst = '0' else --MUX
214 | | | | | conv_integer(Instruction(15 downto 11));
215
216 Write_data <= x"0000"&Instruction(15 downto 0) when RegImport = '1' else
217 | | | | | x"0000"&"000000"&SWin when SWImport = '1' else --"00000000000000000000"
218 | | | | | ALUResult when MemtoReg = '0' else
219 | | | | | Read_mem_data;
220
221
222
223 Register_part:
224 Registers port map (clk,Reset,RegWrite,Read_register1,Read_register2,Write_register,
225 | | | | | Write_data,Read_data1,Read_data2);
226
227
228 --ALU
229 ALU_B <= Read_data2 when ALUSrc = '0' else
230 | | | | | Extendout;
231
232 ALU_part:
233 ALU port map (Read_data1,ALU_B,Operation,ALUresult,Zero);
234
235
236 --Data Memory
237
238 Data_mem:
239 Data_memory port map (clk,conv_integer(ALUResult(7 downto 2)),Read_data2,MemWrite,MemRead,Read_mem_data);
240
241 --Control
242
243 Control_part:
244 Control port map (Instruction(31 downto 26),SWImport,Jump,Halt,RegImport,RegOutport,RegDst,Branch,MemRead,MemtoReg,ALUOp,MemWrite,ALUSrc,RegWrite, OUTLED);
245
246 --ALUControl
247
248 ALUControl_part:
249 ALUControl port map (ALUOp,Instruction(5 downto 0),Operation);
250
251
252 --Sign_extend
253
254 Sign_extend_part:
255 Sign_extend port map (Instruction(15 downto 0),Extendout);
256
257
258 --Add_32bit
259 Add_32bit_A <= PC + 4;
260 Add_32bit_B <= Extendout(29 downto 0) & "00";
261
262 Add_32bit_part:
263 Add_32bit port map (Add_32bit_A,Add_32bit_B,AddResult);
264
265 PCout <= PC(5 downto 0);
266
267
268
269 -----
270 process(clk,Reset)
271 begin
272   if Reset = '0' then
273     output <= x"00000000";
274   elsif clk'event and clk = '0' then --fall edge
275     if RegOutport = '1' then
276       Output <= Read_data1;
277     end if;
278   end if;
279 end process;

```

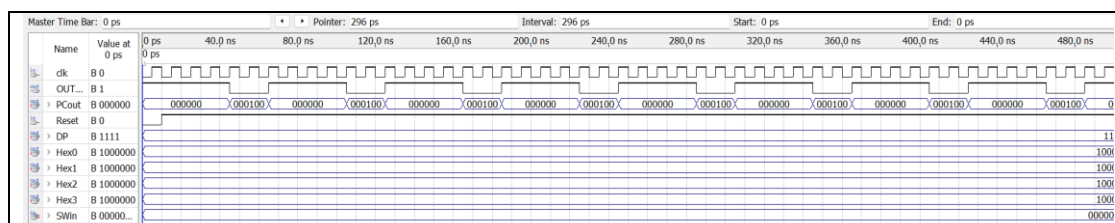
```

281
282
283     process(clk)
284     begin
285         if clk'event and clk = '1' then
286             cnt <= cnt + 1;
287         end if;
288     end process;
289
290
291 U1:digits_large port map(Conv_integer(Output),num(7),num(6),num(5),num(4),num(3),num(2),num(1),num(0));
292
293
294 show(0) <= conv_std_logic_vector(num(0),4) when cnt(25) = '0' else
295         conv_std_logic_vector(num(4),4);
296
297 show(1) <= conv_std_logic_vector(num(1),4) when cnt(25) = '0' else
298         conv_std_logic_vector(num(5),4);
299
300 show(2) <= conv_std_logic_vector(num(2),4) when cnt(25) = '0' else
301         conv_std_logic_vector(num(6),4);
302
303 show(3) <= conv_std_logic_vector(num(3),4) when cnt(25) = '0' else
304         conv_std_logic_vector(num(7),4);
305
306
307
308
309 HEX0_part:decoder_7seg port map(show(0),HEX0);
310 HEX1_part:decoder_7seg port map(show(1),HEX1);
311 HEX2_part:decoder_7seg port map(show(2),HEX2);
312 HEX3_part:decoder_7seg port map(show(3),HEX3);
313
314 DP<= "0000" when cnt(25) = '1' else
315     "1111";
316
317 end MIPS;

```

主要更改的部分是 Program Counter 部分，目的是為了延長 PC 被執行的時間，故不可以讓每次時脈執行時都 PC+4，而是讓一個 delay 信號計數到指定時間再執行。至於 for 迴圈沒有實作的理由是，迴圈一定要做到與時脈相關才會延遲計數否則迴圈跑的步數再多也很快就執行完了，又這行為應該是每次時脈觸發都要做，故捨棄 for 迴圈。

可以看到其中註解了兩行延長為要求時間的 code，是為了呈現結果才這麼做。



為了清楚看到確實有延長，而非照時脈，故意設定最小的延長不與時脈同步。而結果也照著要求按比例縮小，從結果圖也可以觀察到確實指令只有執行 0、4 循環，符合要求，且 OUTLED 的信號也確實影響著結果。

## 加分題

### 程式碼

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 use ieee.std_logic_unsigned.all;
4
5 entity Instruction_memory is
6     port(
7         Reset:in std_logic;
8         Read_address:in integer range 0 to 63;
9         Instruction:out std_logic_vector(31 downto 0)
10    );
11 end Instruction_memory;
12
13 architecture Instruction_memory of Instruction_memory is
14     type mem_array is array (0 to 63) of std_logic_vector( 31 downto 0 );
15     signal memory : mem_array := (
16         x"68000000",x"68000000",x"08000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",
17         x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",
18         x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",
19         x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",
20         x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",
21         x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",
22         x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000");
23 begin
24
25     Instruction <= memory(Read_address);
26
27     process(Reset)
28     begin
29         if Reset = '0' then
30             memory(0) <= x"68000000"; -- 令第一個LED亮起
31             memory(1) <= x"68000000"; -- 令第二個LED亮起
32             memory(2) <= x"08000000"; -- 令第三個LED亮起，並跳回第一個指令
33
34             for i in 3 to 63 loop
35                 memory(i) <= (others => '0');
36             end loop;
37         end if;
38     end process;
39 end;
```

指令只有三個，與基本題不同的是，沒有 LED 暗的，只要設定每次亮的位置不同即可，故第一個指令亮最左邊，第二個亮中間，最後一個亮最右邊，並跳回第一個指令。

```

1
2  library ieee;
3  use ieee.std_logic_1164.all;
4  use ieee.std_logic_unsigned.all;
5  use ieee.std_logic_arith.all;
6
7  entity MIPS is
8      port(
9          clk:in std_logic;
10         SWin:in std_logic_vector(9 downto 0);
11         Reset:in std_logic;
12         PCout:out std_logic_vector(5 downto 0);
13         DP:out std_logic_vector(3 downto 0);
14         Hex0,Hex1,Hex2,Hex3:out std_logic_vector(6 downto 0);
15         OUTLED:out std_logic_vector(2 downto 0)
16     );
17 end MIPS;
18 architecture MIPS of MIPS is
19
20     component Registers is
21     port(
22         clk:in std_logic;
23         Reset:in std_logic;
24         RegWrite:in std_logic;
25         Read_register1:in integer range 0 to 31;
26         Read_register2:in integer range 0 to 31;
27         Write_register:in integer range 0 to 31;
28         Write_data:in std_logic_vector(31 downto 0);
29         Read_data1:out std_logic_vector(31 downto 0);
30         Read_data2:out std_logic_vector(31 downto 0)
31     );
32     end component;
33

```

```

34 component ALU is
35     port(
36         A:in std_logic_vector(31 downto 0);
37         B:in std_logic_vector(31 downto 0);
38         Operation:in std_logic_vector(3 downto 0);
39         ALUresult:out std_logic_vector(31 downto 0);
40         Zero:out std_logic
41     );
42 end component;
43
44 component Data_memory is
45     port(
46         clk:in std_logic;
47         Address:in integer range 0 to 63;
48         Write_data:in std_logic_vector(31 downto 0);
49         MemWrite:in std_logic;
50         MemRead:in std_logic;
51         Read_data:out std_logic_vector(31 downto 0)
52     );
53 end component;
54
55 component Instruction_memory is
56     port(
57         Reset:in std_logic;
58         Read_address:in integer range 0 to 63;
59         Instruction:out std_logic_vector(31 downto 0)
60     );
61 end component;
62
63 component Sign_extend is
64     port(
65         Inst15_0:in std_logic_vector(15 downto 0);
66         Extendout:out std_logic_vector(31 downto 0)
67     );
68 end component;
69
70

```



```

71  ✓ component Control is
72  ✓     port(
73         Inst31_26:in std_logic_vector(5 downto 0);
74         SWImport:out std_logic;
75         Jump:out std_logic;
76         Halt:out std_logic;
77         RegImport:out std_logic;
78         RegOutport:out std_logic;
79         RegDst:out std_logic;
80         Branch:out std_logic;
81         MemRead:out std_logic;
82         MemtoReg:out std_logic;
83         ALUOp:out std_logic_vector(1 downto 0);
84         MemWrite:out std_logic;
85         ALUSrc:out std_logic;
86         RegWrite:out std_logic;
87         OUTLED:out std_logic
88     );
89     end component;
90
91  ✓ component ALUControl is
92  ✓     port(
93         ALUOp:in std_logic_vector(1 downto 0);
94         Inst5_0:in std_logic_vector(5 downto 0);
95         Operation:out std_logic_vector(3 downto 0)
96     );
97     end component;
98
99  ✓ component Add_32bit is
100  ✓     port(
101         A:in std_logic_vector(31 downto 0);
102         B:in std_logic_vector(31 downto 0);
103         Result:out std_logic_vector(31 downto 0)
104     );
105     end component;
106

```

```

107     component digits_large is
108     port(
109         BIN:in integer range 0 to 99999999;
110         num7: out integer range 0 to 9;
111         num6: out integer range 0 to 9;
112         num5: out integer range 0 to 9;
113         num4: out integer range 0 to 9;
114         num3: out integer range 0 to 9;
115         num2: out integer range 0 to 9;
116         num1: out integer range 0 to 9;
117         num0: out integer range 0 to 9
118     );
119
120     end component;
121
122     component decoder_7seg is
123     PORT(
124         BCD:in std_logic_vector(3 downto 0);
125         HEX:out std_logic_vector(6 downto 0)
126     );
127     end component;
128
129
130     type INT_array is Array (0 to 3) of integer range 0 to 9;
131     type INT_array_large is Array (0 to 7) of integer range 0 to 9;
132     type LOGIC_array is Array (0 to 3) of std_logic_vector(3 downto 0);
133     signal num1,num2: INT_array;
134     signal num: INT_array_large;
135     signal P:std_logic_vector(19 downto 0);
136     signal show:LOGIC_array;
137     signal cnt:std_logic_vector(25 downto 0);
138
139     signal Instruction:std_logic_vector(31 downto 0);
140     signal Extendout:std_logic_vector(31 downto 0);
141     signal Read_data1:std_logic_vector(31 downto 0);
142     signal Read_data2:std_logic_vector(31 downto 0);
143
144     signal ALUResult:std_logic_vector(31 downto 0);
145     signal ALU_B:std_logic_vector(31 downto 0);
146     signal Operation:std_logic_vector(3 downto 0);
147     signal Zero:std_logic;
148
149
150     signal Read_mem_data:std_logic_vector(31 downto 0);
151     signal PC:std_logic_vector(31 downto 0);
152     signal SWImport,Jump,Halt,RegImport,RegOutport:std_logic;
153     signal Branch,RegDst,MemRead,MemtoReg,MemWrite,ALUSrc,RegWrite, OUTLEDsig:std_logic;
154     signal ALUOp:std_logic_vector(1 downto 0);
155
156     signal Read_register1,Read_register2,Write_register:integer range 0 to 31;
157     signal Write_data:std_logic_vector(31 downto 0);
158
159     signal Add_32bit_A:std_logic_vector(31 downto 0);
160     signal Add_32bit_B:std_logic_vector(31 downto 0);
161     signal AddResult:std_logic_vector(31 downto 0);
162     signal Output:std_logic_vector(31 downto 0);
163
164     signal delay:integer range 0 to 150000000;
165     signal flag0, flag1, flag2:std_logic := '0';

```

```

167 begin
168 --Program Counter
169 process(clk,reset)
170 begin
171     if reset = '0' then
172         PC <= (others => '0');
173         delay <= 0;
174         flag0 <= '1';
175         OUTLED(0) <= '1';
176         OUTLED(1) <= '0';
177         OUTLED(2) <= '0';
178     elsif clk'event and clk = '1' then
179         if Branch = '1' and Zero = '1' then
180             PC <= AddResult;
181
182         elsif Jump = '1' and flag2 = '1' and delay = 24999999 then --直到暫停的時間結束，PC指到下一個指令
183             --elsif Jump = '1' and flag0 = '1' and delay = 1 then
184             --else
185                 delay <= 0;
186                 flag0 <= '1';
187                 flag2 <= '0';
188                 OUTLED(0) <= '1';
189                 OUTLED(2) <= '0';
190                 PC <= PC(31 downto 28)&Instruction(25 downto 0)&"00";
191             elsif Halt = '0' then
192                 --else
193                 delay <= delay + 1;
194             end if;
195
196         if OUTLEDsig = '1' and flag0 = '1' and delay = 14999999 then --直到暫停的時間結束，PC指到下一個指令
197             --if OUTLEDsig = '1' and flag2 = '1' and delay = 11 then
198                 delay <= 0;
199                 OUTLED(0) <= '0';
200                 OUTLED(1) <= '1';
201                 flag0 <= '0';
202                 flag1 <= '1';
203                 PC <= PC +4;
204             elsif OUTLEDsig = '1' and flag1 = '1' and delay = 14999999 then --直到暫停的時間結束，PC指到下一個指令
205                 --elsif OUTLEDsig = '1' and flag1 = '1' and delay = 11 then
206                 delay <= 0;
207                 OUTLED(1) <= '0';
208                 OUTLED(2) <= '1';
209                 flag1 <= '0';
210                 flag2 <= '1';
211                 PC <= PC +4;
212             end if;
213         end if;
214     end process;

```

```

215 --Instruction Memory
216
217 Inst_Memory_part:
218 Instruction_memory port map (Reset,conv_integer(PC(7 downto 2)),Instruction);
219
220
221
222 --Registers
223 v Read_register1 <= conv_integer(Instruction(20 downto 16)) when RegOutport = '1' else
224     conv_integer(Instruction(25 downto 21));
225
226 Read_register2 <= conv_integer(Instruction(20 downto 16));
227
228 v Write_register <= conv_integer(Instruction(20 downto 16)) when RegImport = '1' else --MUX
229     conv_integer(Instruction(20 downto 16)) when SWImport = '1' else --MUX SWIN
230     conv_integer(Instruction(20 downto 16)) when RegDst = '0' else --MUX
231     conv_integer(Instruction(15 downto 11));
232
233 v Write_data <= x"0000"&Instruction(15 downto 0) when RegImport = '1' else
234     x"0000"&"000000"&SWin when SWImport = '1' else --"00000000000000000000"
235     ALUResult when MemtoReg = '0' else
236     Read_mem_data;
237
238
239
240 Register_part:
241 v Registers port map(clk,Reset,RegWrite,Read_register1,Read_register2,Write_register,
242     Write_data,Read_data1,Read_data2);
243
244
245 --ALU
246 v ALU_B <= Read_data2 when ALUSrc = '0' else
247     Extendout;
248
249 ALU_part:
250 ALU port map (Read_data1,ALU_B,Operation,ALUresult,Zero);

```

---

```

253 --Data Memory
254
255 Data_mem:
256 Data_memory port map (clk,conv_integer(ALUResult(7 downto 2)),Read_data2,MemWrite,MemRead,Read_mem_data);
257
258 --Control
259
260 Control_part:
261 Control port map (Instruction(31 downto 26),SWImport,Jump,Halt,RegImport,RegOutport,RegDst,Branch,MemRead,MemtoReg,ALUOp,MemWrite,ALUSrc,RegWrite, OUTLEDSig);
262
263 --ALUControl
264
265 ALUControl_part:
266 ALUControl port map (ALUOp,Instruction(5 downto 0),Operation);
267
268
269 --Sign_extend
270
271 Sign_extend_part:
272 Sign_extend port map (Instruction(15 downto 0),Extendout);
273
274
275 --Add_32bit
276 Add_32bit_A <= PC + 4;
277 Add_32bit_B <= Extendout(29 downto 0) & "00";
278
279 Add_32bit_part:
280 Add_32bit port map (Add_32bit_A,Add_32bit_B,AddResult);
281
282 PCout <= PC(5 downto 0);

```



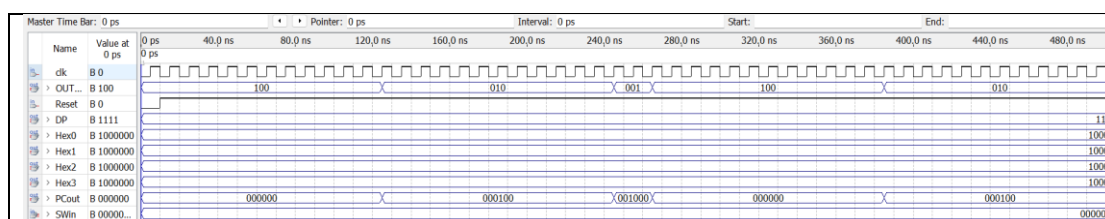
```

286 -----
287 process(clk,Reset)
288 begin
289     if Reset = '0' then
290         output <= x"00000000";
291     elsif clk'event and clk = '0' then --fall edge
292         if RegOutput = '1' then
293             Output <= Read_data1;
294         end if;
295     end if;
296 end process;
297
298 -----
299
300     process(clk)
301     begin
302         if clk'event and clk = '1' then
303             cnt <= cnt + 1;
304         end if;
305     end process;
306
307
308     U1:digits_large port map(Conv_integer(Output),num(7),num(6),num(5),num(4),num(3),num(2),num(1),num(0));
309
310
311     show(0) <= conv_std_logic_vector(num(0),4) when cnt(25) = '0' else
312         conv_std_logic_vector(num(4),4);
313
314     show(1) <= conv_std_logic_vector(num(1),4) when cnt(25) = '0' else
315         conv_std_logic_vector(num(5),4);
316
317     show(2) <= conv_std_logic_vector(num(2),4) when cnt(25) = '0' else
318         conv_std_logic_vector(num(6),4);
319
320     show(3) <= conv_std_logic_vector(num(3),4) when cnt(25) = '0' else
321         conv_std_logic_vector(num(7),4);

```

加分題改最多的依然是 Program Counter 的部分，opcode 一樣延用 26，那這指令只知道要亮，所以添加三個 flag 控制信號確認延長時間到了後該關掉哪個 LED，並使哪個 LED 點亮。

與基本題相同，為了呈現出結果的正確性，將要求的時間註解，並按比例縮小呈現出結果。



可以清楚看到圖中呈現出 LED 亮的相對位置，並看到 PC 確實延長了接收下一道指令的時間。從 PCout 中可以看到指令確實只有 0、4、8 再輪流執行，符合題目要求。



## 心得

透過此次的實習，我更了解到了如何運用 VHLD 使硬體部件照的我的意願行動，並知悉自己也是有能力設計出生活中常見例子的，如題目要求的紅綠燈，我真正了解到自己原來離生活沒有差距如此之大。