Report
---Data Lab

黄子骕 21307130013

1.bitXor

用(x & y)去找出 x,y 中那些都为 1 的位，用(~x & ~y)找出 x，y 中那些都为 0 的位，由于最后结果我们是要这些位为 0，而在 x，y 中一个 1 一个 0 的位为 1，因此~(x & y) & ~(~x & ~y)即可有此效果。

2.tmin

因为 1 的二进制表示为 000....1,所以我们令它左移 31 位就可得到 100...0，即为补码表示最小整数。

3.isTmax

首先，补码表示的最大整数为 01111...1，即除了最高位为 0，其他位均为 1。我们可以用(~x ^ (x + 1)) 来验证是否处最高位外均为 1，因为只有均为 1， 在+1 后不断进位，(~x ^ (x + 1))的值才能为 0。

同时，由于-1（11111...1）除最高位外也都为 1，因此需要额外排除它的干扰，只有当 x 不为-1 时，！(~x)的值为 0，因此!(((~x ^ (x + 1))) | !(~x))即可保证 x 为 0111...1 时候返回 1。

其他时候返回 0。

本题出现问题：一开始没有考虑到-1 的情况，只判断了低 31 位导致产生错误。

4.allOddBIts

前两行代码可以构建一个二进制表示为 101010...10 的数。

进一步利用这个数与 x 进行&运算，提取出 x 的奇数位。

最后将上一步结果与 101010....10 作^运算来比较 x 奇数位是否均为 1，并对结果做！运算，此时 x 若奇数位均为 1，结果返回 1。

5.negate

使用公式~x + 1 = -x。

6.isAsciiDigit

首先使用 is_less 和 is_larger 来保存 x 与上下限的差值，之后仅需判断差值的符号位是否均为 0 即可（由于在此题的条件中,若即使出现溢出时，is_less 和 is_larger 的符号位不可能同时为 0，因此可以这样判断）。

7.conditional

当 x 为 0 时，test 为 000...0，当 x 不为 0 时，test 为 111...1，此时，test 与~test 之间必定会有一个为 000...0，为 000...0 的那一侧便不输出，因此用(test & y) | (~test & z)， 利用 test 是否为 000...0 来做判断输出 y 还是 z。

8.isLessOrEqual

当 x，y 符号位相同，且 x <= y 时， y-x 的符号位一定为 0;

当 x，y 符号位不同时候，仅需确认 x 的符号位为 1 即可;

最后一行分别表示上述两种情况，只要有一种情况成立（即&的一侧为 0000..0）则返回 1。

9.logicalNeg

在这里我考虑 x 与 -x 符号位相同时，x 为 0 或 INT_MIN，因此只需验证 x 与-x 的符号位是否相同，并排除 INT_MIN 干扰即可（即保证符号位为 0 而不为 1），当同号且 x 不为 INT_MIN 时，((sign_x ^ sign_neg_x) | sign_x)结果为 000...0，其他时候为 111...1。

本题出现问题：一开始未考虑到 INT_MIN 的干扰导致出现错误。

10.howManyBits

首先将正负数都统一成统一情况（若负数则取反），即找最高位的 1 并+1（符号位）得到表示 x 的最小位数。

先检查最高的 16 位，若含 1，则表明要表示 x 需要（32-16）=16 位以上，因此使 test_16 保存为 16，同时将 x 右移 16 位以此使得最高的 16 位都变为 0，进而可以无影响的研究剩余位；若不含 1，则表明要表示 x 需要 小于等于 16 位，此时 test_16 为 0，同时在这情况下最高的 16 位同样为 0，可以直接继续研究剩余位。

重复上述步骤，直到只剩一位无法再分。

将所有测试结果相加并+1（符号位）得到结果 。

11.floatScale2

首先用 sign，exp，frac 分别保存符号位，阶码以及尾数；

若 exp 为 special 状态，直接返回 uf；

若 exp 为 denormalized 状态，此时有几种可能性：

1 - 尾数的最高位为 0，此时其他不变将尾数左移 1 位即可；

2 - 尾数的最高位为 1，此时尾数左移 1 位同时 exp+1（因为 exp+1 后变为 normalized 状态，M 的整数位变为 1）；

3 - 尾数为 0，则直接返回 uf（0 乘一切都 0）；

若 exp 为 normalized 状态，先令 exp+1，后则有两种可能：

1 - 若变为 special 状态则直接返回同符号的无穷；

2 – 若仍为 normalized 状态则其他部分不变，返回 exp 部分+1 后的结果。

12.floatFloat2Int

首先记录 uf 的符号位，阶码，尾数；

若为 denormalized 状态，则直接返回 0，其他情况则计算出 E；

以下根据 E 的大小有三种可能：

① 若 E>31,则超出 int 表示范围，返回 0x80000000；

② 若 E<0，则直接返回 0；

③ 若 E>=0 且 E<=31，此时将尾数适当左移或右移，使得小数点在适当的位置（即只保存整数部分，此时有四种情况：

1 - uf 为正，同时移动后未覆盖符号位，此时直接返回；

2 - uf 为正，移动后覆盖了符号位，此时 E 一定为 31，此时一定超过 int 表示范围，返回 0x80000000；

3 - uf 为负，移动后未覆盖符号位，此时返回相反数；

4 – uf 为负，移动后覆盖了符号位，此时 E 一定为 31，此时当结果不为 100...0 时，超出负数表示最小值，返回 0x8000000，若结果为 100...0，根据题意，同样返回 0x80000000。

13.floatPower2

将 x 取值范围划分为三个部分：

1 - x>127 时,超出表示范围，返回正无穷；

2 - x<-149 时，无法表示，返回 0；

3 - -126<=x<=127 时，用 normalized 状态表示，frac 为 0，exp 为 x+Bias；

4 - -149<=x<-126 时，用 denormalized 状态表示，exp 为 0，frac 根据 x 大小通过移位操作确定。