ZPC 2020 ZHEJIANG UNIVERSITY 浙江大学计算机学院

2020

运算器设计

咱们工人有力量！

下岗工人再就业队

东 4-509

2020-8-22

运算器设计报告

小组基本信息

组名：下岗工人再就业队

项目完成人员：黄仁泓、令狐鹏

完成时间：2020.8.22

指导老师：楼学庆

使用手册

操控

SW[15:13]三位控制从有八个A数值的内存中取出某一个作为A的数值，SW[12:10]控制取B的值，他们的值的更改在txt文档里实现

SW[8:6]控制整数的乘除法方面

```
            if(flag==3'b000)
                ALU_out <= Shang1;
            else if(flag==3'b001)
                ALU_out <= Yushu1;
            else if(flag==3'b100)
                ALU_out <= Shang2;
            else if(flag==3'b101)
                ALU_out <= Yushu2;
            else if(flag[1:0]==2'b10)
                ALU_out <= Div;
```

这三位数是000的时候ALU显示的是无符号除法的商的64位，001是无符号除法的余数，同理对100是有符号数等等。当中间及SW[7]为1的时候，显示的就是新的被除数，是64位的被除数。

SW[5:4]控制显示Seg的内容

```
case(SW[5:4])
            2'b00:SegShow<=ALU_out[31:0 ];
            2'b01:SegShow<=ALU_out[63:32];
            2'b10:SegShow<=A;
            2'b11:SegShow<=B;
        endcase
```

00的时候显示低32位的ALU输出结果，01显示高32位。

10显示A数，11显示B数

SW[3:0]控制显示什么运算的结果：

```
    case (mode)
      4'b0000: ALU_out <= {32'b0,A};
      4'b0001: ALU_out <= {32'b0,B};
      //Logic Operation
      4'b0010: ALU_out <= {32'b0,And};
      4'b0011: ALU_out <= {32'b0,Or};
      4'b0100: ALU_out <= {NotB,NotA};
      4'b0101: ALU_out <= {32'b0,Xor};
        //Shift Operation
      4'b0110: ALU_out <= {ShiftL,ShiftR};
      4'b0111: ALU_out <= {ShiftC,ShiftAR};
      //Int Operation
      4'b1000: begin
           ALU_out <= {32'b0,Add};
            //符号SF、进位CF、溢出OF、零标志ZF、奇偶PF
            //进位 溢出 符号(为1说明a比b小，减法) 奇偶 零标志
            flagOut<={CFs[0],OFs[0],SFs[0],PFs[0],ZFs[0]};
            end
      4'b1001: begin
           ALU_out <= {32'b0,Sub};
            flagOut<={CFs[1],OFs[1],SFs[1],PFs[1],ZFs[1]};
            end
      4'b1010: ALU_out <= {Multiply};
      4'b1011: begin
            if(flag==3'b000)
                ALU_out <= Shang1;
            else if(flag==3'b001)
                ALU_out <= Yushu1;
```

```
            else if(flag==3'b100)
                ALU_out <= Shang2;
            else if(flag==3'b101)
                ALU_out <= Yushu2;
            else if(flag[1:0]==2'b10)
                ALU_out <= Div;
                end
        //Float Operation
        4'b1100: ALU_out <= {32'b0,Float_Add};
        4'b1101: ALU_out <= {32'b0,Float_Sub};
        4'b1110: ALU_out <= {32'b0,Float_Multiply};
        4'b1111: ALU_out <= {30'b0,OF,Float_Shang};
    endcase
```

开关控制的具体内容都在代码里面了，名称都比较好看懂

显示

调用了小的十六位数码管显示了我们按键的值，第一位显示A的选择数，然后是B的选择数，后面两个显示的是5-0的SW选择数或者在加减法的时候是作为了标志信号的输出，依次的内容是：进位 溢出 符号(为1说明a比b小，减法) 奇偶 零标志

大的八位数码管是作为结果的显示的，比如加减法的结果，乘除法的结果等。也可以显示我们的操作数。

输入

我们的输入也是在txt文档的方法上面进行的，A和B的txt文档都是8个32位二进制数组成的，一行为一个数，根据选择读取对应的数来进行计算。比较特殊的是整数除法，除法必须要64位的操作数，所以我单独开了一个txt文档读入64位的被除数。可以修改文件获得其他的输入数据。

运算器基本信息

运算器支持基本的运算操作

移位、与、或、加减乘除、浮点计算等操作。符合课程设计要求。

分工信息

移位等+浮点运算：黄仁泓

整数+报告书写：令狐鹏

浮点类型运算研究

一、简要介绍

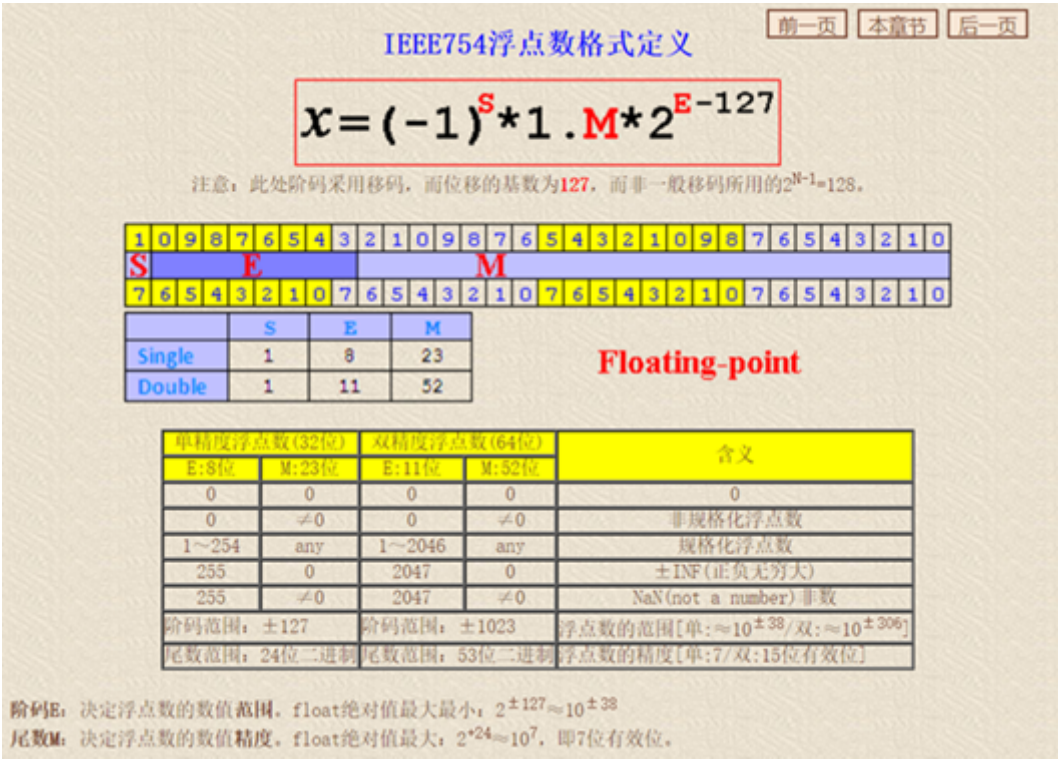
本次大实验，我负责的是浮点数运算部分，浮点运算总共分为三大块，分别是浮点数加法和减法、浮点数乘法、浮点数除法。

二、具体算法推导、实现与模拟

研究浮点类型运算，首先先从浮点数格式定义入手。

单精度浮点数共32bits，分为符号位S,阶码E,尾码M，因此处理单精度浮点数需首先将浮点数划分为三块分别进行加减乘除的运算。

32位浮点数 = 第31位为符号位 + 第30~23位为偏移码（阶码）+ 第22~0位为尾数**



	S	E	M
Single	1	8	23
Double	1	11	52

单精度浮点数(32位) E:8位	单精度浮点数(32位) M:23位	双精度浮点数(64位) E:11位	双精度浮点数(64位) M:52位	含义
0	0	0	0	0
0	≠0	0	≠0	非规格化浮点数
1～254	any	1～2046	any	规格化浮点数
255	0	2047	0	±INF(正负无穷大)
255	≠0	2047	≠0	NaN(not a number)非数
阶码范围：±127		阶码范围：±1023		浮点数的范围[单:≈$10^{\pm 38}$/双:≈$10^{\pm 306}$]
尾数范围：24位二进制		尾数范围：53位二进制		浮点数的精度[单:7/双:15位有效位]

图表 1 浮点格式

1.浮点加法：

实现X±Y运算,要用如下五步完成:

(1) 对阶操作,即比较两个浮点数的阶码值的大小.求△E=Ex-Ey。当其不等于零时,首先应使两个数取相同的阶码值。将原来阶码小的数的尾数右移|△E|位,其阶码值加上|△E|,即每右移一次尾数要使阶码加1,则该浮点数的值不变。尾数右移时,对原码形式的尾数,符号位不参加移位,尾数高位补0。

(2) 实现尾数的加(减)运算,对两个完成对阶后的浮点数执行求和(差)操作。

(3) 规格化处理,若得到的结果不满足规格化规则,就必须把它变成规格化的数,采取右规或左规策略

(4) 舍入操作。在执行对阶或右规操作时,会使尾数低位上的一位或多位的数值被移掉,使数值的精度受到影响,可以把移掉的几个高位的值保存起来供舍入使用。常用的办法有"0"舍"1"入法,即移掉的最高位为1时 则在尾数末位加1，为0时则舍去移掉的数值。尾数加上1后又可能出现溢出，故还需继续规范化。

(5) 判结果的正确性,即检查阶码是否溢出,若阶码正常，加(减)运算正常结束；若阶码下溢,要置运算结果为浮点形式的机器零,若上溢,则置溢出标志。

特殊情况处理

\1. 移位时发生上溢则置浮点0x7F8xxxxx(这个范围的浮点数在IEEE标准中均为NAN标志)。

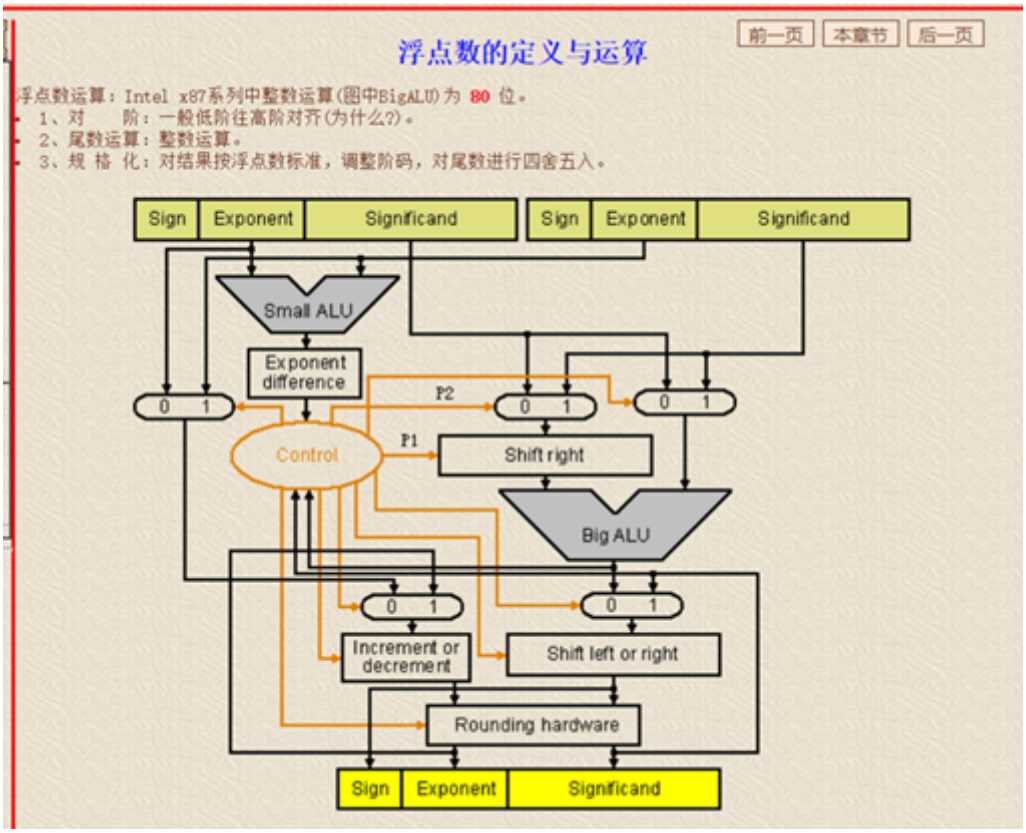
若发生下溢，则置浮点0x00000000，这个很好理解，因为当他指数足够小就可以看成0。

\2. 舍入方式：

在IEEE754标准中，舍入处理提供了四种可选方法：就近舍入、朝0舍入，朝+∞舍入

朝-∞舍入，但经过比较，就近舍入的方式比较适合，因为朝0舍入只是简单的截尾，无论是正数还是负数，这种方法容易导致误差的积累。而朝单方向舍入也会造成误差积累。

\3. 如果移位移动超过了24bit，也即将尾数基本移出，显然有个数相对于另一个数大很多（几乎是2^24倍）,我们可以忽略另一个加数，直接返回较大的数。



图表 2 浮点数加法器

设计原理：

采用状态机的思想，通过时序实现浮点数的定义和运算。对应着上述算法，总共给状态机分配五种状态：

状态机：**start->zerock->exequal->addm->infifl->over**

Start**：**Start**分离阶码、尾数、符号位

Zerock**：**Zerock**特殊情况处理（如0）

Exequal**：**Exequal**指数处理，使得指数相等——对阶

Addm**：**Addm**带符号位和保留进位的尾数相加——尾数加减运算

Infifl**：**Infifl**尾数规格化处理和舍入处理

Over: 输出阶段

程序代码：

```
module float_add(
  input [31:0]ix,
  input [31:0]iy,
  input clk,
  output reg [31:0]oz
  );

wire[31:0] ix,iy;
wire ost;
reg[25:0] xm, ym, zm;  //尾数（空余多位用以近似）
reg[7:0]  xe, ye, ze;  //阶码
reg[2:0] state;        //浮点加法状态
```

```
parameter
start=3'b000,zerock=3'b001,exequal=3'b010,addm=3'b011,infifl=3'b100,over
=3'b110;
assign ost = (state == over) ? 1 : 0;

reg temp=0;

always@(posedge ost) //若已经完成，进行赋值
   begin
     oz <= {zm[25],ze[7:0],zm[22:0]};
   end

always@(posedge clk)
  begin
    case(state)
      start:        //划分阶段
        begin
          xe <= ix[30:23];
          xm <= {ix[31],1'b0,1'b1,ix[22:0]};
          ye <= iy[30:23];
          ym <= {iy[31],1'b0,1'b1,iy[22:0]};
          state <= zerock;
        end
      zerock:
        begin
            //若指数为0，默认为0
          if(ix==0)
            begin
              {ze, zm} <= {ye, ym};
              state <= over;
            end
          else if(iy==0)
            begin
              {ze, zm} <= {xe, xm};
               state <= over;
            end
          else if(xe==8'hFF&&ix[22:0]!=0)
                begin
                    ze<=8'hFF;
                     zm<=26'h3FFFFF;
                    state <= over;
                end
             else if(xe==8'hFF&&ix[22:0]==0)
                begin
                    ze<=8'hFF;
                     zm<=26'h0;
                    state <= over;
                end
            else
            state <= exequal;
        end
      exequal: //对阶
        begin
          if(xe == ye)//若指数相同直接相加
            state <= addm;
          else if(xe > ye) //指数不同
            begin
              ye <= ye + 1;
```

```
            ym[24:0] <= {1'b0, ym[24:1]};
            if(ym == 0)
              begin
                zm <= xm;
                ze <= xe;
                state <= over;
              end
            else
              state <= exequal;
          end
        else
          begin
            xe <= xe + 1;
            xm[24:0] <= {1'b0,xm[24:1]};
            if(xm == 0)
              begin
                zm <= ym;
                ze <= ye;
                state <= over;
              end
            else
          state <= exequal;
          end
      end
    addm:
      begin
        if ((xm[25]^ym[25])==0) //符号位相同
          begin
            zm[25] <= xm[25];
            zm[24:0] <= xm[24:0]+ym[24:0];
          end
        else
          if(xm[24:0]>ym[24:0])  //根据尾数大小比较
          begin
            zm[25] <= xm[25];
            zm[24:0] <=xm[24:0]-ym[24:0];
          end
        else
          begin
            zm[25] <= ym[25];
            zm[24:0] <=ym[24:0]-xm[24:0];
          end
        ze <= xe;
        state <= infifl;
      end
    infifl://规范化
      begin
        if(zm[24]==1&&zm[0]==0)  //右规（只需要规范一次）
          begin
            zm[24:0] <= {1'b0,zm[24:1]};
            ze <= ze + 1;   //指数增加
            state <= over;
          end
        if(zm[24]==1&&zm[0]==1)  //右规（只需要规范一次）
          begin
            zm[24:0] <= {1'b0,zm[24:1]}+1;
            ze <= ze + 1;   //指数增加
            state <= over;
```

```
            end
          else if(zm[23]==0)   //左规
            begin
              zm[24:0] <= {zm[23:0],1'b0};
              ze <= ze - 1;
              state <= infifl;
            end
          else
            state <= over;
        end
      over:
        begin
         state<= start;
        end
      default:
        begin
          state<= start;
        end
    endcase
  end


endmodule
```

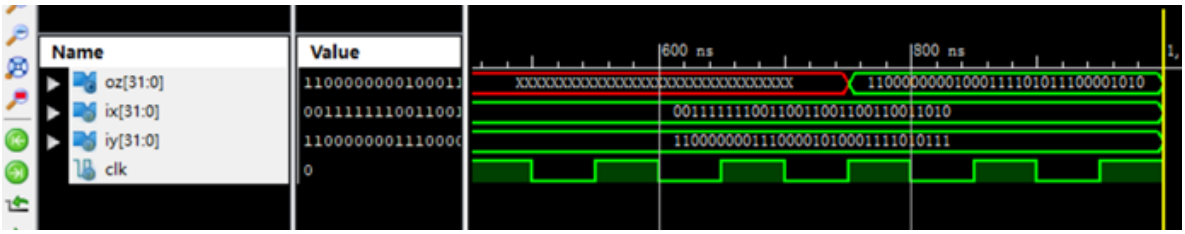
由于代码较长，可见上传附件

程序模拟：

测试代码如下：（test1）

```
module test1;



    // Inputs

    reg [31:0] ix;

    reg [31:0] iy;

    reg clk;

    // Outputs

    wire [31:0] oz;

    // Instantiate the Unit Under Test (UUT)

    float_add uut (

        .ix(ix),

        .iy(iy),

        .clk(clk),

        .oz(oz)

```

```
29      );
30
31      initial begin
32
33          // Initialize Inputs
34
35          ix = 0;
36
37          iy = 0;
38
39          clk = 0;
40
41          // Wait 100 ns for global reset to finish
42
43          \#100;
44
45      ix=32'b00111111_10011001_10011001_10011010;
46
47      iy=32'b11000000_01110000_10100011_11010111;
48
49          forever
50
51            \#50 clk=~clk;
52
53    end
54
55  endmodule
```



$$ix = 1.2000000476837158 iy = -3.7599999904632570z = -2.559999942779541$$

与预期完全相符

2.浮点减法：

浮点减法就是建立在浮点数加法的基础上进行，类比补码的加减法，如果我们要计算A-B我们只需要计算A+(-B)即可。而参考浮点的格式，只需要将符号位取反即可获得负的浮点数。

特殊情况处理：

\1. 溢出和舍入：因为调用的是浮点加法器，所以情况同浮点加法相同。

\2. 关于0：因为在IEEE标准中，我们定义了0x00000000为0的浮点表示，而我们通过符号位取反，将导致出现0x80000000情况，而0x80000000的值为下图所示，因此我们只需判断是否为0时跳过符号位即可。

-0.0E0

Most accurate representation = -0.0E0

程序代码：

```
module float_sub(
  input [31:0]ix,
  input [31:0]iy,
  input clk,
  output reg [31:0]oz
  );
wire [31:0] result;
float_add(.ix(ix),.iy({~iy[31],iy[30:0]}),.clk(clk),.oz(result));

always @*
  begin
    oz <= result;
  end

endmodule
```

由于与只在加法基础上做了小小改动，因此测试代码与加法器相同。

3.浮点乘法

正如我们一开始提出的，处理浮点数需要分开三部分处理。

设两个规格化浮点数分别为$A = Ma * 2^{Ea}$，$B = Mb * 2^{Eb}$，
$A + B = (Ma + Mb * 2^{-(Ea-Eb)}) * 2^{Ea}$ (假设Ea>=Eb)
$A - B = (Ma - Mb * 2^{-(Ea-Eb)}) * 2^{Ea}$ (假设Ea>=Eb)
$A * B = (Ma * Mb) * 2^{Ea+Eb}$
$A/B = (Ma/Mb) * 2^{Ea-Eb}$
上述运算结果可能出现以下几种情况：

阶码上溢	一个正指数超过了最大允许值	+∞/-∞/溢出
阶码下溢	一个负指数超过了最小允许值	+0/-0
尾数溢出 (尾数溢出，结果不一定溢出)	最高有效位有进位	右规
非规格化尾数	数值部分高位为0 右规或对阶时，右段有效位丢失	左规 尾数舍入

指数相加，尾数相乘，符号位异或。

对于指数相加，鉴于时移码运算，可推导为：

$$X_{移} + Y_{移} = X+Y-127$$

而对于尾数乘法，我们可推导一直便于计算的方法：

$$m * 1.n = (1 + 0.m)(1 + 0.n) = 1 + 0.m + 0.n + 0.mn = 1.mn + 0.m + 0.n$$

其中m,n即为ix，iy的尾数部分，所以两尾数乘完后必定出现1，（而且准确的说经过推导一定在2x+1位，x为尾数的位数）但我们只取24bit（其中最高位为1要省去），需要进行右规，同时右规过程中考虑舍入，如果舍去为1，则尾数加一否则则直接舍去。

特殊情况：

\1. 遇到溢出，则类似加法器判断是上溢还是下溢，上溢则置NAN，下溢则置0，同时溢出信号置1。

\2. 遇到其中一个乘数为0，则直接返回结果0；

硬件实现：

类似加法器，我们也采用状态机操作

状态机：**start->calculate->over->exp->out**

1. **Start****：**分离阶码、尾数、符号位
2. **calculate****：**尾数相乘
3. **over****：**规范化尾数乘法结果
4. **exp****：**指数相加操作
5. **out****：**输出步骤

程序代码

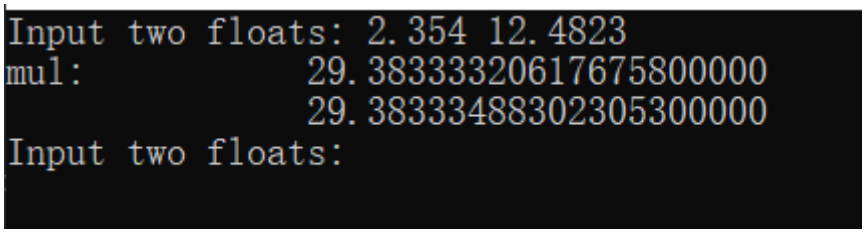
```
module float_mul(
  input [31:0] ix,
  input [31:0] iy,
  input clk,
  input rst,
  output reg [31:0] oz,
  output reg Yichu
  );

reg s1,s2,s3;
reg [7:0] exp1,exp2,exp3;  //指数
reg [8:0] temp3;           //指数运算
reg [22:0] man1,man2,man3; //尾数
reg n;
reg [23:0] temp;
reg [45:0] comeout;
reg [23:0] all;            //状态机
reg [1:0] zheng;
reg [2:0] state = 0;

always@(posedge clk) //浮点乘法状态机
  begin
    if (state == 0)//提取符号，指数，尾数
     begin
        s2 <= iy[31];
        exp2 <= iy[30:23];
        man2 <= iy[22:0];
        s1 <= ix[31];
        exp1 <= ix[30:23];
        man1 <= ix[22:0];
        state <= 1;
          if(ix==0||iy==0)
            begin
            Yichu <= 0;
            oz <= 32'h0;
            state <= 0;
            end
          else if(exp1==8'hFF&&ix[22:0]==0)
            begin
               Yichu <= 0;
                oz <= 32'h7F800000;
                state <= 0;
             end
```

```
            else if(exp1==8'hFF&&ix[22:0]!=0)
             begin
               Yichu <= 0;
                oz <= 32'hFFFFFFFF;
                state <=0;
             end
    end

    if (state == 1)
     begin
        comeout <= man1 * man2;
        temp <= man1 + man2;    //1.m*1.n=1+(0.m+0.n)+(0.m*0.n)
      all <= temp[22:0] + comeout[45:23];
      zheng <= 1'b1 + temp[23] + all[23];
      state <= 4;
    end

    if (state == 4)
     begin
      if(zheng[1]==1)
          begin
            n <= 1'b1; //用于移位
            man3[22:0] <= {zheng[0],all[22:1]};
          end
      else
          begin
            n <= 1'b0;
            man3 <= all[22:0];
          end
      state <= 2;
    end

    if (state == 2)  //处理指数
     begin
       temp3 <= exp1 + exp2 - 127 + n;
       state <= 3;
   end

    if (state == 3)
     begin
        s3 <= s1^s2;
        Yichu <= temp3[8];
        oz <= {s3,temp3[7:0],man3[22:0]};
       state <= 0;
   end
  end

endmodule
```

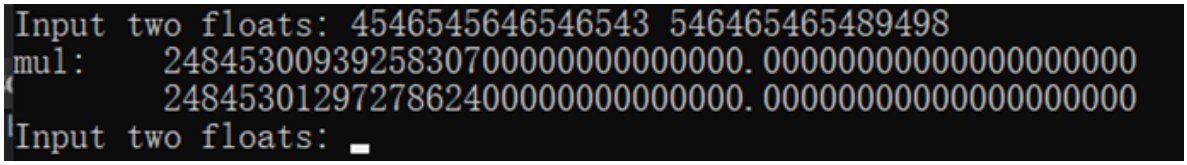
代码较长见上传附件。

程序模拟：

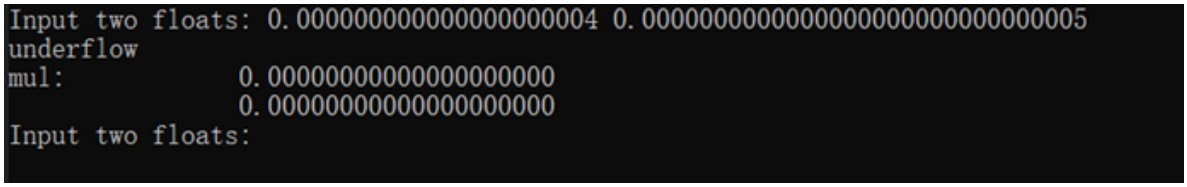
这次我们采用C程序来验证乘法算法：(上面是应用了我们的算法获得的，下面是利用计算机本身的乘法获得)



由此可见，在前7位都是与标准实现相同，而后面的误差可能是由于舍入规则不同导致。



大数处理，也是7位相同



溢出处理

均与实验预期相符

4.浮点除法**

类比浮点乘法,处理浮点数需要分开三部分处理。

符号位异或，尾数相除，指数相减

尾数相除算法：

其实总结加减乘除，我们处理浮点数最重要是求得尾数部分的值，而除法也和乘法类似，通过移位，从而控制处理后我们能取得24bits的尾码（最高位的1最后转化要舍弃）。

具体算法如下：

假设我们有ix，iy的尾码分别为xm,ym，

如果**xm>ym，**则1.xm/1.ym必定保证了结果为1.xxxx，若想保证x的位数为尾码的位数（23bits），只需将xm左移23bits，再进行除法操作即可。

如果**xm<ym，**则1.xm/1.ym必定保证了结果为0.1xxx，若想保证x的位数为尾码的位数（23bits），则将xm左移24bits，再进行除法操作，注意若执行这个操作，则相当于左移了一位，在之后的指数运算则需纳入考虑。

指数相减：

对于指数相加，鉴于时移码运算，可推导为：

$$X_{移} - Y_{移} = X\text{-}Y+127$$

特殊情况

\1. 遇到溢出，则类似加法器判断是上溢还是下溢，上溢则置NAN，下溢则置0，同时溢出信号置1。

\2. 遇到被除数为0，则直接返回结果0；

\3. 遇到除数为0，则返回错误的信号，置为INF；

硬件实现：

类似加法器，我们也采用状态机操作

状态机：**start->div->ex->out**

1. **Start****：**分离阶码、尾数、符号位
2. **div****：**尾数相除
3. **ex****：**指数相减操作
4. **out****：**输出步骤

程序代码

```
module float_div(
  input [31:0] ix,
  input [31:0] iy,
  input clk,
  input rst,
  output reg [31:0] oz,
  output reg [1:0] Yichu
  );

reg [1:0] overflow;
reg s1,s2,s3; //符号位
reg [7:0] exp1,exp2,exp3; //指数
reg [22:0] man1,man2,man3; //尾数
reg n;
reg [8:0] temp1;
reg [8:0] temp2;
reg [8:0] temp3;
reg [23:0] temp4;
reg [23:0] temp5;
reg [24:0] tepman1;
reg [24:0] tepman2;
reg [47:0] tepman3;
reg [1:0] state = 0;


always@(posedge clk) //浮点数除法状态机
  begin
    if (state == 0)
     begin
            if(iy==32'b0)
              begin
                     Yichu <=0;
                     oz <=32'hFFFFFF;
                     state<=0;
              end
            else if(ix==32'b0)
              begin
                     Yichu <=0;
                     oz <=32'h0;
                     state<=0;
              end
            else if(ix==32'h7F800000)
              begin
```

```
                    Yichu <=0;
                    oz <=32'h7F800000;
                    state<=0;
              end
               else if(ix[30:23]==8'hFF&&ix[22:0]!=0)
              begin
                    Yichu <=0;
                    oz <=32'hFFFFFFFF;
                    state<=0;
              end
            else if(ix==32'b11000000000000000000_00000000000000)
              begin
                s1<=ix[31];
                exp1<=ix[30:23];
                man1<=ix[22:0];
                s2<=1'b0;
                exp2<=8'b00000000;
                man2<=23'b0;
                    state <= 2'b01;
              end
            else if(iy==32'b11000000000000000000_00000000000000)
              begin
                s2<=iy[31];
                exp2<=iy[30:23];
                man2<=iy[22:0];
                s1<=1'b0;
                exp1<=8'b00000000;
                man1<=23'b0;
                    state <= 2'b01;
              end
            else
                  begin
                s2<=iy[31];
                exp2<=iy[30:23];
                man2<=iy[22:0];
                s1<=ix[31];
                exp1<=ix[30:23];
                man1<=ix[22:0];
                    state <= 2'b01;
              end
    end

    if (state == 1)
     begin
      temp4 <= man1 + ~man2;
      temp5 <= temp4 +1;
      if(temp5[23] == 1)  //判断是否移位
        n<= 1'b1;
      else
        n<= 1'b0;
      if(n == 0)
      begin
      tepman1 <={n,1'b1,man1};
      tepman2 <={2'b01,man2};
      tepman3 <={tepman1,23'b0};
      man3 <= tepman3 / tepman2;
      end
      else
```

```
      begin
      tepman1 <={1'b1,man1,1'b0};
      tepman2 <={2'b01,man2};
      tepman3 <={tepman1,23'b0};
      man3 <= tepman3 / tepman2;
      end

      state <= 2;
    end

    if (state == 2)
     begin
      temp3 <= exp1-exp2+127- n;
      exp3 <= temp3[7:0];
      state <= 3;
   end

    if (state == 3)
     begin
        s3 <= s1^s2;
      Yichu <= temp3[8];
        oz <= {s3,exp3,man3[22:0]};
       state <= 0;
   end
  end

endmodule
```

代码较长见上传附件。

程序模拟：

这次我们采用C程序来验证乘法算法：(上面是应用了我们的算法获得的，下面是利用计算机本身的乘法获得)

```
Input two floats: 4556.15213 12312.11111
div:              0.37005448341369629000
                  0.37005451155580577000
```

与乘法类似，因为舍入规则不同导致最后一位可能与计算机算的出现误差，因此基本上保证7位相同（可以从23bit的尾码能确定7位精确的数）

```
Input two floats: 0 123131565
div:              0.00000000000000000000
                  0.00000000000000000000
```

被除数为0，自动为0

```
Input two floats: 23132651 0.00000000000000000000000000000000000000000001
overflow!
div:              1.#QNAN000000000000000
```

溢出处理

```
Input two floats: 4531.65165 0
        error!
div:              1.#INF0000000000000000
                  1.#INF0000000000000000
```

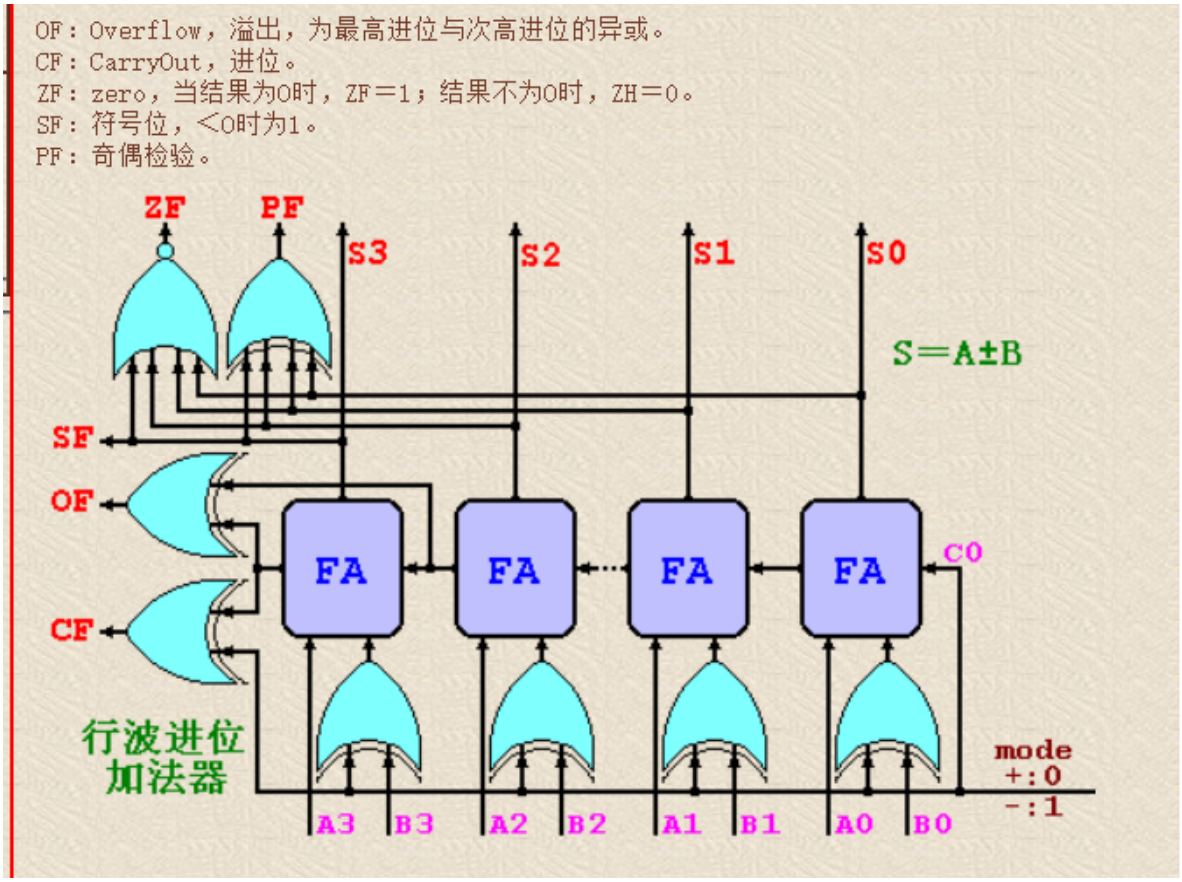
当除数为0，报错并设定值为INF(0x7F800000)

整数类型运算研究

因为移位啊和或啥的比较简单，就不进行说明了。下面说明一下加减乘除的方法。

加法、减法

加减法采用的是行波进位加法器的方法，虽然比较慢但是其标志位比较明显有用，便于操作，具体实验图如下：



多个FA模块进行加减操作，可以很方便地求出各个标志位、进位和最后的结果

mode模块就是用于控制加法或者是减法，只需要改变就可以实现代码重用，非常方便。

各个符号位是全部组合成一个五位数然后集体输出的，当是加减法的时候会显示在小数码管上面后两个。

具体加减法器代码如下：

```
module FA_32bits(
    input Ctrl,
    input [31:0]A,B,
    output [31:0]s,
    output OF,CF,ZF,SF,PF
    );
    wire [31:0]ca;
        FA f0(.a(A[0]),.b(B[0]^Ctrl),.c0(Ctrl), .s(s[0]),.Ca(ca[0]));
        FA f1(.a(A[1]),.b(B[1]^Ctrl),.c0(ca[0]),.s(s[1]),.Ca(ca[1]));
        FA f2(.a(A[2]),.b(B[2]^Ctrl),.c0(ca[1]),.s(s[2]),.Ca(ca[2]));
        FA f3(.a(A[3]),.b(B[3]^Ctrl),.c0(ca[2]),.s(s[3]),.Ca(ca[3]));
        FA f4(.a(A[4]),.b(B[4]^Ctrl),.c0(ca[3]),.s(s[4]),.Ca(ca[4]));
```

```
        FA f5(.a(A[5]),.b(B[5]^Ctrl),.c0(ca[4]),.s(s[5]),.Ca(ca[5]));
        FA f6(.a(A[6]),.b(B[6]^Ctrl),.c0(ca[5]),.s(s[6]),.Ca(ca[6]));
        FA f7(.a(A[7]),.b(B[7]^Ctrl),.c0(ca[6]),.s(s[7]),.Ca(ca[7]));
        FA f8(.a(A[8]),.b(B[8]^Ctrl),.c0(ca[7]),.s(s[8]),.Ca(ca[8]));
        FA f9(.a(A[9]),.b(B[9]^Ctrl),.c0(ca[8]),.s(s[9]),.Ca(ca[9]));
        FA f10(.a(A[10]),.b(B[10]^Ctrl),.c0(ca[ 9]),.s(s[10]),.Ca(ca[10]));
        FA f11(.a(A[11]),.b(B[11]^Ctrl),.c0(ca[10]),.s(s[11]),.Ca(ca[11]));
        FA f12(.a(A[12]),.b(B[12]^Ctrl),.c0(ca[11]),.s(s[12]),.Ca(ca[12]));
        FA f13(.a(A[13]),.b(B[13]^Ctrl),.c0(ca[12]),.s(s[13]),.Ca(ca[13]));
        FA f14(.a(A[14]),.b(B[14]^Ctrl),.c0(ca[13]),.s(s[14]),.Ca(ca[14]));
        FA f15(.a(A[15]),.b(B[15]^Ctrl),.c0(ca[14]),.s(s[15]),.Ca(ca[15]));
        FA f16(.a(A[16]),.b(B[16]^Ctrl),.c0(ca[15]),.s(s[16]),.Ca(ca[16]));
        FA f17(.a(A[17]),.b(B[17]^Ctrl),.c0(ca[16]),.s(s[17]),.Ca(ca[17]));
        FA f18(.a(A[18]),.b(B[18]^Ctrl),.c0(ca[17]),.s(s[18]),.Ca(ca[18]));
        FA f19(.a(A[19]),.b(B[19]^Ctrl),.c0(ca[18]),.s(s[19]),.Ca(ca[19]));
        FA f20(.a(A[20]),.b(B[20]^Ctrl),.c0(ca[19]),.s(s[20]),.Ca(ca[20]));
        FA f21(.a(A[21]),.b(B[21]^Ctrl),.c0(ca[20]),.s(s[21]),.Ca(ca[21]));
        FA f22(.a(A[22]),.b(B[22]^Ctrl),.c0(ca[21]),.s(s[22]),.Ca(ca[22]));
        FA f23(.a(A[23]),.b(B[23]^Ctrl),.c0(ca[22]),.s(s[23]),.Ca(ca[23]));
        FA f24(.a(A[24]),.b(B[24]^Ctrl),.c0(ca[23]),.s(s[24]),.Ca(ca[24]));
        FA f25(.a(A[25]),.b(B[25]^Ctrl),.c0(ca[24]),.s(s[25]),.Ca(ca[25]));
        FA f26(.a(A[26]),.b(B[26]^Ctrl),.c0(ca[25]),.s(s[26]),.Ca(ca[26]));
        FA f27(.a(A[27]),.b(B[27]^Ctrl),.c0(ca[26]),.s(s[27]),.Ca(ca[27]));
        FA f28(.a(A[28]),.b(B[28]^Ctrl),.c0(ca[27]),.s(s[28]),.Ca(ca[28]));
        FA f29(.a(A[29]),.b(B[29]^Ctrl),.c0(ca[28]),.s(s[29]),.Ca(ca[29]));
        FA f30(.a(A[30]),.b(B[30]^Ctrl),.c0(ca[29]),.s(s[30]),.Ca(ca[30]));
        FA f31(.a(A[31]),.b(B[31]^Ctrl),.c0(ca[30]),.s(s[31]),.Ca(ca[31]));

     assign CF=ca[31]^Ctrl;
     assign OF=ca[31]^ca[30];
     assign SF=s[31];
     assign ZF=~(s[0]|s[1]|s[2]|s[3]|s[4]|s[5]|s[6]|s[7]|s[8]|s[9]|s[10]
|s[11]|s[12]|s[13]|s[14]|s[15]|s[16]|s[17]|s[18]|s[19]|s[20]
|s[21]|s[22]|s[23]|s[24]|s[25]|s[26]|s[27]|s[28]|s[29]|s[30]|s[31]);
     assign PF=s[0]^s[1]^s[2]^s[3]^s[4]^s[5]^s[6]^s[7]^s[8]^s[9]^s[10]
^s[11]^s[12]^s[13]^s[14]^s[15]^s[16]^s[17]^s[18]^s[19]^s[20]
^s[21]^s[22]^s[23]^s[24]^s[25]^s[26]^s[27]^s[28]^s[29]^s[30]^s[31];

endmodule

```

整体结构简单干净，易理解。时间关系所以没有上先行进位加法器。

乘法模块

乘法需要考虑是有符号或者是无符号的乘法

无符号乘法直接完全并行，全部计算就可以了，具体代码如下：

```
module Mul32(
    input clk,
    input [31:0]A,B,
    input Ctrl,//控制有符号或者无符号乘除

```

```
    output reg [63:0]Mul_Out
    );
     reg [61:0]mul_out;

     reg [30:0]C,D;
     always@(*)begin
     if(A[31]==1'b1)
        C<=(~A[30:0])+1;
     else
        C<=A[30:0];
     if(B[31]==1'b1)
        D<=(~B[30:0])+1;
     else
        D<=B[30:0];
     case(Ctrl)
     1'b0:begin
    Mul_Out<={32'b00000000000000000000000000000000, {32{B[0]}}&A    }
         +{31'b0000000000000000000000000000000, {32{B[1]}}&A, 1'b0  }
         +{30'b000000000000000000000000000000, {32{B[2]}}&A, 2'b00  }
         +{29'b00000000000000000000000000000, {32{B[3]}}&A, 3'b000  }
         +{28'b0000000000000000000000000000, {32{B[4]}}&A, 4'b0000  }
         +{27'b000000000000000000000000000, {32{B[5]}}&A, 5'b00000  }
         +{26'b00000000000000000000000000, {32{B[6]}}&A, 6'b000000  }
         +{25'b0000000000000000000000000, {32{B[7]}}&A, 7'b0000000  }
         +{24'b000000000000000000000000, {32{B[8]}}&A, 8'b00000000  }
         +{23'b00000000000000000000000, {32{B[9]}}&A, 9'b000000000  }
         +{22'b0000000000000000000000, {32{B[10]}}&A, 10'b0000000000}
         +{21'b000000000000000000000, {32{B[11]}}&A, 11'b00000000000}
         +{20'b00000000000000000000, {32{B[12]}}&A, 12'b000000000000}
         +{19'b0000000000000000000, {32{B[13]}}&A, 13'b0000000000000}
         +{18'b00000000000000000, {32{B[14]}}&A, 14'b00000000000000}
         +{17'b0000000000000000, {32{B[15]}}&A, 15'b000000000000000}
         +{16'b000000000000000, {32{B[16]}}&A, 16'b0000000000000000}
         +{15'b00000000000000, {32{B[17]}}&A, 17'b00000000000000000}
         +{14'b0000000000000, {32{B[18]}}&A, 18'b000000000000000000}
         +{13'b000000000000, {32{B[19]}}&A, 19'b0000000000000000000}
         +{12'b00000000000, {32{B[20]}}&A, 20'b00000000000000000000}
         +{11'b0000000000, {32{B[21]}}&A, 21'b000000000000000000000}
         +{10'b000000000, {32{B[22]}}&A, 22'b0000000000000000000000}
         +{9'b00000000, {32{B[23]}}&A, 23'b00000000000000000000000 }
         +{ 8'b0000000, {32{B[24]}}&A, 24'b000000000000000000000000}
         +{7'b000000, {32{B[25]}}&A, 25'b0000000000000000000000000 }
         +{6'b00000, {32{B[26]}}&A, 26'b00000000000000000000000000 }
         +{5'b0000, {32{B[27]}}&A, 27'b000000000000000000000000000 }
         +{ 4'b000, {32{B[28]}}&A, 28'b0000000000000000000000000000}
         +{3'b00, {32{B[29]}}&A, 29'b00000000000000000000000000000 }
         +{2'b0, {32{B[30]}}&A, 30'b000000000000000000000000000000 }
         +{1'b0, {32{B[31]}}&A, 31'b0000000000000000000000000000000 };
            end
    1'b1:begin
    mul_out<={31'b0000000000000000000000000000000,{31{D[0]}}&C[30:0]}
        +{30'b000000000000000000000000000000,{31{D[1]}}&C[30:0],1'b0}
        +{29'b00000000000000000000000000000,{31{D[2]}}&C[30:0],2'b00}
        +{28'b0000000000000000000000000000,{31{D[3]}}&C[30:0],3'b000}
        +{27'b000000000000000000000000000,{31{D[4]}}&C[30:0],4'b0000}
        +{26'b00000000000000000000000000,{31{D[5]}}&C[30:0],5'b00000}
        +{25'b0000000000000000000000000,{31{D[6]}}&C[30:0],6'b000000}
        +{24'b000000000000000000000000,{31{D[7]}}&C[30:0],7'b0000000}
```

```
        +{23'b00000000000000000000000,{31{D[8]}}&C[30:0],8'b00000000}
        +{22'b00000000000000000000000,{31{D[9]}}&C[30:0],9'b000000000}
        +{21'b000000000000000000000,{31{D[10]}}&C[30:0],10'b0000000000}
        +{20'b00000000000000000000,{31{D[11]}}&C[30:0],11'b00000000000}
        +{19'b00000000000000000000,{31{D[12]}}&C[30:0],12'b000000000000}
        +{18'b00000000000000000000,{31{D[13]}}&C[30:0],13'b0000000000000}
        +{17'b00000000000000000000,{31{D[14]}}&C[30:0],14'b00000000000000}
        +{16'b0000000000000000,{31{D[15]}}&C[30:0],15'b000000000000000}
        +{15'b000000000000000,{31{D[16]}}&C[30:0],16'b0000000000000000}
        +{14'b00000000000000,{31{D[17]}}&C[30:0],17'b00000000000000000}
        +{13'b0000000000000,{31{D[18]}}&C[30:0],18'b000000000000000000}
        +{12'b000000000000,{31{D[19]}}&C[30:0],19'b0000000000000000000}
        +{11'b00000000000,{31{D[20]}}&C[30:0],20'b00000000000000000000}
        +{10'b0000000000,{31{D[21]}}&C[30:0],21'b000000000000000000000}
        +{9'b000000000,{31{D[22]}}&C[30:0],22'b0000000000000000000000}
        +{8'b00000000,{31{D[23]}}&C[30:0],23'b00000000000000000000000}
        +{7'b0000000,{31{D[24]}}&C[30:0],24'b000000000000000000000000}
        +{6'b000000,{31{D[25]}}&C[30:0],25'b0000000000000000000000000}
        +{5'b00000,{31{D[26]}}&C[30:0],26'b00000000000000000000000000}
        +{4'b0000,{31{D[27]}}&C[30:0],27'b000000000000000000000000000}
        +{3'b000,{31{D[28]}}&C[30:0],28'b0000000000000000000000000000}
        +{2'b00,{31{D[29]}}&C[30:0],29'b00000000000000000000000000000}
        +{1'b0,{31{D[30]}}&C[30:0],30'b000000000000000000000000000000};
    Mul_Out<={A[31]^B[31],1'b0,mul_out};
              end
    endcase
    end
endmodule
```

无符号的时候是完全并行再加起来，32位一起计算，

有符号的时候必须记住，输入的是补码，对补码进行加减乘除是要先将补码转为原码再进行计算。所以当Ctrl信号是1的时候，我先判断A和B是不是负数，是负数的话就把后面31位先按位取反再加1，获得原码，然后31位进行乘法操作，一定要注意乘法的位数是多少，这样写的时候是很容易出错的，所以要注意位数。在获得了后31位的结果之后，还要处理乘法最后的符号位，就是两个数的高位进行异或就是符号位，再添加一个0就获得了有符号数的结果了。

最重要的就是要记得乘法的输入是补码！！！

除法

除法也需要像乘法一样考虑是否是有符号的数，如果是无符号的就直接除。

用的方法的每次都把被除数的当前位数和除数进行比较，大于的话就可以商1，小于就只左移被除数一位。一共移64位，即是把被除数全部移完，这样剩下的高位就是余数，低位就是商，然后分别放出去展示。还需要注意的是我们需要64位的被除数来进行除法运算。

代码如下：

```
module Div_32bits(
        input[63:0] a,
        input[31:0] b,
        output reg [63:0] shang,
        output reg [63:0] yu
        );

        reg[127:0] cuna;//进行移位操作时的操作数以及存储最终结果
        reg[127:0] cunb;
```

```
        reg[63:0] ta;//把ab的值拿出来
        reg[63:0] tb;

always @(a or b)//要有循环才可以进行赋值 否则会语法错误
begin
    ta <= a;
    tb <= {32'b00000000_00000000_00000000_00000000,b};
end

integer j;//循环变量
always @(ta or tb)//模拟除法进行移位 够除就减 不够就不管继续移位
begin
    cuna = {64'b00000000_00000000_00000000_00000000_00000000_00000000_00000000_00000000,ta};//初始操作，先把初始的东西放进去
    cunb = {tb,64'b00000000_00000000_00000000_00000000_00000000_00000000_00000000_00000000};
    for(j=0;j < 64;j = j + 1)//循环进行 移位32次则循环完成
        begin
            cuna = {cuna[126:0],1'b0};//要有移位 把前面的移掉 每次移一位
            if(cuna[127:64] >= tb)//够除
                cuna = cuna - cunb + 1'b1;//直接减 而且商是1
            else
                cuna = cuna;//否则就不动
        end

    shang <= cuna[63:0];//最后前32位是余数 后32位就是商
    yu <= cuna[127:64];
end
endmodule


```

有符号的除法和有符号的乘法一样，需要进行特殊的处理。有符号的除法是用的补码，所以一样，要先把补码转换成为原码，然后再进行计算。正数的原码是本身，负数的补码是其取反再加，这一点要注意。

其他的过程和无符号的除法一样，就是位数到底是多少比较容易弄错，中间到底是怎么样的过程也需要想清楚。

代码如下：

```
module Div_32bits_flag(
        input[63:0] a,
        input[31:0] b,
        output reg [63:0] shang,
        output reg [63:0] yu
        );

        reg[125:0] cuna;//进行移位操作时的操作数以及存储最终结果 少2
        reg[125:0] cunb;
        reg[62:0] ta;//把ab的值拿出来
        reg[62:0] tb;
        reg[1:0] flag;

always @(a or b)//要有循环才可以进行赋值 否则会语法错误
```

```
begin
    if(a[63]==1'b1)
        ta <= (~a[62:0])+1;
    else
        ta <= a[62:0];
    if(b[31]==1'b1)
        tb <= {31'b00000000_00000000_00000000_0000000,(~b[30:0])+1};
    else
        tb <= {31'b00000000_00000000_00000000_0000000,b[30:0]};
     flag={a[63],b[31]};
end

integer j;//循环变量
always @(ta or tb)//模拟除法进行移位 够除就减 不够就不管继续移位
begin
    cuna = {63'b00000000_00000000_00000000_00000000_00000000_00000000_00000000_0000000,ta};//初始操作，先把初始的东西放进去
    cunb = {tb,63'b00000000_00000000_00000000_00000000_00000000_00000000_00000000_0000000};
    for(j=0;j < 63;j = j + 1)//循环进行 移位32次则循环完成
        begin
            cuna = {cuna[124:0],1'b0};//要有移位 把前面的移掉 每次移一位
            if(cuna[125:63] >= tb)//够除
                cuna = cuna - cunb + 1'b1;//直接减 而且商是1
            else
                cuna = cuna;//否则就不动
        end
     case(flag)
        2'b00:begin shang<={1'b0,cuna[62:0]};yu<={1'b0,cuna[125:63]};end
        2'b01:begin shang<={1'b1,cuna[62:0]};yu<={1'b0,cuna[125:63]};end
        2'b10:begin shang<={1'b1,cuna[62:0]};yu<={1'b1,cuna[125:63]};end
        2'b11:begin shang<={1'b0,cuna[62:0]};yu<={1'b1,cuna[125:63]};end
     endcase
end

endmodule
```

而最后商和余数的正负号是需要特殊处理的。如上面看到的，具体的处理如下表

被除数	除数	商	余数
+	+	+	+
+	-	-	+
-	+	-	-
-	-	+	-

处理表格来自网上搜索得出。

仿真结果

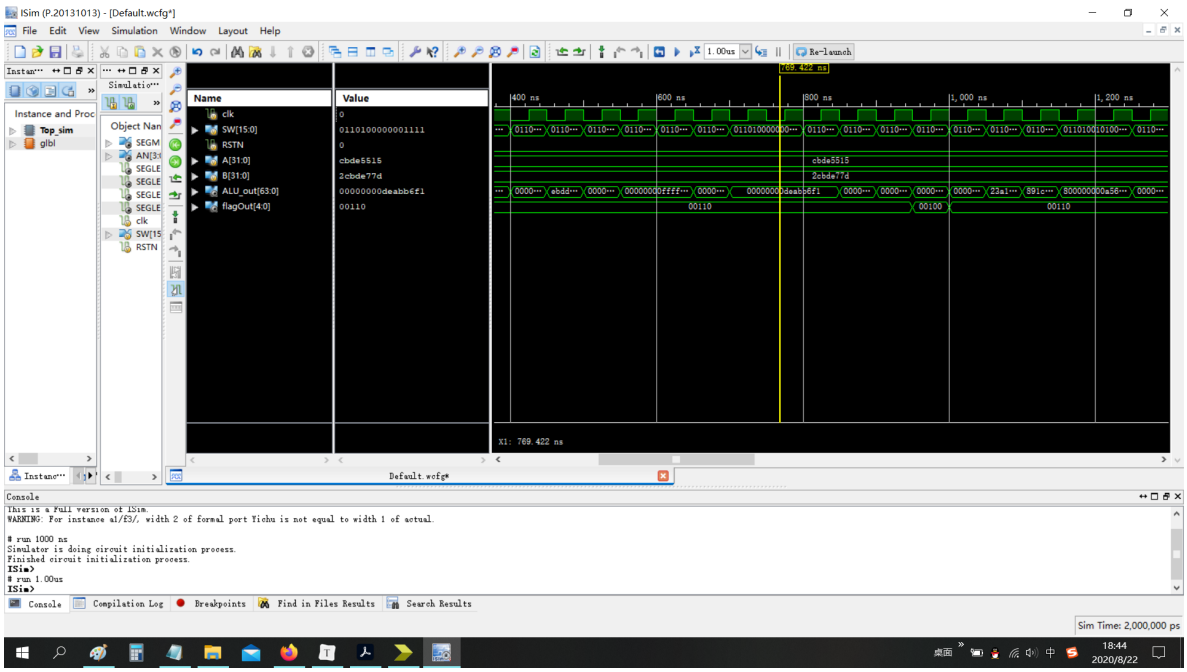
对Top模块仿真，仿真代码如下：

```
module Top_sim;

    // Inputs
    reg clk;
    reg [15:0] SW;
    reg RSTN;

    // Outputs
    wire [7:0] SEGMENT;
    wire [3:0] AN;
    wire SEGLED_CLK;
    wire SEGLED_CLR;
    wire SEGLED_DO;
    wire SEGLED_PEN;

    // Instantiate the Unit Under Test (UUT)
    Top uut (
        .clk(clk),
        .SW(SW),
        .RSTN(RSTN),
        .SEGMENT(SEGMENT),
        .AN(AN),
        .SEGLED_CLK(SEGLED_CLK),
        .SEGLED_CLR(SEGLED_CLR),
        .SEGLED_DO(SEGLED_DO),
        .SEGLED_PEN(SEGLED_PEN)
    );

    initial begin
        // Initialize Inputs
        SW = 0;
        RSTN = 0;
    #100;SW[15:13]=3'b011;
          SW[12:10]=3'b010;

    #50;SW[3:0]=0;
    #50;SW[3:0]=1;
    #50;SW[3:0]=8;
    #50;SW[3:0]=9;
    #50;SW[3:0]=10;
    #50;SW[3:0]=11;
    #50;SW[7]=1;
    #50;SW[7]=0;
    #50;SW[3:0]=12;
    #50;SW[3:0]=13;
    #50;SW[3:0]=14;
    #50;SW[3:0]=15;

    #100;SW[6]=1'b1;
    #50;SW[3:0]=0;
    #50;SW[3:0]=1;
    #50;SW[3:0]=8;
    #50;SW[3:0]=9;
    #50;SW[3:0]=10;#50;SW[8]=1;
    #50;SW[3:0]=11;

    #50;SW[7]=0;
    #50;SW[3:0]=12;
```

```
59      #50;SW[3:0]=13;
60      #50;SW[3:0]=14;
61      #50;SW[3:0]=15;
62
63      end
64      initial forever begin
65      clk=0;#25;
66      clk=1;#25;
67     end
68
69  endmodule
70
71
```

基本上涵盖了所有的算术方法，其中也有有符号无符号的运算。

具体的运算验证较为麻烦，需要结合计算器一个一个地算。所以不进行赘述，验证在上机的时候就验证过了。



仿真结果如图，如有兴趣可以自行下载程序直接进行验证。

遇到的困难

1. 对于运算法则有一些忘记了，较为久远，所以花了较多的时间去找寻正确的方法，最后在老师的第一次验收后有了心得思路，成功做出来了实验。
2. 运算的时候整数的乘除法是补码，是补码，是补码。第一次验收的时候我以为是原码，就拿着原码去验收，结果被打回来重新做了。。。
3. 加法的那个图的c0是Ctrl接入的，并不是自己控制接入的。
4. 时间比较紧迫，我们组又只有两个人，所以做得任务较多完成的也有些仓促。
5. IEEE标准下的浮点运算，无论是加减还是乘除均有多个步骤，（状态机）设计较为复杂。
6. 浮点运算存在着多种特殊情况，比如输入数存在NAN、0、无穷等情况，加减乘除运算会发生指数可能超过可以表示范围。需要较为复杂的判断各种情况。
7. 浮点运算设计验证实验的时候，选择的数字也是需要仔细考虑，比如如何体现浮点数的四舍五入等。
8. 浮点运算乘除法的简便运算需要一定的数学推导。

实验心得

1. 通过学习写这个运算器，让我对整数浮点数都有了较深入的了解，记起来了几个月前学的知识，对这些东西有了一定的理解，会怎么运算了，像之前就一直比较懵。
2. ise仿真里面的relaunch是一个好东西，很方便看到中间变量的值，很容易找到自己的问题所在。
3. 对verilog的语法有了更加深入的了解，对上板子这样的事情也不再有很大的惧怕心理。
4. 看到自己的程序完美运行是很开心的。
5. 本次实验事实上存在一定的难度，浮点乘除法和加减法对于一般人来说本来就较为复杂，而且在处理过程需要考虑一系列的异常状况，舍入条件等。
6. 在实际上板子的时候，也存在较多困难，比如仿真与板子运行不符合等等情况。

致谢老师

最后，谢谢老师一个学期的努力，是您的辛勤付出让我们对计组知识有了深入的了解，对自己的未来的课程学习有很大的帮助，让自己更加理解计算机，乃至于最后做一个cpu出来，虽然是一个很粗糙的产品，但是也是我们付出努力之后认真获得的，这样让我们很有成就感。再次谢谢老师！