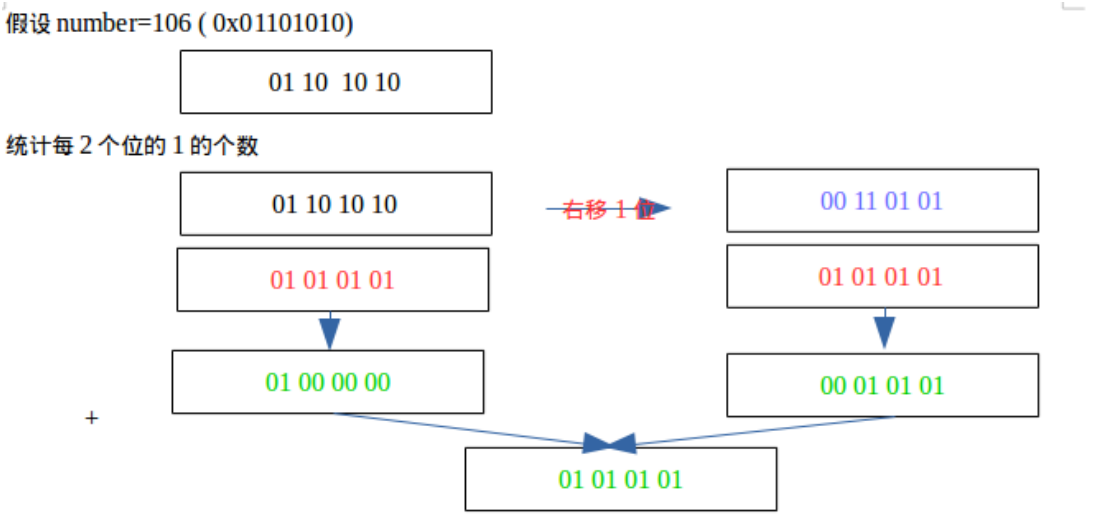
G.lx 的 bitCount 简要思路

正如提示所说，使用位运算分组求和。图中给出了 8 位整数，即 1 个字节的计算方法。由于题目保证 x 在 int 范围内，因此 x 是 32 位整数。32 位整数和 8 位整数做法很类似，在 8 位整数做法的基础上，只要再利用右移 8 位(00000000 11111111……00000000 11111111 = 0x00ff…00ff)和 16 位(00000000 00000000 11111111 11111111……00000000 00000000 11111111 11111111= 0x0000ffff…0000ffff)得到的结果（掩码，mask）进行计算，就可以得到答案。



不就代表每 2 位各有 1 个 1 咯。
同理，利用右移 1 位得到的结果 01 01 01 01 我们与 00 11 00 11 （右移 2 位）进行上图计算，再利用右移 2 位得到的结果与 00 00 11 11 （右移 4 位）进行上图计算，最后通过 00 00 11 11 求出来之后就是答案，很神奇。

```
/*
 * bitCount - returns count of number of 1's in word
 *   Examples: bitCount(5) = 2, bitCount(7) = 3
 *   Legal ops: ! ~ & ^ | + << >>
 *   Max ops: 40
 *   Rating: 4
 */
int bitCount(int x) {
  int mask1 = 0x55; // mask1 = 0101...0101 = 0x55...55
  mask1 = mask1 | (mask1 << 8);
  mask1 = mask1 | (mask1 << 16);
  int mask2 = 0x33; // mask2 = 0011...0011 = 0x33...33
  mask2 = mask2 | (mask2 << 8);
  mask2 = mask2 | (mask2 << 16);
  int mask3 = 0xf; // mask3 = 00001111...00001111 = 0x0f..0f
  mask3 = mask3 | (mask3 << 8);
  mask3 = mask3 | (mask3 << 16);
  int mask4 = 0xff | (0xff << 16); // mask4 = 00000000 11111111... = 0x00ff...00ff
  int mask5 = 0xff | (0xff << 8);  // mask5 = 00000000 00000000 11111111 11111111... = 0x0000ffff...0000ffff
  int 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;
}
```