浙江大学实验报告

课程名称：数字系统　　任课老师：李宇波

实验名称：Verifying EC-1　　实验日期：2023/6/12

1　实验目的和要求

1.1 实验目的

To verify the design of EC-1.

验证 EC-1 的设计。

1.2 实验要求

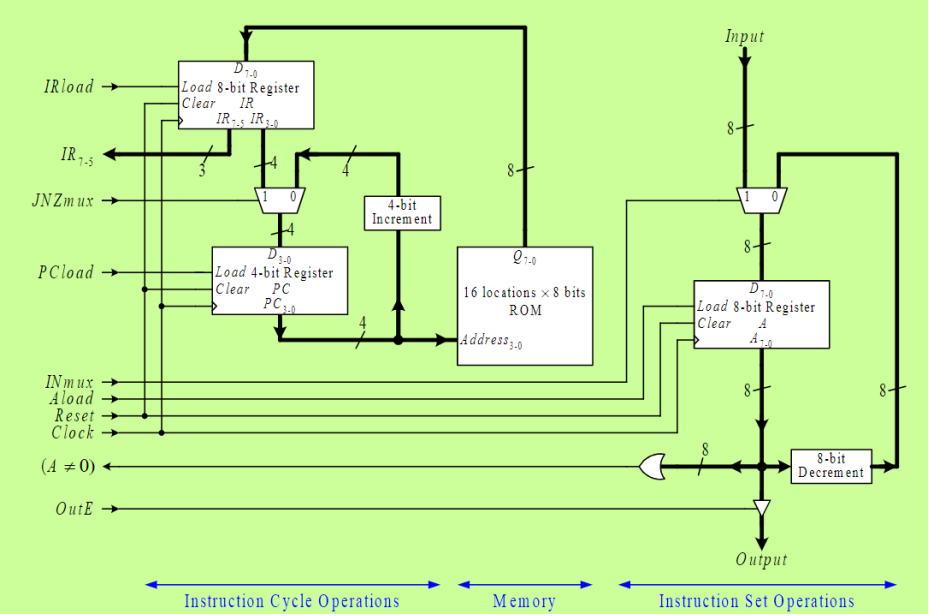
For the EC-1 microprocessor, implement the datapath circuit and FSM circuit separately in Verilog module, connect them together by using a top module, and implement it in your Basys3 board. Note that you need to put the compiled assemble code, i.e., the binary executable code in the ROM in your datapath circuit to run the program in the microprocessor. Connect the switches in board to your input pin, and connect the output pin to a LED.

对于 EC-1 微处理器，在 Verilogmodule 中分别实现数据通路和 有限状态机电路，使用顶部模块将它们连接在一起，然后在 Basys3 板中实现。请注意，你需要将编译后的汇编代码（即二进制可执行代码）放入 数据通路的 ROM 中，以便在微处理器中运行程序。将电路板上的开关连接到输入引脚，并将输出引脚连接到 LED。

2　数据通路

1） 确定器件：根据算法，数据通路需要：三个寄存器；一个自增器；一个自减器；两个数据选择器；一个 ROM；其他外围电路。

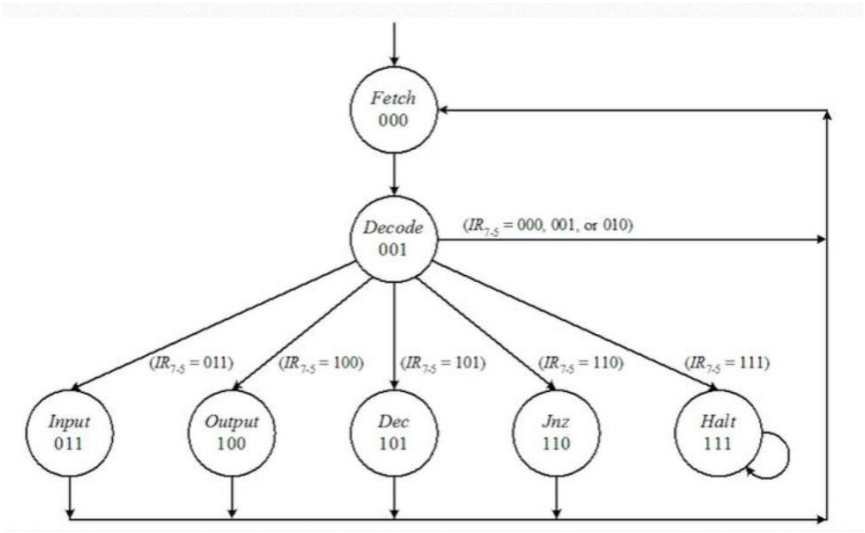
2） 连线



3 控制单元设计

1） 设计状态和状态图如下：

状态	执行操作
S000	Fetch 指令
S001	Decode
S011	Input A
S100	Output A
S101	Decrement A
S110	JNZ address
S111	Halt



实现上述状态转换需要两种状态反馈信号，分别用于判断 A 是否为 0、返回 IR7-5 位的数据。

2） 输出控制字设计

	IRload	PCload	INmux	Aload	JNZmux	OutE
000	1	1	0	0	0	0
001	0	0	0	0	0	0
011	0	0	1	1	0	0
100	0	0	0	0	0	1
101	0	0	0	1	0	0
110	0	IF (A≠0) THEN 1 ELSE 0	0	0	1	0
111	0	0	0	0	0	0

4 行为级模型代码

行为级模型代码设计按层次分类：顶层模块；状态机模块、数据通路模块、分频器模块；（数据通路下的）两个数据选择器模块、三个寄存器模块、一个自加器模块、一个自减器模块、一个 ROM 模块。完整代码见附录 1 。

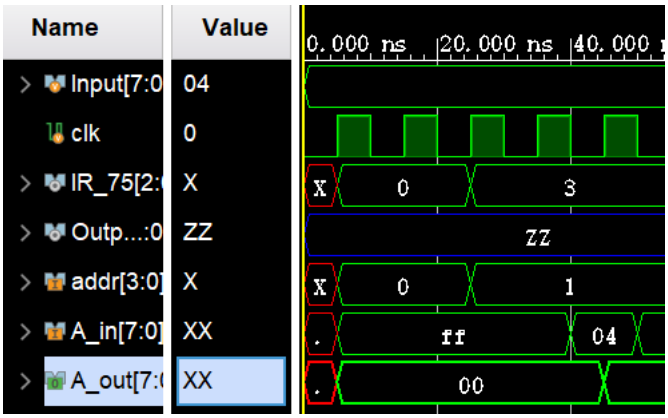
5 行为级仿真

编写仿真代码（见附录 2），模拟实际的 N 输入。

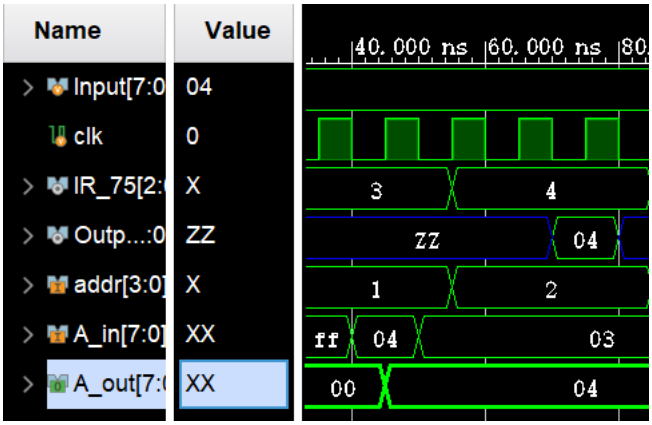
1）数据通路仿真

测试了第一个循环：Input->Output->Decrement->JNZ.

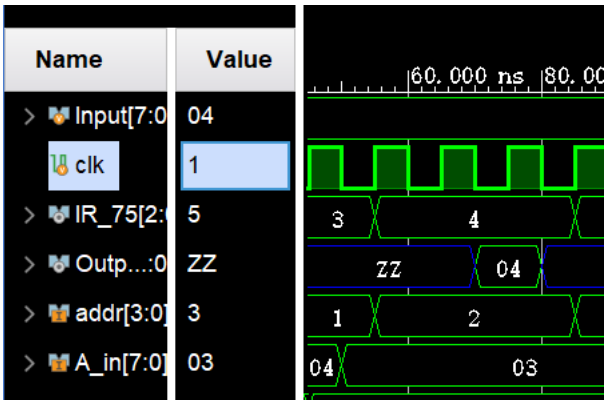
前三个周期：000->001->001，代表 Input，可以看到 A_IN=0000_0100，等于 Input。



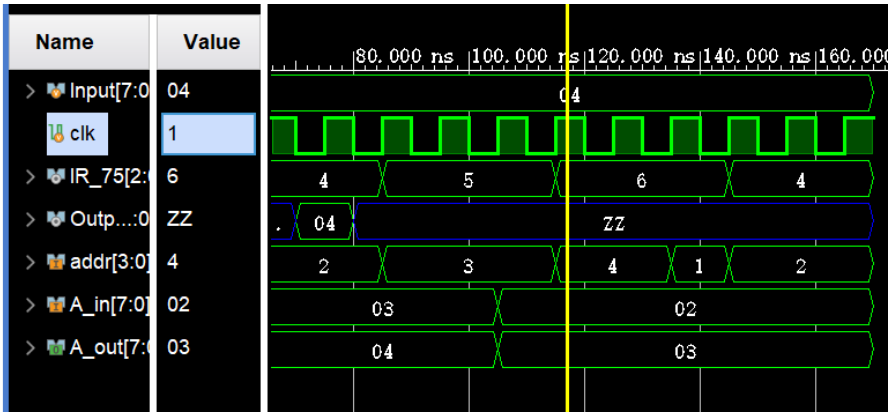
第二个三周期：000->001->100，代表 Output，可以看到 A_out =0000_0100.



第三个三周期：000->001->101，代表 Decrement，可以看到 A_in 减 1（见上图）



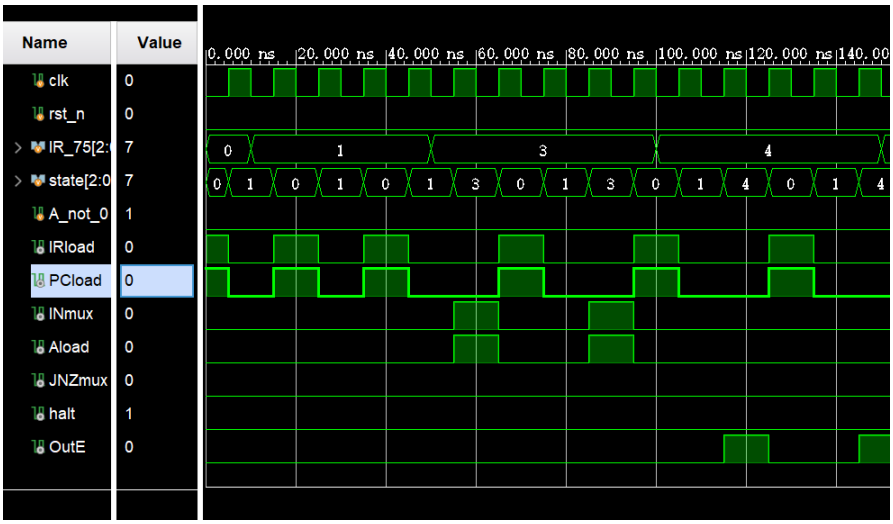
第四个三周期：000->001->110，代表 JNZ，可以看到 address 从 4 变成 1。

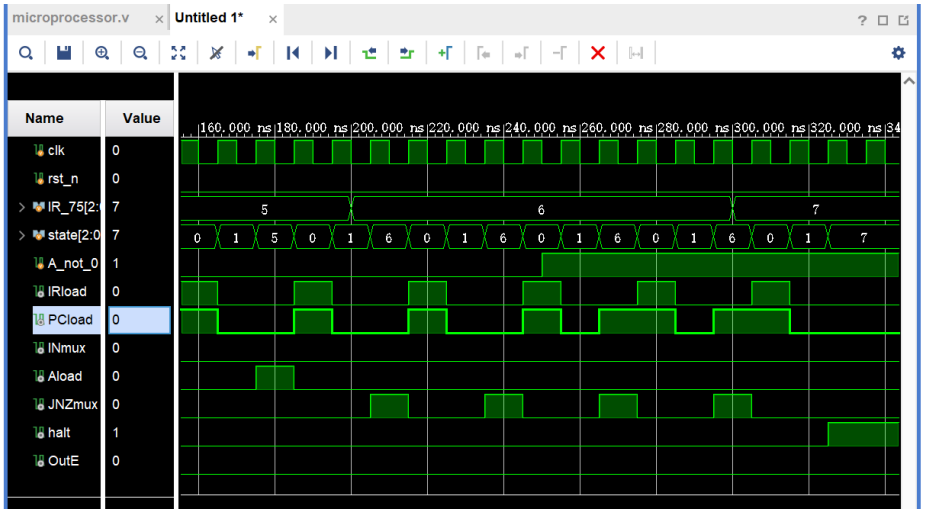


2）有限状态机仿真

先后设定`[2:0] IR_75`为`001`、`011`、`110`、`101`、`110`、`111`；并在`110`的第五个时钟周期改`A_not_0`为`1`。

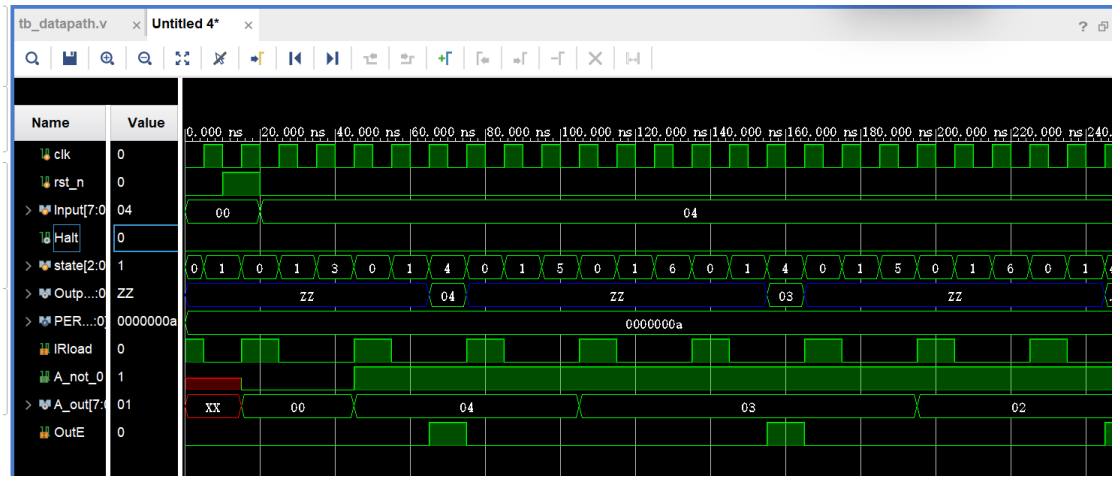
即测试了状态 000->001->000，000->001->011，000->001->100，000->001->101，000->001->110，000->001->111 的转换。状态对应的控制字输出也正确对应。

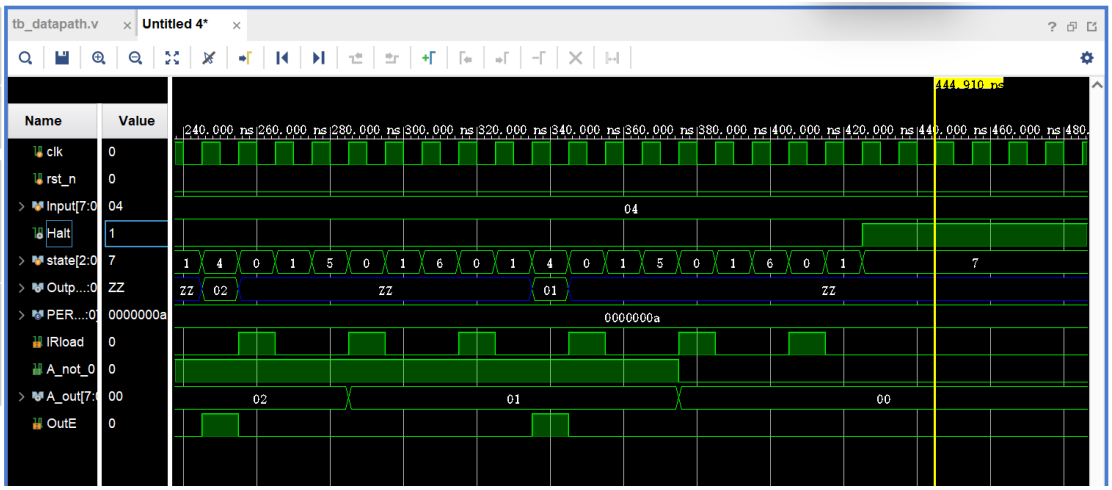




3）处理器仿真

输入 0000_0100，测试输出 Output 经过 0000_0100-> 0000_0011 -> 0000_0010 -> 0000_0001 -> 0000_0000（不输出，此时输出 Halt=1）。





综合以上仿真结果，认为设计无误。

附录 1

1. 顶层模块

```
module GeneralMicro (
    input clk,
    input rst_n,
    input [7:0] Input,
    output Halt,
    output [7:0] Output
);

wire IRload, JNZmux, PCload, INmux, Aload, OutE, A_not_0;
wire [2:0] IR_75;
wire clk_2s;
/*Simulation
assign clk_2s = clk;
*/

clk_div clk2s(
    .clk(clk),
    .reset(rst_n),
    .clk_2s(clk_2s)
);



datapath dp(
    .Input(Input),
    .IRload(IRload),
    .JNZmux(JNZmux),
    .PCload(PCload),
    .INmux(INmux),
    .Aload(Aload),
    .OutE(OutE),
    .rst_n(rst_n),
    .clk(clk_2s),
    .A_not_0(A_not_0),
    .IR_75(IR_75),
    .Output(Output)
);
```

```
FSM fsm(
    .clk(clk_2s),
    .rst_n(rst_n),
    .IR_75(IR_75),
    .A_not_0(A_not_0),
    .IRload(IRload),
    .PCload(PCload),
    .INmux(INmux),
    .Aload(Aload),
    .JNZmux(JNZmux),
    .halt(Halt),
    .OutE(OutE)
);

endmodule
```

2. FSM 模块

```
module FSM (
    input clk,
    input rst_n,
    input [2:0] IR_75,
    input A_not_0,
    output reg IRload,
    output reg PCload,
    output reg INmux,
    output reg Aload,
    output reg JNZmux,
    output reg halt,
    output reg OutE
);

parameter Fetch  =  3'b000;
parameter Decode =  3'b001;
parameter Input  =  3'b011;
parameter Output =  3'b100;
parameter Dec    =  3'b101;
parameter JNZ    =  3'b110;
parameter Halt   =  3'b111;

reg [2:0] state, state_next;
initial begin
```

```
    state = Fetch;
end

always @(*) begin
    case (state)
        Fetch: state_next = Decode;
        Decode:begin
            case (IR_75)
                3'b011: state_next = Input;
                3'b100: state_next = Output;
                3'b101: state_next = Dec;
                3'b110: state_next = JNZ;
                3'b111: state_next = Halt;
                default:state_next = Fetch;
            endcase
        end
        Halt: state_next = Halt;
        default: state_next = Fetch;
    endcase
end

always @(posedge clk) begin
    if(rst_n)
        state = Fetch;
    else
        state = state_next;
end

always @(*) begin
    case (state)
        Fetch:begin
            IRload = 1; PCload = 1;
            INmux = 0;  Aload = 0;
            JNZmux = 0; OutE = 0;
            halt = 0;
        end
        Decode:begin
            IRload = 0; PCload = 0;
            INmux = 0; Aload = 0;
            JNZmux = 0; OutE = 0;
            halt = 0;
        end
        Input:begin
            IRload = 0; PCload = 0;
```

```
            INmux = 1; Aload = 1;
            JNZmux = 0; OutE = 0;
            halt = 0;
        end
        Output:begin
            IRload = 0; PCload = 0;
            INmux = 0; Aload = 0;
            JNZmux = 0; OutE = 1;
            halt = 0;
        end
        Dec:begin
            IRload = 0; PCload = 0;
            INmux = 0; Aload = 1;
            JNZmux = 0; OutE = 0;
            halt = 0;
        end
        JNZ:begin
            IRload = 0;
            if(A_not_0) PCload = 1;
            else PCload = 0;
            INmux = 0; Aload = 0;
            JNZmux = 1; OutE = 0;
            halt = 0;
        end
        Halt:begin
            IRload = 0; PCload = 0;
            INmux = 0; Aload = 0;
            JNZmux = 0; OutE = 0;
            halt = 1;
        end
    endcase
end

endmodule
```

3. 数据通路模块

```
module datapath (
    input [7:0] Input,
    input IRload,
    input rst_n,
    input clk,
```

```
    input JNZmux,
    input PCload,
    input INmux,
    input Aload,
    input OutE,
    output reg A_not_0,
    output [2:0] IR_75,
    output reg [7:0] Output
);

wire [7:0] IR_D, Decrement_out, A_out, IN_out;
wire [3:0] IR_30, JNZ_out, PC_out, Increment_out;

RegisterIR reg_IR(
    .load(IRload),
    .clk(clk),
    .clear(rst_n),
    .IR_in(IR_D),
    .IR_75(IR_75),
    .IR_30(IR_30)
);

RegisterPC reg_PC(
    .load(PCload),
    .clk(clk),
    .clear(rst_n),
    .PC_in(JNZ_out),
    .PC_out(PC_out)
);

MUX mux_JNZ(
    .sw(JNZmux),
    .in1(IR_30),
    .in0(Increment_out),
    .out(JNZ_out)
);

Increment Inc(
    .in(PC_out),
    .out(Increment_out)
);

ROM rom(
    .addr(PC_out),
```

```
    .out(IR_D)
);

MUX mux_IN(
    .sw(INmux),
    .in1(Input),
    .in0(Decrement_out),
    .out(IN_out)
);

RegisterA reg_A(
    .load(Aload),
    .clk(clk),
    .clear(rst_n),
    .A_in(IN_out),
    .A_out(A_out)
);

Decrement dec(
    .in(A_out),
    .out(Decrement_out)
);

always @(*) begin
    A_not_0 = (A_out != 8'b0);
end

always @(*) begin
    Output = (OutE?A_out:8'bz);
end

endmodule
```

4. 分频器模块

```
module clk_div(
    input clk,
    input reset,
    output reg clk_2s
    );
    reg [100:0] count = 0;
    always@(posedge clk or posedge reset)
    begin
    if(reset)
    begin
```

```
    count<=0;
    clk_2s<=0;
    end
    else if(count == 20000000)
    begin
    clk_2s<=~clk_2s;
    count <= 0;
    end
    else if(count < 20000000)
    begin
    count <= count+1;
    end
    end
endmodule
```

5. 数据选择器模块

```
module MUX (
    input sw,
    input [7:0] in1,
    input [7:0] in0,
    output reg [7:0] out
);

always @(*) begin
    if(sw)
        out = in1;
    else
        out = in0;
end

endmodule
```

6. 寄存器模块（以 IR 寄存器为例）

```
module RegisterIR (
    input load,
    input clk,
    input clear,
    input [7:0] IR_in,
    output reg [2:0] IR_75,
    output reg [3:0] IR_30
);

always @(posedge clk) begin
    if(clear)begin
```

```
        IR_75 = 3'b0;
        IR_30 = 4'b0;
    end
    else if(load)begin
        IR_30 = IR_in[3:0];
        IR_75 = IR_in[7:5];
    end
end
endmodule
```

7. 自增模块

```
module Increment (
    input [3:0] in,
    output reg [3:0] out
);

always @(*) begin
    out = in + 4'b1;
end

endmodule
```

8. 自减模块

```
module Decrement (
    input [7:0] in,
    output reg [7:0] out
);

always @(*) begin
    out = in - 8'b1;
end

endmodule
```

9. ROM 模块

```
module ROM (
    input [3:0] addr,
    output [7:0] out
);

wire [7:0] rom [0:15];

assign rom[0] = 8'b011_00000;
assign rom[1] = 8'b100_00000;
```

```
assign rom[2] = 8'b101_00000;
assign rom[3] = 8'b110_00001;
assign rom[4] = 8'b111_11111;
assign rom[5] = 8'b0;
assign rom[6] = 8'b0;
assign rom[7] = 8'b0;
assign rom[8] = 8'b0;
assign rom[9] = 8'b0;
assign rom[10] = 8'b0;
assign rom[11] = 8'b0;
assign rom[12] = 8'b0;
assign rom[13] = 8'b0;
assign rom[14] = 8'b0;
assign rom[15] = 8'b0;

assign out = rom[addr];

endmodule
```

附录 2

1. FSM 仿真代码

```
initial
begin
    #10
    IR_75 = 001;
    // back to 000

    #40
    IR_75 = 011;

    #50
    IR_75 = 100;

    #50
    IR_75 = 101;

    #50
    IR_75 = 110;
    #50
    A_not_0 = 1;
    #50
    IR_75 = 111;
end
```

2. 数据通路仿真代码

```
initial
begin
    Input = 8'b0000_0100;
    rst_n = 1;
    #10
    rst_n = 0;
    #10
    //000
    IRload = 1; PCload = 1;
    INmux = 0; Aload = 0;
    JNZmux = 0; OutE = 0;
    #10
    //001
    IRload = 0; PCload = 0;
    INmux = 0; Aload = 0;
```

```
    JNZmux = 0; OutE = 0;
    #10
    //011 Input, test A_out
    IRload = 0; PCload = 0;
    INmux = 1; Aload = 1;
    JNZmux = 0; OutE = 0;

    #10
    //000
    IRload = 1; PCload = 1;
    INmux = 0; Aload = 0;
    JNZmux = 0; OutE = 0;
    #10
    //001
    IRload = 0; PCload = 0;
    INmux = 0; Aload = 0;
    JNZmux = 0; OutE = 0;
    #10
    //100 Output, test Output
    IRload = 0; PCload = 0;
    INmux = 0; Aload = 0;
    JNZmux = 0; OutE = 1;

    #10
    //000
    IRload = 1; PCload = 1;
    INmux = 0; Aload = 0;
    JNZmux = 0; OutE = 0;
    #10
    //001
    IRload = 0; PCload = 0;
    INmux = 0; Aload = 0;
    JNZmux = 0; OutE = 0;
    #10
    //101 Dec, A_out
    IRload = 0; PCload = 0;
    INmux = 0; Aload = 1;
    JNZmux = 0; OutE = 0;

    #10
    //000
    IRload = 1; PCload = 1;
    INmux = 0; Aload = 0;
    JNZmux = 0; OutE = 0;
```

```
    #10
    //001
    IRload = 0; PCload = 0;
    INmux = 0; Aload = 0;
    JNZmux = 0; OutE = 0;
    #10
    //110 JNZ(A!=0)
    IRload = 0; PCload = 1;
    INmux = 0; Aload = 0;
    JNZmux = 1; OutE = 0;

    #10
    //000
    IRload = 1; PCload = 1;
    INmux = 0; Aload = 0;
    JNZmux = 0; OutE = 0;
    #10
    //001
    IRload = 0; PCload = 0;
    INmux = 0; Aload = 0;
    JNZmux = 0; OutE = 0;
    #10
    //110 JNZ(A=0)
    IRload = 0; PCload = 0;
    INmux = 0; Aload = 0;
    JNZmux = 1; OutE = 0;
    #10
    //111 Halt
    IRload = 0; PCload = 0;
    INmux = 0; Aload = 0;
    JNZmux = 0; OutE = 0;
    $finish;
end
```

3. 处理器仿真代码

```
initial
begin
    #10
    rst_n = 1;
    #10
    Input = 8'b0000_0100;
    rst_n = 0;

end
```