实验：datalab

1.实验目的

利用位运算和部分逻辑运算实现二进制数层级的操作

2.实验思路及相应代码

1.bitxor - x^y using only ~ and &

Legal ops: ~ &

因为亦或的逻辑表达式为(~x&y)|(x&~y)，其中有一个本题中非法字符|，因此首先想到取反再取反：~(~(~x&y)&~(x&~y))，但这超过了instructor的字符数。又由亦或的定义得当x与y不等时真值为1，相等时为0，由此推出(x|y)&(~x|~y)。

因此最终代码为

```
int a=~x&~y;
int b=x&y;
return ~a&~b;
```

2.evenBits - return word with all even-numbered bits set to 1

Legal ops: ! ~ & ^ | + << >>

由题意得，最终返回结果为0x55555555。但题目限制能出现的十六进制数最大为0xff，因此可先定义一个变量a=0x55，再由移位操作得到。

```
int a=0x55;
a=(a<<8)+a;
a=(a<<16)+a;
return a;
```

3.fitsShort - return 1 if x can be represented as a 16-bit, two's complement integer.

Legal ops: ! ~ & ^ | + << >>

如果x能用16位来表示，那么用int表示时，前16位的值应该无意义，即全为0或1，因此只需判断前16位是否全为1或0。故首先可将x右移16位，所得数若所有位位0或1即返回1，但此操作所用操作符个数会多于instructor。注意到第17位也相当于是符号位，故必定和前16位的值相等。因此若x符合条件，那么将x右移15位所得值必定与x右移16位相等。

```
int a=x>>15;
return !(a^(a>>1));
```

4.isTmax - returns 1 if x is the maximum, two's complement number,and 0 otherwise

Legal ops: ! ~ & ^ | +

若x的值为0x7fffffff，则返回1。注意到0x7fffffff+1=0x10000000，两者相加后可得0xffffffff，取反后得0，且仅有0x11111111和0x7fffffff有这样的性质，又0x11111111+1=0，取非后得1，因此可将两者相并，即可得最终结果：

```
int a=x+1;
return !(~(x+a)|!a);
```

5.fitsBits - return 1 if x can be represented as an n-bit, two's complement integer.

Legal ops: ! ~ & ^ | + << >>

若x能被表示成n位的二进制数，则倒数n-1位为有效值，倒数第n位为符号位，那么剩余位的值必和倒数第n位相等。由第三题可类比使用将x分别右移相应位数再两者亦或，此题需一开始现将x右移n-1位，再与x右移n位亦或。而-1可由~0得到，故最后代码为：

```
int a=x>>(n+~0);
return !(a^(a>>1));
```

6.upperBits - pads n upper bits with 1's

Legal ops: ! ~ & ^ | + << >>

需要将int类型的前n位置为1，只需令首位为1，然后右移相应位数（n-1位）即可，因此可以

```
m=n+~0;
x=1<<31>>m;
```

但此方法没有考虑n=0的情况，因此需做出改进，注意到若n不为0，则首位为1；反之为0，因此可以将1换成!(!n)。故最终代码为

```
int m=n+~0;
int x=!(!n)<<31;
return x>>m;
```

7.allOddBits - return 1 if all odd-numbered bits in word set to 1

Legal ops: ! ~ & ^ | + << >>

要检验是否每个奇数位都为1，可以先定义一个奇数位都为1的变量。由第2题所用方法可类比得

```
int a=0xAA;
a=(a<<8)+a;
a=(a<<16)+a;
```

若x奇数位都为1，那么x&a的奇数位必全为1，偶数位全为0，即x&a=a，再用亦或取非的方法可得：`return !((a&x)^a);`

8.byteSwap - swaps the nth byte and the mth byte

Legal ops: ! ~ & ^ | + << >>

若将n与m分别乘上8(1 byte= 8 bits)，可得相应字节所在位置到最后八位的距离。又x^(x^y)=y，因此可先分别将x右移n位与m位亦或并只保留最后8位得到需交换的字节的“混合体”y，再将y分别左移n位与m位与x亦或可得最终结果。

```
int y;
n=n<<3;
m=m<<3;
y=((x>>n)^(x>>m))&0xff;
return x^(y<<n)^(y<<m);
```

9.absVal - absolute value of x

Legal ops: ! ~ & ^ | + << >>

若x位负数，则需取反加一；若x位整数，则不需要对x的值进行改变。而int类型的首位含有符号信息，故可先`int a=x>>31;`得到符号位，又x^0xffffffff=~x,x^0=x，恰好接近要求，故`int b=a^x;`最后+1的操作又可由~a+1实现。

```
int a=x>>31;
int b=a^x;
return b+~a+1;
```

10.divpwr2 - Compute x/(2^n), for 0 <= n <= 30 Round toward zero

Legal ops: ! ~ & ^ | + << >>

若x为整数，则直接将x左移n位即可得到结果；若x为负数，则由于舍入即补码的性质可得，若后n-1位含有1，则x右移n位后还需加1。故首先用`int a=x>>31;`判断x的符号。又想到若x的后n-1位中含有1，则加上一个n位的全为1的二进制数，倒数第n位必为1，右移n位后可得正确结果。因此可利用a得到一个后n-1位都为1的数，即`~(a<<n)&a`，恰好当x为正数时，此数值为0，使x加上该数后右移n位可得最终结果。

```
int a=x>>31;
int b=~(a<<n);
int c=a&b;
return (x+c)>>n;
```

11.leastBitPos - return a mask that marks the position of the least significant 1 bit. If x == 0, return 0

Legal ops: ! ~ & ^ | + << >>

若x倒数第n位为1，即x最后n-1位全为0，则当x取反后，最后n-1位全为1，那么将~x加一后，只有最后n位的值会发生变化。而x&~x=0，因此可得：

```
return (~x+1)&x;
```

12.logicalNeg - implement the ! operator, using all of the legal operators except !

Legal ops: ~ & ^ | + << >>

首先考虑，对任意x，~x|x=0xffffffff。而对0，0|(~0+1)=0，且仅有0如此，因此可以令x|(~x+1)，又当x非0时，此值看似没有共同点，但首位必为1，因此可将此值右移31位，得最终结果：

```
return ((x|(~x+1))>>31)+1;
```

13.bitMask - Generate a mask consisting of all 1's lowbit and highbit

Legal ops: ! ~ & ^ | + << >>

首先想到~0可得所有位都是1的数，又发现若在某位上加一，则该位及前面所有位都会被置为0，因此可借此方法，先分别将1左移highbit位和lowbit位，然后与~0相加，可分别得到最后highbit+1（需并上1<<highbit）位和lowbit位为1的数，然后再对后者取反并交前者，可得最终结果。

```
int a,b,n;
n=~0;
a=1<<highbit;
a=(a+n)|a;
b=(1<<lowbit)+n;
return a&~b;
```

14.isLess - if x < y then return 1, else return 0

Legal ops: ! ~ & ^ | + << >>

首先考虑y-x(y+~x)，当其符号为0时则返回1，但会卡在特殊情况（最大值和最小值的overload情况），又想到判断两者符号，若像x<0,y>0的情况可直接返回一。但总是会出错或者运算符数量超过instructor。所以干脆先根据如下代码列了个真值表（已舍去一些感觉没用的）：

```
int a,c,d,e;
a=y+~x;
a=(a>>31);
c=x>>31;
d=y>>31;
e=c+~d;
```

	a	e	!a	!e	~e	a^e	返回值
0<x<y	0	11...11	1	0	0	11....11	1
x<0<y	0	11...10	1	0	1	11...10	1
x<y<0	0	11...11	1	0	0	11...11	1
0<y<x	11...11	11...11	0	0	0	0	0
y<0<x	11...11	0	0	1	11...11	11...11	0
y<x<0	11...11	11...11	0	0	0	0	0
x=-max,y=max	11...11	11...10	0	0	1	1	1
x=max,y=-max	0	0	1	1	11...11	0	0

注意到a^e与理论返回值十分相近，可两次取非得到，仅有y<0<x时存在误差，恰好!e在此情况下与其他赋值为1的值相反，因此最终代码为 `return !(!(a^e)|!e);`

15.logicalShift - shift x to the right by n, using a logical shift

Legal ops: ! ~ & ^ | + << >>

首先考虑直接将x右移n位，但若x为负数时，前面会自动补为1，因此只需将x右移后的值交上一个前面n位为0，其余位为1的数即可。而需要的补码可类比第13题，由以下代码得到：

```
a=1<<(32+~n);
a=a|(a+~0);
```

最终代码：

```
int a,b;
a=1<<(32+~n);
a=a|(a+~0);
b=x>>n;
return a&b;
```

16.satMul2 - multiplies by 2, saturating to Tmin or Tmax if overflow

Legal ops: ! ~ & ^ | + << >>

要令x变成原来的两倍，在二进制层面只需将x左移一位，因此首先`int a=x<<1`。接着判断x左移之后是否溢出，注意到若x乘2之后会溢出，则x的首位必与a的首位互异（否则只是单纯的扩大一倍）。因此可以设置一个标记来判断x*2是否溢出：

```
int b=x>>31;
int c=a>>31;
int d=b^c;
```

若a不溢出，则不改变a的值；若a溢出，则对a重新赋值。当a溢出时，d=0xffffffff；当a不溢出时，d=0。因此可以令a&~d。又d左移31位后得到0x80000000，是溢出情况的其中一种结果，而另一种结果0x7fffffff=0x80000000-1。-1可通过d&c得到。因此最终代码为

```
int a=x<<1;
int b=x>>31;
int c=a>>31;
int d=b^c;
int g=d&c;
int f=a&~d|(d<<31)+g;
return f;
```

17.subOK - Determine if can compute x-y without overflow

Legal ops: ! ~ & ^ | + << >>

首先计算x-y。类似第14题比较x y的大小，可以通过判断x,y,x-y的符号来判断是否溢出。因此可列真值表：

x（的首位，下同）	y	x-y	真值
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

因此只有两种情况时真值为0，寻找其中的特点发现。真值为0时y与x-y同号，但第一行和最后一行的y也和x-y同号。有观察得这两号与中间两行相比，x与y同号。故最终代码为

```
  int a=~y+1;
  a=(x+a);
  //return (~(~x&y&a)&(~x|y|a))>>31&1;
  return (y^a|~(x^y))>>31&1;
```

18.bang - Compute !x without using !

Legal ops: ~ & ^ | + << >>

首先考虑，对任意x，~x|x=0xffffffff。而对0，0|(~0+1)=0，且仅有0如此，因此可以令x|(~x+1)，又当x非0时，此值看似没有共同点，但首位必为1，因此可将此值右移31位，得最终结果：

```
return ((x|(~x+1))>>31)+1;
```

19.bitParity - returns 1 if x contains an odd number of 0's

Legal ops: ! ~ & ^ | + << >>

当x中有奇数个0时，1的个数也是奇数；当0为偶数个时，1的个数也是偶数。在位运算中，^可将这两种情况区分开来，但该方法要对所有位进行^操作。要对所有位进行操作，可不断“对折”x，将“对折”后的两段x进行亦或操作。因此得

```
  x=x^(x>>16);
  x=x^(x>>8);
  x=x^(x>>4);
  x=x^(x>>2);
  x=x^(x>>1);
  return x&1;
```

20.isPower2 - returns 1 if x is a power of 2, and 0 otherwise

Legal ops: ! ~ & ^ | + << >>

首先排除以下情况：1.x<0，2.x=0。若x为2的幂数，则x的二进制为00…010..0，x-1的二进制为00…001…1，将两者|在加1之后所得结果为x的两倍，可由此判断x是否符合要求。因此得最终代码

```
int a=((x+~0)|x)+1;
return !((x<<1)^a|(x>>31)|!x);
```

21.float_i2f - Return bit-level equivalent of expression (float) x

Result is returned as unsigned int, but it is to be interpreted as the bit-level representation of a single-precision floating point values.

Legal ops: Any integer/unsigned operations incl. ||, &&. also if, while

首先判断x是否等于0，如是，则直接返回。

统一x：定义`sign=x>>31`判断x正负性，若x为负数，则对x取反加一。

判断x第一个1出现的位置：从x首位开始查找，若碰到1或x中各位被遍历，则跳出循环。

```
n=32;
while(!(x_&a)&&--n)
{
  x_=x_<<1;
  --m;
}
```

对如0x7fffffff的进位处理：因为如果有进位情况，则进位之后首位必为零（首位1+进位1）。由此可判断是否要给指数加一。

```
x__=x_;
x__=(x__+0x80)&~0xff;
if(!(x__&a))
  m=m+1;
```

最后定义`e=126+m`得出float内的指数。

最后判断是否符合进位条件：因为x会从32位的表示被舍到24位。又因为首位的1仍未被舍掉，因此只需判断最后8位是否大于0xff或第8,9位都为1即可。

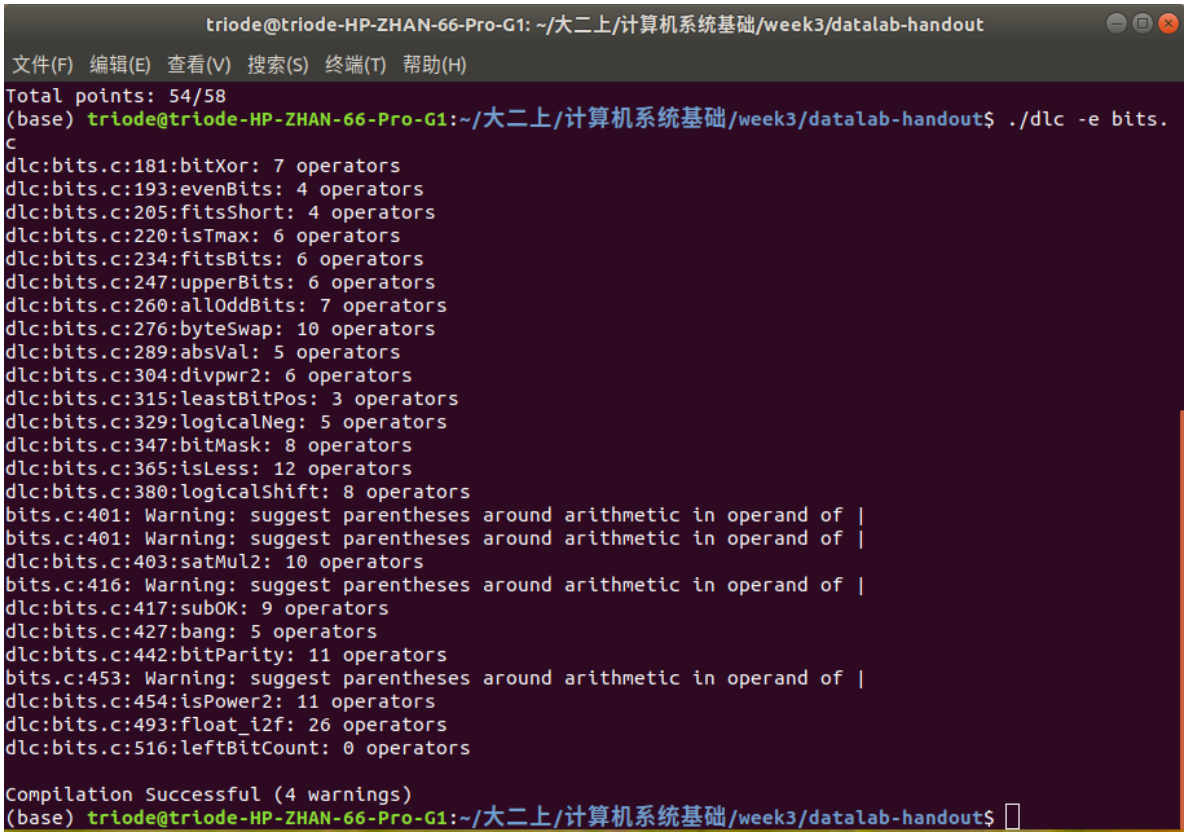
```
if((x_&0xff)>0x80||(x_&c)==c)
  x_=x_+0x80;
```

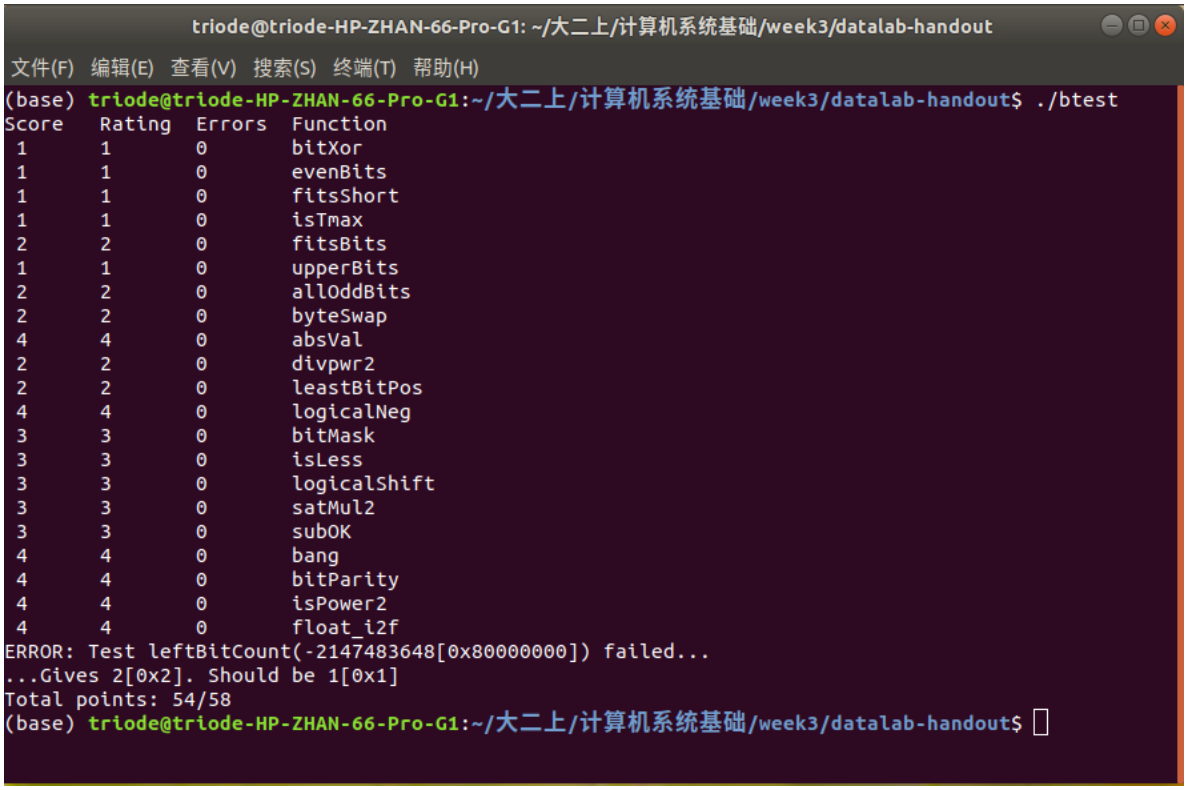
最终代码：

```
int sign,e=127,m=31,n;
int a=1<<31,c=3<<7;
unsigned final,x_=x,x__;
if(!x)  return x;
sign=x>>31;
if(sign)
{
  x_=~x;
  ++x_;
}
n=32;
while(!(x_&a)&&--n)
{
  x_=x_<<1;
  --m;
```

```
    }
    x__=x_;
    x__=(x__+0x80)&~0xff;
    if(!(x__&a))
      ++m;
    ++m;
    while(--m)
      ++e;
    if((x_&0xff)>0x80||(x_&c)==c)
      x_=x_+0x80;
    x_=x_<<1;
    final=(sign<<31)|(e<<23)|(x_>>9);
    return final;
```

3.实验结果截图





7 4 4 6 6 6 7 10 5 6 3 5 8 12 8 10 9 5 11 11 26 — — — 2018202196

4.总结

通过这次datalab的作业，让我对位运算有了一些新的认识，比如~0=-0xffffffff。而且因为每题都尽量做到比instructor好，形成了一定的思维惯性，在最近碰到的一些和位操作有关的作业中，也会考虑"这样的方法是不是运算符最少的"这样的问题。以后的代码之中会更加关注代码效率。

不过有点遗憾的是前面的题目为了凑instructor的操作符导致最后一题没有写出来，之后尽量把lab做完。