1、c

2、b

3、b

4、

5、d

6、c

7、d

8. 欲产生序列信号 11010111，则至少需要（　　）级触发器。

A. 2　　B. 3　　C. 4　　D. 5

可以使用状态机实现序列产生，此时需要耗费 3 个触发器：

每一个状态产生一个输出，那么 8 个输出需要 8 个状态，而 8 个状态需要 3 位的状态变量即可，也就是需要 3 个触发器即可实现 8 个状态的跳转，从而实现了 8 位序列的产生。

9、题目：移位寄存器由 8 级触发器构成，则构成的扭环计数器有多少个有效状态？

环形计数器？线性反馈移位寄存器？

扭环计数器的实现是设置一个初始状态，将最高位取反，作为最低位的输入，通过移位即可得到。

也是基于移位寄存器的计数器，是对环形计数器的改进，对于 n 个移位寄存器构成的计数器，有 2n 个有效状态

环形也是基于移位寄存器的计数器，对于 n 个移位寄存器构成的计数器，只有 n 个有效状态。

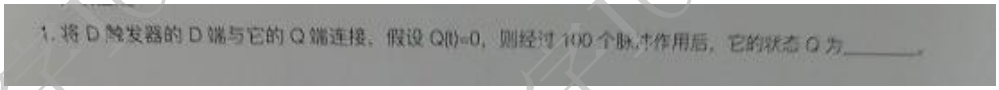
设置一个初始状态，通过移位即可得到。

线性反馈移位寄存器有 2^n - 1

https://blog.csdn.net/qq_44113393/article/details/89852994

10、d

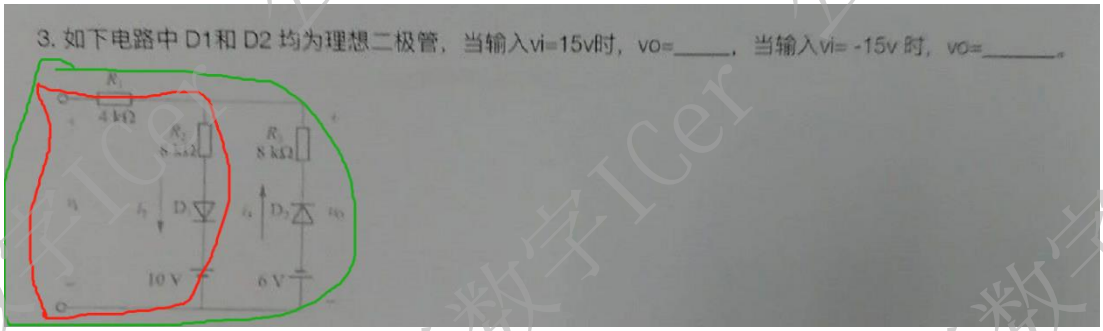
二



这不是废话吗？对自己一直延迟，最后还是自己：0；

2、i = dq/dt 代入参数即可

3、vi = 15v,分析红色回路，vi = -15v，分析绿色回路



4

5、 10/10^12

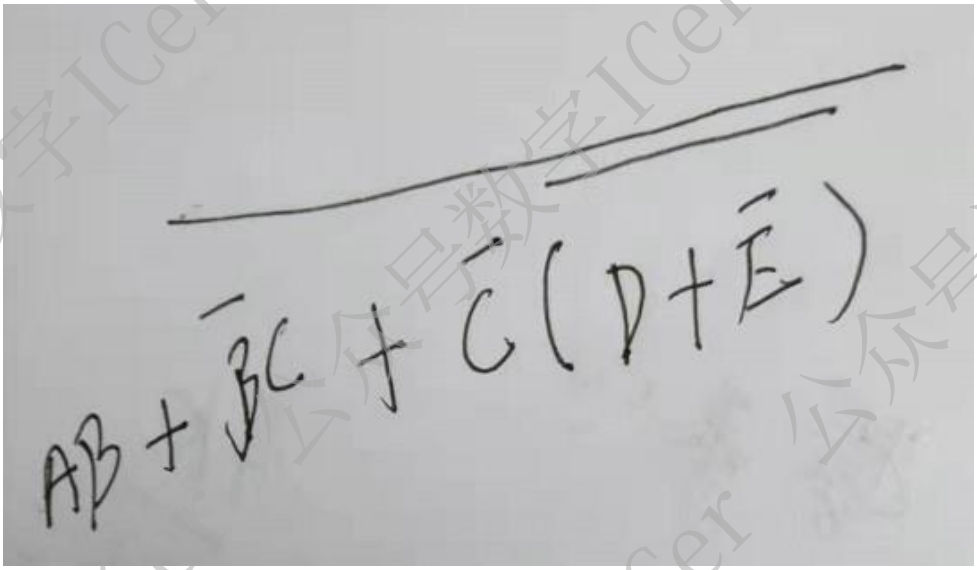
6、8

7、自己看

8、11 个输入，12 个输出

9、不知道

10 记不清了，自己推推



第一题代码：

```
module add5(
    input [4:0] a,
    input [4:0] b,
    input clk,
    input reset,
    output reg [9:0] out
    );
    reg [9:0] mid0 = 0, mid1 = 0, mid2 = 0, mid3 = 0, mid4 = 0;
    parameter s0 = 3'd0, s1 = 3'd1, s2 = 3'd2, s3 = 3'd3, s4 = 3'd4;
    reg [2:0] cur_state, nxt_state;
    always@(posedge clk or posedge reset) begin
        if(reset) cur_state <= s0;
        else cur_state <= nxt_state;
    end
```

```
always@(*) begin
    nxt_state = s0;
    case(cur_state)
    s0: begin
        nxt_state = s1;
    end
    s1: begin
        nxt_state = s2;
    end
    s2: begin
        nxt_state = s3;
    end
    s3: begin
        nxt_state = s4;
    end
    s4: begin
        nxt_state = s4;
    end
    default: nxt_state = s0;
    endcase
end
always@(*) begin
```

```
out = 0;
case(cur_state)
    s0: begin
        if(b[0]) mid0 = {5'b00000, a};
        else mid0 = 0;
    end
    s1: begin
        if(b[1]) mid1 = {4'b0000, a, 1'b0};
        else mid1 = 0;
    end
    s2: begin
        if(b[2]) mid2 = {3'b000, a, 2'b00};
        else mid2 = 0;
    end
    s3: begin
        if(b[3]) mid3 = {2'b00, a, 3'b000};
        else mid3 = 0;
    end
    s4: begin
        if(b[4]) mid4 = {1'b0, a, 4'b0000};
        else mid4 = 0;
        out = mid0 + mid1 + mid2 + mid3 + mid4;
```

```
            end
            default:;
        endcase
    end
endmodule
```

最后一题代码：

```
module fsm_game #(
    parameter A = 3'b010
    )(
    input clk,
    input reset,
    input [2:0] in,
    output reg [2:0] out //最低为表示小于，次低位表示大于，最高位表示等于;
    );

    parameter START = 3'd0, GREAT = 3'd1, LESS = 3'd2, EQUAL = 3'd3, END =
3'd4;
    reg [2:0] cur_state, nxt_state;
    reg [4:0] cnt = 0;
    always@(posedge clk or posedge reset) begin
        if(reset) cur_state <= START;
        else cur_state <= nxt_state;
```

```
end
always@(*) begin
    nxt_state = START;
    case(cur_state)
    START: begin
        cnt = cnt + 1;
        if(cnt <= 20 && in < A) begin
            nxt_state = LESS;
        //    cnt = cnt + 1;
        end
        else if(cnt <= 20 && in > A) begin
            nxt_state = GREAT;
        //    cnt = cnt + 1;
        end
        else if(cnt <= 20 && in == A) begin
            nxt_state = EQUAL;
        //    cnt = cnt + 1;
        end
        else if(cnt > 20) begin
            nxt_state = END;
        //    cnt = cnt + 1;
        end
```

```
        end
        GREAT: nxt_state = START;
        LESS: nxt_state = START;
        EQUAL: nxt_state = EQUAL;
        END: nxt_state = END;
        default: nxt_state = START;
        endcase
    end
    always@(*) begin
        out = 3'b000;
        case(cur_state)
        START: out = 3'b000;
        GREAT: out = 3'b010;
        LESS: out = 3'b001;
        EQUAL: out = 3'b100;
        END: out = 3'b011;
        default: out = 3'b000;
        endcase
    end
endmodule
```