

datalab 报告

姓名：吴政坤

学号：2023202317

总分	bitXor	logtwo	byteSwap	reverse	...
36	1	4	4	3	...

test 截图：

解题报告

亮点

- 1. reverse
- 2. float_i2f
- 3. logtwo

reverse

```
// 附上题目解题代码
unsigned reverse(unsigned x)
{
    x = (x & 0x55555555) << 1 | (x & 0xAAAAAAAA) >> 1; // 交换相邻的1位
    x = (x & 0x33333333) << 2 | (x & 0xCCCCCCCC) >> 2; // 交换相邻的2位
    x = (x & 0x0F0F0F0F) << 4 | (x & 0xF0F0F0F0) >> 4; // 交换相邻的4位
    x = (x & 0x00FF00FF) << 8 | (x & 0xFF00FF00) >> 8; // 交换相邻的8位
    x = (x & 0x0000FFFF) << 16 | (x & 0xFFFF0000) >> 16; // 交换相邻的16位
    return x;
}
```

通过循环32次，逐位x 的每一位移动到 result 的相应位置。我觉得这个思路比较巧妙。
每次循环中，将 x 右移一位，将 result 左移一位，并将 x 的最低位添加到 result 的最低位。
当 x 变为0时，循环结束，result 即为反转后的结果。
为了减少操作数，使用分治法，逐步将32位分成更小的部分，然后逐级反转。

float_i2f

```
unsigned float_i2f(int x)
{
    if (x == 0)
        return 0; // x = 0 的情况
    if (x == 0x80000000) // x = TMin, -2147483648
        return 0xCF000000;
    unsigned int e = 0, E, M, temp_x, round;
    unsigned int result;
```

```

unsigned s = x >> 31 & 1;

if (x < 0)
{
    x = -x;
}

temp_x = x;
while ((temp_x >> 1) >= 1)
{
    temp_x >>= 1;
    e++;
}
E = e + 127;
x <<= (31 - e); //小数部分
M = (x >> 8) & 0x7fffff; //使尾数对齐
//一个 int 有 32 位，其中有 31 位可以用以表示精度，而 float 的尾数位有 23 位，所以我们将 int 的精度位右移 8 位（即损失 8 位精度），从而得到 float 的尾数位。
round = x & 0xff; //要舍入的小数部分，取x的最低八位。根据“四舍六入五成双”的原则，对于最低位是否要舍入进行判断。
if (round > 0x80) //10000000，即128，如果尾数部分的最低8位大于128，说明我们需要对尾数进行向上舍入
    M += 1; //阶码++
else if (round == 0x80)
{
    if (M & 1)
    {
        M += 1;
    }
} //如果最低8位等于128，根据“四舍六入五成双”舍入规则，
//如果当前尾数 M 的最低位为1，则我们加1。
//如果最低位为0，则不做变化。
if (M >> 23)
{
    M &= 0x7fffff;
    E += 1;
} //检查尾数 M 是否超出了23位。如果最高位为1（即 M >> 23 为真），这意味着尾数溢出，
//因此保留尾数的低23位，阶码 E 加1

result = (s << 31) | M | (E << 23);

return result;
}

```

这道题80%比较好写，但是我一开始忽略了8位精度的损失，通过网上查阅资料才发现到这个点，我认为这个点设计的很巧妙

logtwo

```

int logtwo(int v)
{
    int result = 0;
    int shift16 = (v > 0xFFFF) << 4; //值为0或16
    result |= shift16;
}

```

```

v >>= shift16;

int shift8 = (v > 0xFF) << 3;
result |= shift8;
v >>= shift8;

int shift4 = (v > 0xF) << 2;
result |= shift4;
v >>= shift4;

int shift2 = (v > 0x3) << 1;
result |= shift2;
v >>= shift2;

result |= (v > 0x1); //这里result直接与真值或运算，可以减少步骤

return result;
}

```

思路比较简单，检查最高位是否在16位以上,如果是result+16，以此类推；若最高位多于16位，v就右移16位，否则右移0位，以此类推

比较困难的是把步骤压到25以下，因为很容易写出26步的版本

我通过或运算和左移，优化掉了+ (例如result |= shift16;v >>= shift16;)

在最后一次并行运算时result直接与真值或运算，可以减少步骤

反馈/收获/感悟/总结

前前后后大概花了15个小时，我认为难度比较合适，我遇到困难点主要在：前期的环境配置、熟悉操作、个别题目的分析

我认为最卡我的地方是，float_i2f这个题，我在没有题目信息提示的情况下，根本没有想到精度损失的问题，应该是我对IEEE 754 还不够熟悉

通过datalab，我熟悉各种位运算，了解了部分计算的底层逻辑，了解了int、float类型的数的构成方式
总体而言，还是很有意思的))

参考的重要资料

更适合北大宝宝体质的 Data Lab 踩坑记

<https://arthals.ink/blog/data-lab>