《计算机系统》

datalab实验报告

[datalab实验报告 I](#_Toc1180165116)

[1 实验二 3](#_Toc876140920)

[1.1实验名称 3](#_Toc5074961)

[1.2实验目的 3](#_Toc1543007294)

[1.3实验资源 3](#_Toc311069086)

[2 实验任务 4](#_Toc1162931604)

[2.1实验规则： 4](#_Toc1142797081)

[函数分析： 4](#_Toc2044285246)

[验证： 17](#_Toc711261169)

[3 总结 19](#_Toc1272488181)

[3.1实验中出现的问题 19](#_Toc2066701241)

[3.2心得体会 19](#_Toc1647250909)

# 实验二

## 1.1实验名称

datalab

## 1.2实验目的

1. 完善bits.c函数，并利用目录内文件检验其正确性。
2. 学会机器的位级表示以及位级逻辑操作与运算。
3. 掌握整型和浮点数之间的转换。

## 1.3实验资源

1. datalab-handout.rar
2. 《深入理解计算机系统》
3. CSDN

# 实验任务

## 2.1实验规则：

**整数：**

1. 不能使用任何控制结构，例如 if、do、while、for、switch 等。

2. 不能定义或使用任何宏。

3. 不能在此文件中定义任何附加函数。

4. 不能调用任意函数。

5. 不能使用任何其他操作，例如 &&、||、- 或 ?：

6. 不能使用任何形式的铸件。

7. 不能使用除 int 之外的任何数据类型。 这意味着您不能使用数组、结构体或联合。

**浮点数：**

1. 不能定义或使用任何宏。

2. 不能在此文件中定义任何附加函数。

3. 不能调用任意函数。

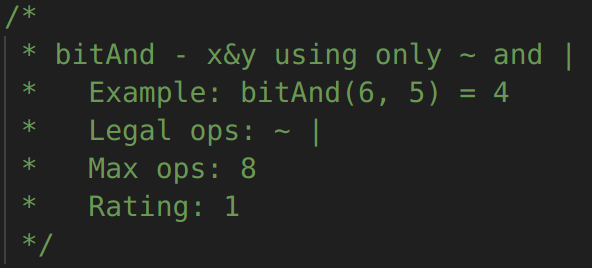
4. 不能使用任何形式的铸件。

5. 不能使用除 int 或 unsigned 之外的任何数据类型。 这意味着你不能使用数组、结构体或联合。

### 函数分析：

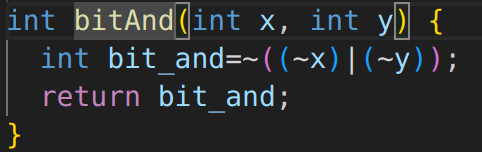
#### 2.2.1 bitAnd

要求：只使用按位取或，按位取反操作实现按位求与操作。



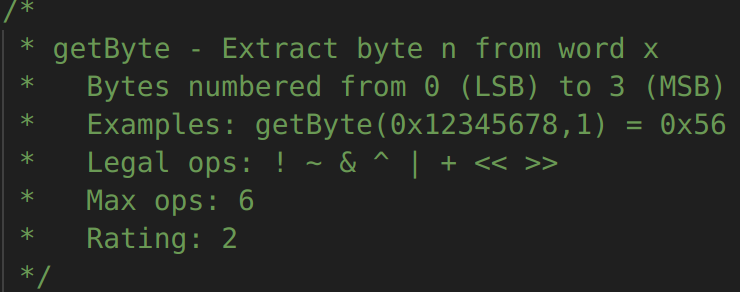
分析：根据离散数学的德摩根律，a&b=!(!a|!b)

代码：



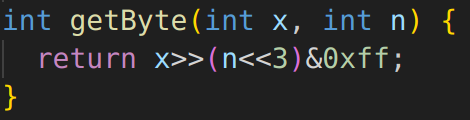
#### 2.2.2 getByte

要求：从x中得到第n个字节



分析：将所需字节右移到最低位，然后与0xff取与得到低八位，这里是4字节数据，32位，由于从低位开始给字节编码，一个字节占8位，则对于第n个字节，需右移（n<<3）位。

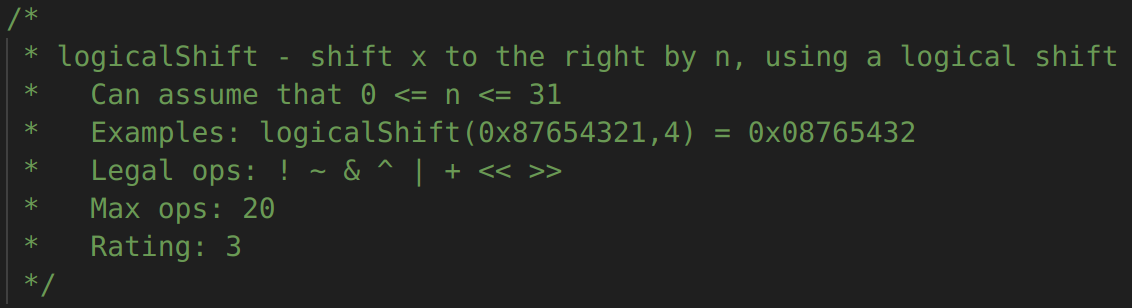
代码：



#### 2.2.3 logicalShift

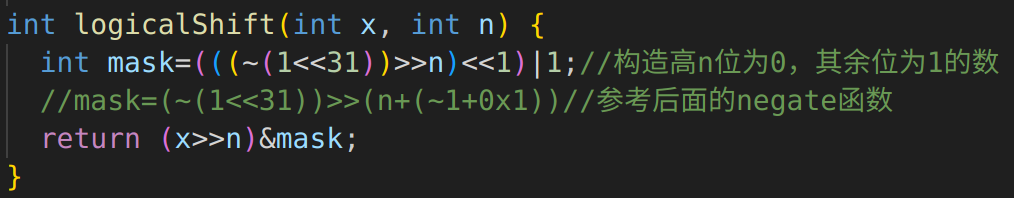
要求：

实现逻辑右移。



分析：由于传入x的数据类型是int，x>>是算术右移，那么当x是负数时，会导致符号位向右补1。所以我们将x右移之后还需要将高n位变成0，这里可以通过与00..11..(左n位都是0，其余位都是1)相与&。

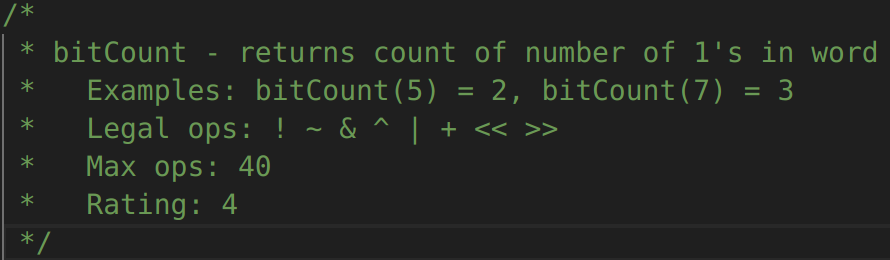
代码：



#### 2.2.4 bitCount

要求：

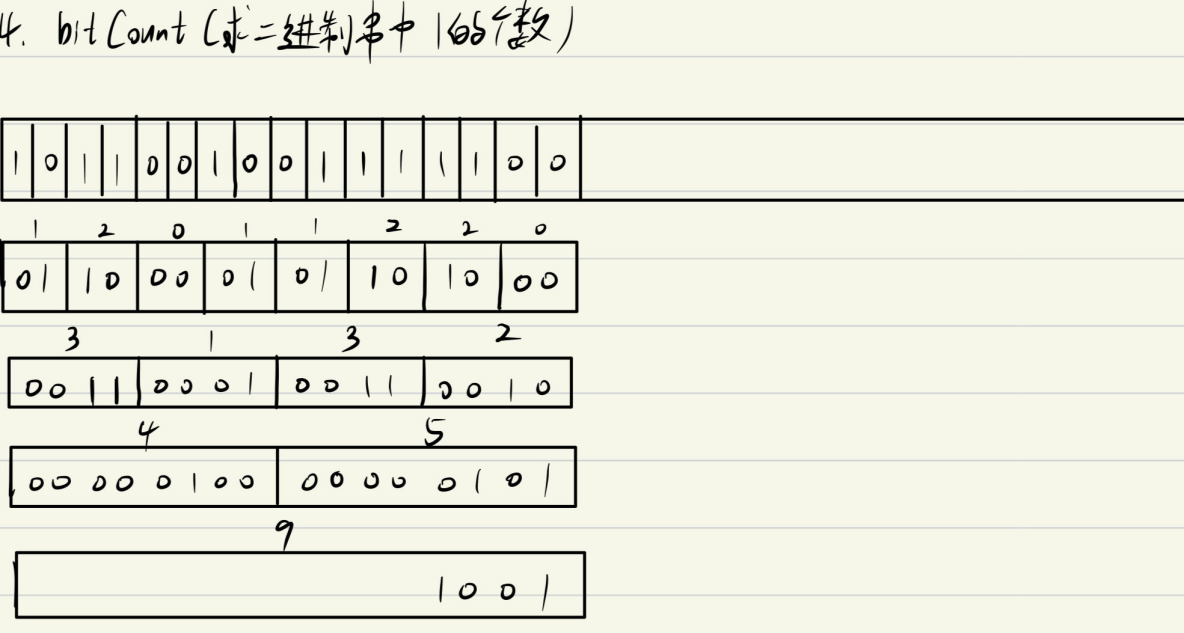
计算x对应的二进制串中1的个数。



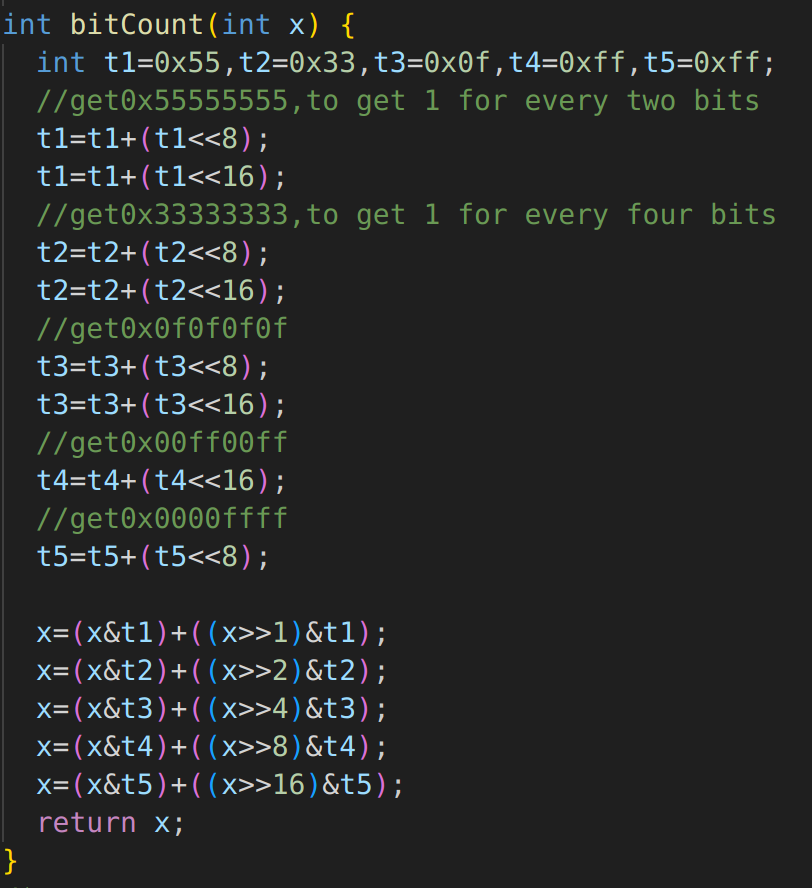
分析：采用二分与分治法思想，先算每2位有几个1，再每4、8、16、32。

那么如何算每2位的1的个数呢？

先看每两位的低位是否为1，然后看每两位的高位是否为1。这里通过掩码实现对每2、4、8、16取1位。



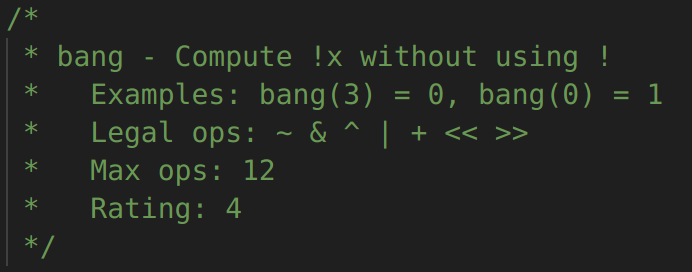
代码：



#### 2.2.5 bang

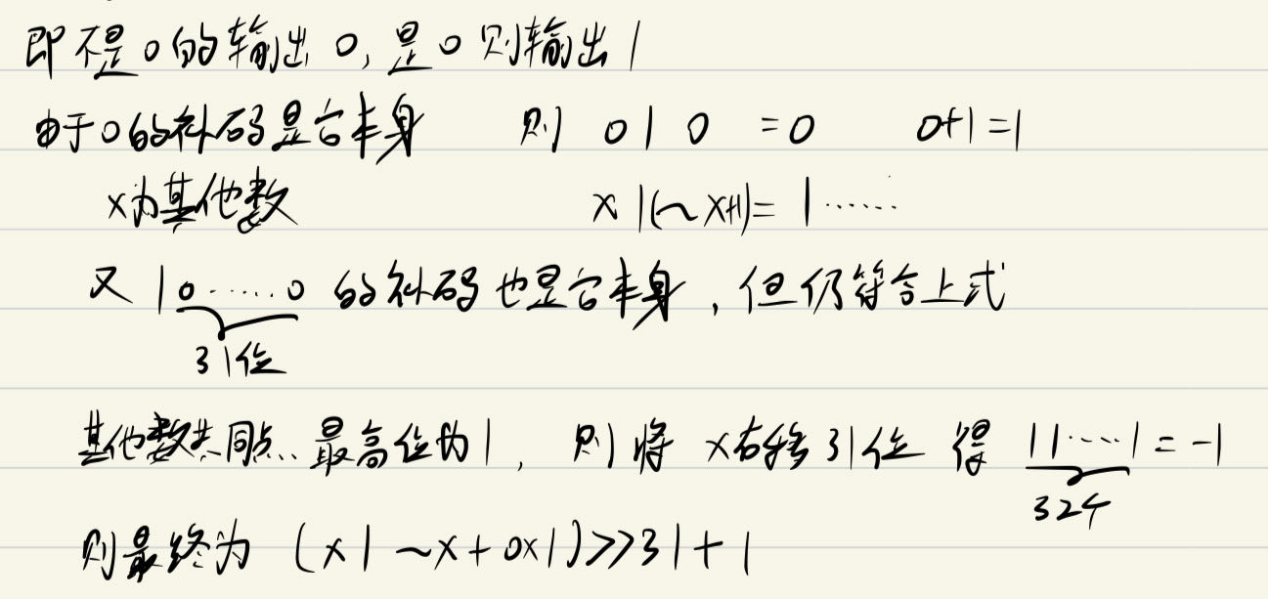
要求：

不用！实现逻辑非操作。

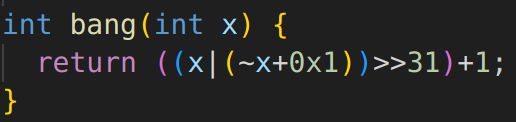


分析：逻辑非即 不是0的数输出0，是0则输出1。

只需要写出一个式子，使得x=0时输出1，x=其他数时输出0。



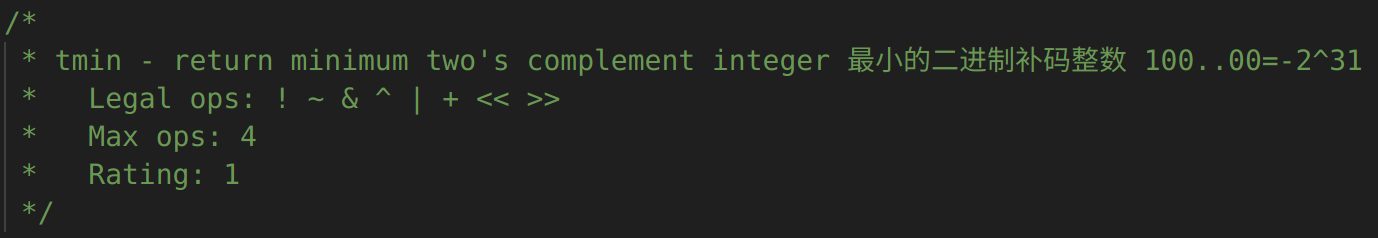
代码：



#### 2.2.6 tmin

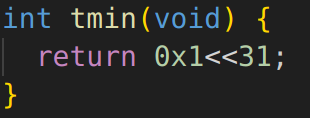
要求：

输出最小的二进制补码整数，即100...00=-2^31



分析：将1算术左移31位即可得到0x80000000。

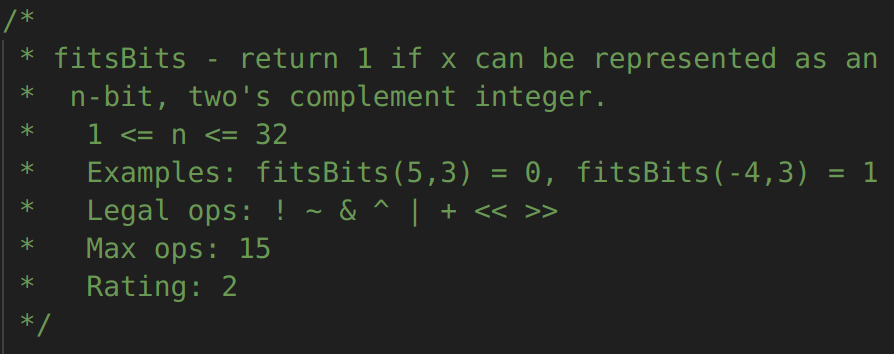
代码：



#### 2.2.7 fitsbits

要求：

判断x是否能被n位补码表示

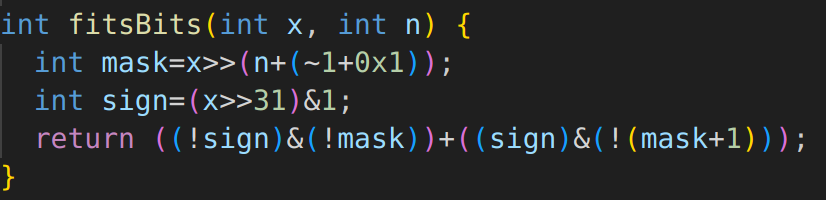


分析：我理解的就是x用补码表示后，要求除了符号位，只有n-1位含有1或0（对应就是前31-n位都是0或都是1）。这里我从正负分别判断，如果为正数，也就是要求前31-n位都是0；如果为负数，也就是要求前31-n位都为1，用mask=x>>n-1，去掉低n-1位。

若正数能表示，mask=0，（！sign）&（！mask)=1；

若负数能表示，mask=111..111，sign&（！(mask+1)）。

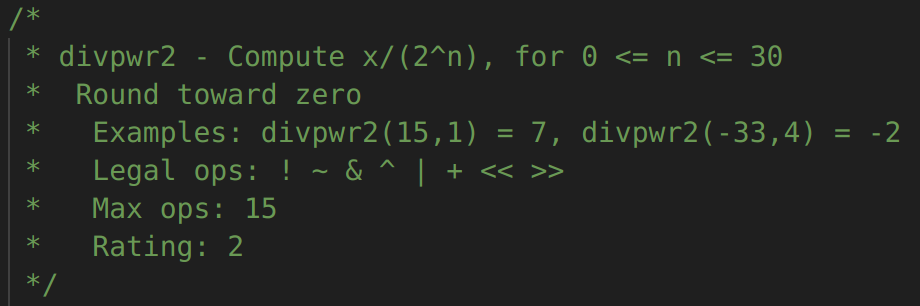
代码：



#### 2.2.8 divpwr2

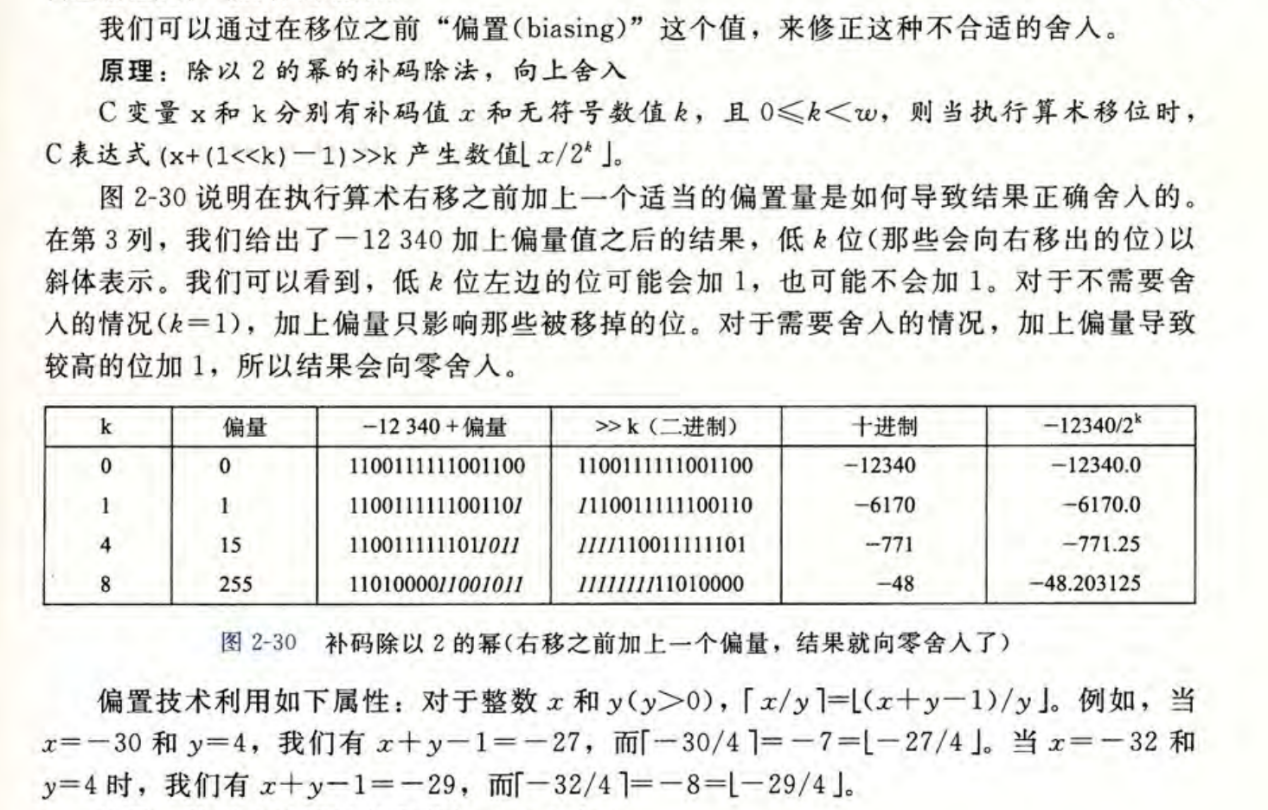
要求：

将x除以2^n。

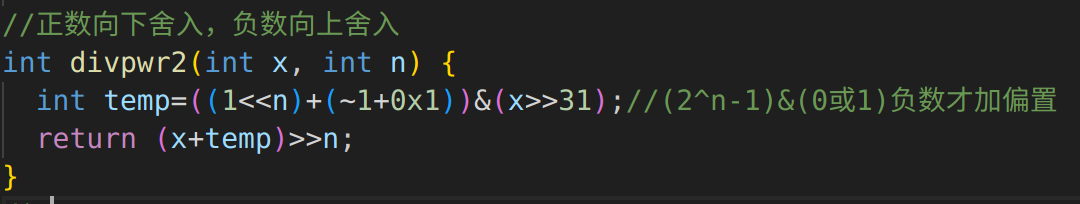


分析：由于只保留整数，样例中整数向下保存，负数向上保存，对补码表示的数字，直接算术右移的结果是向下舍入的，但是负数直接这样计算是不对的。为了保证结果向0舍入，负数需要加上偏置值。

偏置值为2^n-1,即n-1个1。



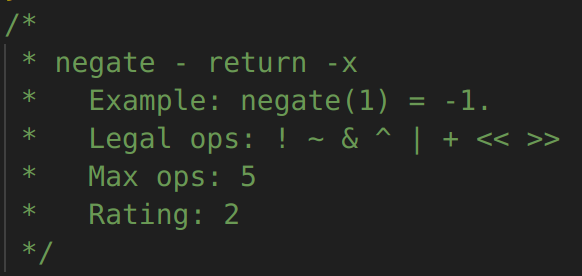
代码：



#### 2.2.9 negate

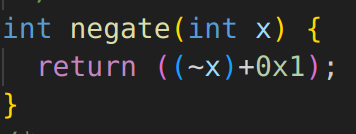
要求：

求相反数



分析：由于x以补码的形式存储，而-x的补码对于x的补码来说就是x按位取反再加1。

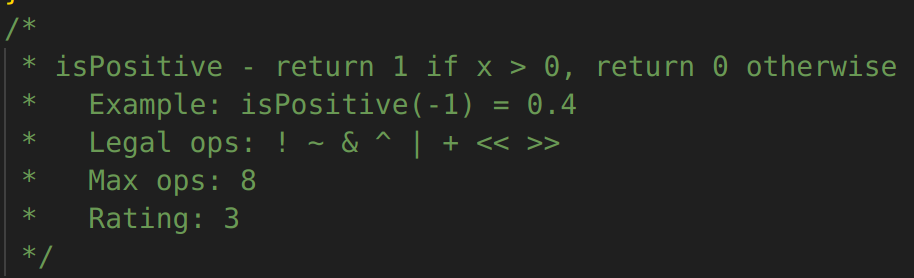
代码：



#### 2.2.10 isPositive

要求：

判断是否为正数



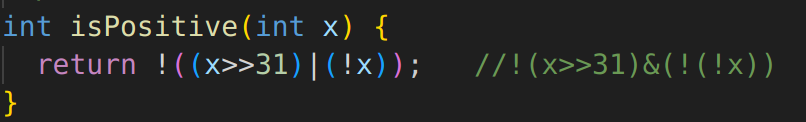
分析：根据符号位判断，但是由于0符号位也为0，所以当x=0，输出0；

当符号位为1，将符号位右移，使全为1或0。

我们需要的是当x>>31全为1或者x=0时，输出0。

那么通过式子!((x>>31)|(!x))可实现。

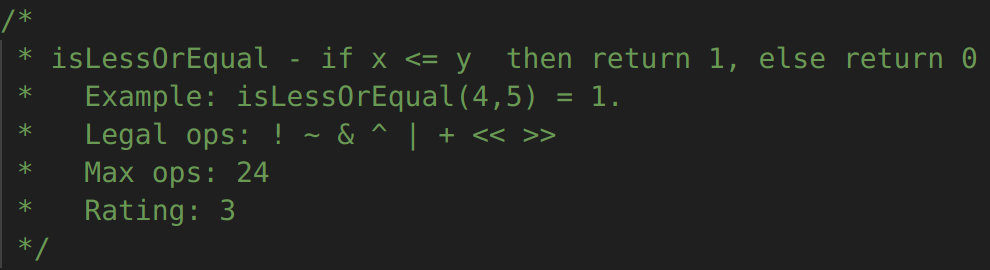
代码：



#### 2.2.11 isLessOrEqual

要求：

判断x<=y



分析：分成x与y同号，异号。

同号：y-x=y+(~x+0x1)符号位是0时，成立。

异号：x符号位为1时成立。

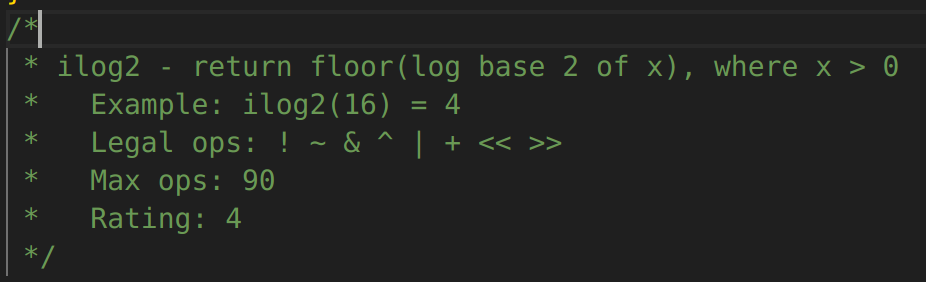
代码：



#### 2.2.12 ilog2

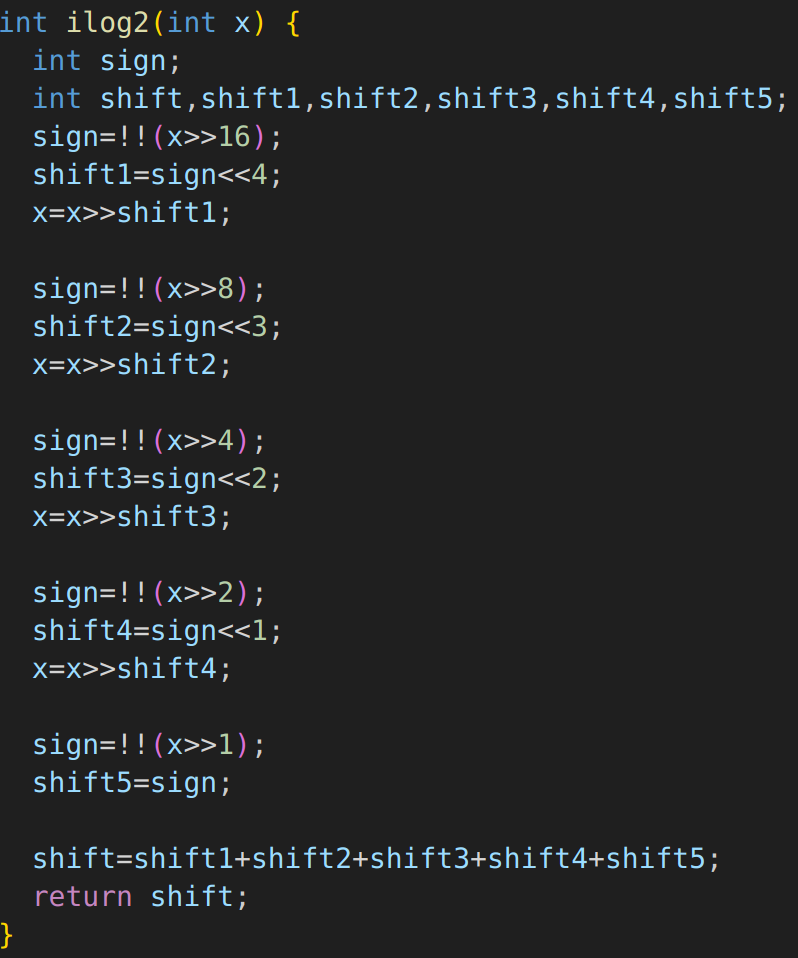
要求：

log2(n)<=x<=log2(n+1),求n



分析：即找x二进制表示的最高位1的位置。采用二分法，先看前16位，如果有，那么让位置加16，并将x右移16位，如果没有，那么就在后16位，看这16位的前8位，如果有，那么让位置加8，以此类推。

代码：

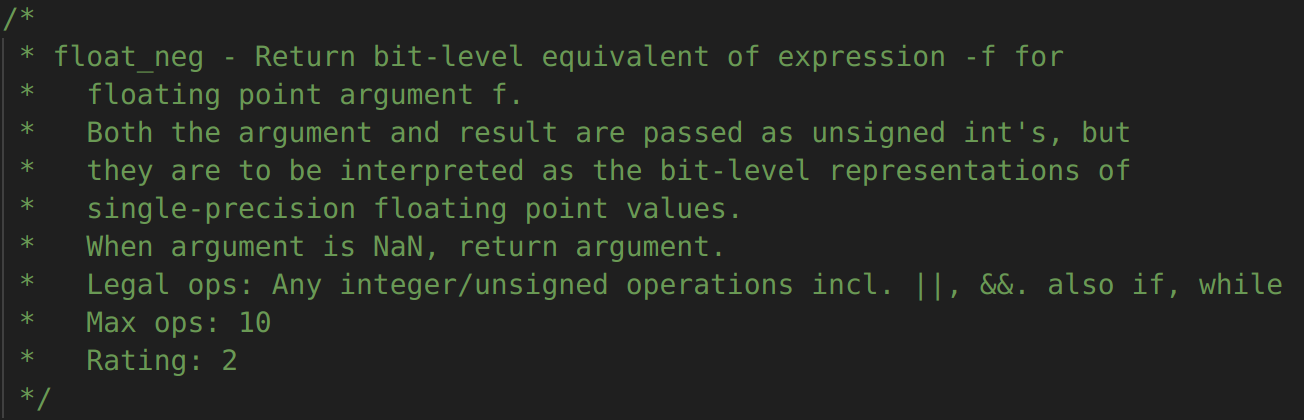


#### 2.2.13 float\_neg

要求：

求浮点数的相反数。

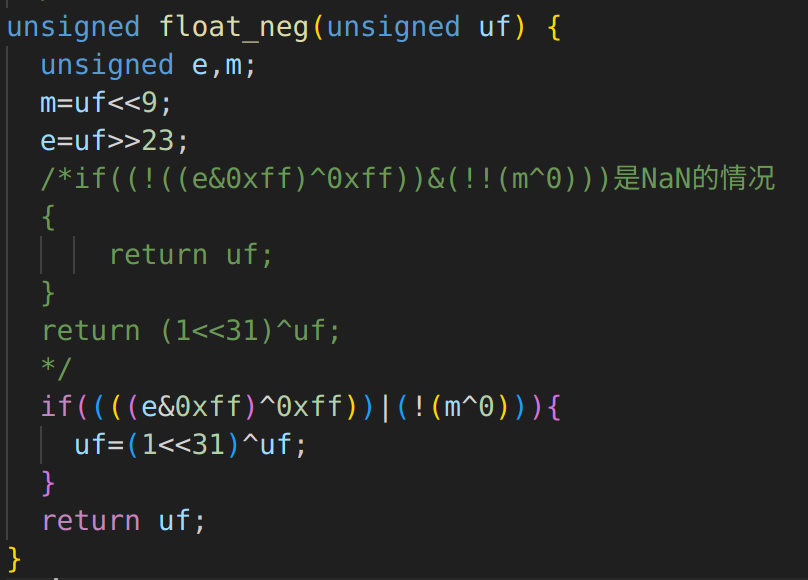
当uf=NaN时，返回uf。



分析：当浮点数是NaN,即阶码全为1，并且尾码不全为0，直接返回uf。否则，将符号位取反，然后返回即可。

阶码：(uf>>23)&0xff。

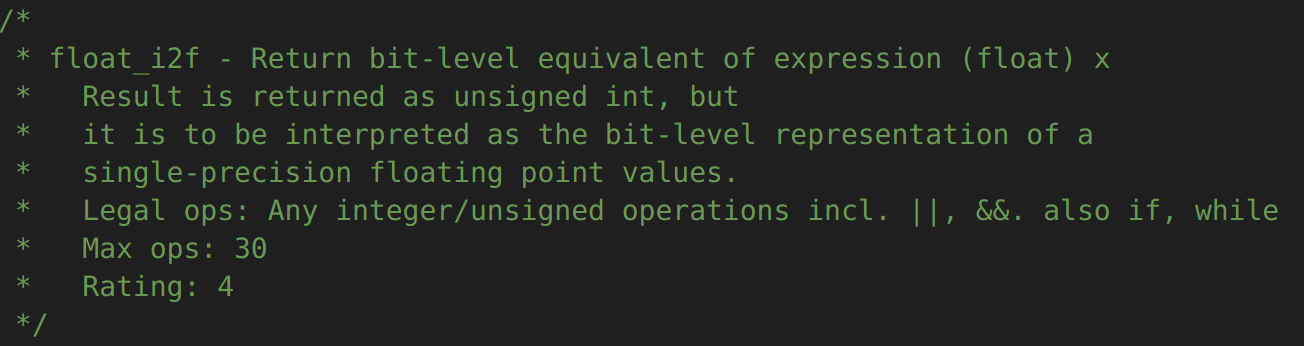
代码：



#### 2.2.14 float\_i2f

要求：

将整数转为浮点数的位级表示



分析：分符号位、阶码、尾码三部分求。

符号位略过。

阶码：由于转换形式为x=1.xxxxxx\*2^E，1为x二进制表示除符号位最高位的1，则求出1的位置后，可以间接求出E的值，关系num+E+1=32。

尾码：将1.xxxxxx中小数点后23位移到低23位。

虽然到这基本都求出来了，但是会出现舍入的情况。对于E>23时，需判断第23位是1或者0，判断23位以后形式是大于等于小于100000，如果大于，那么尾数加1；如果等于且23位为1，尾数加1；否则尾数不变。

对于尾数全为1，并且需要进位的情况，需要将阶码加1，并将尾数置0。

代码：

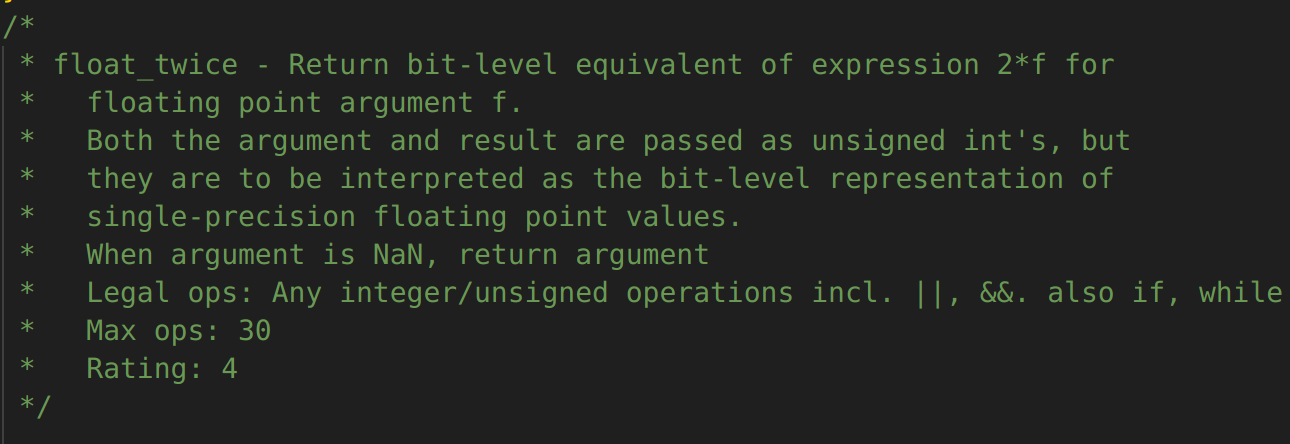


#### 2.2.15 float\_twice

要求：

将浮点数加倍。

对于uf为NaN、﹢∞、-∞（虽然题目只要求NaN，但是实际测试中+-∞也应该直接返回），需要返回原值。



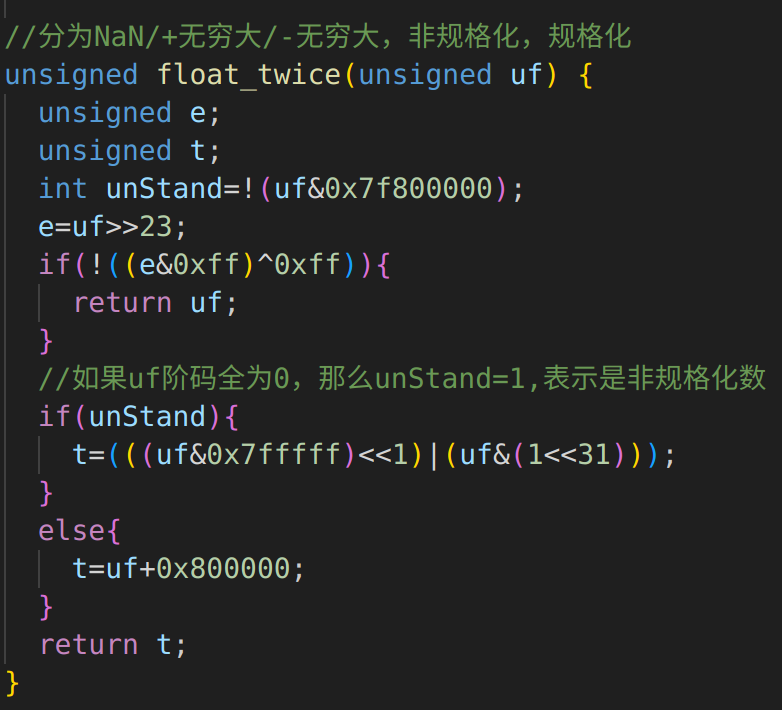
分析：将直接返回的特殊情况判断出来。

特殊情况条件：阶码位全是1。

非规格化：由于E不变，2^（1-bias）\*0.xxxxx，此时需要将尾数乘2，那么左移即可，而如果尾数第一位为1，左移后也不影响，因为变为规格化后，E仍为1-bias,其他不变即可。例如 0 00000000 1000…001 变成 0 00000001 000…0010，这样可以平滑地从非规格数过渡到规格化数。

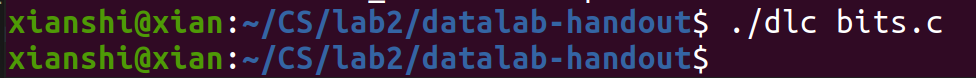
规格化：阶码加1即可。

代码：

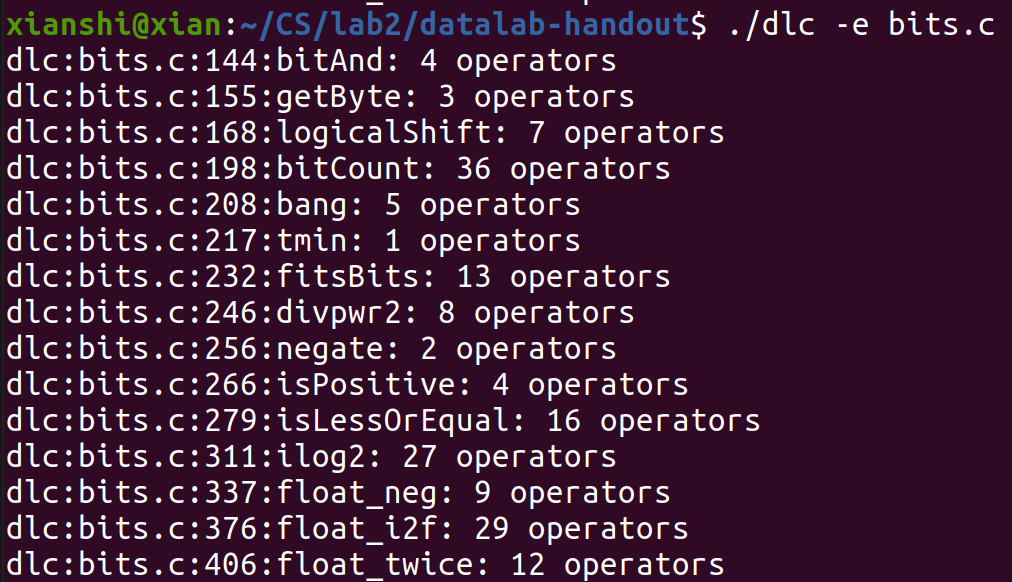


### 验证：

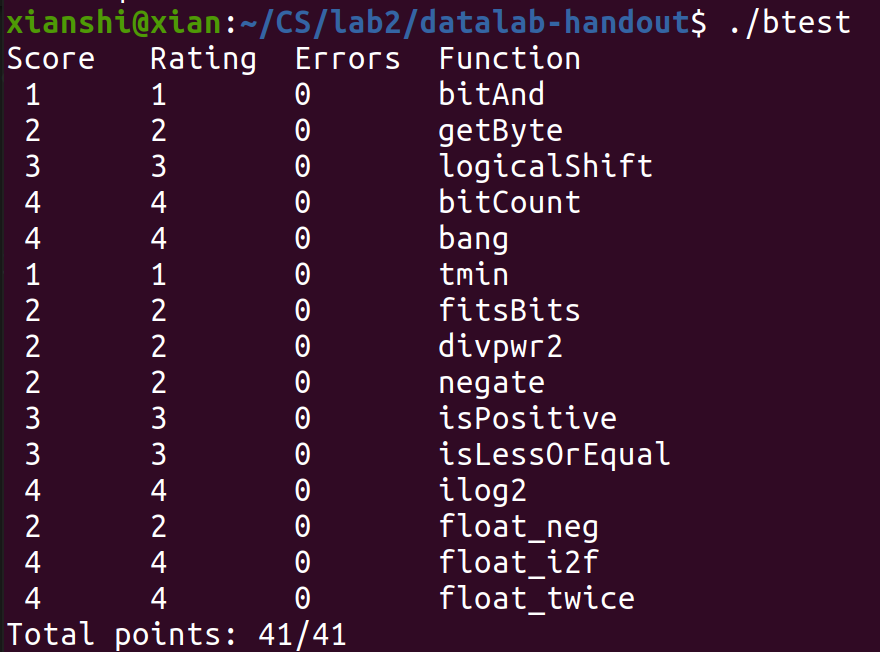
使用dlc对bits.c进行编译，编译通过，即书写合法



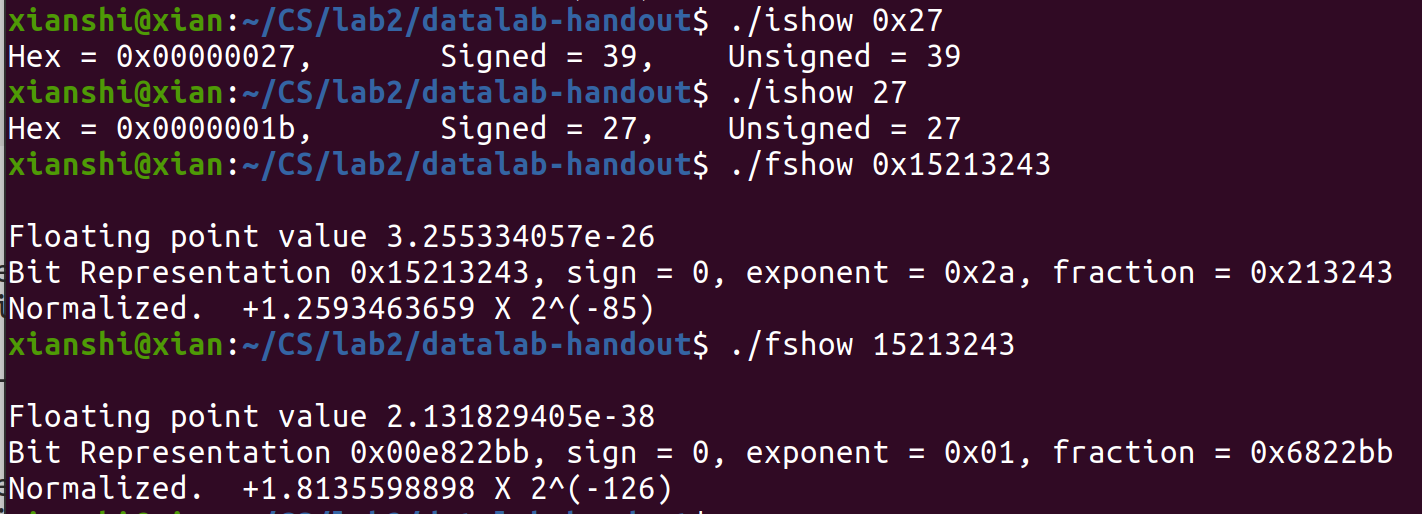
通过./dlc -e验证符号以及数量合法性：



通过make clean、make btest、./btest验证函数可行性：



可以使用./ishow和./fshow 十六进制来显示计算机中表示的整型或浮点型数值：



# **总结**

## 3.1实验中出现的问题

因为之前都是在用C语言等高级语言，其中很多符号和函数都是直接用，基本没想过它们的原理是什么，在进行bits.c文件完善时，对于使用一些限制的符号来实现一些功能不太理解，以及不能很好地完成。

1. 对于算数右移和逻辑右移理解的还不够透彻，导致一开始思路会出现错误，以为简单的左移右移就行，比如divpwr2函数。

对于无符号数，移位都是逻辑移位；对有符号数，移位是算术移位，左移补0，右移补符号位。

1. 在bitCount、ilog2函数卡住了，一直想不到应该用什么办法，最后是通过二分法解决的，现将大问题分成若干个类似的小问题，解决小问题，然后大问题也就解决了。
2. 有的函数出现了符号超出要求的问题，说明我的代码不够简化或者思路不太好。

## 3.2心得体会

通过datalab实验，我对于位运算、IEE浮点数表示有了更好的理解，并能够根据不同情况、数据，实现位运算。对于实验中遇到的问题，通过上网查资料、请教同学，我也基本能搞懂某个函数如何实现，基本原理是什么，以及用代码如何表示。

通过不断尝试，更改代码、变换思路、机器验证，最终还是把实验完成，很满足。

此外，我还总结了一些印象深刻的函数以及其中的部分技巧：

1. 要学会利用数学逻辑，比如德摩根律；
2. 减法可以通过对数取反然后加1实现；
3. 使用一些算法，比如二分法、分治法，可以先将问题分为若干相等的小问题，然后解决小问题后，大问题也就迎刃而解；
4. 分情况考虑问题，对于一些问题，可以将有共同点的情况分在一起，然后根据题目要求在进行对应的函数编写；
5. 提取某位可以通过构造掩码，然后将数与掩码取&；
6. 根据学过的知识仔细想想如何构建关系式，并且对于多种可能的关系式，选择最优项。