Lab01:

1. bitAnd：
2. 用 ~ 和 | 代替 & ，从 | 和 & 符号的特点出发。
3. “|”的特点：当两个操作数都为0时，结果为0；“&”的特点：当两个操作数都为1时，结果为1。
4. 因此，首先猜想（~x）|（~y）= x & y，发现对于1-bit的数来说，等号两边刚好相反，因此~（（~x）|（~y））= x & y。
5. getByte：
6. 获取指定位，则应该首先利用移位操作使“目标值”处于“特定”位上，之后通过掩码来获取该值即可。
7. 移位应当左移还是右移？

答：右移。

1. 因为如果左移，则应当移23（3-n）位，含有“-”号；
2. 右移移动8n位，即n<<3，且右移之后“目标值”位于最后，&0xFF（8位）即可得到。
3. logicalShift：
4. 此题的难点就在于，对于负数，怎样使右移之后补0？

答：如果右移n位之后的前n位为0，可以利用1&0=0，0&0=0来实现。

1. 那么，问题的难点就转化为了如何得到一个只有前n位为0，其他位都为1的数？

答：移位。

1. 首先想到的是0xFFFF FFFF右移n位，前面补0，但是这产生了新的问题：通过算术右移，前面补的是1，这也就是这个函数应该要解决的问题。
2. 为避免该问题，想到0x7FFF FFFF右移（n-1）位。
3. 接下来解决（n-1）的问题，根据合法的符号，其中没有减号，那么如何右移（n-1）位呢？

答：

1. 先右移n位，再左移1位。
2. 此时最后一位补0，因此需要加一。
3. 之后通过 & 操作，就实现了逻辑右移。
4. bitCount：
5. 得到含有1的位数，使用 & 操作符。
6. 参考网上的解法，采用二分法：2位一组，统计其中1的个数，合成4位一组，8位一组，16位一组，最终得到总数
7. 两位一组：x & 0b0101…0101，得到奇数位上的1的个数，之后(x >> 1) & 0b0101…0101,可以得到偶数位上的1的个数，二者相加，即可得到含1总数（两位表示原值两位中1的个数）。
8. 四位一组：两位一组得到的结果(count + count >> 2) & 0x0011…0011，可以得到每四位中1的个数。
9. 八位一组，十六位一组，三十二位一组与上面类似，分别 & 0b00001111……00001111,& 0b1111 1111 0000 0000 1111 1111, &0b1111 1111 1111 1111。
10. 如何使常数不超过8 bits？

答：移位。常数分别由0x55 0x33 0x0f通过一系列移位操作得到。

1. bang：
2. 由bang（0）= 1入手，知0 + 1 = 1；特殊地，又有0xFFFF FFFF+1=0;因此可以通过将非零的数变化为0xFFFFF FFFF,最后返回时+1来实现。
3. 那么如何转化为0xFFFF FFFF呢 ？

答：

1. 首先猜想取反后 | 原值，发现0生成的也是0xFFFF FFFF，因此需要重新寻找其他方法。
2. 若想保证0 & 一个由0经过一系列操作得到的数（o）仍旧得到0，则o = 0。由-0 = 0
3. 因此选用补码。发现非零数 | 该数的补码，其他位不确定，但首位必定为1，因此，右移31位得到0xFFFF FFFF。
4. tmin：

最小的二进制补码0x8000 0000，直接通过1 << 31得到。

1. fitsBits：
2. 类比short转换为int类型的数，当一个负数转换为int类型时，前面需要填充1，正数则填充0；
3. 因此，倒过来，想要看该数能否用n-bit表示，就是将该数前面的（32-n）位全部转换为与后n位的首位相等的数（0或1），之后判断与原始的数是否相等，若相等，则表示可以用n位表示。
4. 如何使前（32-n）位与n-bit表示时的符号位一致呢？

答：先左移（32-n）位，再右移（32-n）位。

1. divpwr2：
2. 该题的特征：正数取下界，负数取上界。但是直接相除时，取的都是上界，因此对于正数而言，是可以直接通过右移来实现的，但是对于负数而言，则需要进行一定的变换。
3. 从结果入手：直接右移，之后对结果操作。

尝试一：

1. 先对操作数x右移n位得到x1，之后对（-x）右移n位得到（-x2），然后（x1+x2）>> 1，得到结果。
2. 发现，当x=0x8000 0000，在int范围内，没有（-x）与之对应，产生溢出错误。

尝试二：

1. 因为负数取的是上界，因此对得到的结果+1：首先直接对x右移n位，之后加上负数产生的误差（(x >> 31) & 1）即可。
2. 发现，并不是所有的负数都会产生误差，当整除时，不存在误差。
3. 从结果入手没有实现，因此，可以直接从操作数x入手，消除误差。对负数产生的误差，先消除后，再移位：
   1. 因为当x（小于0）不被整除时右移n与正确结果差了一个 -1，即一个2n;当x正好被整除时，右移n与正确答案相同;因此设偏置值为2n。
   2. 但是由于整除时没有误差，因此，设偏置值为2n-1（-2时会产生新的误差）。
   3. 如何对负数生成2n-1，正数生成0？

答：看符号位，对x右移得到0xFFFF FFFF或0，之后 & （1 << n）+ （~0）。（~0+1=0）

1. negate：

取反加一生成-x

1. isPositive：
2. 正数分为正数，负数，0，正数和0的符号位都是0，负数的符号为是1。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | x >> 31 | !x | (x>>31)|(!x) | !((x>>31)|(!x)) |
| 正数 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 |
| 负数 | 0xFFFFFFFF | 0 | 0xFFFFFFFF | 0 |

1. 关键点在于：如何区分0和正数。
2. isLessOrEqual：
3. 主要思路：相减
4. 什么情况下x-y＜0，但是x＞y？

答：x>0,y<0，二者相减后溢出。

1. 如何表示x-y？

答：利用补码，取反加一得到（-y）。

1. 如何判断x-y的符号？

答：右移31位，之后（&1），得到0（x-y≥0）或0xFFFFFFFF（x-y<0），但是并不能直接通过x-y的符号判断x与y的关系，因为不能排除溢出的情况。

1. 如何排除x-y<0，但是x≥0＞y的情况（即相减后溢出的情况）？

答：判断x和y的符号是否相同，右移31位 &1得到1或0，之后使用抑或判断，若不同返回1，若相同返回0；要保证x＞0但y＜0，还需要将抑或后的结果&x的符号位，若为1，表示x为负数，x＜y。

1. 除x＜y，x=y时也要返回1，因此同理（5），判断符号位是否相同，取非值即可。
2. 最后，三项的结果“|”运算即可。
3. ilog2：
4. 该函数即为判断操作数x中最高非0位的位置（最低位的下标为0）。
5. 如何得到最高非0位？

答：类似于bitCount，使用二分法，不过与之相反的是，从顶层向下找：

1. 首先，判断最高非0位位于前十六位还是后十六位，具体操作：

bitsNumber = (!!(x >> 16)) << 4;

右移得到16或0,即若在前十六位得到16,若不在，得到0;

1. 判断最高非0位位于上一步得到包含最高非0位的前八位还是后八位，具体操作：

bitsNumber + ((!!(x >> (bitsNumber + 8))) << 3);

右移上一步的结果加上8，得到前八位，再加上上一步得到的结果;

1. 八位分成四位，与上一步同理，具体操作：

bitsNumber + ((!!(x >> (bitsNumber + 4))) << 2);

1. 四位分为两位：

bitsNumber + ((!!(x >> (bitsNumber + 2))) << 1);

1. 两位具体到指定位：

bitsNumber + (!!(x >> (bitsNumber + 1)));

1. float\_neg:
2. 求-f，首先应当将符号位改反，即抑或首位为1，其他位都为1的数。
3. 但是，因为存在NaN的情况，即除符号位外，后面8位全为1，之后若其中含有1，则为NaN，因此，首先应当忽略符号位，但其他位不变，所以&0x7FFF FFFF,之后与0x7F80 0000比较大小，若大于该值，即表示NaN，返回去除符号位剩下的值，否则，直接返回将符号位改反的值。
4. float\_i2f：
5. 首先将浮点数分为三部分：符号位、阶码和尾数；将整型的x也分为三类：=0、＞0和＜0；
6. 特殊地，对于x=0，其浮点数表示仍然为0；
7. 因为正数与负数除符号位以外，其他的处理方式一样，因此，先将负数的符号位取反，之后与正数做相同的处理；
8. 阶码：先得到右移n位得到科学记数法的形式，之后（127+32-n）得到阶码，左移23位到阶码相应位置；
9. 尾数：取科学记数法形式中小数点后所有位，之后右移九位到尾数对应位置；
10. 注意：因为右移九位会导致最后的九位被舍去，因此需要考虑舍入问题，存在以下三种情况（与二分之一比较）：

1）如果舍弃的这九位的最高位为0，那么说明舍去的数值小于保留下来的最低位表示的值的二分之一，那么我们不需要舍入。

2）如果舍弃的这九位的最高位为1，并且后面的位有数值，那么说明舍去的数值大于第9位表示的值的二分之一，这个时候我们需要舍入，也就是把最终结果加一。

3）如果舍弃的这九位的最高位为1，并且后面的位都是0，这个时候正好就是第9位表示的值的二分之一。那么这个时候我们就要看第9位，如果第9位为0，那么不舍入。如果第9位为1，那么进行舍入，也就是把最终结果加一。

1. 最后三项合并，加上进位。
2. float\_twice：
3. 两种情况讨论：非规格化和规格化；
4. 非规格化：左移，注意：
   1. 补上符号位；
   2. 如果左移后符号位溢出，刚好成为阶码；
5. 规格化：指数加一即可。