ICS Datalab Report

胡译文 2021201719

Results

Quick Takeaways

- `int sign = x >> 31;` 符号位填充满整个变量，可用于有条件取反
- `(~x) == (-x-1)` 一定程度上起到负号（减法）的作用（除 `0` 和 `Tmin`）
- `!(x | y)` 连等于一个常数（如零）
- `!(x ^ y)` 判断两数是否相等
- `补码 = [](int x) { return (~x) + 1; }`
- 超出位数的位移运算是未定义行为，在某些编译器上位移运算会取模处理（不要在其他地方使用该性质）
- 熟练运用德摩根律

Solutions

bitXor

重点在于利用`&`和`~`表示`|`。`x^y = ~(x&y) & ~(~x & ~y)`

thirdBits

一开始做的时候有一个歧义，每三位一个1的1从哪里开始。解决完歧义后剩下的就是定义变量，指数级复制。（在 logicalNeg 证明了以 `2` 为底是最优解）

fitsShort

最暴力思路：判断除符号位、低15位以外有没有1（负数是判断有没有0）（op数太多orz）

优化：判断有没有 1（或 0 ）就是其实在判断有没有非符号位。利用右移填充符号位的性质+高位应与符号位相同的性质，判断填充前和填充后是否相等即可。

```
int trunc = x >> 15;
return !((x >> 31) ^ trunc);
```

isTmax

本题关键在于探索 `Tmax` 的性质：1. `Tmax+1==Tmin` 2. `~Tmax==Tmin`。但同时拥有这个性质的还有 `-1` （观察可以发现两者仅符号位不同）。令 `y=x+1`：

一开始利用同时利用两个性质再排除 `x+1==0` 情况：`(!( x ^ y ^ ~(!x))) & (!!y)` 同时还实验了好几个思路：`!((y+x)^(~(!y)))`，`!(~(y+x+!y))`）

优化思路时发现 `Tmax+1` 的性质：`(Tmax+1) * 2 == 0`，遂变成：`!((y+y) | (!y))`。

fitsBits

直接将 fitsShort 第一种解法中的 15 更换为 ~0 + n，结果操作符超标~~尝试德摩根律优化（位运算不符合）~~。而若将第二种解法的 16 直接更换为 n，则报错:-)。

同学提示“位移会取模”，遂优化第二种解法的减法：

```
int trunc = x >> (n + 31);
return !((x >> 31) ^ trunc);
```

upperBits

构造掩码即可，一开始将 ~0 左移 32 - n，其中 n == 0 || n == 32 时返回都是 ~0，特判 0 的情况 +1 即可。最后运算符太多。本题灵活度很高，尝试各种姿势的构造：

```
int nn = !n;
return ret = ((~nn) << (~n + !nn));
```

但可能左移操作已经到顶，尝试上一题的优化，改用减法和右移：

```
int tmin = 1 << 31;
return (tmin >> (n + 31)) + !n;
```

anyOddBit

构造掩码即可，返回 !!(odd & x) 即可。

byteSwap

模拟思路：（~~一开始蠢蠢的连 +8 都不会写了，写了三次加法~~）把对应位数用掩码取下来，交换，再放回去。int get_n = ((x & mask_n) >> octuple_n);

同学提示“异或”，遂想起来无临时变量交换int值的方法，可以不需要把原数“挖洞”，简化操作：

```
int octuple_n = n << 3;
int octuple_m = m << 3;
int mask_n = 0xff << octuple_n;
int mask_m = 0xff << octuple_m;
int get_n = ((x & mask_n) >> octuple_n);
int get_m = ((x & mask_m) >> octuple_m);
int mix = (get_n ^ get_m) & 0xff;
return x ^ (mix << octuple_m) ^ (mix << octuple_n);
```

最后发现其实没必要获得掩码 mask_n）

```
int octuple_n = n << 3;
int octuple_m = m << 3;
int get_n = (x >> octuple_n);
int get_m = (x >> octuple_m);
int mix = (get_n ^ get_m) & 0xff;
return x ^ (mix << octuple_m) ^ (mix << octuple_n);
```

absVal

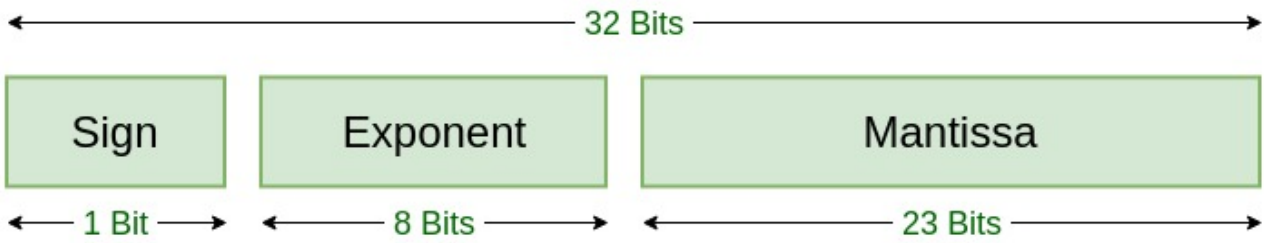
简单版：`(x ^ sign) + (sign & 1)`

压缩版：`(x + sign) ^ sign`（分析发现除加一部分`sign & 1`都不能省略，遂考虑更换运算顺序。`x` 取反之前减一即取反之后加一，遂得。）

divpwr2

（~~蠢蠢的不会算除法~~）右移是向下取整，题目要求是向零取整，区别在于负数。解决方法就是把负数增加一个小量（这里说的是字面值，不是绝对值），强制“退位”。小量试过直接 2^n-1，但运算符太多。遂将 `-1` 利用取反简化：`(sign << n) ^ sign`。

float_neg



Single Precision
IEEE 754 Floating-Point Standard

logicalNeg

暴力思路：判断每一位有没有值。$\log_m(32)\times 2(m-1)$ 次运算有点多，其中 m 表示以多少为底数（2 时取得最小值 10）。

聪明思路：与 isTmax 类似，本题找 `0` 的特征：`(~0) == Tmin`, `Tmin + 1 == Tmax`。其中最重要的特征是 `补码(0) == 0 && 补码(Tmin) == Tmin`，不改变符号位。`((x | ((~x) + 1)) >> 31) + 1`

bitMask

isGreater

logicalShift

第一思路就是构造符号位的mask，抵消填充符号位带来的影响：

```
int msk_sign = ((x >> 31) << (~n)) << 1;
return msk_sign ^ (x >> n);
```

这里 `n == 31` 左移 `32` 的情况需要拆分成左移 `31+1` 位。考虑优化：

trueThreeFourths

损失量的主要规律是 `3->2`，`2->1`，`1->0`。需要一点一点磨细节：

- `(~(x >> 31))` 是正负的特判
- `!two` 是0的特判

```
int y = x >> 2;
int two = x & 0x3;
int min = two + (~(x >> 31)) + !two;
return (y << 1) + y + min;
```

References