实验报告

trick（已失效）

设置全局变量所需符号不会被编译器计数

bitXor

运用德摩根律获得恒等式可得答案

thirdBits

考虑通过移位再按位或构造所需位为 1，其余位为 0 的数，基础数为 73（0x01001001）

fitsShort

若 x 可以被 16 位表示，则代表 x 的 16 到 31 位的每一位都与第 31 位相同，据此两次移位后异或判断可得答案

isTmax

若 x 为 Tmax，则加 1 后所得数 y 的两倍为 0，再排除 y 为 0 可得答案

fitsBits

与 fitsShort 类似，不过这次需要移动 n-1 位与移动 31 位异或
优化：(n+~0)%32==(n+31)%32，可以节省一个运算符

upperBits

考虑到 n=0 时数为 0，故先将最高位设置为!!n，再右移 n-1 位，优化过程同 fitsBits

anyOddBit

同 thirdBits 一样，先取基础数 0xaa 通过移位和或运算构造一个奇数位为 1 偶数位为 0 的数，与原数按位与后判断是否为 0 可得答案

byteSwap

考虑到性质：a^(a^b)=b; b^(a^b)=a
先移位获得 m 和 n 字节，将两者异或再&0xff 即可得到(a^b)，再将其左移到对应的原位置分别异或即可实现交换

absVal

先左移 31 位获取符号位 i（正为 0，负为-1），正数时 x 不变，负数时变为~x+1
考虑到~x+1==~(x-1)==(x-1)^(-1)，故取(x+i)^i 即为答案，检验可得对非负数依然成立

divpwr2

先左移 31 位获取符号位 sign，考虑将 x 加上偏移量再右移 n 位
而 sign 为 0 时偏移量为 0，sign 为-1 时偏移量为 2^n，故偏移量(sign<<n)^sign 即满足要求

float_neg

根据浮点数 NaN 的性质，其与 0x7fffffff 异或后所得 temp 的 31 位为 0，23-30 位为 1,0-22 位不全为 0,分段判断可得答案
优化：temp 满足 NaN 条件时必>0x7f800000，可节省运算符

logicalNeg

取 x 和-x 按位与后的符号位，符号位为 0 则 x 为 0，为-1 则 x 非 0，加 1 即可得到答案

bitMask

本题直接构造较难，考虑先取~0 使得所有位为 1，再分别向左移动 lowbit 和 highbit+1 位，两者进行异或
由于 lowbit>highbit 时返回 0，故所得数再与左移 lowbit 位按位与后可保证满足所有条件
优化：注意到对于 highbit，可以构造一个 highbit 左边全为 0，自身及其右边全为 1 的数再与前者按位与，而~0+(2<<highbit)即可满足条件，节省了一个运算符

isGreater

考虑等价条件 y-x<0，因为这样可以把 y-x=0 归入正号类型中
本题进行分类讨论，先取 p=~x,t=y+~x+1(即 t=y-x),y&p 和 y^p 有且仅有一个为 1，另一个为 0，成为控制开关
从而当 x 和 y 异号时，y&p 即为所求；x 和 y 同号时，根据 t 的符号返回 0 和 1，故表达式(x&p)|((x^p)&r)满足要求，右移 31 位&1 可得答案
优化：x>y 时有类似情况，不过可令 t=y+~x(t=y-x-1)，这样 y=x 被划入负号类型，进行相似操作后所得数符号为 0 时 x>y，为-1 时 x<=y，取!可得答案，节省了一个运算符

logicalShift

将 x>>n 后，与 upperBits 中所得数取反的结果按位与即可，需要注意由于 n 不取 32，故可以直接用(1<<31)>>(n-1)得到桥梁数，而 n=0 时应为 0，故先>>n 再<<1 即可满足条件
优化：考虑到 31-n 等价于 31^n，故令 t=1<<(31^n)，与 x>>n 相加后异或 t 即可

satMul2

观察发现当乘以 2 溢出时，该数的最高位和次高位必不相同，将最高位和次高位做异或操作之后通过移位操作使得所得数 sign 为 0 或-1，据此设置控制开关（与 isGreater 类似）选择得到 x<<1 或是 tmin/tmax(tmin+sign)
优化：最后一个优化的题，从 10 到 9 到 8 到 7 到 6，几乎睡觉都在想，总算弄出来了（兴奋）
主要是利用 k=-1 时移位 k 等于移位 31 的这个特点，设置同时能得到 tmin 的控制开关，即(t>>k)^(k<<k)，这成为最佳答案

subOK

对最高位分类讨论：当 x 和 y 最高位相同时，必不会溢出；两者不同时，比较 x 和 x-y 的最高位是否相同，相同则没有溢出，否则溢出，本题主要操作是异或
优化：与 isGreater 类似，x-y 与 x 同号等价于(y-x-1)与 y 同号，省略一个运算符

trueThreeFourths

与 divpwr2 题目类似，x 先和 3 按位与得到余数 mod，本题我通过尝试几组正负数数据来寻找偏移量
可以发现当 x 为非负数时，偏移量为 3*mod/4*；为负数时，偏移量为*(3*mod-3)/4，故偏移量表达式为((mod+mod+mod+(sign&0x3))>>2)
优化：最最后一个优化的题，柳暗花明又一村
首先还是要得到 mod，不过这次可以让(x>>2)+(x>>1)来代替 3*(x>>2)，从而写出正负号下模与偏移值的对应关系，得到偏移量为(mod+1)>>(sign+2)即为所求

isPower2

若 x 符合条件，则 x&(x-1)为 0，但此时 x 包括了非正数 0x80000000 和 0，故!(x<<1)，即为排除方法
优化：可以发现对于 y=x-1，y>>30 仅在 x 为 0x80000000 和 0 时不为 0，故可以节省一个运算符

float_i2f

整体思路：分而治之，将浮点数分为符号位 sign，指数位 index，浮点位 t 和四舍五入尾数 round
首先判断 x 是否为 0，为 0 则直接返回 x，不为 0 则取出符号为并得到 x 的绝对值设为 t，接着将 t 左移直到其最高位为 1，之后再左移 1 位即可得到指数，此时 t 表示小数部分。由于浮点位只能保留 23 位，故 t>>9 即为对应位置的浮点部分。最后考虑浮点数的舍入，根据向偶数舍入规则，只有 t 的最后 9 位>0x100 或者最后 10 位为 0x300 才会进位(round=1)，再将各个位相加即可
优化:由于 index=(158-higbit)<<23，故可令 index=0x4f000000，highbit=highbit-0x800000;此外判断 round 可以使用 round=((t&0x1ff)>0x100)来避免 if 语句，加上 volatile 关键字后可以确保输出结果正确

howManyBits

采用二分的办法，先判断该数的正负性，若为负则取反（不用+1，位数是一样的），然后依次右移 16,8,4,2,1 位，判断所得数是否为 0，最后再把各个操作所得结果相加，再加上固定的 1 个符号位即为所需位数

优化：若考虑向左移位，则可以去掉一个!符号，其次由于每次移动位数是 0,2,4,8,16，故首先定义 sum=31，利用异或实现减法，可以节省较多运算符。

float_half

首先判断 round 是否为 1，并将 x 符号位设成 0，接着对 x 分类讨论:
如果指数位为 0 或者 1，那么 x>>1 后加上 round 再加上符号位即可
如果 x 指数位全为 1 且浮点位不全为 0，则 x 位 NaN，返回 x，否则直接将指数位-1 即可
优化：指数位为 0 或者 1 等价于<=0x800000，指数位全为 1 且浮点位不全为 0 等价于 x>=0x7f800000