lab7

Lab 7: General Microprocessor Lab – Verifying EC-1

(2 hours)

Goal

To verify the design of EC-1.

Procedure

1) 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.

通用微处理器设计的一般流程

1、定义处理器的指令集（即微处理器支持什么操作）

2、设计出能满足指令集需求的数据通路

3、设计控制单元：状态机和控制字输出

4、数据通路+控制单元形成完整的通用微处理器

1、指令集：

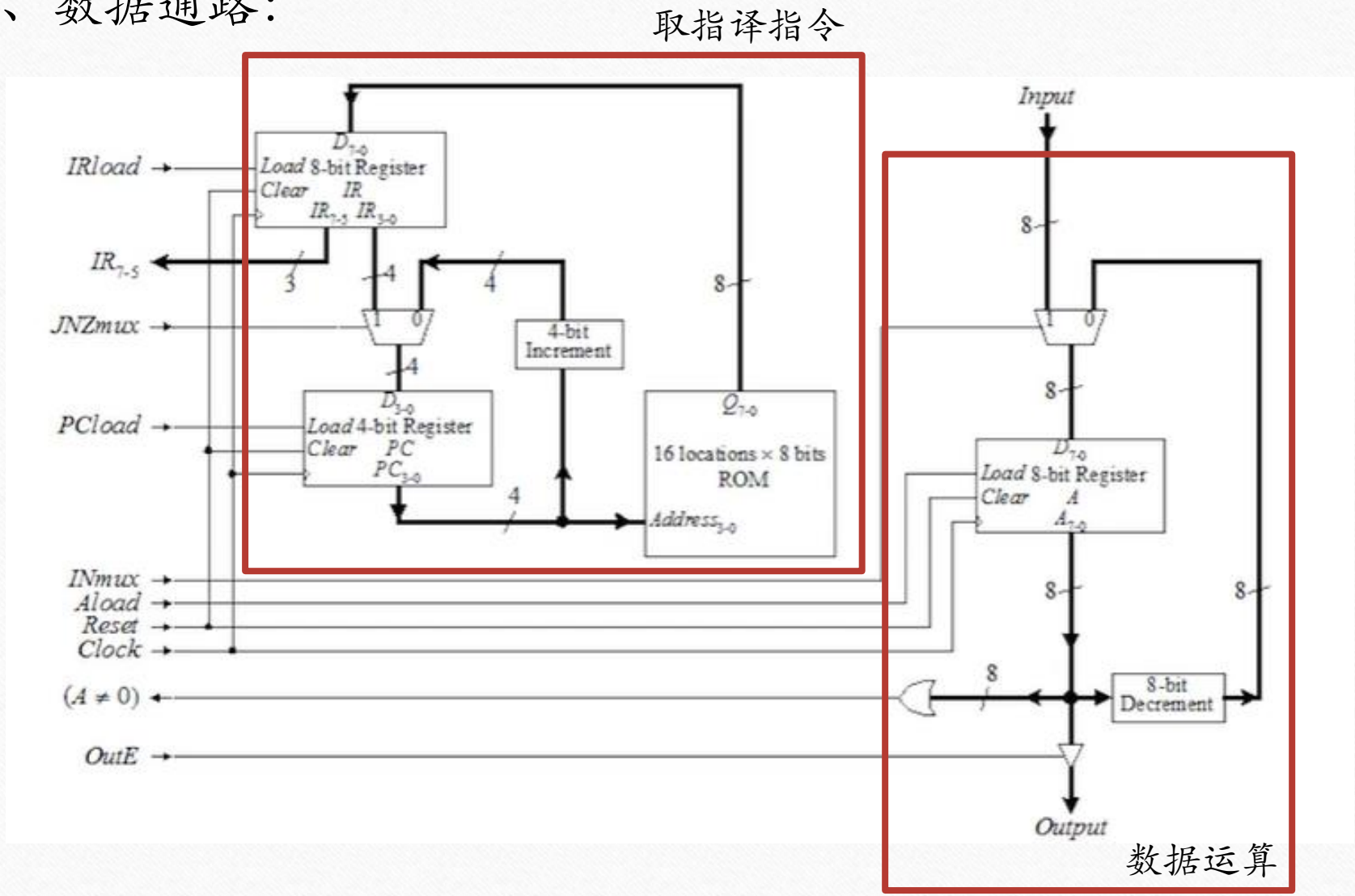
Instruction	Encoding		Operation	Comment
IN A	011	××××××	*A* ← *Input*	Input to *A*
OUT A	100	××××××	*Output* ← *A*	Output from *A*
DEC A	101	××××××	*A* ← *A* – 1	Decrement *A*
JNZ address	110	×aaaa	IF (*A* != 0) THEN *PC* = aaaa	Jump to address if *A* is not zero 跳转指令
HALT	111	××××××	Halt	Halt execution

操作码　操作数

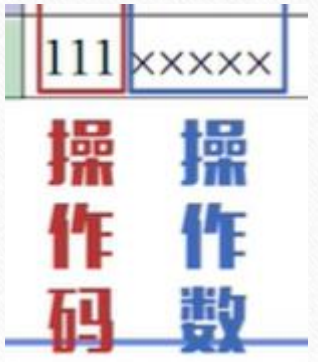
Notations: A=accumulator PC=program counter
AAAA=four bits specifying a memory address

PC：程序指针，记录下一条要执行的指令所在内存单元的地址

2、数据通路：

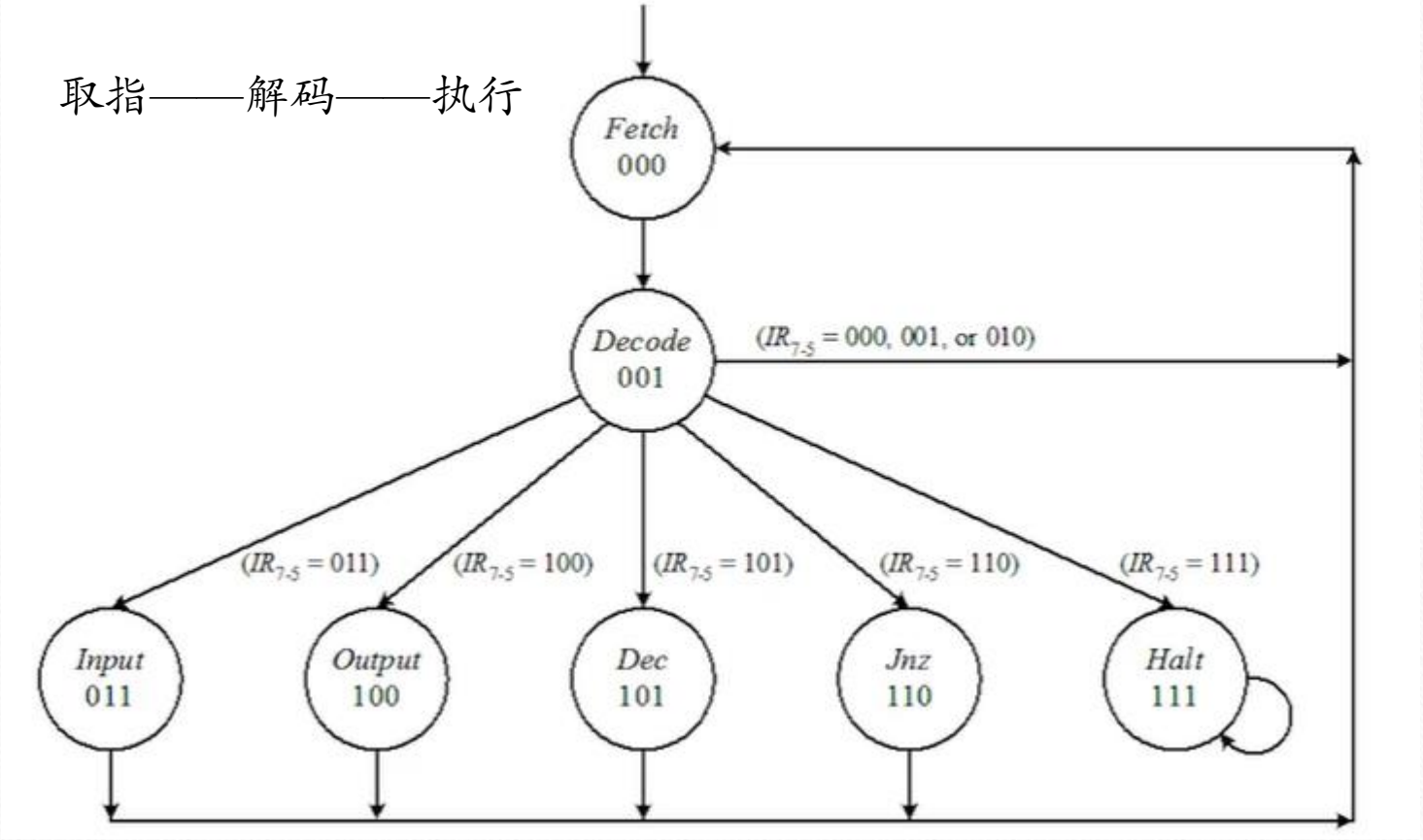


IN A
OUT A
DEC A
JNZ address
HALT



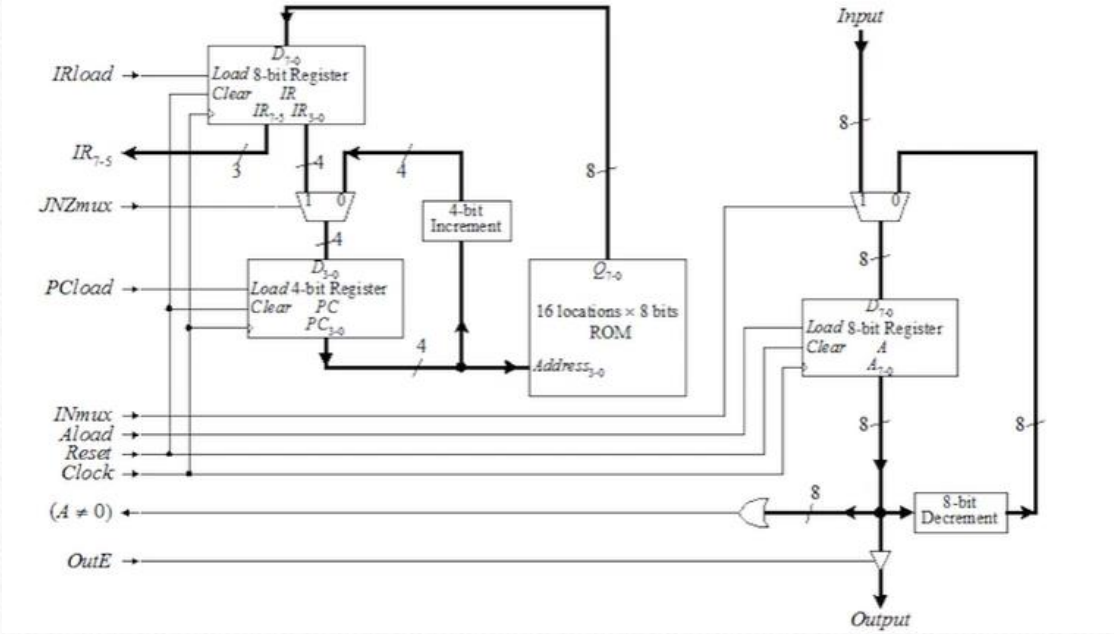
3、状态转换：

取指——解码——执行



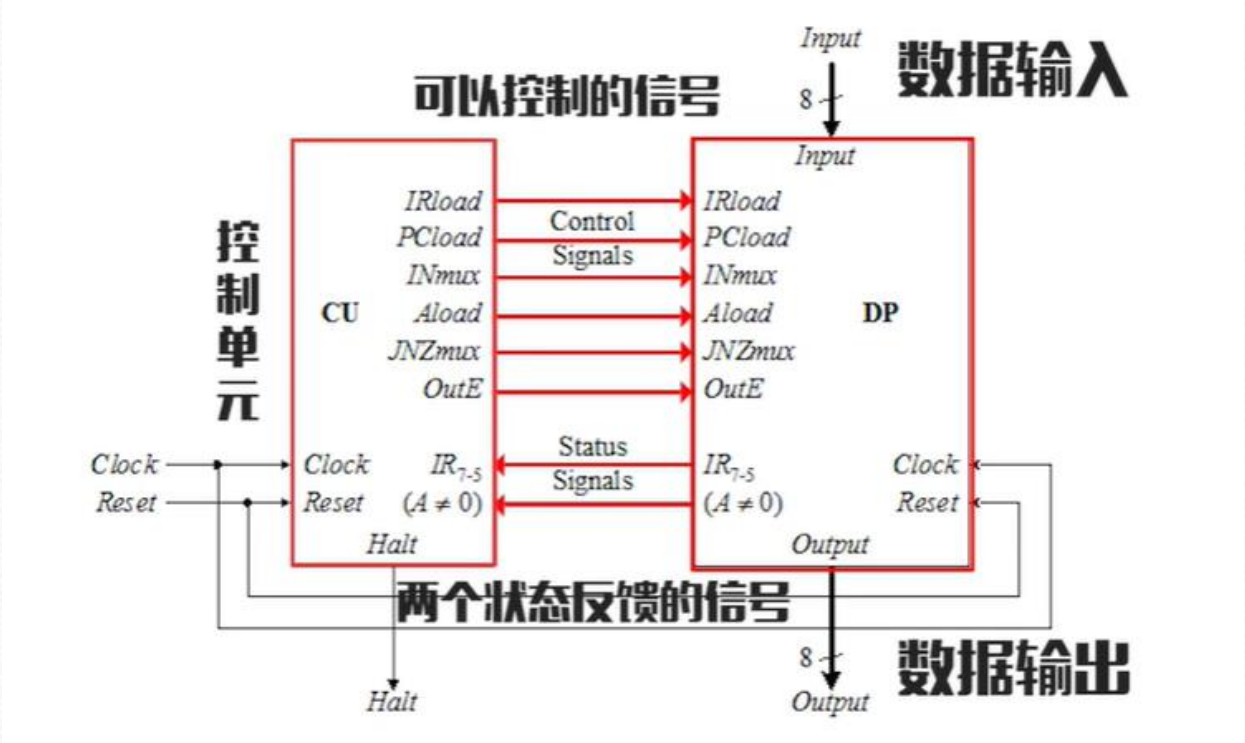
Instruction	Encoding
IN A	011 xxxxx
OUT A	100 xxxxx
DEC A	101 xxxxx
JNZ address	110 xaaaa
HALT	111 xxxxx

3、控制字输出



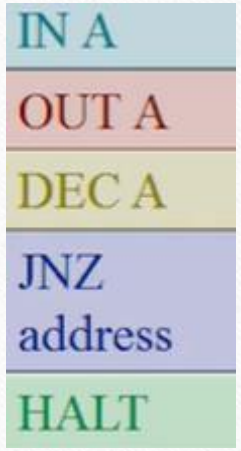
Control Word	State $Q_2Q_1Q_0$	IRload	PCload	INmux	Aload	JNZmux	OutE	Halt
1	000 *Fetch*	1	1	0	0	0	0	0
2	001 *Decode*	0	0	0	0	0	0	0
3	011 *Input*	0	0	1	1	0	0	0
4	100 *Output*	0	0	0	0	0	1	0
5	101 *Dec*	0	0	0	1	0	0	0
6	110 *Jnz*	0	IF $(A \neq 0)$ THEN 1 ELSE 0	0	0	1	0	0
7	111 *Halt*	0	0	0	0	0	0	1

4、数据通路+控制单元



完成简易通用处理器的设计

5、用指令集的指令，实现一段代码




```
        IN A
LOOP:OUT A
        DEC A
        JNZ LOOP
        HALT
```

代码功能：输入一个A，A自减至0

ROM：

程序存到ROM里

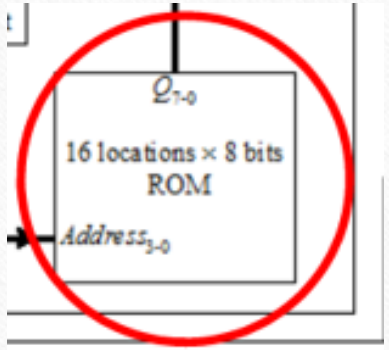
IN A

LOOP:OUT A

DEC A

JNZ LOOP

HALT




```
module ROM_register(
    input [3:0] ina,
    output reg [7:0] out
    );

    always @(*) begin
        case (ina)
            4'b0000: out <= 8'b01100000;
            4'b0001: out <= 8'b10000000;
            4'b0010: out <= 8'b10100000;
            4'b0011: out <= 8'b11000001;
            4'b0100: out <= 8'b11111111;
            default: out <= 8'bzzzzzzzz;
        endcase
    end

endmodule
```

```
memory      instruction
address     encoding
0000        01100000;    -- IN A
0001        10000000;    -- OUT A
0010        10100000;    -- DEC A
0011        11000001;    -- JNZ loop
0100        11111111;    -- HALT
```

Instruction	Encoding
IN A	011 xxxxx
OUT A	100 xxxxx
DEC A	101 xxxxx
JNZ address	110 xaaaa
HALT	111 xxxxx

顶层代码

```
module EC1(
    input [7:0] A,
    input clk, reset,
    output [7:0] led,
    output H
    );
    wire clk_2s;
    clk_div div1(.clk(clk), .reset(reset), .clk_2s(clk_2s));
    wire IRload, JNZmux, PCload,INmux, Aload, OutE;
    wire AnotZero;
    wire [2:0] IR_OpWord;
    Datapath d1(.clk(clk_2s),.reset(reset), .IRload(IRload),.JNZmux(JNZmux), .PCload(PCload),
                .INmux(INmux),.Aload(Aload), .OutE(OutE),.A(A),.IR_OpWord(IR_OpWord),
                .AnotZero(AnotZero), .led(led),.H(H));
    Control c1(.clk(clk_2s), .reset(reset), .IR_OpWord(IR_OpWord),.AnotZero(AnotZero),
            .IRload(IRload),.JNZmux(JNZmux),.PCload(PCload),.INmux(INmux), .Aload(Aload),
            .OutE(OutE));
endmodule
```

数据通路

```
module Datapath(
    input clk, reset, IRload, JNZmux, PCload, INmux, Aload, OutE,
    input [7:0] A,
    output [2:0] IR_OpWord,
    output  AnotZero,
    output [7:0] led,
    output H
    );
    wire [3:0] IR_o, Increment_o, mux2_4bit_o ,PC_o;
    wire [7:0] ROM_o, mux2_8bit_o, A_o, decrement_o;
    IR_Register r1(.ina(ROM_o), .load(IRload), .reset(reset), .clk(clk), .OpWord(IR_OpWord), .OpNum(IR_o));
    mux2_4bit m1(.ina(IR_o), .inb(Increment_o), .JNZmux(JNZmux), .out(mux2_4bit_o));
    PC_register r2(.ina(mux2_4bit_o), .load(PCload), .reset(reset), .clk(clk), .out(PC_o));
    Increment i1(.ina(PC_o), .out(Increment_o));
    ROM_register r3(.ina(PC_o), .out(ROM_o));
    mux2_8bit m2(.ina(A), .inb(decrement_o), .INmux(INmux), .out(mux2_8bit_o));
    A_register r4(.ina(mux2_8bit_o), .load(Aload), .clk(clk),.reset(reset), .out(A_o));
    decrement d1(.ina(A_o), .out(decrement_o));
    assign AnotZero = A_o != 0? 1:0;
    assign H = A_o == 0?1:0;
    Triger t1(.ina(A_o), .OutE(OutE), .out(led));
endmodule
```

控制单元

```
module Control(
    input clk,reset,
    input [2:0] IR_OpWord,
    input AnotZero,
    output reg IRload, JNZmux, PCload, INmux, Aload, OutE
    );

    parameter FETCH = 0 ; parameter DECODE = 1; parameter INPUT = 2;
    parameter OUTPUT = 3; parameter DEC = 4; parameter JNZ = 5;
    parameter HALT = 6;
    reg [2:0] state, next_state;
    always @(posedge clk or posedge reset) begin
        if(reset)begin
            state <= FETCH;
        end
        else
            state <= next_state;
    end
```

```
always @(*) begin
    case (state)
        FETCH: next_state = DECODE;
        DECODE: if(IR_OpWord == 3'b011) next_state = INPUT;
                else if(IR_OpWord == 3'b100)    next_state = OUTPUT;
                else if(IR_OpWord == 3'b101)    next_state = DEC;
                else if(IR_OpWord == 3'b110)    next_state = JNZ;
                else if(IR_OpWord == 3'b111)    next_state = HALT;
                else    next_state = FETCH;
        HALT: next_state = HALT;
        default: next_state <= FETCH;
    endcase

end
```

```
always @(*) begin
    case (state)
        FETCH: begin IRload = 1; PCload = 1; INmux = 0; Aload = 0; JNZmux = 0; OutE = 0;  end
        DECODE: begin IRload = 0; PCload = 0; INmux = 0; Aload = 0; JNZmux = 0; OutE = 0;  end
        INPUT: begin IRload = 0; PCload = 0; INmux = 1; Aload = 1; JNZmux = 0; OutE = 0;  end
        OUTPUT: begin IRload = 0; PCload = 0; INmux = 0; Aload = 0; JNZmux = 0; OutE = 1;  end
        DEC: begin IRload = 0; PCload = 0; INmux = 0; Aload = 1; JNZmux = 0; OutE = 0;  end
        JNZ: begin IRload = 0; PCload = AnotZero==1? 1:0; INmux = 0; Aload = 0; JNZmux = 1; OutE = 0;  end
        HALT: begin IRload = 0; PCload = 0; INmux = 0; Aload = 0; JNZmux = 0; OutE = 1;  end
        default: begin IRload = 1'bz; PCload = 1'bz; INmux = 1'bz; Aload = 1'bz; JNZmux = 1'bz; OutE = 1'bz;  end
    endcase
end
endmodule
```

分频器

```
module clk_div(
    input clk,
    input reset,
    output reg clk_2s
    );
    reg [31:0] count;
    always @(posedge clk or posedge reset ) begin
        if(reset)   begin
            count <= 0; clk_2s <= 0;
        end
        else if(count == 30000000) begin
            count <= 0; clk_2s <= ~clk_2s;
        end
        else count = count+1;
    end
endmodule
```

要求：

验收
实验报告