VLSI Introduction Final Project

徐聖凱、鄭淑容、陳林志洋

完整程式碼儲存於：https://github.com/TigerXSK/VLSI_Final_Submit_ver.git

模擬軟體：Vivado, HDLBits, VSCode

- **Module**

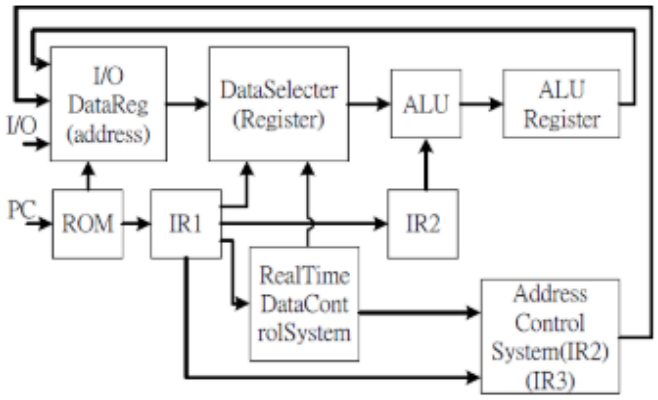
1. **DataReg(address)**
2. **Data Selector System**
3. **Address Control System**
4. **Stack Index**
5. **Stack System**
6. **Program Counter 1**
7. **Program Counter 2**
8. **Condition Code System**
9. **Control System 1**

- **Designing Process**

我們把 Microprocessor Architecture，分為好幾個 Submodule 對 Chat GPT 做訪問。搭配老師所提供的範例 Code，以人工的方式，判斷生成的 Verilog Code 是否存在錯誤。包括 testbench ，有的時候 chatgpt(4o)未必能生成出邏輯正確的 Design 和 Testbech。我們判定 4o 模型無法有效的整合 sequential 和 combinational 電路來進行邏輯和資料更新之間的模擬。但是對於撰寫方式，ChatGPT 確實給了我們很大的啓發，我們可以借由 o1 生成出來的範例程式碼，按照其概念表達出我們自己想要的邏輯。

驗證 Design 時，我們使用 Vivado 和 HDLBits 進行模擬。兩者各有優缺點。Vivado 的優點是功能比較完整，其中尤其是波形圖的部分，提供我們一種最重要的 debug 方法。然而 HDLBits 的功能就比較簡單 ，因爲他主打的就是一個對於新人一個低成本的學習網站。他的優點就是將 module 全部貼上去就可以進行模擬並輸出。然而缺點是要遵守他的 module 命名規則，如 top module 只能取名 top_module()，還有模擬的時候沒有提示模擬狀態，而有些情況會導致難以 debug，

由於訪問的內容過多，我們以 PC1 作為訪問 Chat GPT 的示例：



我要做一個 Microprocessor Architecture ，架構如附圖，請幫我生出 Verilog code，我會給你各個架構的input output 以及其Truth table。

✓ Submodule PC1

- **文字詢問**

幫我做一個submodule PC1，他的input output等，用verilog code 可以表示如下， input [11:0] PCounter , A1; input jump; output[11:0] PCadd; reg[11:0] PCadd; jump為0時請幫我A1 dont care，讓Pcadd=Pccounter+1 jump為1時請幫我A1 assingn為B 然後讓 PCadd=PCcounter+B

```
module PC1(
    input [11:0] PCounter,
    input [11:0] A1,
    input jump,
    output reg [11:0] PCadd
);

    reg [11:0] B; // Internal register to hold A1 when jump is 1

    always @(*) begin
        if (jump) begin
            B = A1; // Assign A1 to B when jump is 1
            PCadd = PCounter + B;
        end else begin
            PCadd = PCounter + 12'd1; // Increment PCounter by 1 when jump is 0
        end
    end

endmodule
```

- Truth Table 詢問

Jump enable	Input		Output
jump	PCcounter	A1	PCadd
0	PC	X	PC+1
1	PC	B	PC+B

PC1的Truth table 長這樣，請再幫我撰寫一次

```
module PC1(
    input [11:0] PCounter,
    input [11:0] A1,
    input jump,
    output reg [11:0] PCadd
);

    always @(*) begin
        if (jump) begin
            PCadd = PCounter + A1; // When jump is 1, add PCounter and A1
        end else begin
            PCadd = PCounter + 12'd1; // When jump is 0, increment PCounter by 1
        end
    end

endmodule
```

✓ Testbench of Submodule PC1

- 首度詢問

```
module pcounter_v1 (PCounter, B, jump, PCadd);
input [11:0] PCounter, B;
input jump;
output [11:0] PCadd;
reg [11:0] PCadd;

always @(jump or B)
begin
  if (jump == 0)
    PCadd = PCounter + 1;
  else
    PCadd = PCounter + B;
end

endmodule
```

幫我寫一個testbench，滿足這兩個測資pccount=10 jump=0 B=5 則PCadd要等於11、pccount=20 jump=1 B=10則PCadd要等於30。通過則顯示pass，不通過則顯示fail

```
module pcounter_v1_tb;
    reg [11:0] PCounter, B;
    reg jump;
    wire [11:0] PCadd;

    // Instantiate the DUT (Device Under Test)
    pcounter_v1 uut (
        .PCounter(PCounter),
        .B(B),
        .jump(jump),
        .PCadd(PCadd)
    );

    initial begin
        // Test Case 1
        PCounter = 12'd10; // Set PCounter to 10
        B = 12'd5;         // Set B to 5
        jump = 1'b0;       // Set jump to 0
        #10;               // Wait for 10 time units
        if (PCadd == 12'd11)
            $display("Test Case 1: PASS");
        else
            $display("Test Case 1: FAIL - Expected: 11, Got: %d", PCadd);

        // Test Case 2
        PCounter = 12'd20; // Set PCounter to 20
        B = 12'd10;        // Set B to 10
        jump = 1'b1;       // Set jump to 1
        #10;               // Wait for 10 time units
        if (PCadd == 12'd30)
            $display("Test Case 2: PASS");
        else
            $display("Test Case 2: FAIL - Expected: 30, Got: %d", PCadd);

        $finish; // End simulation
    end
endmodule
```

- 二次詢問

```
module pcounter_v1_tb;
    reg [11:0] PCounter, B;
    reg jump;
    wire [11:0] PCadd;

    // Instantiate the DUT (Device Under Test)
    pcounter_v1 uut (
        .PCounter(PCounter),
        .B(B),
        .jump(jump),
        .PCadd(PCadd)
    );

    initial begin
        // Test Case 1
        PCounter = 12'd10; // Set PCounter to 10
        B = 12'd5;         // Set B to 5
        jump = 1'b0;       // Set jump to 0
        #10;               // Wait for 10 time units
        if (PCadd == 12'd11)
            $display("Test Case 1: PASS");
        else
            $display("Test Case 1: FAIL - Expected: 11, Got: %d", PCadd);

        // Test Case 2
        PCounter = 12'd20; // Set PCounter to 20
        B = 12'd10;        // Set B to 10
        jump = 1'b1;       // Set jump to 1
        #10;               // Wait for 10 time units
        if (PCadd == 12'd30)
            $display("Test Case 2: PASS");
        else
            $display("Test Case 2: FAIL - Expected: 30, Got: %d", PCadd);

        $finish; // End simulation
    end
endmodule
```

✓ **GPT 行爲分析**

(1) ChatGPT 列出不同問法(prompt)的 Code，並比較每種問法產生的 Verilog Code 結果有何不同？

當以文字敘述 (例如「assign B = A1 的動作」) 請求 ChatGPT 生成 Verilog Code 時，因為描述可能過於簡單或缺乏邏輯細節，生成的代碼容易遺漏一些重要條件，尤其是在邏輯複雜的情況下更為明顯。相比之下，提供 Truth Table 則能明確列出所有輸入與輸出對應，這種具體的結構能幫助模型更準確地理解並生成代碼，減少邏輯遺漏的風險。

(2) 哪一種問法（prompt）所產生的結果比較正確？

直接提供 Truth Table 所產生的 Verilog Code 正確率較高，因為 Truth Table 是一種結構化且無歧義的表示方式，清晰展現了輸入與輸出間的對應關係。相比之下，文字敘述容易因資訊不完整或語意模糊導致誤解，因此在要求準確性時，Truth Table 是更可靠的選擇。

(3) Testbench 的正確性

對於複雜邏輯，ChatGPT 初次生成的 Testbench 通常不會完全正確，因為測試條件和邊界情況可能沒有完全覆蓋。然而，通過反覆提示與調整，用戶可以逐步改進生成的 Testbench，使其更接近正確。實際模擬與驗證仍是最終確認正確性的關鍵。

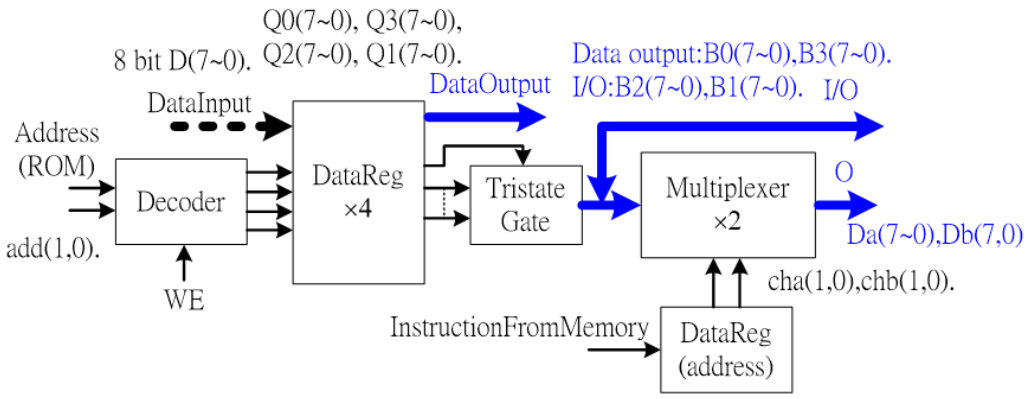
(4) 詢問的語法正確與否

若提示的語法存在疏漏或描述不清，ChatGPT 很可能生成不符合預期的代碼。清晰且邏輯正確的描述能有效提升生成結果的準確性，因此在編寫提示時應盡量表達完整，避免語意模糊或過於簡略。

(5) 重複問同樣問題是否有一致的答案？

重複問相同的問題時，ChatGPT 通常會生成一致的 Verilog Code，邏輯部分大致相同，但註解內容可能因表達方式不同而略有變化。若需檢查邏輯一致性，建議側重於代碼本身的功能，而非註解的差異。

- 模擬結果
  - ✓ DataReg(address)



- Design:

```
module io_v1 (D, B1, B2, addr, cha, chb, clk, WE, B0, B3, Da, Db);
    input [7:0] D;
    input [7:0] B1, B2;
    input [1:0] addr, cha, chb;
    input clk, WE;
    output [7:0] B0, B3, Da, Db;

    reg [7:0] Q3, Q2, Q1, Q0;
    wire [7:0] K1, K2;
    reg [1:0] choicetema, choicetemb;
    reg [7:0] Da, Db;

    // Decoder + Data register
    always @(posedge clk) begin
        if (WE) begin
            case (addr)
                2'b00: Q0 <= D;
                2'b01: Q1 <= D;
                2'b10: Q2 <= D;
                2'b11: Q3 <= D;
            endcase
        end else begin
            Q0 <= Q0;
            Q1 <= Q1;
            Q2 <= Q2;
            Q3 <= Q3;
        end
    end
```

```
// Tristate buffer and I/O
buffif1 t1_0(.y(K1[0]), .a(Q1[0]), .en(Q0[0]));
buffif1 t1_1(.y(K1[1]), .a(Q1[1]), .en(Q0[0]));
buffif1 t1_2(.y(K1[2]), .a(Q1[2]), .en(Q0[0]));
buffif1 t1_3(.y(K1[3]), .a(Q1[3]), .en(Q0[0]));
buffif1 t1_4(.y(K1[4]), .a(Q1[4]), .en(Q0[0]));
buffif1 t1_5(.y(K1[5]), .a(Q1[5]), .en(Q0[0]));
buffif1 t1_6(.y(K1[6]), .a(Q1[6]), .en(Q0[0]));
buffif1 t1_7(.y(K1[7]), .a(Q1[7]), .en(Q0[0]));

buffif1 t2_0(.y(K2[0]), .a(Q2[0]), .en(Q0[1]));
buffif1 t2_1(.y(K2[1]), .a(Q2[1]), .en(Q0[1]));
buffif1 t2_2(.y(K2[2]), .a(Q2[2]), .en(Q0[1]));
buffif1 t2_3(.y(K2[3]), .a(Q2[3]), .en(Q0[1]));
buffif1 t2_4(.y(K2[4]), .a(Q2[4]), .en(Q0[1]));
buffif1 t2_5(.y(K2[5]), .a(Q2[5]), .en(Q0[1]));
buffif1 t2_6(.y(K2[6]), .a(Q2[6]), .en(Q0[1]));
buffif1 t2_7(.y(K2[7]), .a(Q2[7]), .en(Q0[1]));

assign B0 = Q0; // Data output
assign B1 = K1; // I/O
assign B2 = K2; // I/O
assign B3 = Q3; // Data output

// Address register and I/O
always @(posedge clk) begin
    choicetema <= cha;
    choicetemb <= chb;
end
always @(choicetema or Q3 or K2 or K1 or Q0) begin
    case (choicetema)
        2'b00: Da <= Q0;
        2'b01: Da <= K1;
        2'b10: Da <= K2;
        2'b11: Da <= Q3;
    endcase
end
// Data register and I/O
always @(choicetemb or Q3 or K2 or K1 or Q0) begin
```

```
        case (choicetemb)
            2'b00: Db <= Q0;
            2'b01: Db <= K1;
            2'b10: Db <= K2;
            2'b11: Db <= Q3;
        endcase
    end

endmodule

module buffif1 (
    y,
    a,
    en
);
    output reg y;
    input wire a;
    input wire en;

    always @(*) begin
        if (en) begin
            y = a;
        end else begin
            y = 1'bz;
        end
    end
endmodule
```

- Testbench:

  Overall:

  四個 test_case，分別屬 truth table 的四個 case

```v
//==============================================//
//      ioi_v1_tb_n.v testbench by 徐聖凱        //
//==============================================//
`define CYCLE_TIME 10

module IO_PATTERN(
    //Output Port
    clk,
    rst_n,
    D,
    B1,
    B2,
    addr,
    cha,
    chb,
    WE,
    //Input Port
    B0,
    B3,
    Da,
    Db
);
//==============================================//
//          Input & Output Declaration          //
//==============================================//
output reg clk, rst_n;
output reg [7:0] D;
output reg [7:0] B1, B2;
output reg [1:0] addr, cha, chb;
output reg WE;

input [7:0] B0, B3, Da, Db;

io_v1 u_io_v1 (
    .D(D),
    .B1(B1),
    .B2(B2),
    .addr(addr),
    .cha(cha),
    .chb(chb),
    .clk(clk),
    .WE(WE),
    .B0(B0),
    .B3(B3),
    .Da(Da),
    .Db(Db)
);
```

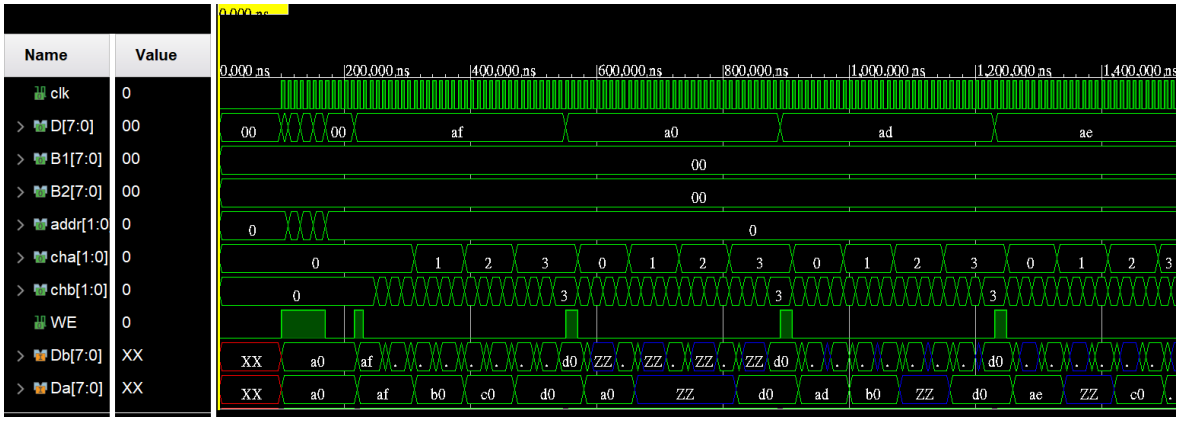
```v
//==============================================//
//               Parameter & Integer            //
//==============================================//
integer CYCLE = `CYCLE_TIME;
integer i;

//==============================================//
//                  Main Function               //
//==============================================//
initial begin
    reset_task;

    // Start test cases
    test_case_1; // Q0(0)=1, Q0(1)=1
    test_case_2; // Q0(0)=0, Q0(1)=0
    test_case_3; // Q0(0)=1, Q0(1)=0
    test_case_4; // Q0(0)=0, Q0(1)=1

    pass_task;
end
```

```v
//==============================================//
//           Clock and Reset Function           //
//==============================================//
// clock
always #(CYCLE/2) clk = ~clk;

task reset_task; begin ···
end endtask

//==============================================//
//              Test Cases                      //
//==============================================//

task test_case_1; begin // Q0(0)=1, Q0(1)=1 (Q0 = AF) ···
end endtask

task test_case_2; begin // Q0(0)=0, Q0(1)=0 (Q0 = A0) ···
end endtask

task test_case_3; begin // Q0(0)=1, Q0(1)=0 (Q0 = AD) ···
end endtask

task test_case_4; begin // Q0(0)=0, Q0(1)=1 (Q0 = AE) ···
end endtask

//==============================================//
//          Fail and Pass Functions             //
//==============================================//
task fail_task(input [255:0] msg, input [255:0] sig, input [15:0] expected, input [15:0] actual); begin ···
end endtask

task pass_task; begin
    $display("============================================================");
    $display("                  Congratulations!!");
    $display("                All Test Cases Pass");
    $display("============================================================");
    $finish;
end endtask

endmodule
```

reset_task: 主要是將 Q0 Q1 Q2 Q3 分別先給定一個值，否則每一次 task 都要花費大量週期初始化資料。

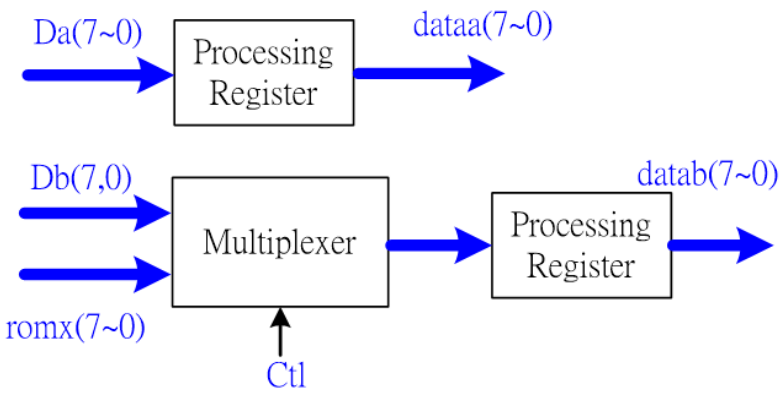
Test_case: 將 truth table 完整實現，匹配 input(cha, chb)和 output 的每一個情況(共 16 種)

- **Simulation Result:**




```
# run 2000ns
=> Test Case 1 Passed!
=> Test Case 2 Passed!
=> Test Case 3 Passed!
=> Test Case 4 Passed!
============================================================
                    Congratulations!!
                    All Test Cases Pass
============================================================
```

✓ **Data Selector System**



- **Design:**

```
module direct_v1 (Da, Db, romx, clk, direct, dataa, datab);
input [7:0] Da, Db, romx;
input clk, direct;
output [7:0] dataa, datab;
reg [7:0] dataa, datab;

always @(posedge clk)
begin
    dataa <= Da;
end

always @(posedge clk)
begin
    case (direct)
        0: datab <= Db;
        1: datab <= romx;
    endcase
end

endmodule
```

- **Testbench**

```
`timescale 1ps/1ps

module top_module();
reg [7:0] Da, Db, romx;
reg clk, direct;
wire [7:0] dataa, datab;

direct_v1 uut (
    .Da(Da),
    .Db(Db),
    .romx(romx),
    .clk(clk),
    .direct(direct),
    .dataa(dataa),
    .datab(datab)
);

always #5 clk = ~clk;
initial clk = 0;

initial begin
    //case 1 direct = 0
    direct = 0;
    Da = 8'd10;
    Db = 8'd20;
    romx = 8'd5;
    #5;
    #1;
    if(dataa == Da && datab == Db)
    $display ("Time = %0t | direct = %d, Da = %d, Db= %d, romx= %d, dataa= %d,
datab= %d, passed!", $time, direct, Da, Db, romx, dataa, datab);
    else $display ("Time = %0t | direct = %d, Da = %d, Db= %d, romx= %d,
dataa= %d, datab= %d, failed at case 1 !", $time, direct, Da, Db, romx, dataa,
datab);
```

```
    #5;
    direct = 1;
    Da = 8'd15;
    Db = 8'd20;
    romx = 8'd10;

    #5;
    if(dataa == Da && datab == romx)
    $display ("Time = %0t | direct = %d, Da = %d, Db= %d, romx= %d, dataa= %d,
datab= %d, passed at case 2 !", $time, direct, Da, Db, romx, dataa, datab);
    else $display ("Time = %0t | direct = %d, Da = %d, Db= %d, romx= %d,
dataa= %d, datab= %d, failed at case 2 !", $time, direct, Da, Db, romx, dataa,
datab);
    $stop;
    end

endmodule
```

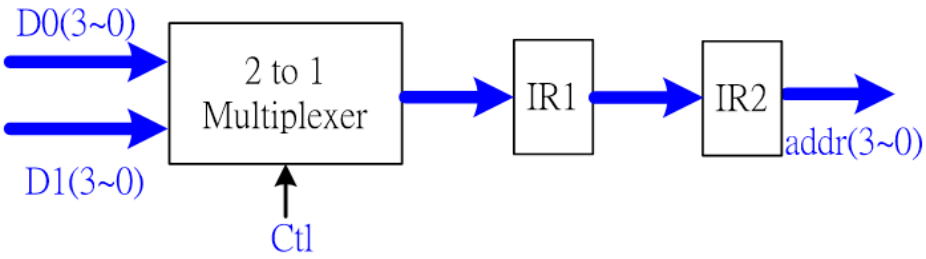
- **Simulation Result:**

```
Running Icarus Verilog simulator...
VCD info: dumping is suppressed.
Time = 6 | direct = 0, Da =  10, Db=  20, romx=   5, dataa=  10, datab=  20, passed!
Time = 16 | direct = 1, Da =  15, Db=  20, romx=  10, dataa=  15, datab=  10, passed at case 2 !
./top_module.v:43: $stop called at 16 (1ps)
Hint: Total mismatched samples is 0 out of 0 samples
Simulation finished at 16 ps
Mismatches: 0 in 0 samples
```

Truth table

Input						Output
Clk	Ctl	Da	Db	romx	dataa	datab
↑	0	A	B	X	A	B
↑	1	A	X	C	A	C

✓ **Address Control System**



- **Design:**

```
module addr_v1 (D0, D1, clk, direct, addr);
input [3:0] D0, D1;
input clk, direct;
output [3:0] addr;
reg [3:0] addr, temp1, temp2;

always @(direct or D0 or D1)
begin
   case (direct)
      0 : temp1 <= D0;
      1 : temp1 <= D1;
   endcase
end

always @(posedge clk)
begin
   addr <= temp2;
   temp2 <= temp1;
end

endmodule
```

- **Testbench:**

```
`timescale 1ps/1ps

module top_module();
    reg clk, direct;
    reg [3:0] D0, D1;
    wire [3:0] addr;

    addr_v1 uut(
        .clk(clk),
        .direct(direct),
        .D0(D0),
        .D1(D1),
        .addr(addr)
    );



    always #5 clk = ~clk;
    initial clk = 0;

    initial begin
        direct = 0;
        D0 = $random % 16;
        D1 = $random % 16;
        @(posedge clk);
        test;

        repeat (4) begin
            @(negedge clk);
                D0 = $random % 16;
                D1 = $random % 16;
                direct = ~direct;
            @(posedge clk);
                test;
```

```
        end
        $finish;
    end

    task test;
        begin
            #10;
            #1;
            if ((direct == 0 && addr == D0) || (direct == 1 && addr == D1)) begin
                $display("passed! Time = %0t | direct = %d, addr = %d, D0 = %d, D1 =
%d", $time, direct, addr, D0, D1);
            end else begin
                $display("Time = %0t | direct = %d, addr = %d,   D0 = %d, D1 = %d,
failed! Expected = %d", $time, direct, addr, D0, D1, (direct ? D1 : D0));
            end
        end
    endtask
endmodule
```

- **Simulation result:**

iverilog — Compile and simulate

Running Icarus Verilog compile. Show Icarus Verilog compile messages...
Running Icarus Verilog simulation. Show Icarus Verilog simulation messages...

```
Running Icarus Verilog simulator...
VCD info: dumping is suppressed.
passed! Time = 16 | direct = 0, addr =  4, D0 =  4, D1 =  1
passed! Time = 36 | direct = 1, addr =  3, D0 =  9, D1 =  3
passed! Time = 56 | direct = 0, addr = 13, D0 = 13, D1 = 13
passed! Time = 76 | direct = 1, addr =  2, D0 =  5, D1 =  2
passed! Time = 96 | direct = 0, addr =  1, D0 =  1, D1 = 13
./top_module.v:37: $finish called at 96 (1ps)
Hint: Total mismatched samples is 0 out of 0 samples
Simulation finished at 96 ps
Mismatches: 0 in 0 samples
```

Truth table IR1

IR1 input				IR1 output
Clk	D0	D1	Ctl	
↑	D0	X	0	D0
↑	X	D1	1	D1

Truth table IR2

Clk	IR2 input	IR2 output	
↑	IR1 output	addr(3~2)	addr(1~0)
↑	A	A(3~2)	A(1~0)

- ✓ **Stack index**
- **Design:**

```
module sp_v1 (Clk, pop, push, sp,sp_in);
input Clk, pop, push;
input [1:0] sp_in;
output [1:0] sp;
reg [1:0] sp;

always @(posedge Clk)
begin
   if (pop)
      sp <= sp_in + 1;
   else if (push)
      sp <= sp_in - 1;
   else sp<= sp_in;
end

endmodule
```

- **Testbench:**

```
`timescale 1ps/1ps

module top_module ();
reg Clk, pop, push;
reg [1:0]sp_in;
wire [1:0] sp;

sp_v1 uut(
    .Clk(Clk),
    .pop(pop),
    .push(push),
    .sp_in(sp_in),
    .sp(sp)
);

always #5 Clk = ~Clk;
initial Clk = 0;

initial begin
    //case 1 pop=1 push =0
    sp_in =2'd0;
    pop = 1;
    push = 0;
    @(posedge Clk);
    #1;
    if(sp == sp_in+1)
    $display ("pass case 1, pop = %d , push = %d , sp_in =%d , sp =%d", pop, push,
sp_in, sp);
    else $display ("Failed!, pop = %d , push = %d , sp_in =%d , sp =%d", pop, push,
sp_in, sp);

    //case 2 pop=0, push=1
    @(negedge Clk);
        sp_in =2'd1;
```

```
        pop = 0;
        push = 1;

    @(posedge Clk);
    #1;
        if(sp == sp_in-1)
        $display ("pass case 2, pop = %d , push = %d , sp_in =%d , sp =%d", pop, push,
sp_in, sp);
        else $display ("Fail case 2, pop = %d , push = %d , sp_in = %d ,sp = %d", pop,
push, sp_in, sp);



    //case 3 pop = 0 , push = 0;
    @(negedge Clk);
        sp_in =2'd2;
        pop = 0;
        push = 0;

    @(posedge Clk);
    #1;
        if(sp == sp_in)
        $display ("pass case 3, pop = %d , push = %d , sp_in =%d , sp =%d", pop, push,
sp_in, sp);
        else $display ("Fail case 3, pop = %d , push = %d , sp_in = %d ,sp = %d", pop,
push, sp_in, sp);

$finish;
end

endmodule
```

- Simulation result:

iverilog — Compile and simulate

Running Icarus Verilog compile. Show Icarus Verilog compile messages...
Running Icarus Verilog simulation. Show Icarus Verilog simulation messages...

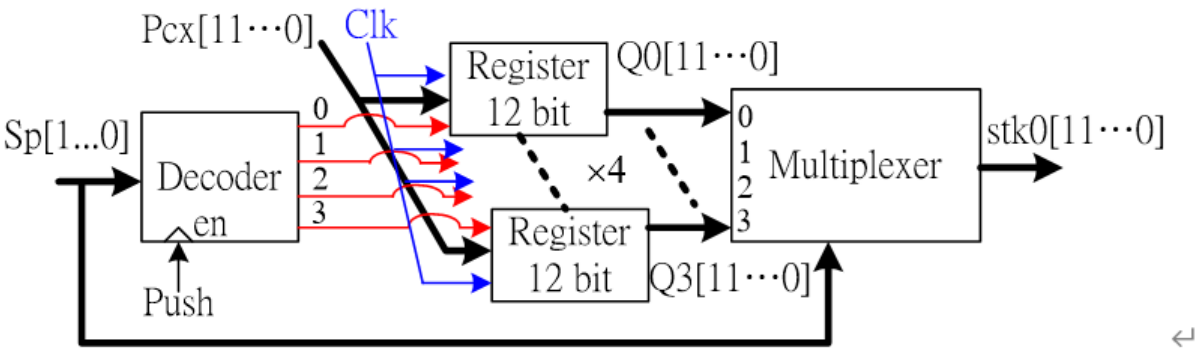
```
Running Icarus Verilog simulator...
VCD info: dumping is suppressed.
pass case 1, pop = 1 , push = 0 , sp_in =0 , sp =1
pass case 2, pop = 0 , push = 1 , sp_in =1 , sp =0
pass case 3, pop = 0 , push = 0 , sp_in =2 , sp =2
./top_module.v:55: $finish called at 26 (1ps)
Hint: Total mismatched samples is 0 out of 0 samples
Simulation finished at 26 ps
Mismatches: 0 in 0 samples
```

Status: Simulation completed

Truth table

Control input			Stack index output
Clk	pop	push	sp
↑	1	X	sp+1
↑	0	1	sp-1
↑	0	0	UC

✓ Stack system



- **Design:**

```
module stk_v1 (pcx, Clk, push, sp, stko);
input [11:0] pcx;
input Clk, push;
input [1:0] sp;
output [11:0] stko;
reg [11:0] Q0, Q1, Q2, Q3;
reg [11:0] stko;

always @(posedge Clk) begin
    if (push) begin
        case (sp)
            2'b00: Q0 <= pcx;
            2'b01: Q1 <= pcx;
            2'b10: Q2 <= pcx;
            2'b11: Q3 <= pcx;
        endcase
    end

    case (sp)
        2'b00: stko <= Q0;
        2'b01: stko <= Q1;
        2'b10: stko <= Q2;
        2'b11: stko <= Q3;
    endcase
end

endmodule
```

- **Testbench:**

```
`timescale 1ns/1ps

module top_module();
    reg [11:0] pcx;
    reg Clk, push;
```

```
reg [1:0] sp;
wire [11:0] stko;

stk_v1 uut (
    .pcx(pcx),
    .Clk(Clk),
    .push(push),
    .sp(sp),
    .stko(stko)
);

always #5 Clk = ~Clk;
initial Clk = 0;

initial begin

    push = 0; sp = 2'b00; pcx = 12'd0;

    // Case 1: push = 1, sp = 0
    @(negedge Clk); push = 1; sp = 2'd0; pcx = 12'd100;
    @(posedge Clk);
    #10;
    #1;
    if (stko == 12'd100)
        $display("Test Case 1 passed: stko = %d", stko);
    else
        $display("Test Case 1 failed: stko = %d", stko);

    // Case 2: push = 1, sp = 1
    @(negedge Clk); sp = 2'd1; pcx = 12'd200;
    @(posedge Clk);
    #10;
    #1;
    if (stko == 12'd200)
        $display("Test Case 2 passed: stko = %d", stko);
```

```
else
    $display("Test Case 2 failed: stko = %d", stko);

// Case 3: push = 1, sp = 2
@(negedge Clk); sp = 2'd2; pcx = 12'd300;
@(posedge Clk);
#10;
#1;
if (stko == 12'd300)
    $display("Test Case 3 passed: stko = %d", stko);
else
    $display("Test Case 3 failed: stko = %d", stko);

// Case 4: push = 1, sp = 3
@(negedge Clk); sp = 2'd3; pcx = 12'd400;
@(posedge Clk);
#10;
#1;
if (stko == 12'd400)
    $display("Test Case 4 passed: stko = %d", stko);
else
    $display("Test Case 4 failed: stko = %d", stko);

// Case 5: push = 0 (保持  stko)
@(negedge Clk); push = 0; sp = 2'd3;
@(posedge Clk);
#10;
#1;
if (stko == 12'd400)
    $display("Test Case 5 passed: stko = %d", stko);
else
    $display("Test Case 5 failed: stko = %d", stko);

// Case 6:  多次写入和读取
@(negedge Clk); push = 1; sp = 2'd0; pcx = 12'd500;
```

```
        @(posedge Clk); #1;
        @(negedge Clk); sp = 2'd1; pcx = 12'd600;
        @(posedge Clk); #1;
        @(negedge Clk); sp = 2'd0;
        @(posedge Clk);
        #1;
        if (stko == 12'd500)
            $display("Test Case 6 passed: stko = %d", stko);
        else
            $display("Test Case 6 failed: stko = %d", stko);

        $finish;
    end
endmodule
```

- **Simulation result:**

iverilog — Compile and simulate

Running Icarus Verilog compile. Show Icarus Verilog compile messages...
Running Icarus Verilog simulation. Show Icarus Verilog simulation messages...

```
Running Icarus Verilog simulator...
VCD info: dumping is suppressed.
Test Case 1 passed: stko =  100
Test Case 2 passed: stko =  200
Test Case 3 passed: stko =  300
Test Case 4 passed: stko =  400
Test Case 5 passed: stko =  400
Test Case 6 passed: stko =  500
./top_module.v:88: $finish called at 136000 (1ps)
Hint: Total mismatched samples is 0 out of 0 samples
Simulation finished at 136000 ps
Mismatches: 0 in 0 samples
```

Status: Simulation completed

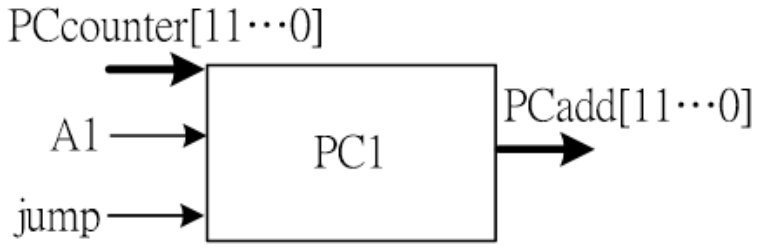
Write-in truth table of the stack system

Write-in							
Data in	Control signal in			Register			
Pcx	Clk	Push	Sp	Q0	Q1	Q2	Q3
A	↑	1	0	A	unchange	unchange	unchange
A	↑	1	1	unchange	A	unchange	unchange
A	↑	1	2	unchange	unchange	A	unchange
A	↑	1	3	unchange	unchange	unchange	A
X	↑	0	X	unchange	unchange	unchange	unchange

Output truth table of the stack system

Control (Sp)	Multiplexer output(stk0)
0	Q0
1	Q1
2	Q2
3	Q3

✓ **Program Counter 1**



- **Design:**

```
module pcounter_v1 (PCounter, B, jump, PCadd);
input [11:0] PCounter, B;
input jump;
output [11:0] PCadd;
reg [11:0] PCadd;

always @(jump or B)
begin
    if (jump == 0)
        PCadd = PCounter + 1;
    else
        PCadd = PCounter + B;
End
```

```
endmodule
```

- **Testbench:**

```
`timescale 1ps/1ps
module top_module ();
reg [11:0] PCounter, B;
reg jump;
wire [11:0] PCadd;

pcounter_v1 uut (
    .PCounter(PCounter),
    .B(B),
    .jump(jump),
    .PCadd(PCadd)
);

initial begin
    //case 1 jump==0
    PCounter = 12'd10;
    jump = 0;
    B = 12'd5;
    #1;
    if(PCadd == PCounter +1)
    $display("passed! case 1: PCcounter = %d, jump = %d, B = %d, PCadd = %d",
PCounter, jump, B, PCadd);
    else $display("fail at case 1 ! PCcounter = %d, jump = %d, B = %d, PCadd = %d",
PCounter, jump, B, PCadd);

    //case 2 jump==1 w/ differ pc & b
    PCounter = 12'd20;
    jump = 1;
    B = 12'd10;
    #1;
    if(PCadd == PCounter +B)
    $display("passed! case 2: PCcounter = %d, jump = %d, B = %d, PCadd = %d",
PCounter, jump, B, PCadd);
```

```
    else $display("fail at case 2 ! PCcounter = %d, jump = %d, B = %d, PCadd = %d",
PCounter, jump, B, PCadd);
    $finish;
end

endmodule
```

- **Simulation result:**

iverilog — Compile and simulate

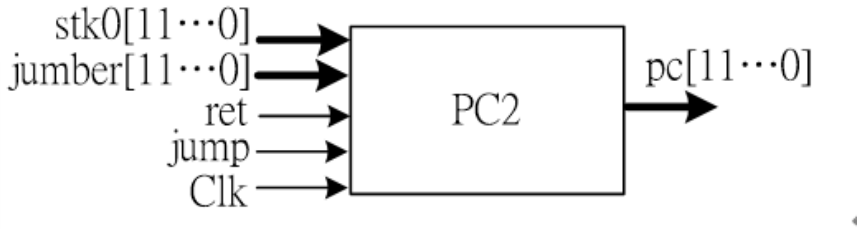
Running Icarus Verilog compile. Show Icarus Verilog compile messages...
Running Icarus Verilog simulation. Show Icarus Verilog simulation messages...

```
Running Icarus Verilog simulator...
VCD info: dumping is suppressed.
passed! case 1: PCcounter =   10, jump = 0, B =    5, PCadd =   11
passed! case 2: PCcounter =   20, jump = 1, B =   10, PCadd =   30
./top_module.v:33: $finish called at 2 (1ps)
Hint: Total mismatched samples is 0 out of 0 samples
Simulation finished at 2 ps
Mismatches: 0 in 0 samples
```

Status: Simulation completed

Jump enable	Input		Output
jump	PCcounter	A1	PCadd
0	PC	X	PC+1
1	PC	B	PC+B

✓ **Program Counter 2**



- **Design:**

```
module pcstk_v1 (clk, jump, ret, jnumber, stko, pc,pc_in);
input [11:0] jnumber, stko, pc_in;
input clk, jump, ret;
output reg [11:0] pc;

always @(posedge clk)
begin
    case (ret)
        0: begin
            if (jump == 0)
                pc <= pc_in + 1;
            else
                pc <= pc_in+ jnumber;
        end
        1: pc <= stko;
    endcase
end

endmodule
```

- **Testbench:**

```
`timescale 1ps/1ps

module top_module ();
reg [11:0] jnumber, stko, pc_in;
reg clk, jump, ret;
wire [11:0] pc;

pcstk_v1 uut(
    .pc_in (pc_in),
    .jnumber(jnumber),
    .stko(stko),
    .clk(clk),
    .jump(jump),
```

```
    .ret(ret),
    .pc(pc)
);

always #5 clk = ~clk;
initial clk = 0;

initial begin
    //case 1 ret = 0, jump = 0
    jump = 0;
    jnumber = 12'd5;
    ret = 0;
    stko = 12'd10;
    pc_in = 12'd20;

    @(posedge clk); #1;
    if(pc == pc_in +1 ) $display ("Passed case 1! , pc_in = %d, pc = %d, stko = %d,
jnumber = %d",pc_in, pc,stko,jnumber) ;
    else $display ("Failed at case 1! , pc_in = %d, pc = %d, ret = %d, jump = %d",pc_in,
pc, ret, jump) ;

    //case 2 ret = 0, jump = 1
    @(negedge clk);
    jump = 1;

    @(posedge clk); #1;
    if(pc == pc_in + jnumber) $display ("Passed case 2! , pc_in = %d, pc = %d, stko =
%d, jnumber = %d",pc_in, pc,stko,jnumber) ;
    else $display ("Failed at case 2! , pc_in = %d, pc = %d, ret = %d, jump = %d,
jnumber = %d ",pc_in, pc, ret, jump, jnumber) ;

    //case 3 ret = 1
    @(negedge clk);
    ret = 1; jump = 1;
```

```
    @(posedge clk); #1;
    if(pc == stko) $display ("Passed case 3! , pc_in = %d, pc = %d, stko = %d, jnumber = %d",pc_in, pc,stko,jnumber) ;
    else $display ("Failed at case 3! , pc_in = %d, pc = %d, ret = %d, stko = %d ",pc_in, pc, ret, stko) ;

$finish;
end

endmodule
```

- **Simulation result**

iverilog — Compile and simulate

Running Icarus Verilog compile. Show Icarus Verilog compile messages...
Running Icarus Verilog simulation. Show Icarus Verilog simulation messages...

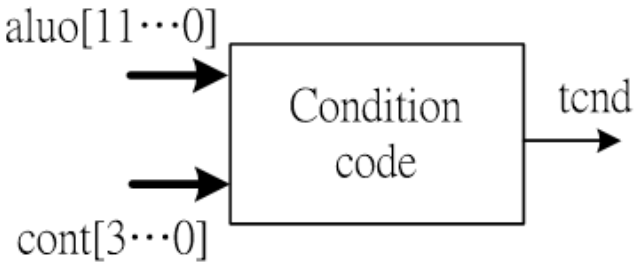
```
Running Icarus Verilog simulator...
VCD info: dumping is suppressed.
Passed case 1! , pc_in =   20, pc =   21, stko =   10, jnumber =    5
Passed case 2! , pc_in =   20, pc =   25, stko =   10, jnumber =    5
Passed case 3! , pc_in =   20, pc =   10, stko =   10, jnumber =    5
./top_module.v:49: $finish called at 26 (1ps)
Hint: Total mismatched samples is 0 out of 0 samples
Simulation finished at 26 ps
Mismatches: 0 in 0 samples
```

Status: Simulation completed

Truth table

Control			Input		Output
Clk	ret	jump	stko	jumper	pc
↑	1	X	A	X	A
↑	0	0	X	X	pc+1
↑	0	1	X	B	pc+B

✓ **Condition Code System**



- **Design:**

```
module tcnd_v1(contl, aluo, tcnd);
input [3:0] contl;
input [8:0] aluo;
output tcnd;
reg tcnd;

always @(contl or aluo)
begin
   case (contl)
      1 : tcnd = 1'b1;              // 無條件回主程式 RET
      2 : tcnd = 1'b1;              // 無條件跳躍 JUMP
      3 : tcnd = 1'b1;              // 無條件呼叫 CALL
      4 : tcnd = ~|aluo[7:0];       // 計算結果為 0 時 JZ
      5 : tcnd = |aluo[7:0];        // 計算結果為非 0 時 JNZ
      6 : tcnd = aluo[8];           // 進位標誌為 1 時 JC
      7 : tcnd = ~aluo[8];          // 進位標誌為 0 時 JNC  // zero flag
      default : tcnd = 1'b0;        // 預設條件
   endcase
end

endmodule
```

- **Testbench:**

```
`timescale 1ns/1ps

module top_module();
    reg [3:0] contl;
    reg [8:0] aluo;
    wire tcnd;

    tcnd_v1 uut (
        .contl (contl),
        .aluo (aluo),
        .tcnd (tcnd)
    );

    initial begin
        contl = 0;
        repeat(15) begin
            aluo = 9'd10;
            #1;
            if (contl <=3 && contl >0) begin
                if (tcnd == 1'b1) begin
                    $display ("Passed! case %d", contl) ;
                end
                else $display ("Failed!, contl = %d ,tcnd = %d", contl,tcnd);
            end
            else if (contl == 'd4) begin
                if (tcnd ==   ~|aluo[7:0]) $display ("Passed! case 4 ") ;
                else $display ("Failed!, contl = %d ,tcnd = %d", contl,tcnd);
            end
            else if (contl == 'd5) begin
                if (tcnd ==   |aluo[7:0]) $display ("Passed! case 5 ") ;
                else $display ("Failed!, contl = %d ,tcnd = %d", contl,tcnd);
            end
            else if (contl == 'd6) begin
                if (tcnd ==   aluo[8]) $display ("Passed! case 6 ") ;
```

```
            else $display ("Failed!, contl = %d ,tcnd = %d", contl,tcnd);
        end
        else if (contl == 'd7) begin
            if (tcnd ==   ~aluo[8]) $display ("Passed! case 7") ;
            else $display ("Failed!, contl = %d ,tcnd = %d", contl,tcnd);
        end
        else begin
            if(tcnd == 0) $display ("Passed! case %d",contl) ;
            else $display ("failed!, contl = %d ,tcnd = %d", contl,tcnd);
        end
        contl = contl +1 ;
    end
    $finish;
    end
endmodule
```

- Simulation result

iverilog — Compile and simulate

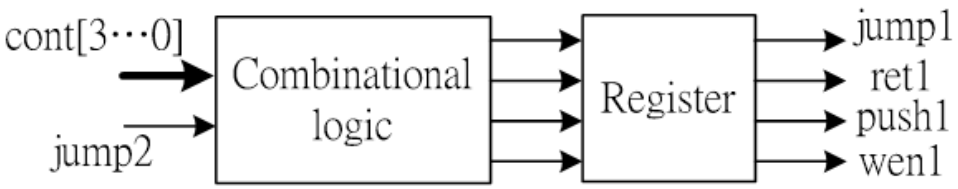
Running Icarus Verilog compile. Show Icarus Verilog compile messages...
Running Icarus Verilog simulation. Show Icarus Verilog simulation messages...

```
Running Icarus Verilog simulator...
VCD info: dumping is suppressed.
Passed! case  0
Passed! case  1
Passed! case  2
Passed! case  3
Passed! case 4
Passed! case 5
Passed! case 6
Passed! case 7
Passed! case  8
Passed! case  9
Passed! case 10
Passed! case 11
Passed! case 12
Passed! case 13
Passed! case 14
./top_module.v:48: $finish called at 15000 (1ps)
Hint: Total mismatched samples is 0 out of 0 samples
Simulation finished at 15000 ps
Mismatches: 0 in 0 samples
```

Condition code

Control code	Instruction	
0001	RET	回主程式
0010	JUMP	無條件跳躍 tcnd=1
0011	CALL	無條件呼叫 tcnd=1
0100	JZ	運算結果為 0 時(0 旗號=1) tcnd=1 運算結果為非 0 時(0 旗號=0) tcnd=0
0101	JNZ	運算結果為 0 時(0 旗號=1) tcnd=0 運算結果為非 0 時(0 旗號=0) tcnd=1
0110	JC	進位旗號為 1 時 tcnd=1 進位旗號為非 1 時 tcnd=0
0111	JNC	進位旗號為 0 時 tcnd=1 進位旗號為非 0 時 tcnd=0

✓ **Control system 1**



- **Design:**

```
module control_v1 (contl, clk, jump2, jump1, ret1, push1, wen1);
input [3:0] contl;
input clk, jump2;
output jump1, ret1, push1, wen1;
reg jump1, ret1, push1, wen1;

initial begin
   jump1 = 1'b0; //  預設：不執行跳躍操作
   ret1 = 1'b0; //  預設：不執行回傳操作
   push1 = 1'b0; //  預設：不執行推入操作
   wen1 = 1'b0; //  預設：不啟用寫入
end
always @(posedge clk)
begin

   //  根據 contl 的值進行控制邏輯
   case (contl)
```

```
4'b0001: begin
    // 情況 1：RET（從子程序返回）
    jump1 <= ~jump2; // 傳遞條件非跳躍信號
    ret1 <= ~jump2;  // 設定回傳信號
    push1 <= push1;  // 保持推入信號不變
    wen1 <= wen1;    // 保持寫入信號不變
end
4'b0010: begin
    // 情況 2：JUMP（無條件跳躍）
    ret1 <= ret1;    // 保持回傳信號不變
    jump1 <= ~jump2;  // 傳遞條件非跳躍信號
    push1 <= push1;  // 保持推入信號不變
    wen1 <= wen1;    // 保持寫入信號不變
end
4'b0011: begin
    // 情況 3：CALL（函數呼叫，保存返回地址）
    push1 <= ~jump2;  // 啟用推入操作以保存地址
    jump1 <= ~jump2;  // 傳遞條件非跳躍信號
    ret1 <= ret1;    // 保持回傳信號不變
    wen1 <= wen1;    // 保持寫入信號不變
end
4'b0100: begin
    // 情況 4：JZ（運算結果為零跳躍）
    jump1 <= ~jump2;  // 傳遞條件非跳躍信號
    ret1 <= ret1;    // 保持回傳信號不變
    push1 <= push1;  // 保持推入信號不變
    wen1 <= wen1;    // 保持寫入信號不變
end
4'b0101: begin
    // 情況 5：JNZ（運算結果為非零跳躍）
    jump1 <= ~jump2;  // 傳遞條件非跳躍信號
    ret1 <= ret1;    // 保持回傳信號不變
    push1 <= push1;  // 保持推入信號不變
    wen1 <= wen1;    // 保持寫入信號不變
end
```

```
        4'b0110: begin
            // 情況 6：JC（進位標誌為 1 跳躍）
            jump1 <= ~jump2;  // 傳遞條件非跳躍信號
            ret1 <= ret1;     // 保持回傳信號不變
            push1 <= push1;   // 保持推入信號不變
            wen1 <= wen1;     // 保持寫入信號不變
        end
        4'b0111: begin
            // 情況 7：JNC（進位標誌為 0 跳躍）
            jump1 <= ~jump2;  // 傳遞條件非跳躍信號
            ret1 <= ret1;     // 保持回傳信號不變
            push1 <= push1;   // 保持推入信號不變
            wen1 <= wen1;     // 保持寫入信號不變
        end
        4'b1xxx: begin
            // 情況 8：資料運算指令
            wen1 <= ~jump2;    // 啟用條件非跳躍信號作為寫入
            ret1 <= ret1;     // 保持回傳信號不變
            push1 <= push1;   // 保持推入信號不變
            jump1 <= jump1;   // 保持跳躍信號不變
        end
        default: begin
            // 其他情況：不執行任何操作
            ret1 <= 1'b0;
            push1 <= 1'b0;
            wen1 <= 1'b0;
            jump1 <= 1'b0;
        end
    endcase
end

endmodule
```

- **Testbench:**

```
`timescale 1ns/1ps

module top_module;

    reg [3:0] contl;
    reg clk, jump2;
    wire jump1, ret1, push1, wen1;

    control_v1 uut (
        .contl(contl),
        .clk(clk),
        .jump2(jump2),
        .jump1(jump1),
        .ret1(ret1),
        .push1(push1),
        .wen1(wen1)
    );

    always #5 clk = ~clk;
    initial clk = 0;

    initial begin
        contl=0;
        jump2=1;
        #10;
        test_case(4'b0001, 1, ~jump2, ~jump2, 0, 0); // RET
        test_case(4'b0010, 1, ~jump2, 0, 0, 0);        // JUMP
        test_case(4'b0011, 1, ~jump2, 0, ~jump2, 0); // CALL
        test_case(4'b0100, 1, ~jump2, 0, 0, 0);        // JZ
        test_case(4'b0101, 1, ~jump2, 0, 0, 0);        // JNZ
        test_case(4'b0110, 1, ~jump2, 0, 0, 0);        // JC
        test_case(4'b0111, 1, ~jump2, 0, 0, 0);        // JNC
        test_case(4'b1000, 1, 0, 0, 0, ~jump2);        //  数据运算指令
        $finish;
```

```
    end

    task test_case;
        input [3:0] t_contl;
        input t_jump2;
        input exp_jump1, exp_ret1, exp_push1, exp_wen1;
        begin
            // 应用输入
            contl = t_contl;
            jump2 = t_jump2;
            #10; // 等待一个时钟周期

            // 检查输出是否正确
            if (jump1 === exp_jump1 && ret1 === exp_ret1 && push1 ===
exp_push1 && wen1 === exp_wen1) begin
                $display("Test Case Passed: contl = %b | jump1 = %b, ret1 = %b, push1
= %b, wen1 = %b", t_contl, jump1, ret1, push1, wen1);
            end else begin
                $display("Test Case Failed: contl = %b | jump1 = %b, ret1 = %b, push1 =
%b, wen1 = %b | Expected: jump1 = %b, ret1 = %b, push1 = %b, wen1 = %b",
                         t_contl, jump1, ret1, push1, wen1, exp_jump1, exp_ret1, exp_push1,
exp_wen1);
            end
        end
    endtask
endmodule
```

- Simulation result

iverilog — Compile and simulate

Running Icarus Verilog compile. Show Icarus Verilog compile messages...
Running Icarus Verilog simulation. Show Icarus Verilog simulation messages...

```
Running Icarus Verilog simulator...
VCD info: dumping is suppressed.
Test Case Passed: contl = 0001 | jump1 = 0, ret1 = 0, push1 = 0, wen1 = 0
Test Case Passed: contl = 0010 | jump1 = 0, ret1 = 0, push1 = 0, wen1 = 0
Test Case Passed: contl = 0011 | jump1 = 0, ret1 = 0, push1 = 0, wen1 = 0
Test Case Passed: contl = 0100 | jump1 = 0, ret1 = 0, push1 = 0, wen1 = 0
Test Case Passed: contl = 0101 | jump1 = 0, ret1 = 0, push1 = 0, wen1 = 0
Test Case Passed: contl = 0110 | jump1 = 0, ret1 = 0, push1 = 0, wen1 = 0
Test Case Passed: contl = 0111 | jump1 = 0, ret1 = 0, push1 = 0, wen1 = 0
Test Case Passed: contl = 1000 | jump1 = 0, ret1 = 0, push1 = 0, wen1 = 0
./top_module.v:36: $finish called at 90000 (1ps)
Hint: Total mismatched samples is 0 out of 0 samples
Simulation finished at 90000 ps
Mismatches: 0 in 0 samples
```

Status: Simulation completed

Truth table

Input			Output				
Clk	contl[3..0]	jump2	jump1	ret1	push1	wen1	
↑	0001	A	Not A	Not A	0	0	回主程式
↑	0010	A	Not A	0	0	0	無條件跳躍
↑	0011	A	Not A	0	Not A	0	無條件呼叫
↑	0100	A	Not A	0	0	0	運算為零跳躍
↑	0101	A	Not A	0	0	0	運算非零跳躍
↑	0110	A	Not A	0	0	0	進位旗號 1 跳躍
↑	0111	A	Not A	0	0	0	進位旗號 0 跳躍
↑	1xxx	A	0	0	0	Not A	資料運算
↑	else	A	0	0	0	0	不動作