

实验报告

学号：2014K8009929022 姓名：孔静 专业：计算机科学与技术
实验序号：1 实验名称：基本部件单元设计

一、 代码以及波形图

Alu:

Part1 代码

alu 部分

```
module alu(A,B,ALUop,Overflow,Zero,CarryOut,Result);
    input [31:0]A,B;
    input [2:0]ALUop;
    output Overflow,Zero,CarryOut;
    output [31:0]Result;
    wire cin;
    reg [31:0]result;
    wire [31:0]result1,C;
    wire [31:0]cout;

    assign C= (ALUop[2]==1)? ~B:B;
    assign cin=ALUop[2];
    add32 part1(cout,result1,A,C,cin);

    always @(A or B or result1 or cout or ALUop)
    begin
        case (ALUop)
            3'b000:result=A&B;
            3'b001:result=A|B;
            3'b010:
            begin
                result=result1;
            end
            3'b110:
            begin
                result=result1;
            end
            3'b111:
            begin
                if ( ~(cout[6]^cout[5]) && result1[31]) || ((cout[6]^cout[5]) && cout[6]) )
                    result=32'b1;
                else
                    result=32'b0;
            end
            default:result=32'b0;
        endcase
    end

    assign Overflow=((~ALUop[2]&ALUop[1]&~ALUop[0])|(ALUop[2]&ALUop[1]&~ALUop[0])) & (cout[6]^cout[5]);
    assign Zero=((~ALUop[2]&ALUop[1]&~ALUop[0])|(ALUop[2]&ALUop[1]&~ALUop[0])) & (~|Result);
    assign CarryOut=((~ALUop[2]&ALUop[1]&~ALUop[0])|(ALUop[2]&ALUop[1]&~ALUop[0])) & cout[6];
    assign Result=result;
endmodule
```

定义信号

调用了加法器，增加进位变量

寄存器，存放结果，最后 assign 给输出 Result

result1 连接加法器的输出结果，C 为中间变量且连接加法器输入

加法，调用加法器 A+B+0，减法调用加法器 A+~B+1

case 根据 ALUop 的不同，给 result 赋值相应结果

执行 slt 操作时，先进行减法操作，根据得数的符号位正负判断大于小于：如果结果溢出，根据进位判断正负；如果结果不溢出，根据符号位判断正负

根据定义，ALUop，result，以及加法器最高位和符号位的进位，给相应结果赋值

加法器部分

```
module add32(cout,sum,a,b,cin);
    input [31:0]a,b;
    input cin;
    output [31:0]sum;
    output [6:0]cout;

    add8 p1(cout[0],sum[7:0],a[7:0],b[7:0],cin);
    add8 p2(cout[1],sum[15:8],a[15:8],b[15:8],cout[0]);
    add8 p3(cout[2],sum[23:16],a[23:16],b[23:16],cout[1]);
    add4 p4(cout[3],sum[27:24],a[27:24],b[27:24],cout[2]);
    add2 p5(cout[4],sum[29:28],a[29:28],b[29:28],cout[3]);
    add1 p6(cout[5],sum[30],a[30],b[30],cout[4]);
    add1 p7(cout[6],sum[31],a[31],b[31],cout[5]);

endmodule
```

因为以前写过 8 位加法器，因此这个 32 位本来是由 4 个 8 位加法器组成，可是 Overflow 等输出需要最高位的进位，所以改写成 3 个 8 位加法器+1 个 4 位加法器+1 个 2 位加法器+2 个 1 位加法器的 7 个加法器组成，cout【5】保存最高位的进位，cout【6】保存符号位的进位

```
module add8(cout,sum,a,b,cin);
    input [7:0]a,b;
    input cin;
    output [7:0]sum;
    output cout;

    wire [7:0]p,g,c;
```

上学期写的 8 位超前进位加法器，输入 [7:0]a, [7:0]b, cin, 输出[7:0]sum, cout

```
    assign p=a|b;
    assign g=a&b;
    assign c[0]=g[0]|(p[0]&cin);
    assign c[1]=g[1]|(p[1]&g[0])|(p[1]&p[0]&cin);
    assign c[2]=g[2]|(p[2]&g[1])|(p[2]&p[1]&g[0])|(p[2]&p[1]&p[0]&cin);
    assign c[3]=g[3]|(p[3]&g[2])|(p[3]&p[2]&g[1])|(p[3]&p[2]&p[1]&g[0])|(p[3]&p[2]&p[1]&p[0]&cin);
    assign c[4]=g[4]|(p[4]&g[3])|(p[4]&p[3]&g[2])|(p[4]&p[3]&p[2]&g[1])|(p[4]&p[3]&p[2]&p[1]&g[0])|(p[4]&p[3]&p[2]&p[1]&p[0]&cin);
    assign c[5]=g[5]|(p[5]&g[4])|(p[5]&p[4]&g[3])|(p[5]&p[4]&p[3]&g[2])|(p[5]&p[4]&p[3]&p[2]&g[1])|(p[5]&p[4]&p[3]&p[2]&p[1]&g[0])|(p[5]&p[4]&p[3]&p[2]&p[1]&p[0]&cin);
    assign c[6]=g[6]|(p[6]&g[5])|(p[6]&p[5]&g[4])|(p[6]&p[5]&p[4]&g[3])|(p[6]&p[5]&p[4]&p[3]&g[2])|(p[6]&p[5]&p[4]&p[3]&p[2]&g[1])|(p[6]&p[5]&p[4]&p[3]&p[2]&p[1]&g[0])|(p[6]&p[5]&p[4]&p[3]&p[2]&p[1]&p[0]&cin);
    assign c[7]=g[7]|(p[7]&g[6])|(p[7]&p[6]&g[5])|(p[7]&p[6]&p[5]&g[4])|(p[7]&p[6]&p[5]&p[4]&g[3])|(p[7]&p[6]&p[5]&p[4]&p[3]&g[2])|(p[7]&p[6]&p[5]&p[4]&p[3]&p[2]&g[1])|(p[7]&p[6]&p[5]&p[4]&p[3]&p[2]&p[1]&g[0])|(p[7]&p[6]&p[5]&p[4]&p[3]&p[2]&p[1]&p[0]&cin);
    assign {cout,sum}=a^b^c,cin;

endmodule
```

```
module add4(cout,sum,a,b,cin);
    input [3:0]a,b;
    input cin;
    output [3:0]sum;
    output cout;

    wire [3:0]p,g,c;
```

```
    assign p=a|b;
    assign g=a&b;
    assign c[0]=g[0]|(p[0]&cin);
    assign c[1]=g[1]|(p[1]&g[0])|(p[1]&p[0]&cin);
    assign c[2]=g[2]|(p[2]&g[1])|(p[2]&p[1]&g[0])|(p[2]&p[1]&p[0]&cin);
    assign c[3]=g[3]|(p[3]&g[2])|(p[3]&p[2]&g[1])|(p[3]&p[2]&p[1]&g[0])|(p[3]&p[2]&p[1]&p[0]&cin);
    assign {cout,sum}=a^b^c,cin;

endmodule
```

```

module add2(cout,sum,a,b,cin);
  input [1:0]a,b;
  input cin;
  output [1:0]sum;
  output cout;

  wire [1:0]p,g,c;

  assign p=a|b;
  assign g=a&b;
  assign c[0]=g[0] | (p[0]&cin);
  assign c[1]=g[1] | (p[1]&g[0]) | (p[1]&p[0]&cin);
  assign {cout,sum}=a^b^(c,cin);

endmodule

```

```

module add1(cout,sum,a,b,cin);
  input a,b;
  input cin;
  output sum;
  output cout;

  wire p,g,c;

  assign p=a|b;
  assign g=a&b;
  assign c=g | (p&cin);
  assign {cout,sum}=a^b^(c,cin);

endmodule

```

Part2Testbench 代码以及部分 simulate 波形图

```

initial
begin
  txt=$fopen("result.txt");
  A=32'b0;
  B=32'b0;
  ALUop=3'b111;
  $fdisplay(txt,"4 A/B/Result O Z C\n");
  for(i=0;i<50;i=i+1)
  begin
    $fdisplay(txt,"A %b\nB %b",A,B);
    ALUop=3'b000;
    #5;$fdisplay(txt,"and %b %b %b %b",Result,Overflow,Zero,CarryOut);
    ALUop=3'b001;
    #5;$fdisplay(txt,"or %b %b %b %b",Result,Overflow,Zero,CarryOut);
    ALUop=3'b010;
    #5;$fdisplay(txt,"add %b %b %b %b",Result,Overflow,Zero,CarryOut);
    ALUop=3'b110;
    #5;$fdisplay(txt,"sub %b %b %b %b",Result,Overflow,Zero,CarryOut);
    ALUop=3'b111;
    #5;$fdisplay(txt,"slt %b %b %b %b\n",Result,Overflow,Zero,CarryOut);
    A={$random};
    B={$random};
  end
end

```

隔 50ns, ALUop 按顺序改变, 000,001,010,110,11

隔 250ns, A,B 改变

```

`timescale 10ns/1ns
module alu_test();
  reg [31:0]A,B;
  reg [2:0]ALUop;
  wire [31:0]Result;
  wire Overflow,Zero,CarryOut;
  integer txt,i;

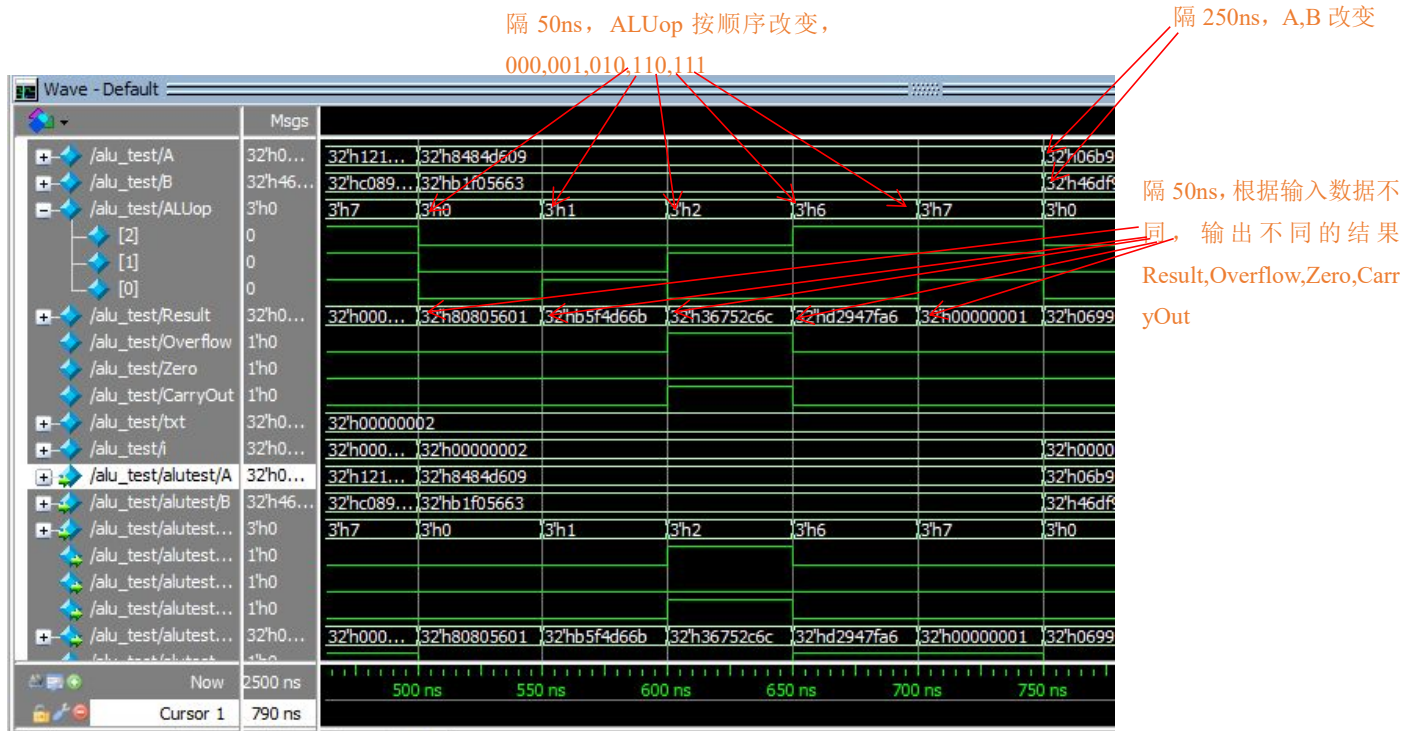
  alu alutest(A,B,ALUop,Overflow,Zero,CarryOut,Result);

```

时间精度设定

定义相应变量

调用 alu



Regfiles:

Part1 代码

```
module regfiles(clk,rst,wen,raddr1,raddr2,waddr,wdata,rdata1,rdata2);
    input clk,rst,raddr1,raddr2,waddr,wen;
    input [31:0]wdata;
    input [4:0]raddr1,raddr2,waddr;
    output [31:0]rdata1,rdata2;
    wire clk;
    reg [31:0]r[31:0];

    always @(posedge clk)
    begin
        if(wen==1 && waddr!=5'b0)
            r[waddr]<=wdata;
        end

        assign rdata1= (raddr1==5'b0)? 32'b0:r[raddr1];
        //assign rdata1= r[raddr1];
        assign rdata2= (raddr2==5'b0)? 32'b0:r[raddr2];

    endmodule
```

当时钟信号上升沿, 且写能信号 wen 为 1, 且写入位置不在 5'b0 的时候, 在寄存器堆相应位置写入数据

判断读取位置是否是 5'b0, 输出相应结果

Part2Testbench 代码以及部分 simulate 波形图

```

timescale 10ns/1ns
module regtest();
    reg [4:0] raddr1, raddr2, waddr;
    reg [31:0] wdata;
    reg wen;
    reg clk, rst;
    wire [31:0] rdata1, rdata2;
    integer result;
    reg [31:0] simreg[31:0];
    integer i;
    reg [31:0] simreg1, simreg2;

    regfiles test (clk, rst, wen, raddr1, raddr2, waddr, wdata, raddr1, raddr2);

    always #5clk=~clk;

    initial
    begin
        clk=0;
        result=$fopen("reg.txt");
        simreg[0]=32'b0;
        #10;
        repeat(1000)
        begin
            wdata=$random;
            raddr1={$random}%32;
            raddr2={$random}%32;
            waddr={$random}%32;
            wen={$random}%2;
            if(wen==1&&waddr!=5'b00000)
            begin
                simreg[waddr]=wdata;
                simreg1=simreg[raddr1];
                simreg2=simreg[raddr2];
                #10;
                if(rdata1==simreg1&&rdata2==simreg2)
                begin
                    $fdisplay(result,"addr: %h %h %h True",wdata,rdata1,rdata2);
                end
                else
                begin
                    $fdisplay(result,"addr: %h %h %h Error",wdata,rdata1,rdata2);
                end
            end
        end
    end
endmodule

```

图 1, 最开始的波形图

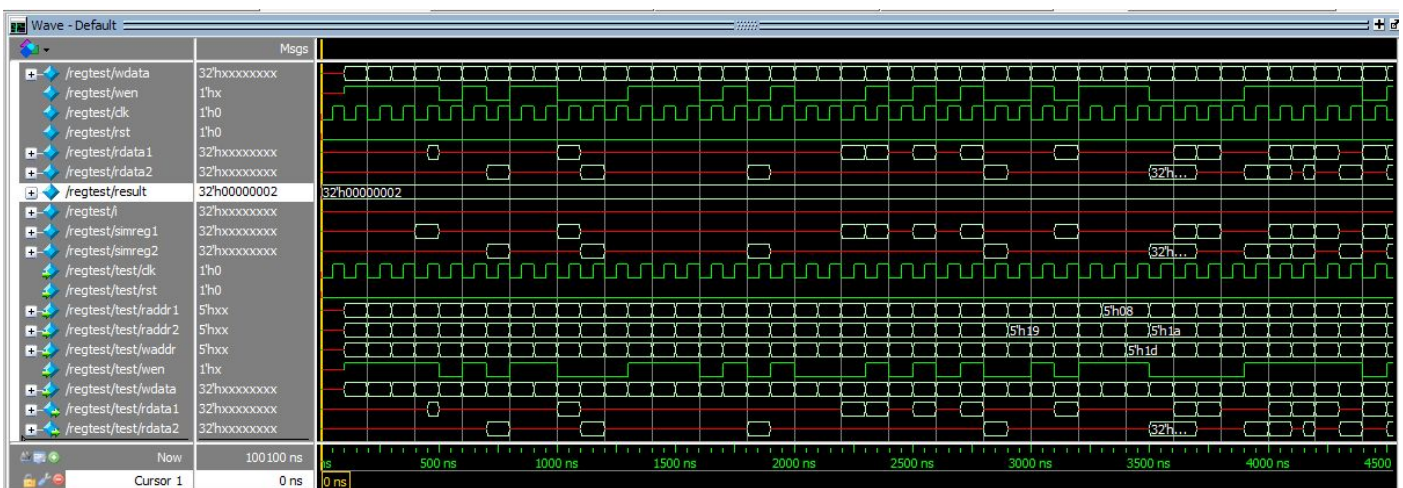
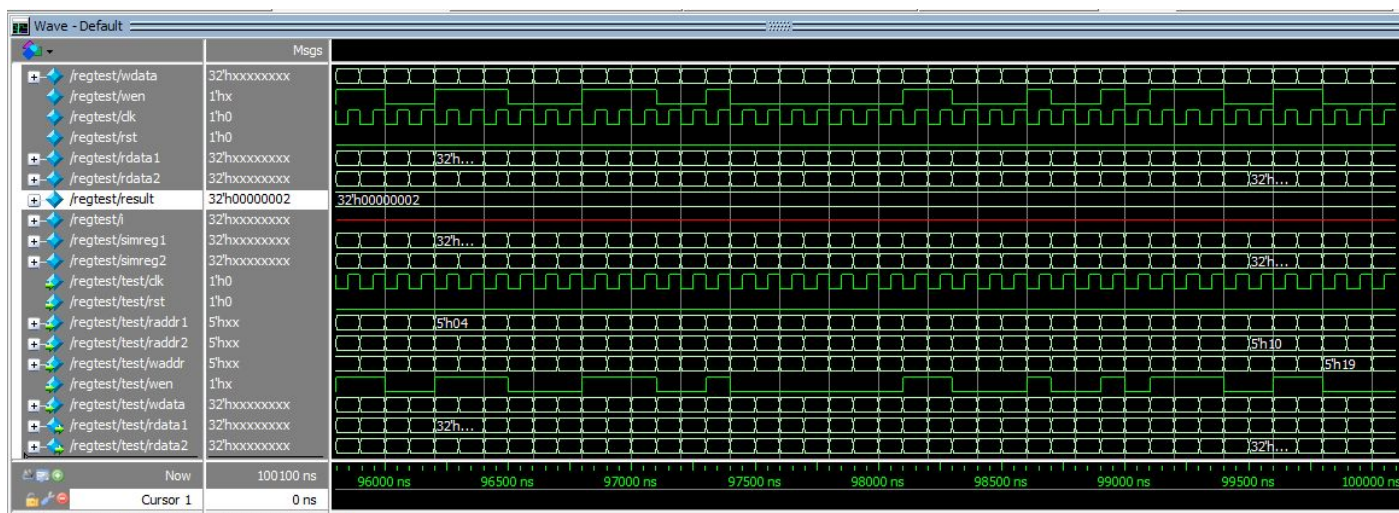


图 2，重复 1000 次读写过程，最后部分的波形图



（因为我的 testbench，所有的数据、读取写入地址以及写能信号全都是随机的，所以开始寄存器里基本没有值，所以基本是红线，最后重复次数多了，基本上所有位置都有了数据，所以红线没有了）
每 10ns，进行一次随机数据，输入输出，相应数据改变。

二、 问题合集

问题有很多，大部分是因为粗心等原因造成的，敲键盘打错（一直把 data 拼成了 date），忘记了“；”号，因为在中文输入法下“；”以及“;”（全半角的错误），最后忘记了“endmodule”和“endcase”等等，此类问题略。其他问题依靠着 transcript、百度还有同学的帮助下解决，但好多都忘记了是什么问题，以后会记得用//绿色保留错误版本。

（注，以下截图代码，绿色部分均为修正前，其他部分为修正后）

Problem1

```
always @(A or B or result1 or cout or ALUop)
//always @(result1 or ALUop)
```

没有将所有 always 内部涉及的数据写在（）内，导致程序出错，没有执行 always 操作。

后将所有涉及的数据都补上了。

Problem2

```
3'b111:
begin
    if( (~ (cout[6]^cout[5])&&result1[31]) || ((cout[6]^cout[5])&&cout[6]) )
        // (~ (cout[6]^cout[5])&&cout[6]) || ((cout[6]^cout[5])&&result1[31])
        result=32'b1;
    else
        result=32'b0;
end
```

逻辑错误，开始情况是溢出时候判断符号位，未溢出时判断符号位进位，判断反了，导致 slt 结果相反。

后根据输出的 txt 文本发现此问题，并修正。

Problem3


```
assign Zero=((~ALUop[2]&ALUop[1]&~ALUop[0])|(ALUop[2]&ALUop[1]&~ALUop[0])) & (~|result);
//assign Zero=((~ALUop[2]&ALUop[1]&~ALUop[0])|(ALUop[2]&ALUop[1]&~ALUop[0])) & ~(result[0]|result[1]|result[2]|
//result[3]|result[4]|result[5]|result[6]|result[7]|result[8]|result[9]|result[10]|result[11]|result[12]|
//result[13]|result[14]|result[15]|result[16]|result[17]|result[18]|result[19]|result[20]|result[21]|
//result[22]|result[23]|result[24]|result[25]|result[26]|result[27]|result[28]|result[29]|result[30]|result[31]);
assign Cout=((~|Hex[23]&|Hex[23]&~|Hex[23])|(|Hex[23]&|Hex[23]&~|Hex[23])) & Cout[6];
```

原本没有使用归约操作符，对每一位进行操作，复杂又容易打错，并且当位数更多时，全部打出显然很复杂。

后在助教老师指导下，看 IEEE Standard for Verilog Hardware Description Language, IEEE Std 1364 -2005 第 5.1 节部分，改用归约符号，简洁方便。

Problem4

```
add8 p1(cout[0],sum[7:0],a[7:0],b[7:0],cin);
add8 p2(cout[1],sum[15:8],a[15:8],b[15:8],cout[0]);
add8 p3(cout[2],sum[23:16],a[23:16],b[23:16],cout[1]);
add4 p4(cout[3],sum[27:24],a[27:24],b[27:24],cout[2]);
add2 p5(cout[4],sum[29:28],a[29:28],b[29:28],cout[3]);
add1 p6(cout[5],sum[30],a[30],b[30],cout[4]);
add1 p7(cout[6],sum[31],a[31],b[31],cout[5]);
//add8 p1(cout[0],sum[7:0],a[7:0],b[7:0],cin);
//add8 p2(cout[1],sum[15:8],a[15:8],b[15:8],cout[0]);
//add8 p3(cout[2],sum[23:16],a[23:16],b[23:16],cout[1]);
//add8 p4(cout[3],sum[31:24],a[31:24],b[31:24],cout[2]);
|
```

原本 32 位加法器中，直接由 4 个 8 位加法器组成，导致最高位的进位未知，许多结果无法得出（如 Overflow）

后将最后一次进行 8 位加法器拆分为，4,2,1,1 位加法器的组成，cout[5]储存着最高位的进位，cout[6]储存着符号位的进位

Problem5

```
always @(posedge clk)
//always @(posedge clk & wen)
//always @(posedge clk and wen)
```

最开始版本是最下面 and wen，但在 compile 环节出错，不知道为啥，但我尝试着改成&后编译成功，逻辑上似乎也没啥错误，当 wen 位 0 的时候，不管 clk 如何，永远没有上升沿，所以不执行写入。

后助教老师看过程序之后，告诉我在 always 里面只能写 posedge/negedge/and/or，如果加入&，会综合出一个 blablabla，具体忘记了，反正就是容易出错。修正如上，外加在 always 内部加上 if 判断 wen。

Problem6

```
assign rdata1= (raddr1==5'b0)? 32'b0:r[raddr1];
//assign rdata1= r[raddr1];
```

一开始直接给 rdata1 赋值 r[raddr1]，这个问题也跟我程序有关，我主体程序中并没有对 r[0]进行赋值，并且如果写入地址为 5'b0 的时候，不进行写入，所以如果当读取地址为 5'b0 的时候，无输出，波形图为红线，高阻态。

后在 assign 内部加一个？：判断语句，修正了这一错误。

三、 对讲义中思考题的理解和回答

Question1

温故而知新

回顾数字电路的知识,寄存器属于组合逻辑电路,还是时序逻辑电路?你还记得什么是时序逻辑电路吗?

寄存器属于时序逻辑电路。

时序逻辑电路，与组合逻辑电路的不同就是，它包含触发器，即它不仅和输入信号有关，还跟以前的电路状态有关。

Question2


对比位宽与地址长度

现在你应该可以理解 `reg [7:0] r` 和 `reg r [7:0]` 的区别了. 尝试总结verilog中各种reg类型变量的定义方式与其含义.

`reg [a:0]r[b:0]` 表示 b+1 个 a+1 位的寄存器

四、 对于此次实验的心得、感受和建议

在以前做过数字电路大作业 alu 的基础上，此次作业并不难，并且操作相对之前那个 alu 少很多，工作量没有那么吓人，（不好意思的说，我之前的 alu 没好好做，因为功能太多了，打开电脑就崩溃的感觉），并且

有  [计算机组成原理实验指导.pdf](#) 清晰的中文解释，所以觉得很好。（如果 IEEE standard for verilog 也能有中文版就好了，看到英文真的头疼。）

写+调试代码的时候，出现了各种大错误小错误，有的自己解决了，有的在同学（在此感谢谈清扬的 mvp，以及张传奇/张北辰/方言歌等同学的助攻）的帮助下顺利解决，还有的由助教帮忙指出并解决。顺便高兴一下，自己也可以帮助解决以前没学过数字电路的同学的部分问题。（话说，助教能给个除了邮件之外的联系方式吗，比如 qq 比如微信什么的，有的东西问同学，只知道怎么改，不知道为什么这样改。还有和同学交流产生疑问，需要跟助教求证，所以想要一下联系方式。）