```
int bitAnd(int x, int y) {
/*
 *用 ~ 和 | 代替 & 摩尔定律的等价形式
 */
   int result = ~(~x|~y);
   return result;
}
/*
 * getByte - Extract byte n from word x
 *     Bytes numbered from 0 (LSB) to 3 (MSB)
 *     Examples: getByte(0x12345678,1) = 0x56
*/
int getByte(int x, int n) {
/*
 *给定 n ， 求出第 n 个字节
 *一个字节=8bit，一个十六进制位是 4bit，所以移动(n << 3)bit 后，将他与 0xFF 取交集
 */
   int temp = n << 3;
   temp = x >> temp;
   temp = temp & 0xff;
   return temp;
}
/*
 * logicalShift - shift x to the right by n, using a logical shift
 *     Can assume that 0 <= n <= 31
 *     Examples: logicalShift(0x87654321,4) = 0x08765432*/
int logicalShift(int x, int n) {
/*
 *实现逻辑右移。逻辑右移左边补 0，而算数右移左边补符号位。
 *所以构造出全 1 右移 n 位，左 n 位都是 0 的数，将原 x 也右移 n 位，与该数取交
 */
   int temp = ~(1 << 31);
   temp = ((temp >> n ) << 1) + 1;
   temp = (x >> n) & temp;
   return temp;
}
/*
 * bitCount - returns count of number of 1's in word
 *     Examples: bitCount(5) = 2, bitCount(7) = 3
*/
int bitCount(int x) {
/*
 *一位一位移动来测可得，但步骤太多。可以利用分治法。
 *依次统计每 2 位、4 位、8 位、16 位、32 位。
 */
   int count;
   int formask1 = (0x55) | (0x55 << 8);
   int mask1 = (formask1 << 16) | (formask1);
   int formask2 = (0x33) | (0x33 << 8);
   int mask2 = (formask2 << 16) | (formask2);
   int formask3 = (0x0f) | (0x0f << 8);
   int mask3 = (formask3 << 16) | (formask3);
   int mask4 = (0xff << 16) | (0xff);
   int mask5 = (0xff << 8) | (0xff);
   count = (x & mask1) + ((x >> 1) & mask1);
   count = (count & mask2) + ((count >> 2) & mask2);
   count = (count & mask3) + ((count >> 4) & mask3);
   count = (count & mask4) + ((count >> 8) & mask4);
   count = (count & mask5) + ((count >> 16) & mask5);
   return count;
}
/*
 * bang - Compute !x without using !
 *     Examples: bang(3) = 0, bang(0) = 1
*/
int bang(int x) {
/*
 *对于 0x0...0,0x0...0 | ((~0x0...0)+1)=0
 *而对于其它，都不可能，因为符号位上必为 1，依据此性质可得。
 */
   int temp = (~x) + 1;
   int result = x | temp;
   result = result >> 31;
   result = result + 1;
   return result;
}
/*
 * tmin - return minimum two's complement integer
*/
int tmin(void) {
/*
 *0x80000000，转化为 32 位知 1 代表负，而 0000000 代表最小的。
 */
   int result = 1 << 31;
   return    result;
}
/*
 * fitsBits - return 1 if x can be represented as an
 *   n-bit, two's complement integer.
 *     1 <= n <= 32
 *     Examples: fitsBits(5,3) = 0, fitsBits(-4,3) = 1
*/
int fitsBits(int x, int n) {
/*
 *如果可以表示，则前 32-n 位与该值无关，只表示符号
 *那么可以先左移 n 位，再算术右移 n 位，判断是否相等，相等则可表示。
 */
   int temp = (x << (32 + ~n + 1)) >> (32 + ~n +1);
   int result = !(temp ^ x);
   return result;
}
/*
 * divpwr2 - Compute x/(2^n), for 0 <= n <= 30
 *   Round toward zero
 *     Examples: divpwr2(15,1) = 7, divpwr2(-33,4) = -2*/
int divpwr2(int x, int n) {
/*
 *正数可直接右移得，而负数有问题，会取远离 0 的那个整数。除非是 2^n。
 * 构造一个 bias 加在 x 上，当非 2^n 时，进位解决了取法问题。2^n 不影响。
 */
   int sign = x >> 31;
   int mask = (1 << n) + (~0);
   int bias = sign & mask;
   int result = (x+bias) >> n;
   return result;
}
/*
 * negate - return -x
 *     Example: negate(1) = -1.
*/
int negate(int x) {
/*
 *取反加一
 */
   int result = (~x) + 1;
   return result;
}
/*
 * isPositive - return 1 if x > 0, return 0 otherwise
 *     Example: isPositive(-1) = 0.
 */
int isPositive(int x) {
 *拒绝两种情况：负（符号位 1）和 0（取反得 1）
 */
   int temp1 = (x >> 31) & 1;
   int temp2 = !x;
   int result = !(temp1 | temp2);
   return result;
}
/*
 * isLessOrEqual - if x <= y   then return 1, else return 0
 *     Example: isLessOrEqual(4,5) = 1.*/
int isLessOrEqual(int x, int y) {
/*
 *考虑到异号计算溢出，有三种情况：
 *1：相等 2：同号，且 x-y<0 3：异号，且 x<0
 */
   int ifequal = !(x ^ y);//if equals
   int sub = x + ~y + 1; //x-y
   int ifsub = (sub >> 31) & 1; // x-y < 0
   int both = ((x ^ y) >> 31) & 1;//both pisitive or negative
   int xne = ((x >> 31) & 1);//sign of x neagtive
   int result = ifequal | (both & xne) | ((both ^ 1) & ifsub);
   return result;
}
/*
 * ilog2 - return floor(log base 2 of x), where x > 0
 *     Example: ilog2(16) = 4*/
int ilog2(int x) {
/*
 *因为 x 大于 0，所以本质是找最左边的 1。
 *采用二分法，先左移 16 位看是否大于 0。
 *如果大于 0，!一次后是 0，再!一次后是 1，<<4 得到 16，可以加入答案.
 *再把范围缩短一半，再左移或右移 8 位，进行同样操作。
 */
   int result = 0;
   result = ((!!(x>>16)) << 4);
   result = result + ((!!(x>>(result+8))) << 3);
   result = result + ((!!(x>>(result+4))) << 2);
   result = result + ((!!(x>>(result+2))) << 1);
   result = result + ((!!(x>>(result+1))) << 0);
   return result;
}
/*
 * float_neg - Return bit-level equivalent of expression -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 representations of
 *    single-precision floating point values.
 *    When argument is NaN, return argument.
 *    Legal ops: Any integer/unsigned operations incl. ||, &&. also if, while
*/
unsigned float_neg(unsigned uf) {
/*
 *nan的条件：E位是255（满1）且F位不是0，若全是0，则是infinity。
 *则先不管符号位，左移一位，找到左8位都是1的。
 *若它与左8位全1其它全0不全等，则直接返回。 其它的，返回^x的结果得反
 */
  unsigned temp1 = 0x80000000;
  unsigned temp2 = 0xFF000000;
  unsigned tempx = uf << 1;
  if((tempx & temp2) == temp2 )
  {
  if(tempx != temp2)
    {
    return uf;
    }
  }
  return temp1 ^ uf;
}
/*
 * 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.*/
unsigned float_i2f(int x) {
/*
 *取最左边作为符号位。转化为绝对值，并且记录下来符号位。
 *计算左边有多少个 0得到指数位。最后把阶码和尾码移动到相应的位置
 */
  unsigned abs = x;
  unsigned sign = 0x0;
  unsigned count = 0;
  unsigned flag = 0;
  unsigned mask = 0x80000000;
  if(x == 0)
  return 0;
  }
  if(x < 0)
  {
```

```
  abs = -x;
  sign = 0x80000000;
  }
  while (!(mask & abs)){
  mask >>= 1;
  count = count + 1;
  }
  abs <<= count + 1;
  if(((abs & 0x1ff) > 0x100) || ((abs & 0x3ff) == 0x300)){
  flag = 1;
  }
  return sign + (abs >> 9) + ((158 - count) << 23) + flag;
  }
/*
 * 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*/
unsigned float_twice(unsigned uf) {
/*
 *根据浮点是否规格化进行操作
 */
  if(( 0x7f800000 & uf) == 0){
  uf = (uf & 0x80000000) | (( uf & 0x007fffff) << 1);
}
  else if((0x7f800000 & uf) != 0x7f800000)
{
  uf = uf + 0x00800000;
}
  return uf;
}
```