实验一：32位ALU

07112005

董若扬

1120202944

1. 实验目的

本实验的目的是设计一个32位ALU，支持至少8种运算，包括加、加无符号、减、减无符号、与、或、异或和取反或，同时输出5个标志符号：SF（符号位）、CF（进位标志）、ZF（零标志）、OF（溢出标志）和PF（奇偶标志）；支持左右移位操作；可支持至少两种舍入操作。

2. 实验环境

Vivado 2019.2

Windows 11 22H2 22621.1972

3. 实验设计

3.1 运算

本实验设计的ALU支持如下8种运算功能：

- 加法（add）
- 加法（addu）
- 减法（sub）
- 减法（subu）
- 与（and）
- 或（or）
- 异或（xor）

- 非（nor）

其中，加法和减法分别有带符号和无符号两种实现方式。addu和subu的CF（进位标志）由运算结果的高位来确定，OF（溢出标志）始终为0。

3.2 标志

输出5个标志符号：SF、CF、ZF、OF和PF。

- SF（符号位）：以运算结果的最高位作为符号位，若结果为正数，则SF=0；若结果为负数，则SF=1。
- CF（进位标志）：对于addu和subu运算，CF由运算结果的高位来确定；对于其他运算，CF始终为0。
- ZF（零标志）：若运算结果为0，则ZF=1；否则ZF=0。
- OF（溢出标志）：对于add和sub运算，当两个操作数均为正数或者均为负数时，若结果的符号位与加法或减法的操作数符号位不同，则OF=1，否则OF=0；对于其他运算，OF始终为0。
- PF（奇偶标志）：以运算结果的最低字节作为参考，若该字节中1的个数为奇数，则PF=0；否则PF=1。

3.3 移位

ALU支持三种移位操作：

- 左移（shl）：将in0向左移in1位，左移时低位补0，高位舍弃。CF记录舍弃的最高位，OF始终为0。
- 右移（shr）：将in0向右移in1位，右移时低位舍弃，高位补0。CF记录被舍弃的最低位，OF始终为0。
- 算术右移（sar）：将in0向右移in1位，右移时低位舍弃，高位补符号位（即最高位）。CF记录被舍弃的最低位，OF始终为0。

4. 实验实现

ALU32模块的输入包括两个32位操作数in0和in1，以及一个11位操作码op。输出包括一个32位运算结果out，以及5个标志符号CF、OF、ZF、PF和SF。

采用always@(*)的组合逻辑实现，根据op的值，进入相应的case分支，对应进行相应的运算。运算结果out根据操作符的不同由不同的模块计算得出，同时计算出CF、OF、ZF、PF和SF。其中，CF和OF需要根据不同的运算进行不同的判断。

- **add**

将输入的两个数 in0 和 in1 相加，将结果存储在 out 变量中。同时，根据加法的进位规则判断是否产生进位（进位标志 CF），根据加法的溢出规则判断是否出现了溢出（溢出标志 OF），并判断结果是否为 0 （零标志 ZF）。

```
        //add
        11'b00000100000:
            begin
            out=in0+in1;
            OF=((in0[31]==in1[31])&&(~out[31]==in0[31]))?1:0;
            ZF=(out==0)?1:0;
            CF=0;
            end
```

- **addu**

与 add 类似，将输入的两个数 in0 和 in1 相加，但不考虑溢出。同时，根据加法的进位规则判断是否产生进位（进位标志 CF），并判断结果是否为 0 （零标志 ZF）。

```
        //addu
        11'b00000100001:
            begin
            {CF,out}=in0+in1;
            ZF=(out==0)?1:0;
            OF=0;
            end
```

- **sub**

将输入的两个数 in0 和 in1 相减，将结果存储在 out 变量中。同时，根据减法的借位规则判断是否产生借位（借位标志 CF），根据减法的溢出规则判断是否出现了溢出（溢出标志 OF），并判断结果是否为 0 （零标志 ZF）。

```
        //sub
        11'b00000100010:
            begin
            out=in0-in1;
            OF=((in0[31]==0&&in1[31]==1&&out[31]==1)||
(in0[31]==1&&in1[31]==0&&out[31]==0))?1:0;
            ZF=(in0==in1)?1:0;
            CF=0;
            end
```

- **subu**

与 sub 类似，将输入的两个数 in0 和 in1 相减，但不考虑溢出。同时，根据减法的借位规则判断是否产生借位（借位标志 CF），并判断结果是否为 0 （零标志 ZF）。

```
        //subu
        11'b00000100011:
            begin
            {CF,out}=in0-in1;
            ZF=(out==0)?1:0;
            OF=0;
            end
```

- **and**

将输入的两个数 in0 和 in1 按位与，将结果存储在 out 变量中。同时，判断结果是否为 0 （零标志 ZF）。

```
        //and
        11'b00000100100:
            begin
            out=in0&in1;
            ZF=(out==0)?1:0;
            CF=0;
            OF=0;
            end
```

- **or**

将输入的两个数 in0 和 in1 按位或，将结果存储在 out 变量中。同时，判断结果是否为 0 （零标志 ZF）。

```
        //or
        11'b00000100101:
            begin
            out=in0|in1;
            ZF=(out==0)?1:0;
            CF=0;
            OF=0;
            end
```

- **xor**

将输入的两个数 in0 和 in1 按位异或，将结果存储在 out 变量中。同时，判断结果是否为 0 （零标志 ZF）。

```
        //xor
        11'b00000100110:
            begin
            out=in0^in1;
            ZF=(out==0)?1:0;
            CF=0;
            OF=0;
            end
```

- **nor**

将输入的两个数 in0 和 in1 按位或取反，将结果存储在 out 变量中。同时，判断结果是否为 0 （零标志 ZF）。

```
        //nor
        11'b00000100111:
            begin
            out=~(in0|in1);
            ZF=(out==0)?1:0;
            CF=0;
            OF=0;
            end
```

- **slt**

将输入的两个数 in0 和 in1 进行有符号比较，将结果存储在 out 变量中。同时，根据比较的结果判断是否出现了溢出（溢出标志 OF），并判断结果是否为 0 （零标志 ZF）。

```
        //slt
        11'b00000101010:
            begin
            if(in0[31]==1&&in1[31]==0)
                out=1;
            else if(in0[31]==0&&in1[31]==1)
                out=0;
            else
                out=(in0<in1)?1:0;
           OF=out;
           ZF=(out==0)?1:0;
           CF=0;
           end
```

- **sltu**

将输入的两个数 in0 和 in1 进行无符号比较，将结果存储在 out 变量中。同时，根据比较的结果判断是否出现了溢出（溢出标志 OF），并判断结果是否为 0（零标志 ZF）。

```
        //sltu
        11'b00000101011:
            begin
                out=(in0<in1)?1:0;
                CF=out;
                ZF=(out==0)?1:0;
                OF=0;
            end
```

- **shl**

将输入的数 in0 向左移动 in1 位，将结果存储在 out 变量中。同时，根据移位的结果判断是否出现了进位（进位标志 CF），并判断结果是否为 0（零标志 ZF）。

```
        //shl
        11'b00000000100:
            begin
            {CF,out}=in0<<in1;
            OF=0;
            ZF=(out==0)?1:0;
            end
```

- **shr**

将输入的数 in0 向右移动 in1 位，将结果存储在 out 变量中。同时，根据移位的结果判断是否出现了进位（进位标志 CF），并判断结果是否为 0（零标志 ZF）。

```
        //shr
        11'b00000000110:
            begin
            out=in0>>in1;
            CF=in0[in1-1];
            OF=0;
            ZF=(out==0)?1:0;
            end
```

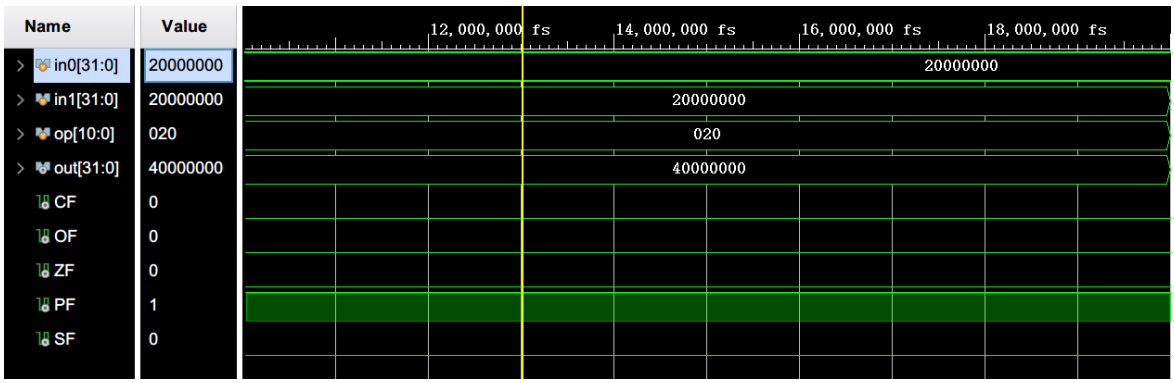
- **sar**

将输入的有符号数 in0 向右移动 in1 位，采用算术右移（符号扩展），将结果存储在 out 变量中。同时，根据移位的结果判断是否出现了进位（进位标志 CF），并判断结果是否为 0 （零标志 ZF）。

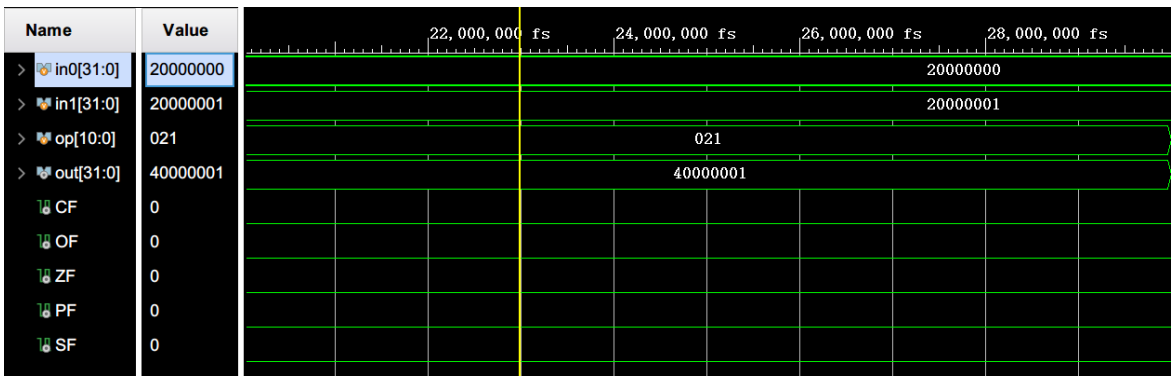
```
        //sar
        11'b00000000111:
            begin
            out=($signed(in0))>>>in1;
            CF=in0[in1-1];
            OF=0;
            ZF=(out==0)?1:0;
            end
```

5. 实验结果

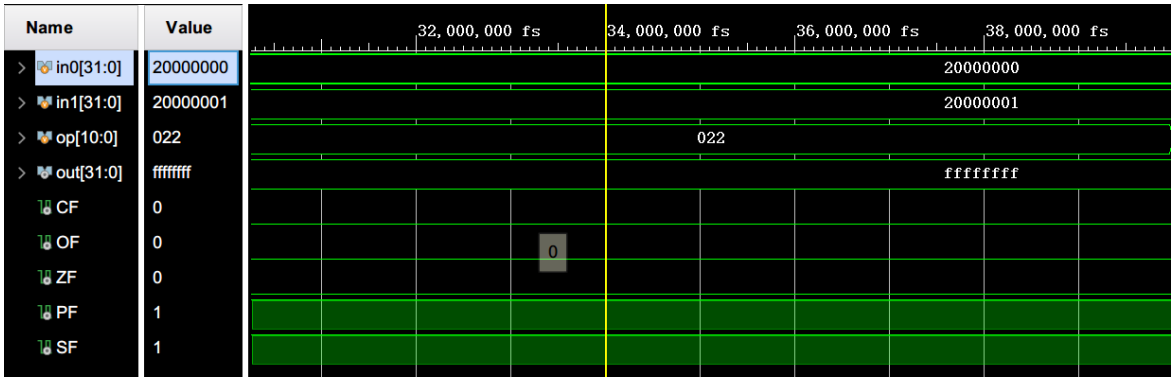
Add: in0=20000000, in1=20000000, out=40000000, CF=0, OF=0, ZF=0, PF=1, SF=0



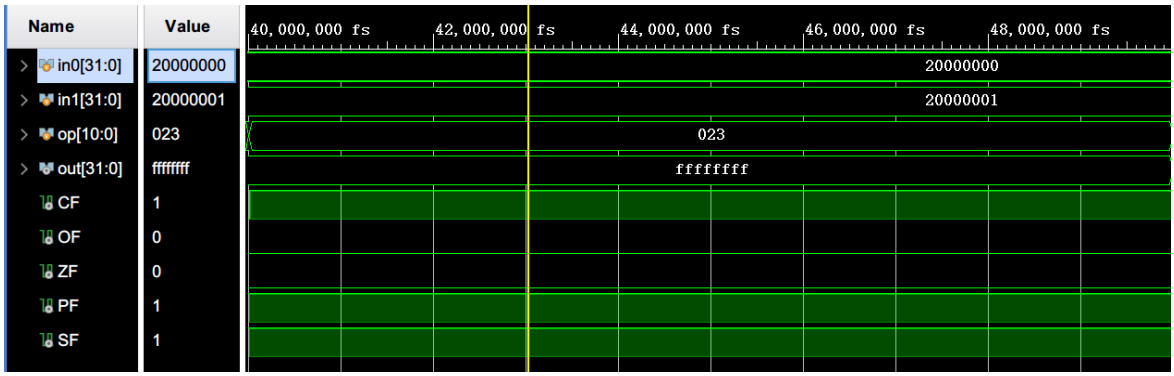
Addu: in0=20000000, in1=20000001, out=40000001, CF=0, OF=0, ZF=0, PF=0, SF=0



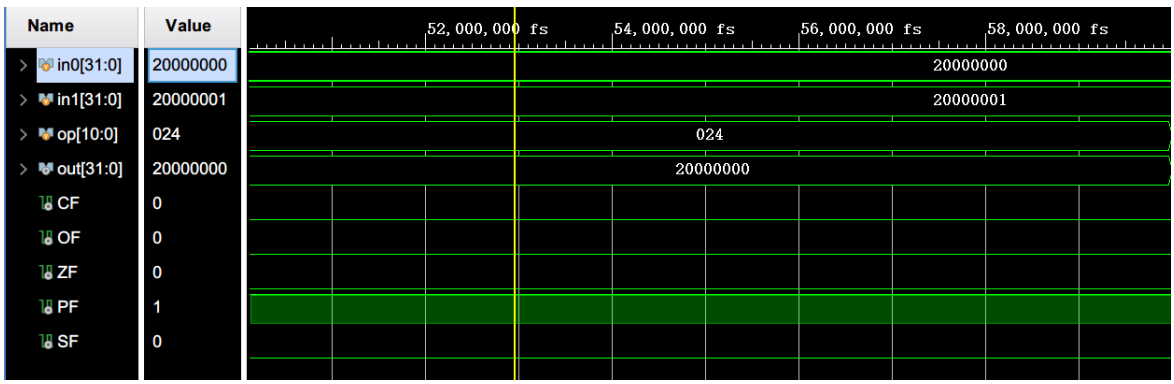
Sub: in0=20000000, in1=20000001, out=FFFFFFFF, CF=0, OF=0, ZF=0, PF=1, SF=1



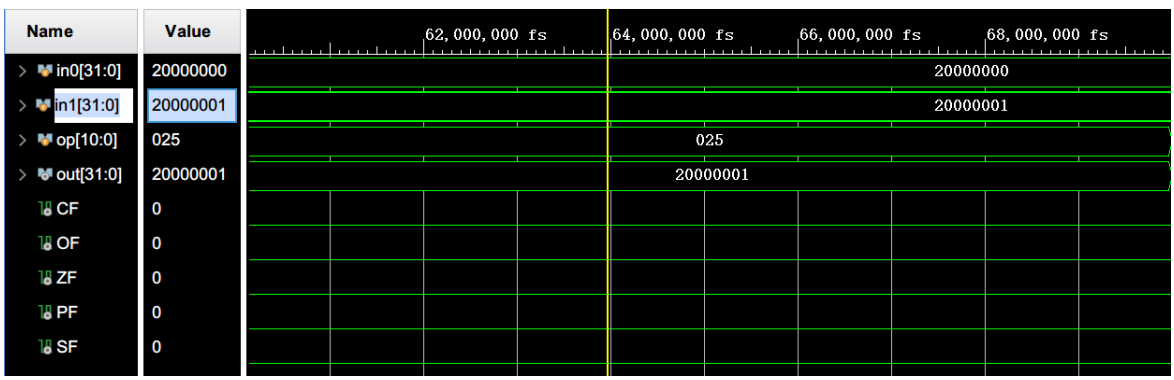
Subu: in0=20000000, in1=20000001, out=FFFFFFFF, CF=1, OF=0, ZF=0, PF=1, SF=1



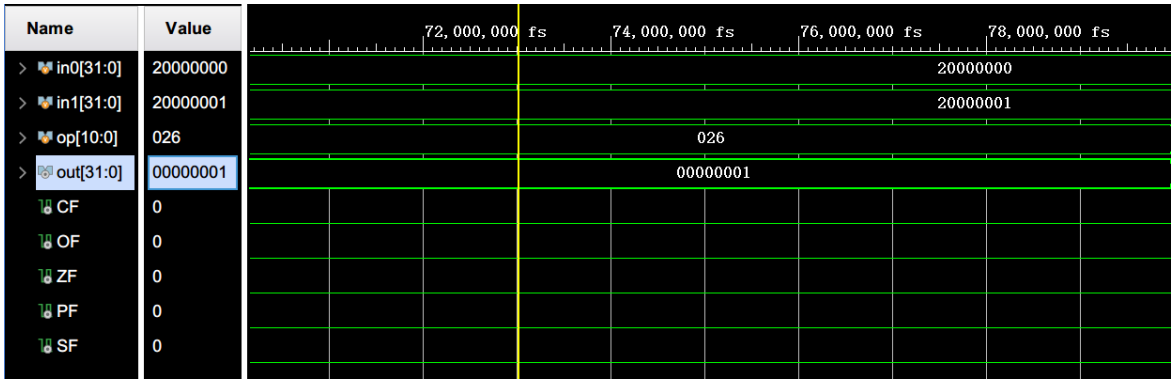
And: in0=20000000, in1=20000001, out=2000000, CF=0, OF=0, ZF=0, PF=1, SF=0



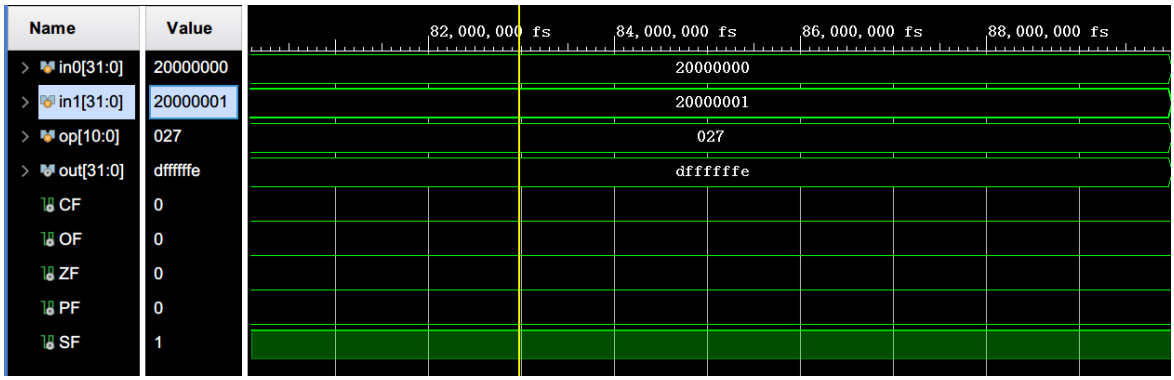
Or: in0=20000000, in1=20000001, out=20000001, CF=0, OF=0, ZF=0, PF=0, SF=0



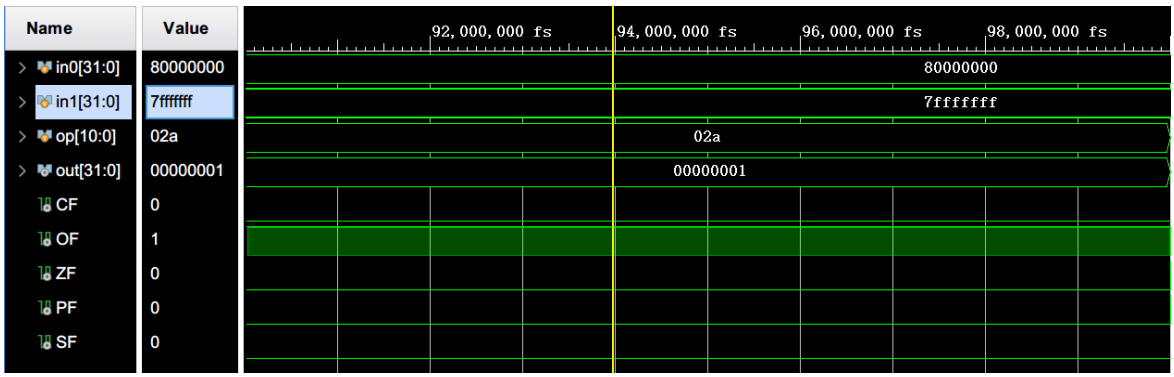
Xor: in0=20000000, in1=20000001, out=00000001, CF=0, OF=0, ZF=0, PF=0, SF=0



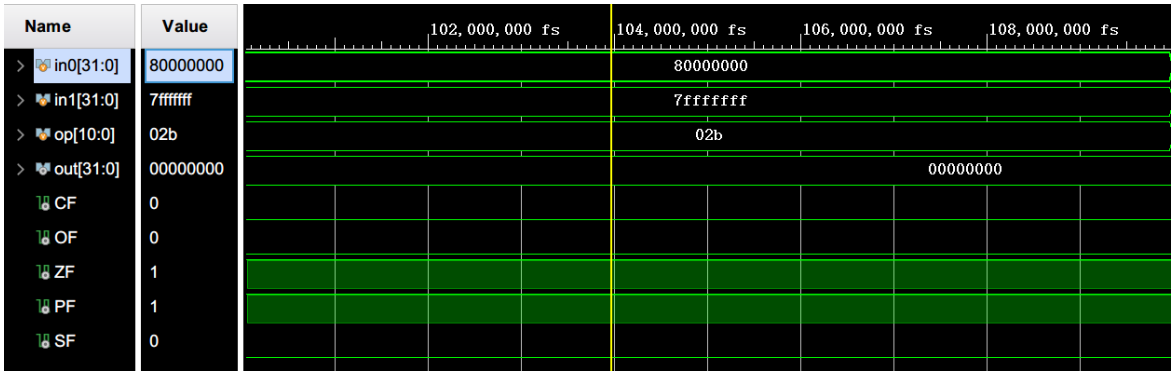
Nor: in0=20000000, in1=20000001, out=DFFFFFFE, CF=0, OF=0, ZF=0, PF=0, SF=1



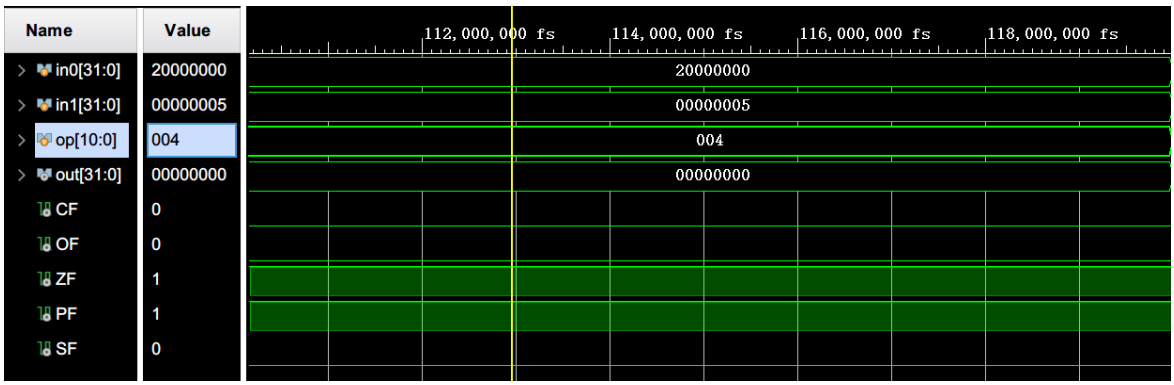
SLT: in0=80000000, in1=7FFFFFFF, out=00000001, CF=0, OF=1, ZF=0, PF=0, SF=0



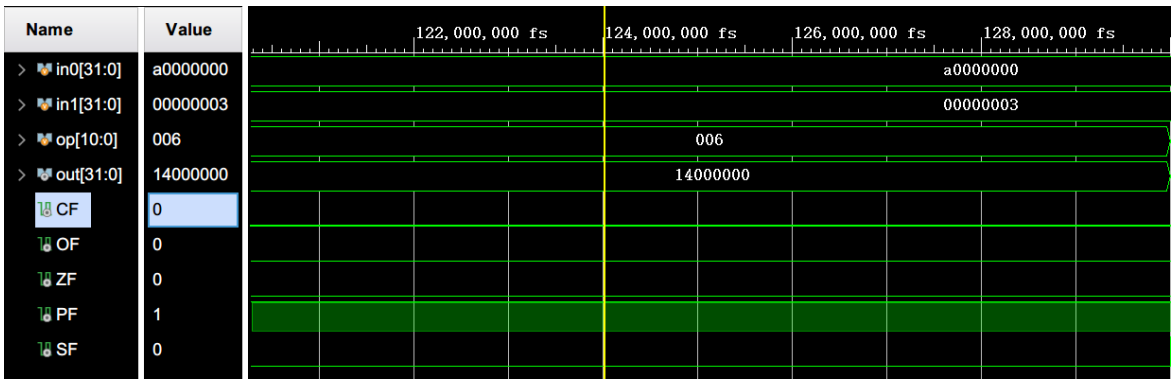
SLTU: in0=80000000, in1=7FFFFFFF, out=00000000, CF=0, OF=0, ZF=1, PF=1, SF=0



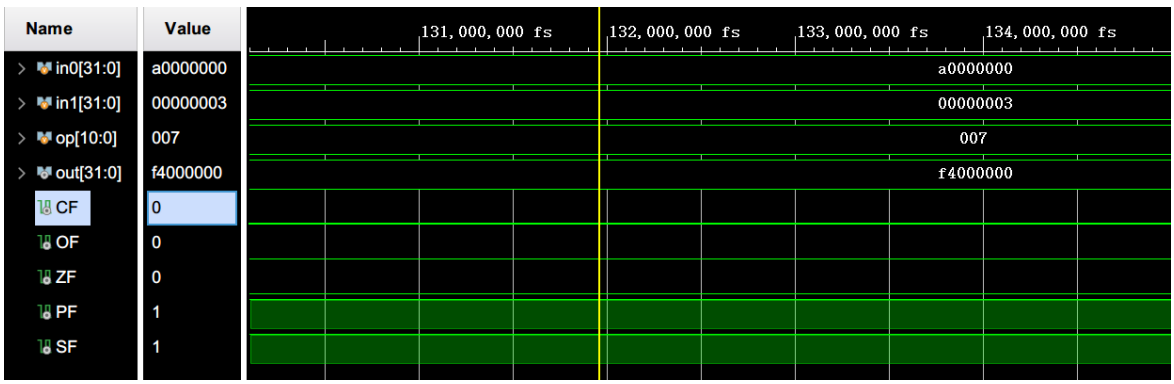
SHL: in0=20000000, in1=00000005, out=00000000, CF=0, OF=0, ZF=1, PF=1, SF=0

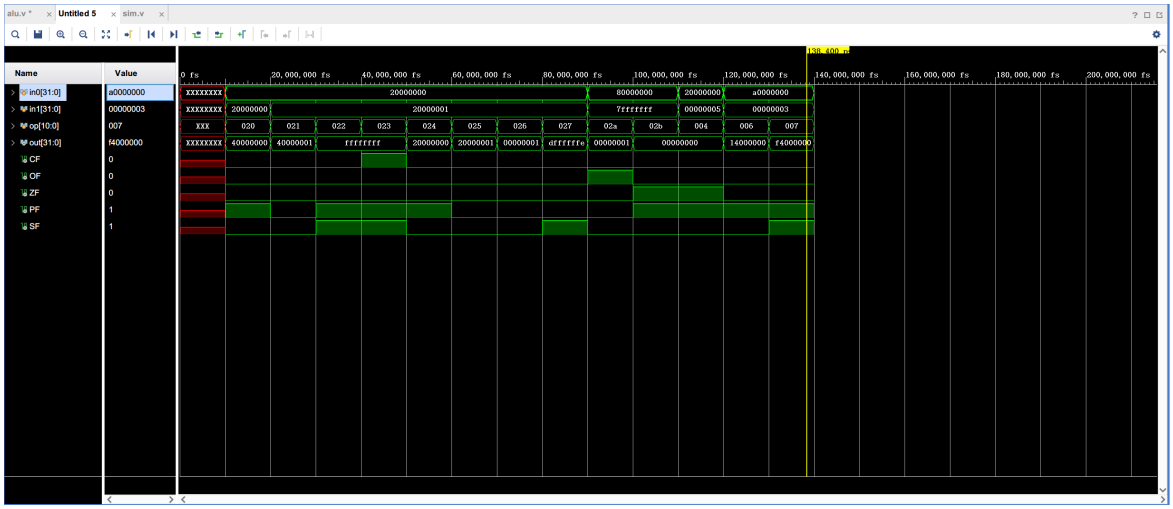


SHR: in0=A0000000, in1=00000003, out=14000000, CF=0, OF=0, ZF=0, PF=1, SF=0



SAR: in0=A0000000, in1=00000003, out=F4000000, CF=0, OF=0, ZF=0, PF=1, SF=1





6. 实验总结

本实验设计了一个支持至少8种运算操作的32位ALU，并输出5个标志符号，通过本实验熟悉硬件描述语言和EDA工具，为接下来设计CPU打好基础。