Assignment 1

整数

bitOr

利用德·摩根律可将 `|` 用 `~` 运算和 `&` 运算表达

getByte

将所求的字节移到最低位，然后用 `0xff` 将其提取出来

logicalShift

算术右移与逻辑右移的区别在于最高位补的数，因此先将符号位去掉（使用一次 `xor`），最后将符号位补上（注意到可以通过 `31 xor x` 来代替 `31-x`）

bitReverse

首先考虑如何交换一对 `bit`，注意到 `01->10,10->01` 相当于 `xor` 上 `11`，而 `00->00,11->11` 相当于 `xor` 上 `00`。通过这一发现，可以将所有的 `4n` 与 `4n+3` 位上的比特交换，`4n+1` 与 `4n+2` 位上的比特交换。最后只需要将 8 个“4 比特组”顺序颠倒即可。由于可用算符数有限，我在诸多细节处采用了较多可以减少算符的设计。

bang

利用每次折半的方法，将所有的位 `or` 起来，通过考察最低位即可判断 `x` 是否为 0

tmax

将 `-1` 的符号位去除即可得到最大的 `int`

fitsBits

首先容易取得 `x` 的绝对值，由于需要符号位，因此只要满足 `x<(1<<(n-1))` 即可 fit。

dividePower2

容易观察到，`(-1)>>1` 是负数中唯一一个不满足向 0 取整的数；考虑将负数转换为正数，计算完后再转换为负数，但 `-2^31` 的绝对值无法表达，因此，对于负数 `x`，先计算 `(-x-1)/(2^n)`，再考虑被减去的 `1` 何时产生贡献，容易发现仅当 `+(((-x-1)>>n)!=(-x)>>n)` 时被减去的 `1` 能导致 `n` 次进位从而产生贡献。
最后使用的 `(x & (-b)) | (ox & b)` 是非常实用的选择语句。

negate

`-x=(~x)+1`

isPositive

取符号位判断即可

isLessOrEqual

分两种情况讨论：

1. `x,y` 异号，那么仅当 `x<0<=y` 时返回为 1
2. `x,y` 同号，那么仅当 `x-y<=0` 时返回为 1

基于这两种情况讨论即可

intLog2

借助倍增（或者说二分）的思路，将判断次数降低到 `log(32)=5` 次，具体实现是显然的

浮点数

floatIsEqual

除了特判 `NaN` 的情况、`+0=-0` 的情况，其余的只要判断整数表达相等即可

floatFloat2Int

只要按照标准进行翻译就行了，比较复杂的地方在于判断浮点数是否超过 `int` 的上下界。

floatScale2

同样也是按照标准，对规格化的表达与非规格化的表达分别进行计算即可。