异或操纵：
共有三种异或操纵

```
a^b=
(a|b)&(~a|~b)
~(~a&~b)&~(a&b)
(a&~b)|(~a&b)
/*
 * bitXor - x^y using only ~ and &
 *   Example: bitXor(4, 5) = 1
 *   Legal ops: ~ &
 *   Max ops: 14
 *   Rating: 1
 */
int bitXor(int x, int y) {
    //即满足既不是1，1也不是0，0
  return ~(~x&~y)&~(x&y);
}
```

返回最小的补码：
最小补码仅符号位为1，其余全为0

```
/*
 * tmin - return minimum two's complement integer
 *   Legal ops: ! ~ & ^ | + << >>
 *   Max ops: 4
 *   Rating: 1
 */
int tmin(void) {
    //符号位为1，其他为全0
  return 0x1<<31;
}
```

判断是否为补码最大值：
补码最大值即符号位为0，其余都为1，我们可以将其进行一系列变换称为0x0000，那么返回值就可以为~0xFFFF

```
/*
 * isTmax - returns 1 if x is the maximum, two's complement number,
 *     and 0 otherwise
```

```
 *   Legal ops: ! ~ & ^ | +
 *   Max ops: 10
 *   Rating: 2
 */
int isTmax(int x) {
    //将其变为全0，这样取反后恰好成立
    x;//011111
    x+1;//100000
    (x+1)^x;//111111,从这一步开始相同
    ~(x+1)^x;//000000
    !(~(x+1)^x);//1
    //排除-1 111111
    return !(~(x+1)^x)&!!(x+1);
}
```

判断是否所有的奇数位都为1

```
/*
 * allOddBits - return 1 if all odd-numbered bits in word set to 1
 *   Examples allOddBits(0xFFFFFFFD) = 0, allOddBits(0xAAAAAAAA) = 1
 *   Legal ops: ! ~ & ^ | + << >>
 *   Max ops: 12
 *   Rating: 2
 */
int allOddBits(int x) {
    //用一个偶数位0xAAAAAAAA的掩码与运算x，取出奇数位1010
    //再与0xAAAAAAAA异或，若结果全0，则证明相等
    //但是只能定义8位，所以0xAAAAAAAA要特殊定义
    int a=0xAA<<8;
    int b=a|0xAA;
    int c=b<<16|b;
    return !((x&c)^(c));
}
```

不使用-取反

```
/*
 * negate - return -x
 *   Example: negate(1) = -1.
 *   Legal ops: ! ~ & ^ | + << >>
```

```
 *   Max ops: 5
 *   Rating: 2
 */
 //直接利用补码取反法则，每位取反，末位加1
int negate(int x) {
  return ~x+1;
}
```

判断属于数字的Ascii码

```
/*
 * isAsciiDigit - return 1 if 0x30 <= x <= 0x39 (ASCII codes for characters '0' to '9')
 *   Example: isAsciiDigit(0x35) = 1.
 *            isAsciiDigit(0x3a) = 0.
 *            isAsciiDigit(0x05) = 0.
 *   Legal ops: ! ~ & ^ | + << >>
 *   Max ops: 15
 *   Rating: 3
 */
 //属于数字的Ascii码为30-39：00110000->00111001，所以除后6位必须都为0
 //一个数是加上比0x39大的数后符号由正变负，另一个数是加上比0x30小的值时是负数。
int isAsciiDigit(int x) {
    //1 保证除后6位必须都为0
    //x>>=6;
    //!!x;
    //若x为0，则值为0，否则为非0，所以第一个表达式为!(!!(x>>6))

    //2 保证后6位与后5位为11
    //x>>=4;
    //x^0b11==0 判断第六第五位是否为11
    //所以第二个表达式为!((x>>4)^0b11)

    //3
    //int c=x&0xf;取后四位
    //d=c-A<0;若此式小于0，表示后四位在范围内
    //!!(d>>31)
    return !(!!(x>>6))&!(x^0b11)&!!(((x&0xf)+(~0xA+1))>>31);
}
```

实现三目运算符

```
/*
 * conditional - same as x ? y : z
 *   Example: conditional(2,4,5) = 4
 *   Legal ops: ! ~ & ^ | + << >>
 *   Max ops: 16
 *   Rating: 3
 */
 //将x转换为全0或全1，便于使用
int conditional(int x, int y, int z) {
    //当x为1时转换为全1，不然为全0
    return ((~(!!x)+1)&y)|(~(~(!!x)+1)&z);
}
```

实现小于等于

```
/*
 * isLessOrEqual - if x <= y  then return 1, else return 0
 *   Example: isLessOrEqual(4,5) = 1.
 *   Legal ops: ! ~ & ^ | + << >>
 *   Max ops: 24
 *   Rating: 3
 */
 //若x,y异号，则容易判断,若x,y同号,则做减法，判断结果
int isLessOrEqual(int x, int y) {
    //取符号
    int signX=(x>>31)&0x1;
    int signY=(y>>31)&0x1;
    //(signX&~signY);异号时x为负的情况，直接判断为小于
    //(signX&~signY)|!(~signX&signY);包含同号情况
    int z=y+(~x+1);
    int flag=z>>31;//取减法结果，大于返回0
    return (signX&~signY)|(!(~signX&signY)&!flag);
}
```

求逻辑非

```
/*
 * logicalNeg - implement the ! operator, using all of
 *              the legal operators except !
 *   Examples: logicalNeg(3) = 0, logicalNeg(0) = 1
```

```
 *   Legal ops: ~ & ^ | + << >>
 *   Max ops: 12
 *   Rating: 4
 */
 //0和10000000其相反数的或运算，符号位为0，其他为1，通过补码符号位区别0和10000000
int logicalNeg(int x) {
    //(x|(~x+1))若x>0或x<0,与其相反数符号位必有1个为1
    return ((x|(~x+1))>>31)+1;
}
```

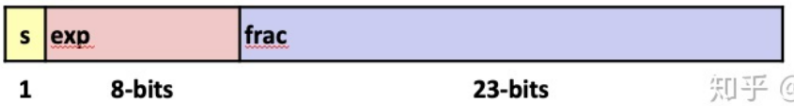
最少多少位补码能表示x

```
/* howManyBits - return the minimum number of bits required to represent x in
 *             two's complement
 *  Examples: howManyBits(12) = 5
 *            howManyBits(298) = 10
 *            howManyBits(-5) = 4
 *            howManyBits(0)  = 1
 *            howManyBits(-1) = 1
 *            howManyBits(0x80000000) = 32
 *  Legal ops: ! ~ & ^ | + << >>
 *  Max ops: 90
 *  Rating: 4
 */
 //如果是一个正数，就找最左边的1，如果是一个负数，就找最左边的0，即取反
 //然后利用二分法原理，查找位置
int howManyBits(int x) {
    int flag=x>>31;
    x=(flag&~x)|(~flag&x);//负数取反处理
    int b16,b8,b4,b2,b1,b0;
    b16=!!(x>>16)<<4;//如果高16为有1，那么低16为必须满足，此时b16为16
    x = x>>b16;
    b8 = !!(x>>8)<<3;//剩余位高8位是否有1
    x = x>>b8;//如果有（至少需要16+8=24位），则右移8位
    b4 = !!(x>>4)<<2;//同理
    x = x>>b4;
    b2 = !!(x>>2)<<1;
    x = x>>b2;
    b1 = !!(x>>1);
    x = x>>b1;
```

```
    b0 = x;
    return b16+b8+b4+b2+b1+b0+1;//加上符号位
}
```

计算2*uf:

- 单精度: 32 bits



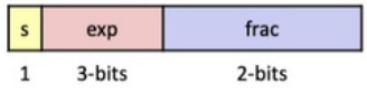

```
/*
 * float_twice - Return bit-level equivalent of expression 2*f for
 *   floating point argument f.
 *   Both the argument and result are passed as unsigned int's, but
 *   they are to be interpreted as the bit-level representation of
 *   single-precision floating point values.
 *   When argument is NaN, return argument
 *   Legal ops: Any integer/unsigned operations incl. ||, &&. also if, while
 *   Max ops: 30
 *   Rating: 4
 */
unsigned float_twice(unsigned uf) {
    //首先考虑特殊情况NaN和无穷
    //将浮点数的三个部分取出来


    unsigned exp = (uf&0x7f800000)>>23;
    unsigned sign=uf>>31&0x1;
    unsigned frac=uf&0x7FFFFF;
    unsigned res;
    if(exp==0xff){//如果exp为255，尾数非0为NaN，全0为无穷大，直接返回
        return uf;
    }else if(exp==0){//如果exp为全0，则为非规格化数，尾数直接返回frac<<2
        frac<<=1;
        res=(sign<<31)|(exp<<23)|frac;
    }else{//指数+1
        ++exp;
        res=(sign<<31)|(exp<<23)|frac;
    }
```

```
    return res;
}
```

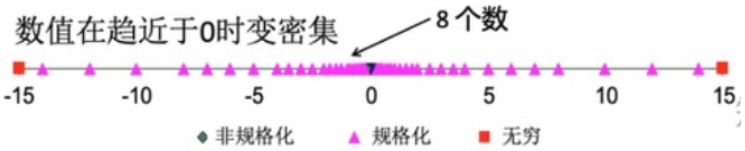
float转换为int

数值分布

- 6-bit类 IEEE格式浮点数
  - e： 阶码(Exponent) 位数3
  - f： 小数位数 2
  - 偏置bias= $2^{3-1}-1 = 3$



- 注意：数值在趋近于0时变密集




```
/*
 * float_f2i - Return bit-level equivalent of expression (int) f
 *   for floating point argument f.
 *   Argument is passed as unsigned int, but
 *   it is to be interpreted as the bit-level representation of a
 *   single-precision floating point value.
 *   Anything out of range (including NaN and infinity) should return
 *   0x80000000u.
 *   Legal ops: Any integer/unsigned operations incl. ||, &&. also if, while
 *   Max ops: 30
 *   Rating: 4
 */
int float_f2i(unsigned uf) {
    //如果是exp=255，情况为NaN和无穷大，返回溢出值0x80000000u
    //首先把小数部分（23位）转化为整数（和23比较），
    //然后判断是否溢出：如果和原符号相同则直接返回，否则如果结果为负（原来为正）
    //则溢出返回越界指定值0x80000000u，否则原来为负，结果为正，则需要返回其补码（相反数）。
    unsigned exp = (uf&0x7f800000)>>23;//取出exp
    int sign=uf>>31&0x1;//取出sign
    unsigned frac=uf&0x7FFFFF;//取出frac
    int E=exp-127;
    if(E<0){//指数部分小于0直接返回0
         return 0;
    }
```

```
    else if(E >= 31){//结果溢出返回越界指定值0x80000000u
        return 0x80000000u;
    }
    else{
        frac=frac|1<<23;//左移23位便于之后处理
        if(E<23) {//左移多了，需要右移
            frac>>=(23-E);
        }else{
            frac<<=(E-23);//左移不够
        }
    }
    if (sign){//根据符号返回正负
        return -frac;
    }else{
        return frac;
    }
}
```

int转换为float

```
/*
 * 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
 *   Max ops: 30
 *   Rating: 4
 */
unsigned float_i2f(int x) {
   if(!x){//如果为0，直接返回
           return x;
   }else if(x == 0x80000000){//溢出值，则返回无穷
           return 0xcf000000;
   }else{
           int s = x>>31&1;
           if(s){//取符号位，转换为正数
                   x = -x;
           }

```

```
21              int i = 30;
22              while(!(x>>i)){ //判断指数字段最大值，找到刚好使尾数小于0的指数
23                      i--;
24              }
25              int exp = i+127;//偏移E=exp-127
26
27              x = x<<(31-i);
28              int frac = (x>>8)&0x7fffff;//取23位
29              x = x&0xff;
30              int dalta = x>128||((x == 128)&&(frac&1));//舍入到偶数
31              frac += dalta;
32              if(frac>>23){//舍入到偶数超过(1<<24)-1,指数加一
33                      frac& = 0x7fffff;
34                      exp++;
35              }

36              return (s<<31)|(exp<<23)|frac;
37      }
38  }
```