lab1 report

17300180070 马逸君

本次作业为我们打开了新世界的大门。

在NOI时代，我因为主要停留在NOIP（省级联赛）层面，对位运算接触不多。一时能想起来的，只有lowbit(x) = x & -x，x & 1等效于x % 2但更快，和有时候会用~x代替x != -1。做完本次作业，我见识了很多。

第一题：用|和~表示&。把&的真值表写出来，从左到右、从上到下分别为0、0、0、1，因为只能用包括~在内的两种算符，我们先设法使得因变量变为1、1、1、0。尝试给自变量做变形，发现两者同时取反后相或即可。

第二题：取出x中的第n字节。思路非常简单，算出需要取出的是哪几位，左移相与保留相应位，回移即可。此题一个小插曲是我把“Byte”和“Bit”弄混了，导致没有一遍过。

第三题：逻辑右移。一个直观的思路是先把最高位置0，然后右移（系统的右移是算术右移），再把原最高位加回去即可。实现上，在用MSB = x & (1 << 31)分离最高位后不能直接MSB >>= n，因为系统默认的右移是算术右移；我们需要利用n计算出新的最高位放在第几位，然后用1左移。

第四题[1]：统计x的二进制表示中1的个数。此题很是考验技巧，我百思不得其解，即使是先做了第五题，还是没能做出来这一题。题解给出的方法是：把相邻位两两相加，循环进行。即一开始是每两位一加，然后是四位一加，最后一次循环就得到32位相加的和了。是二分思想的运用。

第五题：禁止使用“!“运算符，计算!x。显然此题难点在于用余下的几种算符判断x是否为0，最直观的方法就是所有位进行与运算。不能使用循环结构，而把顺序语句拷贝32次不仅会超运算符，还显得特别愚蠢。我一开始想的是把x按byte分割为四个8位数字，再想办法计算较短的数字（这个思路是在做byteCount这题时想出来的，但对于bang这一题有很大启发）。后来转念一想，这不就是分治吗？用高16位和低16位相或，结果为16位，再将其高8位和低8位相或……这样一来，判0只需10个算符，符合要求。后易得。

第六题：求n位补码整型数的最小值。补码有一个等价定义：n位补码最高位的权是-2n，除最高位以外的位，权值与一般的int相同，为2n-1、2n-2、……、1。由此，显然答案最高位为1，其他位全部为0。

第七题：判断给定的x能否用n位补码表示出来。n位补码的取值范围是[-2n-1，2n-1)，加上2n-1以后就变为[0, 2n)，只需判断超出的位（第n位以上的位）是否还有1即可。此题有一个坑点，n=32时，加231会影响到符号位，不适用上述情况；但int本身就是32位，所以所有的int是一定能用32位补码表示的，应直接返回1；题设说0<=n<=32，我们通过n>>5的结果来判断n是否为32，就可以特判了。

第八题：计算x/2n。课本上有讲过，编译器实现除2的幂的时候都是将其转化成右移来计算的，按照向0取整的规则，正数直接右移，负数则需要加上2n-1后右移（纠偏）。虽然不能用条件语句，但我们可以直接利用符号位算术右移来判断正负。完毕。

第九题：禁止使用“-”运算符，计算-x。早在上个学期的数字逻辑电路课程中，就讲过  
(-x)补 = ~(x补) + 1。完毕。

第十题：禁止使用关系运算符，判断是否x>0。容易发现x≠0时直接取出符号位，它的非就是答案。x=0时，直接返回!x。最终答案是两种情况的或。

第十一题：禁止使用关系运算符，判断是否x<=y。容易想到判断是否y-x>=0，这通过取y + (-x)补的符号位完成。然而，在btest中，发现x为-2147483648、y为2147483647时减法溢出了。遂采取如下改进解法：若x与y不同号，这时减法可能溢出，我们通过x、y和0的相对大小关系直接判断；若x和y同号，此时减法不会溢出，再利用y-x的符号判断。值得一提的是，不能使用条件结构，我们设置一个变量dir = sgnx ^ sgny（dir的意义是direction，sgn是符号位的意思），最后再用(s1 & dir) | (s2 & !dir)（s的意义是solution，s1和s2是按上述的前后两种情况分别求得的解）来完成选择操作。完毕。

第十二题[2]：位运算求floor(log2x)。基本思路肯定是判断最高的1所在的位。这题也是想了很久没想出来，在网上查到一个绝佳的解法：记r为需要右移的位数，初始r=0。将x与0xffff比较，如果比0xffff大，说明至少需要右移16位；r += 16（即1<<4 = 10000b）；并将此数右移16位进行更新；如果比0xffff小，什么都不做；r += 0。将更新后的数与0xff比较……依此类推，最后与1比较后得到的r即为最高位所在的位。是倍增思想的运用。

第十三题：给定一个浮点数f的机器数，求-f的机器数。根据浮点数的定义显然只需翻转符号位。但是题目要求若f == NaN则直接返回f；由课本知，阶码全1（0xff）、尾数非0的浮点数为NaN，用一个if语句判断f阶码全1但不是阶码全1尾数全0就可以了。

第十四题：给定int f，求(float)f的机器数。基本思路是先定符号，再定阶码，最后定尾数；求尾数时，尾数过短则左移补0，尾数过长则右移并舍入，舍入时还需要考虑就近可表示数的问题，考虑被丢弃的位上的信息，是难点所在，总之调了很长时间。

第十五题；给定浮点数f的机器数，求f\*2的机器数。首先容易想到特殊情况：无穷大、0、NaN直接返回。规格化数只需要给阶码加1。最后考虑非规格化数，如果尾数最高位为0，直接将整个数左移一位就是答案；如果为1，左移一位后阶码从0变成了1，尾数的1则自动缺省掉了，仍然适用。所以非规格化数左移1位就是答案；非规格化数和0的阶码都是0，不难发现0也适用于左移1位，为了少写一个if语句，我们可以将0和非规格化数合并成一句。至此我们已经讨论完了所有情况，程序上，对输入分类讨论即可。

参考文献：

1. <https://blog.csdn.net/qq_16964363/article/details/76599177>

1. <https://blog.csdn.net/u012138730/article/details/79818162>