2025/10/16

實驗六

序向邏輯練習

姓名：林承羿
學號：01257027
班級：資工 3A
E-mail：IanLin6225@gmail.com

※注意

1. 繳交時一律轉 PDF 檔
2. 繳交期限為下周上課前
3. 一人繳交一份
4. 檔名請按照作業檔名格式進行填寫，未依照格式不予批改
5. 檔名範例：學號_姓名_HW6

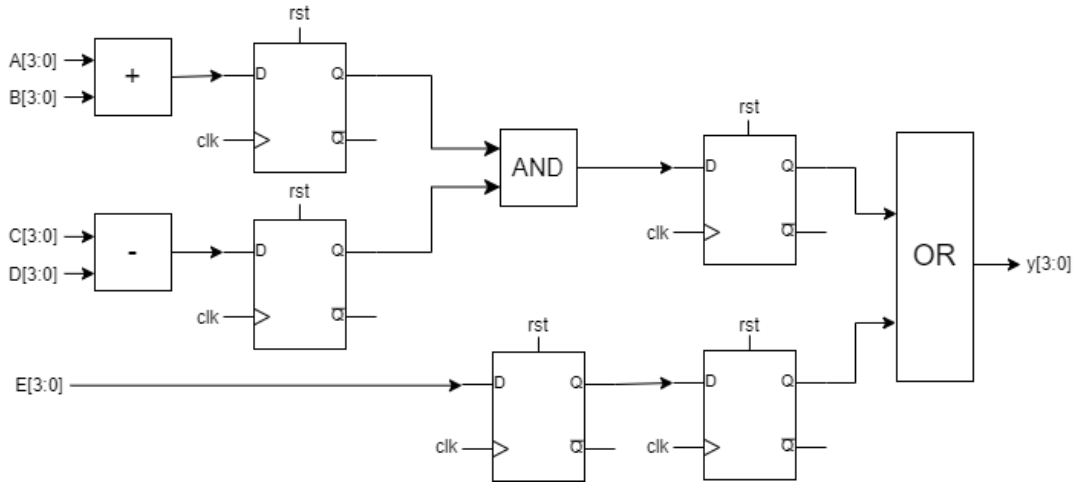
1、 Pipeline

●實驗說明：

1. 使用 pipeline 做 10 筆((A+B)&(C-D))|E 的運算。
2. 測資：

	第 1 筆	第 2 筆	第 3 筆	第 4 筆	第 5 筆	第 6 筆	第 7 筆	第 8 筆	第 9 筆	第 10 筆
A	6	4	1	8	6	11	6	11	7	2
B	7	8	9	7	10	9	8	10	4	8
C	8	7	6	3	3	6	2	4	10	11
D	3	3	3	7	3	5	7	3	7	13
E	10	1	5	2	10	6	1	2	9	5

●系統硬體架構方塊圖（接線圖）：



●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**

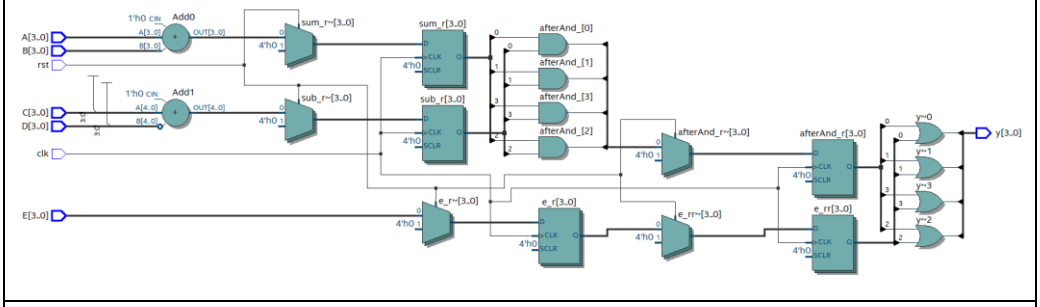
design > simplePipe.sv > ...

```
1  module simplePipe (
2      input logic rst, clk,
3      input logic [3:0] A, B, C, D, E,
4      output logic [3:0] y
5  );
6      // pipe 1
7      logic [3:0] sum_, sum_r, sub_, sub_r, e_, e_r;
8      assign sum_ = A+B;
9      assign sub_ = C-D;
10     assign e_ = E;
11     always_ff @(posedge clk) begin
12         if (rst) begin
13             sum_r <= 0;
14             sub_r <= 0;
15             e_r <= 0;
16         end
17         else begin
18             sum_r <= sum_;
19             sub_r <= sub_;
20             e_r <= e_;
21         end
22     end
23
24     // pipe 2
25     logic [3:0] afterAnd_, afterAnd_r, e_rr;
26     assign afterAnd_ = sum_r & sub_r;
27     always_ff @(posedge clk) begin
28         if (rst) begin
29             e_rr <= 0;
30             afterAnd_r <= 0;
31         end
32         else begin
33             e_rr <= e_r;
34             afterAnd_r <= afterAnd_;
35         end
36     end
37
38     // after pipe2 'or'
39     assign y = afterAnd_r | e_rr;
40 endmodule
```

以上 code 註解也看得出來粗略地分成三個部分，主要以 DFF 作為切分點，並將_設計成組合邏輯輸

出，_r 表示經過一層序向邏輯的輸出，_rr 表示經過兩層序向邏輯的輸出。

- 架構圖：



上圖為 **quartus 19.1** 經過 **compile** 後產生的架構圖，為確認與架構圖相同，經過兩層 **DFF** 後 **OR** 出 **y** 的答案。

- 模擬結果與結果說明：

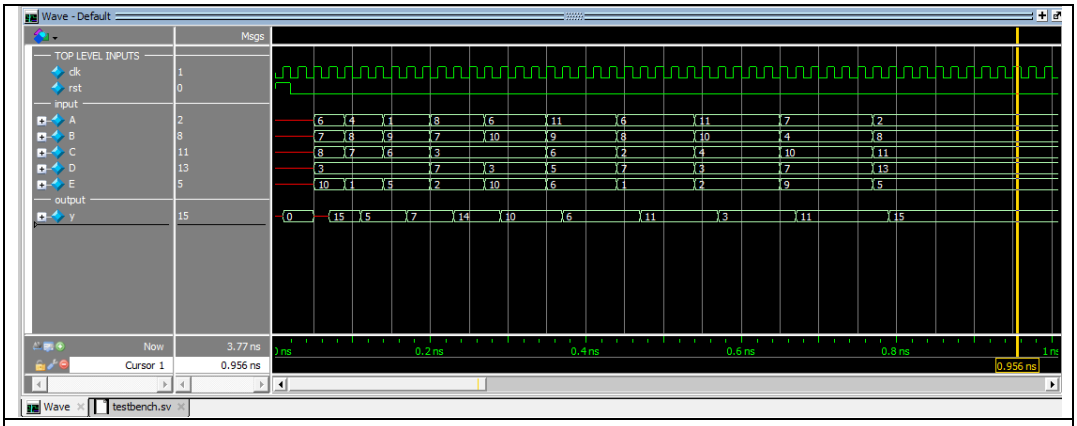
Testbench：

simulation > tb > testbench.sv > testbench

```
module testbench;
  logic rst;
  logic clk;
  logic [3:0] A, B, C, D, E, y;

  simplePipe mySimplePipe (
    .rst (rst),
    .clk (clk),
    .A (A),
    .B (B),
    .C (C),
    .D (D),
    .E (E),
    .y (y)
  );

  initial
  begin
    clk = 0;
    rst = 1;
    #20 rst = 0;
    #30 A = 6; B = 7; C = 8; D = 3; E = 10;
    #40 A = 4; B = 8; C = 7; D = 3; E = 1;
    #50 A = 1; B = 9; C = 6; D = 3; E = 5;
    #60 A = 8; B = 7; C = 3; D = 7; E = 2;
    #70 A = 6; B = 10; C = 3; D = 3; E = 10;
    #80 A = 11; B = 9; C = 6; D = 5; E = 6;
    #90 A = 6; B = 8; C = 2; D = 7; E = 1;
    #100 A = 11; B = 10; C = 4; D = 3; E = 2;
    #110 A = 7; B = 4; C = 10; D = 7; E = 9;
    #120 A = 2; B = 8; C = 11; D = 13; E = 5;
    #3000 $stop;
  end

  always #10 clk = ~clk;
endmodule
```

這是本人此題的 testbench，配合 pipe 說明用到的第一筆輸入，且確認與題目所述的輸入相同。

全部結果：



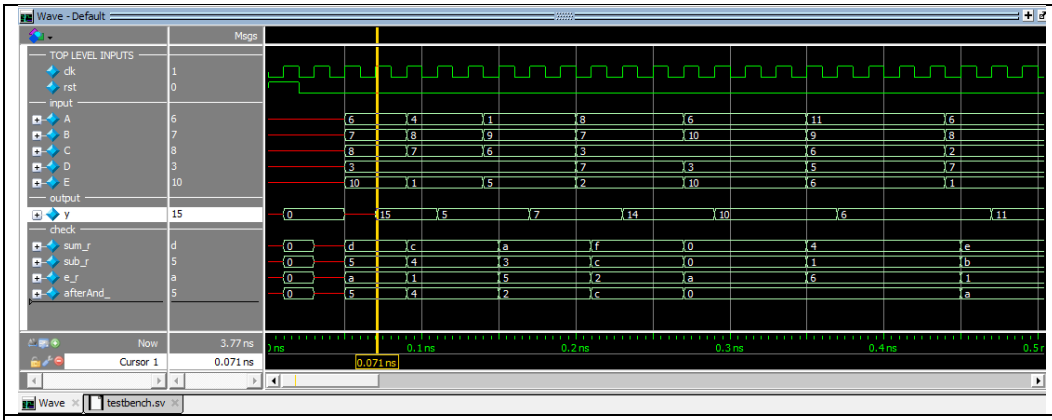
此圖為展示全部輸入跑出的結果。

雙重結果確認：

```
sourceCode > testSv.cc > main(void)
1   # include <bits/stdc++.h>
2   using namespace std;
3
4   int main(void) {
5       vector<int> A {6, 4, 1, 8, 6, 11, 6, 11, 7, 2};
6       vector<int> B {7, 8, 9, 7, 10, 9, 8, 10, 4, 8};
7       vector<int> C {8, 7, 6, 3, 3, 6, 2, 4, 10, 11};
8       vector<int> D {3, 3, 3, 7, 3, 5, 7, 3, 7, 13};
9       vector<int> E {10, 1, 5, 2, 10, 6, 1, 2, 9, 5};
10      vector<int> ans;
11      int mymod = 16;
12      for(int i = 0; i<A.size(); i++) {
13          int tmpsum = (A[i]+B[i]) % mymod;
14          int tmpsub = (C[i]-D[i]) % mymod;
15          int tmpand = (tmpsum & tmpsub) % mymod;
16          int tmpans = (tmpand | E[i]) % mymod;
17          ans.push_back(tmpans);
18      }
19      for(auto &x : ans) cout<<x<<" ";
20      cout<<endl;
21  }

問題  輸出  偵錯主控台  終端機  連接埠

root ➜/workspaces/C++ $ ./a
15 5 7 14 10 6 11 3 11 15
root ➜/workspaces/C++ $
```

上圖為一個本人依照此題撰寫的 C++ code，從 terminal 輸出可看出與波形圖輸出一模一樣。

說明 **pipe**：



此圖加上了 check 欄，為證明有 pipe，可發現在黃色線(表示得到 and 結果)一個 clk 後跑出第一個結果 15，且同時算出第二筆輸入的 and 值 4。

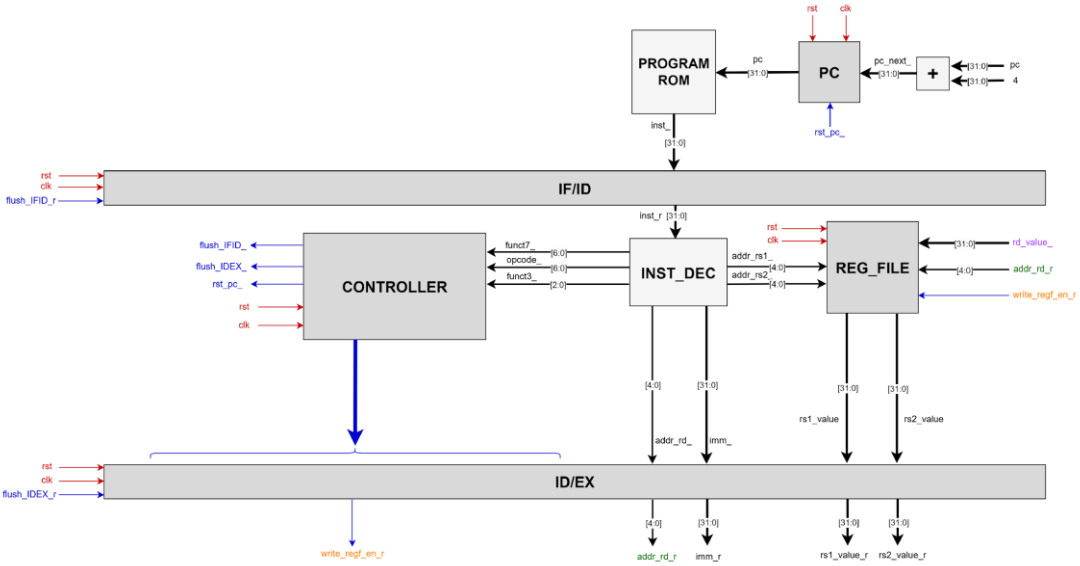
在初始化後，可看到於 rst=1 區間的 posedge clk 後，第一層 pipe 輸出全初始化為 0，且 rst=0 後的第一個 low level(#30)在 testbench 吃到第一筆輸入，經過兩個 posedge clk 後輸出第一筆 y=15，也符合兩層 DFF 需要兩個 clk 才得知 y 的設計。

2、 Pipeline

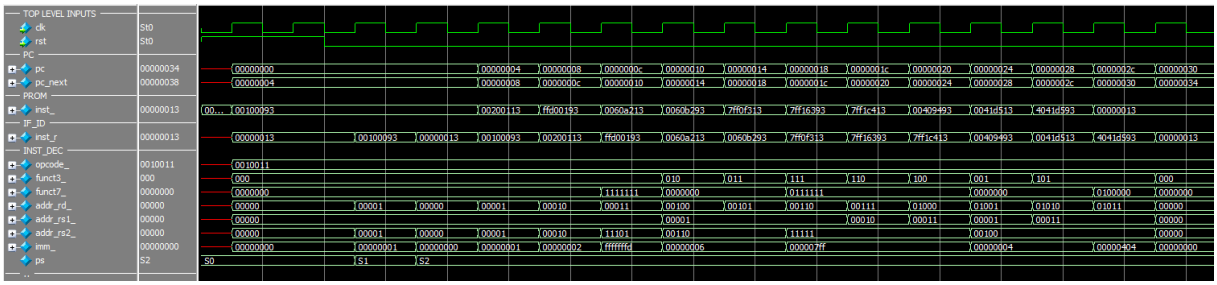
●實驗說明：

1. 完成下圖架構，Program_Rom 已提供
2. 確保程式計數器（pc）正確計數，以及指令（inst and inst_r）被正確提取。

●系統硬體架構方塊圖（接線圖）：



●

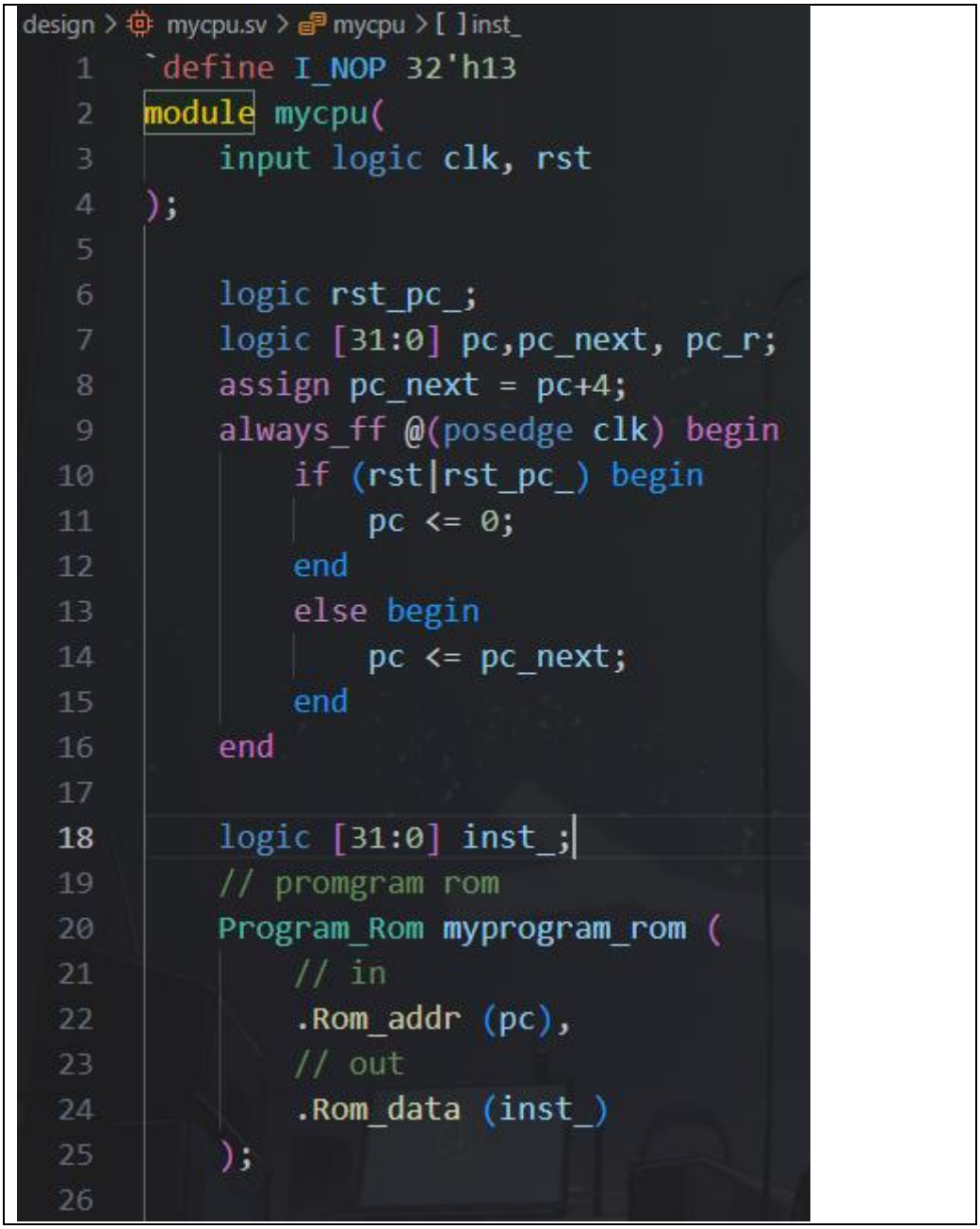


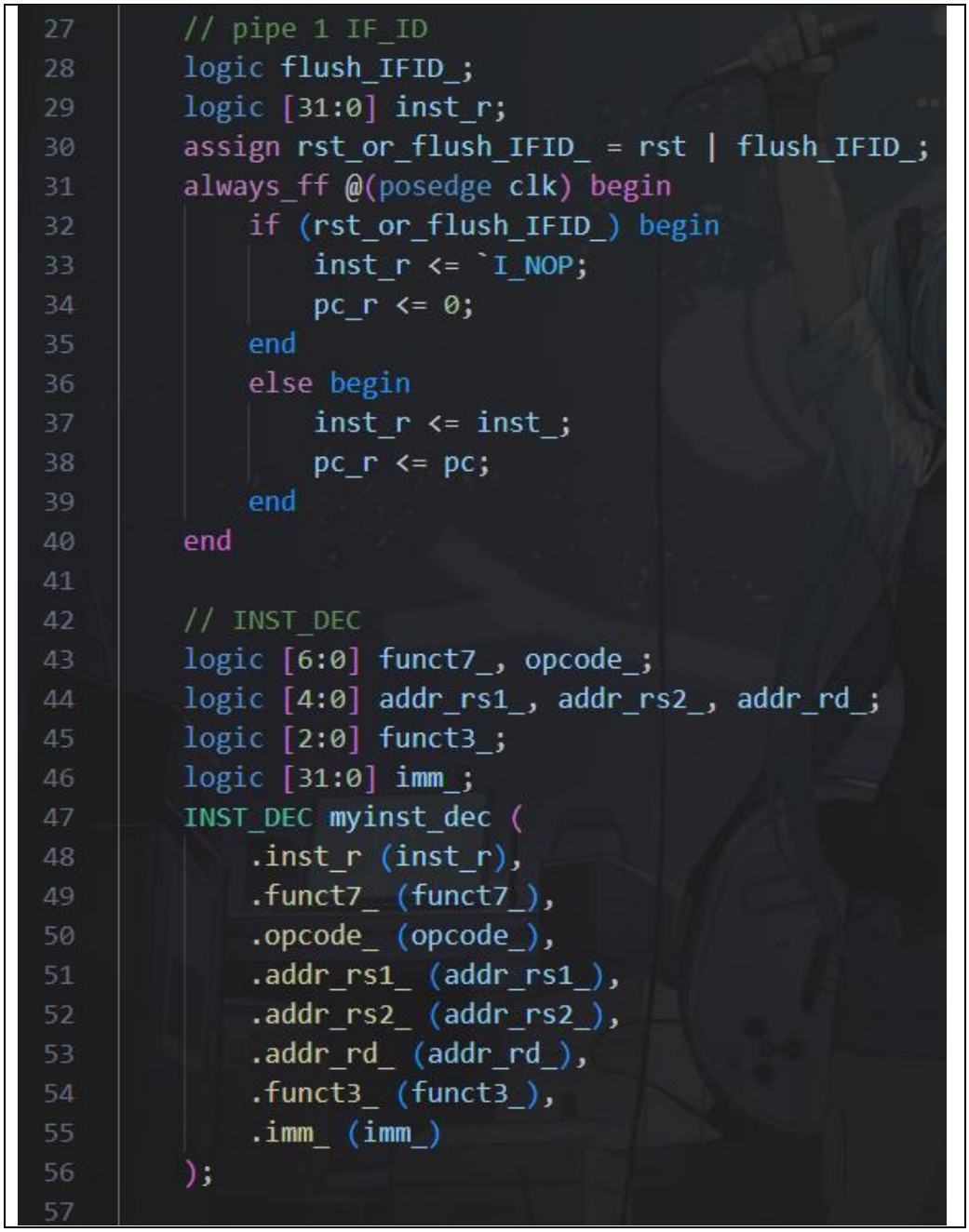
●系統架構程式碼、測試資料程式碼與程式碼說明

截圖請善用 win+shift+S

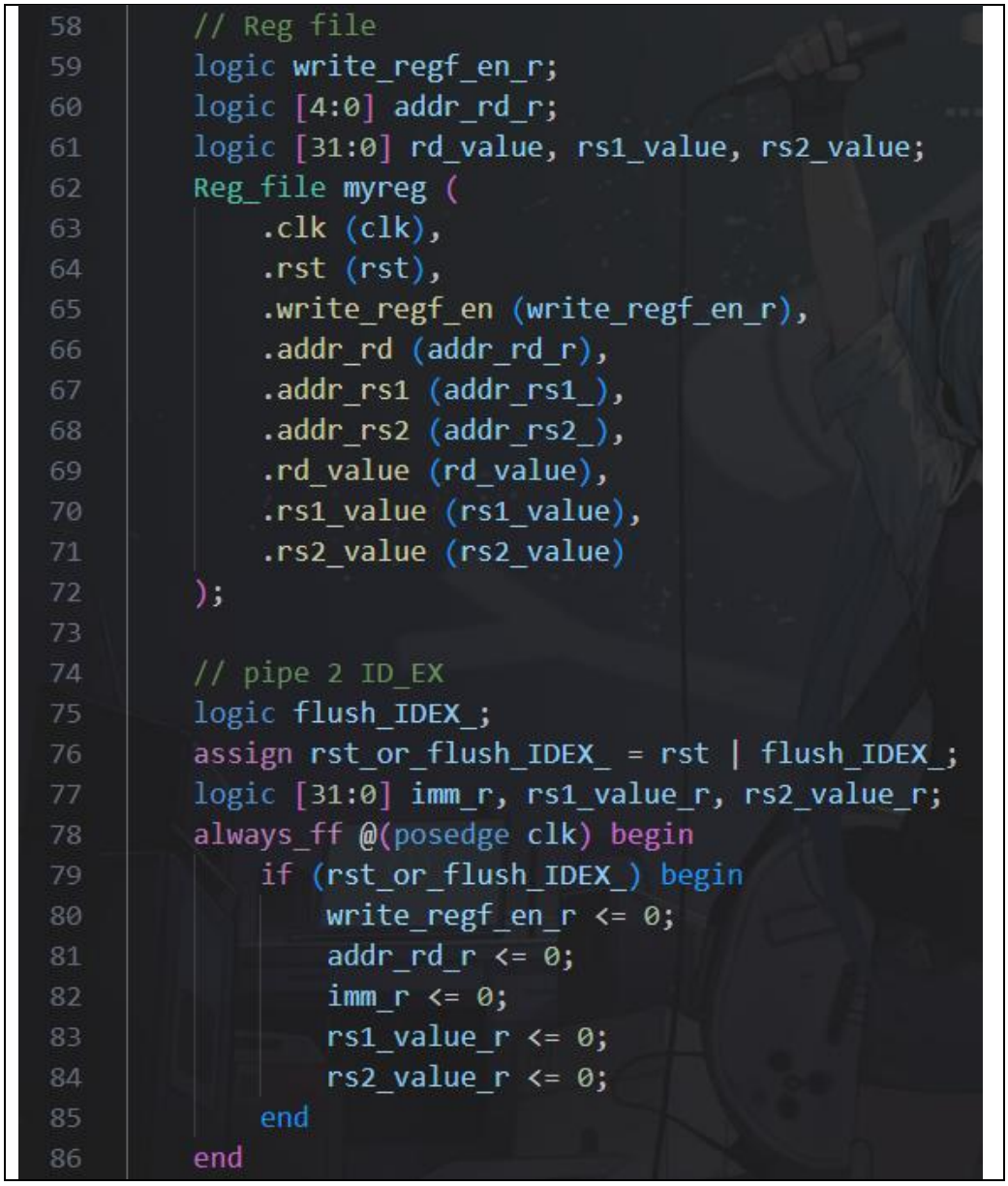
A. 以下截圖按 mycpu.sv 中出現的 module 順序

1. mycpu.sv

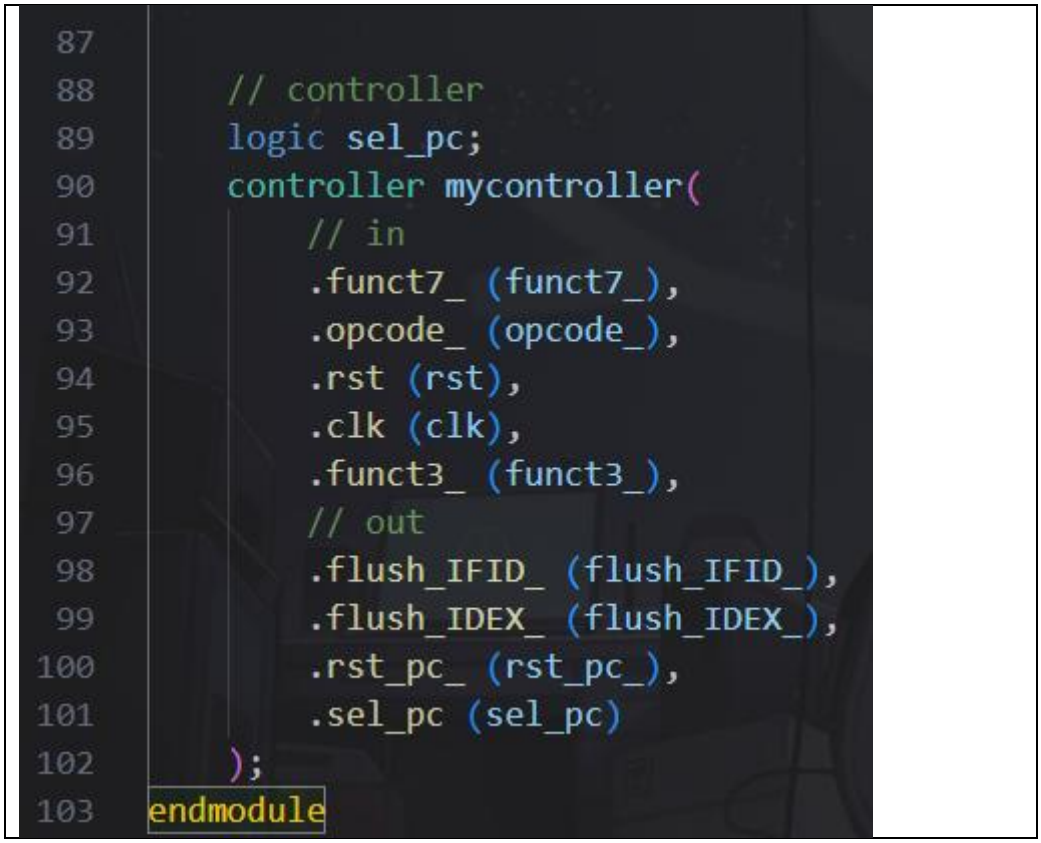

design > mycpu.sv > mycpu > [ ] inst_
1 `define I_NOP 32'h13
2 module mycpu(
3 input logic clk, rst
4);
5
6 logic rst_pc_;
7 logic [31:0] pc,pc_next, pc_r;
8 assign pc_next = pc+4;
9 always_ff @(posedge clk) begin
10 if (rst|rst_pc_) begin
11 pc <= 0;
12 end
13 else begin
14 pc <= pc_next;
15 end
16 end
17
18 logic [31:0] inst_;
19 // promgram rom
20 Program_Rom myprogram_rom (
21 // in
22 .Rom_addr (pc),
23 // out
24 .Rom_data (inst_)
25);
26



```
27      // pipe 1 IF_ID
28      logic flush_IFID_;
29      logic [31:0] inst_r;
30      assign rst_or_flush_IFID_ = rst | flush_IFID_;
31      always_ff @(posedge clk) begin
32          if (rst_or_flush_IFID_) begin
33              inst_r <= `I_NOP;
34              pc_r <= 0;
35          end
36          else begin
37              inst_r <= inst_;
38              pc_r <= pc;
39          end
40      end
41
42      // INST_DEC
43      logic [6:0] funct7_, opcode_;
44      logic [4:0] addr_rs1_, addr_rs2_, addr_rd_;
45      logic [2:0] funct3_;
46      logic [31:0] imm_;
47      INST_DEC myinst_dec (
48          .inst_r (inst_r),
49          .funct7_ (funct7_),
50          .opcode_ (opcode_),
51          .addr_rs1_ (addr_rs1_),
52          .addr_rs2_ (addr_rs2_),
53          .addr_rd_ (addr_rd_),
54          .funct3_ (funct3_),
55          .imm_ (imm_)
56      );
57
```

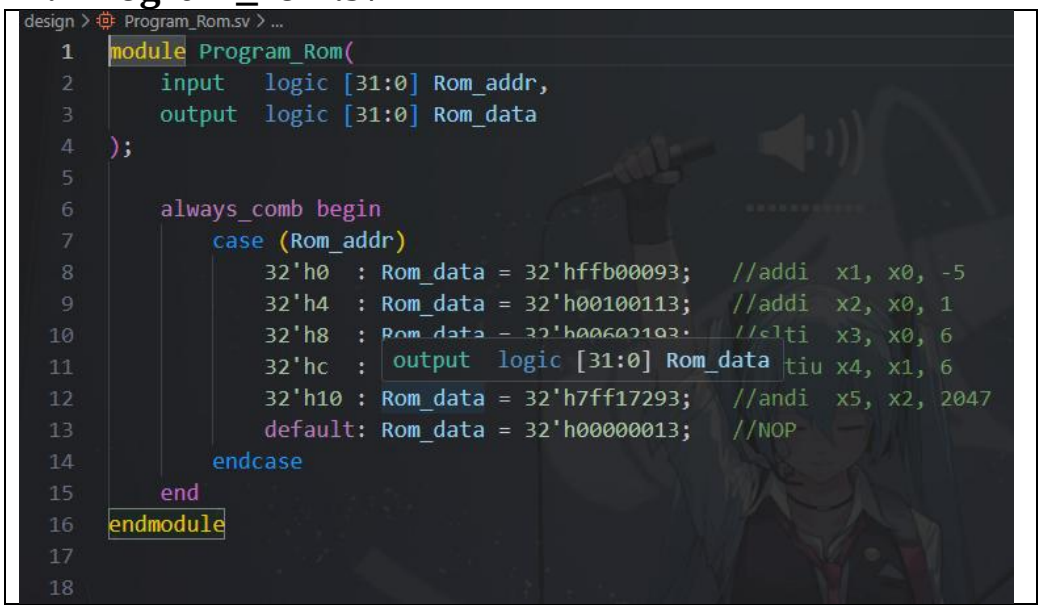


```
58      // Reg file
59      logic write_regf_en_r;
60      logic [4:0] addr_rd_r;
61      logic [31:0] rd_value, rs1_value, rs2_value;
62      Reg_file myreg (
63          .clk (clk),
64          .rst (rst),
65          .write_regf_en (write_regf_en_r),
66          .addr_rd (addr_rd_r),
67          .addr_rs1 (addr_rs1_),
68          .addr_rs2 (addr_rs2_),
69          .rd_value (rd_value),
70          .rs1_value (rs1_value),
71          .rs2_value (rs2_value)
72      );
73
74      // pipe 2 ID_EX
75      logic flush_IDEX_;
76      assign rst_or_flush_IDEX_ = rst | flush_IDEX_;
77      logic [31:0] imm_r, rs1_value_r, rs2_value_r;
78      always_ff @(posedge clk) begin
79          if (rst_or_flush_IDEX_) begin
80              write_regf_en_r <= 0;
81              addr_rd_r <= 0;
82              imm_r <= 0;
83              rs1_value_r <= 0;
84              rs2_value_r <= 0;
85          end
86      end
```



```
87
88      // controller
89      logic sel_pc;
90      controller mycontroller(
91          // in
92          .funct7_ (funct7_),
93          .opcode_ (opcode_),
94          .rst (rst),
95          .clk (clk),
96          .funct3_ (funct3_),
97          // out
98          .flush_IFID_ (flush_IFID_),
99          .flush_IDEX_ (flush_IDEX_),
100         .rst_pc_ (rst_pc_),
101         .sel_pc (sel_pc)
102     );
103 endmodule
```

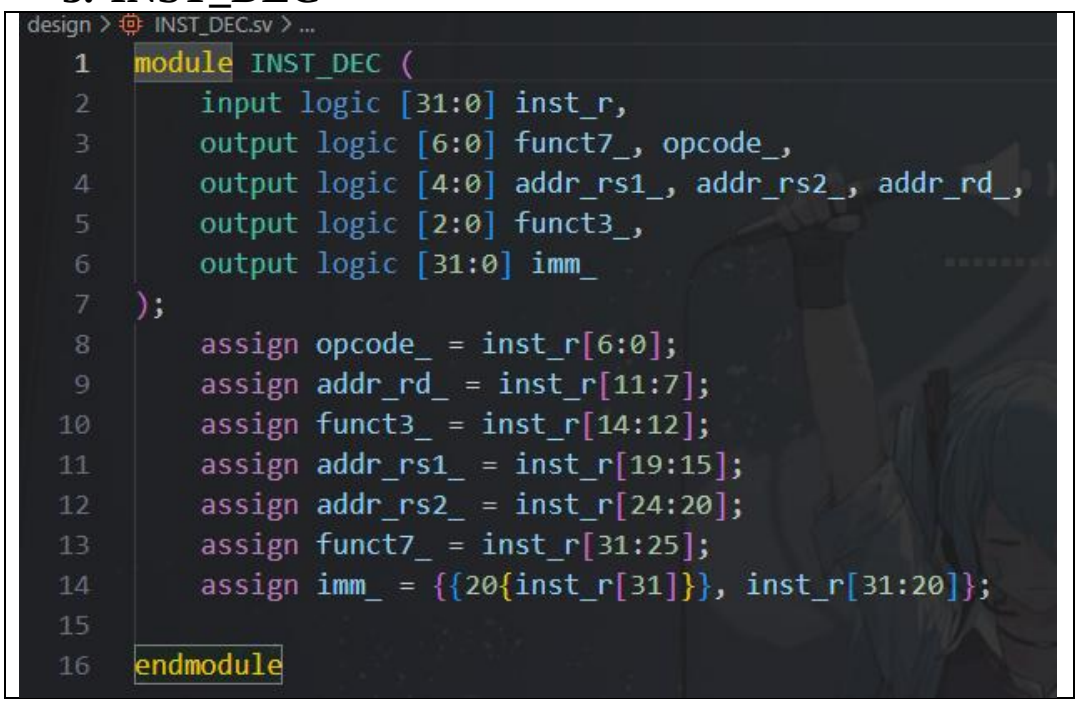
2. Program_rom.sv



```
design > Program_Rom.sv > ...
1  module Program_Rom(
2      input   logic [31:0] Rom_addr,
3      output  logic [31:0] Rom_data
4  );
5
6      always_comb begin
7          case (Rom_addr)
8              32'h0  : Rom_data = 32'hffb00093;   //addi  x1, x0, -5
9              32'h4  : Rom_data = 32'h00100113;   //addi  x2, x0, 1
10             32'h8  : Rom_data = 32'h00602193;   //slti  x3, x0, 6
11             32'hc  : Rom_data = 32'h0061b213;   //sltiu x4, x1, 6
12             32'h10 : Rom_data = 32'h7ff17293;   //andi  x5, x2, 2047
13             default: Rom_data = 32'h00000013;   //NOP
14         endcase
15     end
16 endmodule
17
18
```

output logic [31:0] Rom_data

3. INST_DEC



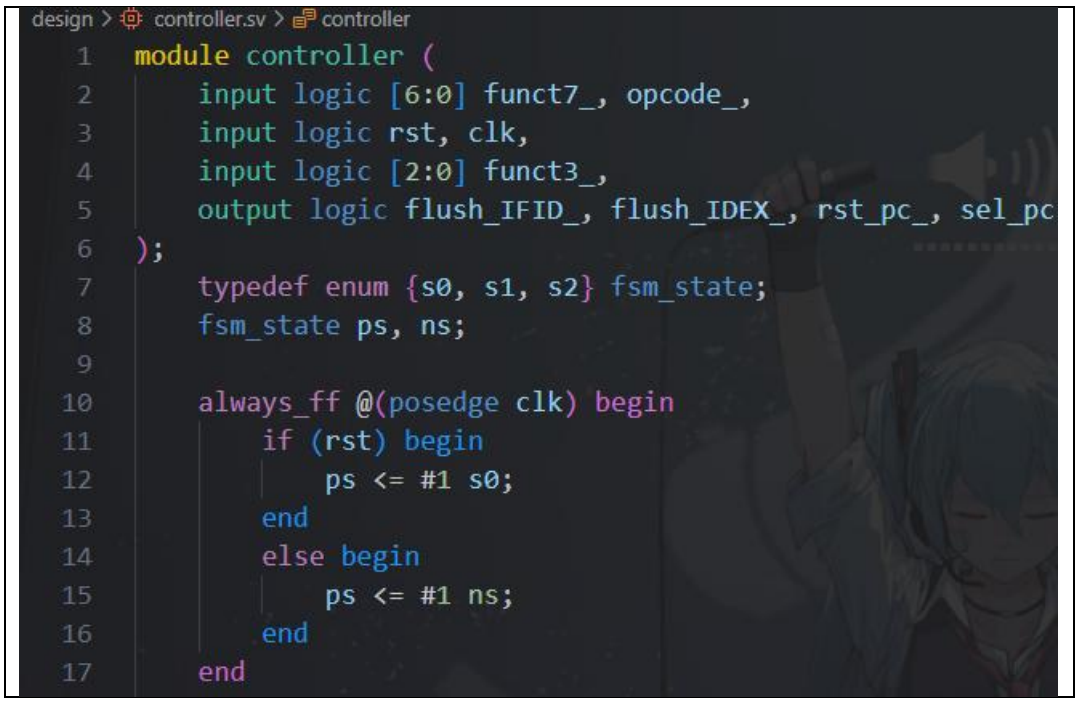
```
design > INST_DEC.sv > ...
1  module INST_DEC (
2      input logic [31:0] inst_r,
3      output logic [6:0] funct7_, opcode_,
4      output logic [4:0] addr_rs1_, addr_rs2_, addr_rd_,
5      output logic [2:0] funct3_,
6      output logic [31:0] imm_
7  );
8      assign opcode_ = inst_r[6:0];
9      assign addr_rd_ = inst_r[11:7];
10     assign funct3_ = inst_r[14:12];
11     assign addr_rs1_ = inst_r[19:15];
12     assign addr_rs2_ = inst_r[24:20];
13     assign funct7_ = inst_r[31:25];
14     assign imm_ = {{20{inst_r[31]}}, inst_r[31:20]};
15
16 endmodule
```

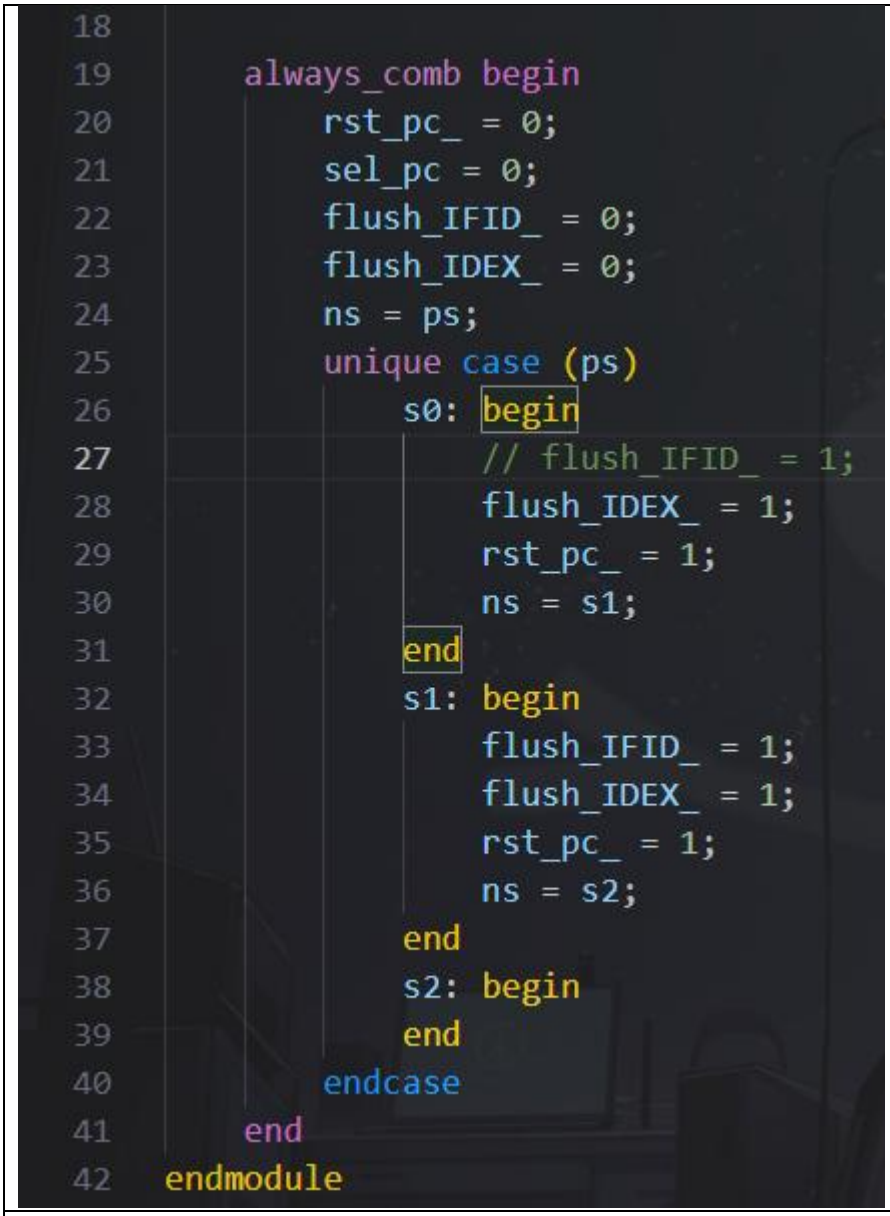
4. Reg_file

design > Reg_file.sv > ...

```
1  module Reg_file(
2      input   logic clk,
3      input   logic rst,
4      input   logic write_regf_en,
5      input   logic [4:0] addr_rd,
6      input   logic [4:0] addr_rs1,
7      input   logic [4:0] addr_rs2,
8      input   logic [31:0] rd_value,
9
10     output  logic [31:0] rs1_value,
11     output  logic [31:0] rs2_value
12 );
13
14     logic [31:0] regs[0:31];
15     logic addr_rd_not_0;
16     integer i;
17
18     assign addr_rd_not_0 = |addr_rd;
19
20     assign rs1_value = regs[addr_rs1];
21     assign rs2_value = regs[addr_rs2];
22
23     always_ff@(posedge clk)
24     begin
25         if(rst) begin
26             for(i = 0; i < 32; i = i+1) begin:rst_keywords
27                 regs[i] <= 0;
28             end
29         end
30         else begin
31             // Write
32             if (write_regf_en && addr_rd_not_0)
33                 regs[addr_rd] <= #1 rd_value;
34         end
35     end
36
37 endmodule
```

5. controller.sv



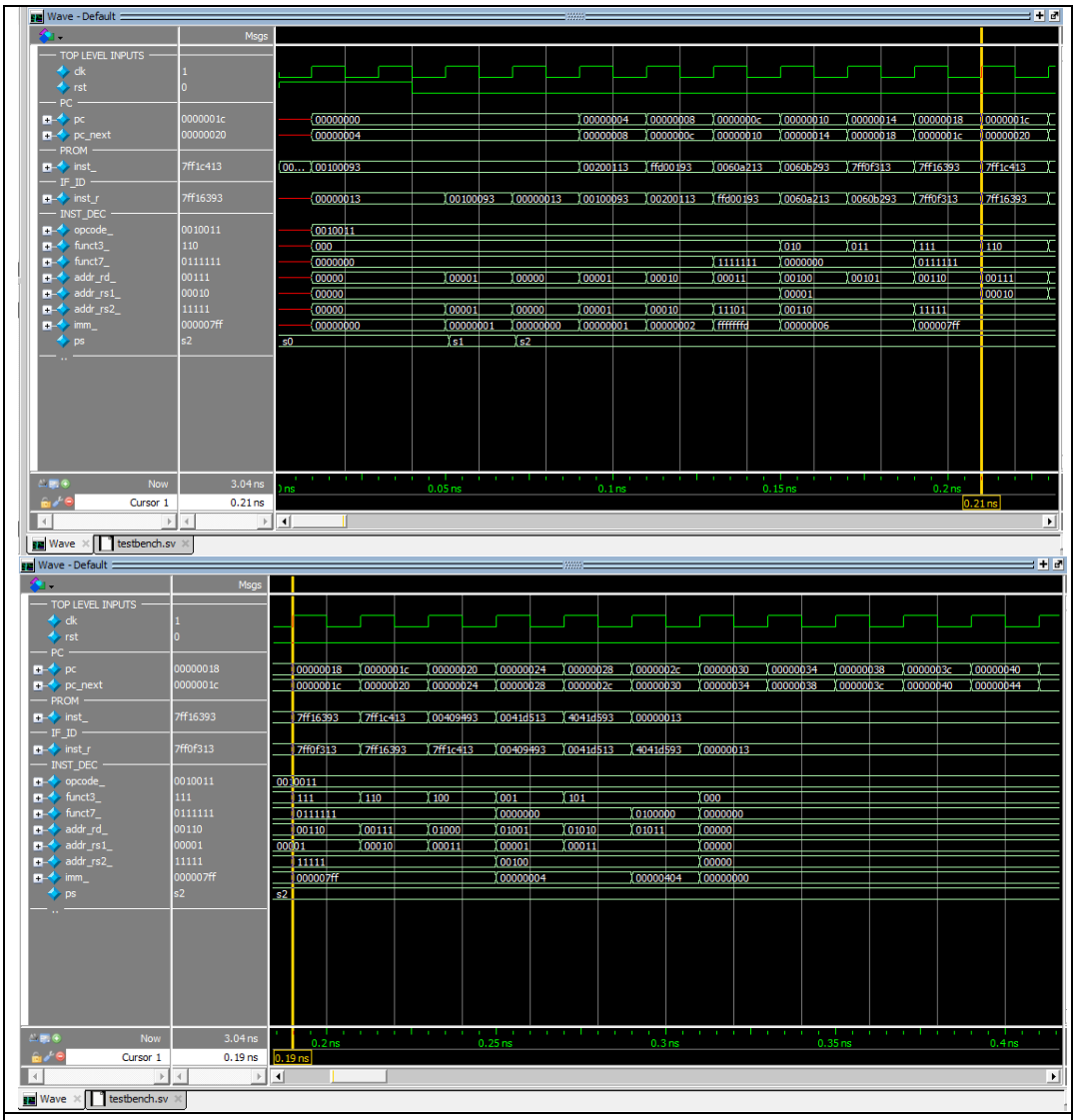


```
18
19      always_comb begin
20          rst_pc_ = 0;
21          sel_pc = 0;
22          flush_IFID_ = 0;
23          flush_IDEX_ = 0;
24          ns = ps;
25          unique case (ps)
26              s0: begin
27                  // flush_IFID_ = 1;
28                  flush_IDEX_ = 1;
29                  rst_pc_ = 1;
30                  ns = s1;
31              end
32              s1: begin
33                  flush_IFID_ = 1;
34                  flush_IDEX_ = 1;
35                  rst_pc_ = 1;
36                  ns = s2;
37              end
38              s2: begin
39              end
40          endcase
41      end
42  endmodule
```

可以看到與上課 **demo** 的主要差別為，狀態 **s1** 的 **flush_IFID_ = 1;**被註解，改取為 **default** 的 **flush_IFID_ = 0;**。

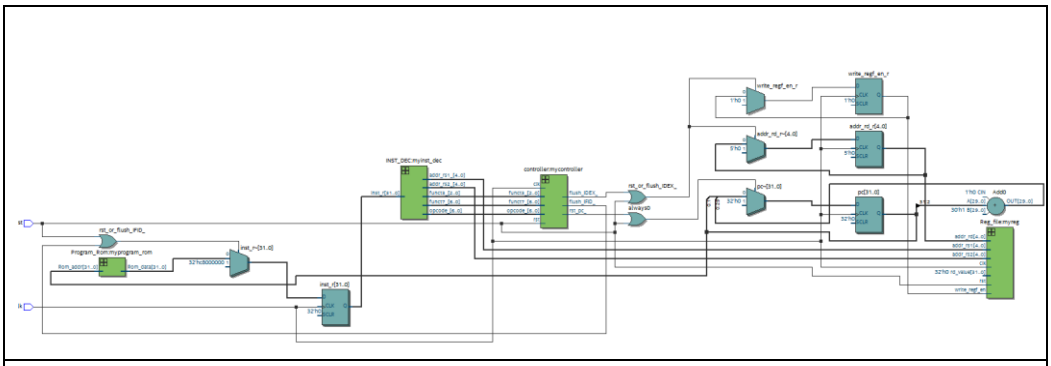
這原因是觀察附的波形圖，在 **s1** 時 **inst_r** 顯示不為 **13 (I_NOP** 之值**)**，即在 **s1** 狀態未 **reset**。而 **controller** 不能控制 **rst** 線，只能控制一同 **or** 的 **flush_IFID_**，且 **rst** 此時為 **0**，即只要 **flush_IFID_**不為 **1** 就不會是 **13**。

●模擬結果與結果說明：



上圖為證明與助教給的範例圖相同(含轉換禁制、加上 **Divider**、照例圖由上到排序波型、**rst** 位置為 **#40)**。

- 架構圖：



就 DFF 來說，我起碼還看得出來是兩層 Pipe，還透過自己命名的 module，大致上可以確認結構跟例圖給的有 87%像。

●結論與心得：

經過此次的實驗，我學會寫 cpu 中的 fetch cycle，刻完了 cpu 大致上的架構(應該：P)，但從 demo 的情況~我學習狀況大概應該算是相當慘烈的。

但我只能說，牢我還是會做穿的，我大概也就只能每次慢慢的把進度追上，慢慢習慣寫硬體的步調，感謝助教跟朋友每次耐心的解決的紅色的線跟問題。