《计算机系统》

实验二报告

目录

[1 实验项目一 3](#_Toc115379316)

[1.1 项目名称 3](#_Toc115379317)

[1.2 实验目的 3](#_Toc115379318)

[1.3 实验资源 3](#_Toc115379319)

[2 实验任务 4](#_Toc115379320)

[2.1 实验任务A 4](#_Toc115379321)

[2.2 实验任务B 6](#_Toc115379322)

[2.3 实验任务C 12](#_Toc115379323)

[3 总结 14](#_Toc115379324)

[3.1 实验中出现的问题 14](#_Toc115379325)

[3.2 心得体会 14](#_Toc115379326)

# 实验项目一

## 项目名称

DataLab

## 实验目的

加强对位级运算的理解及熟练使用的能力

## 实验资源

“datalab-handout.rar”包中的代码文件和dlc工具

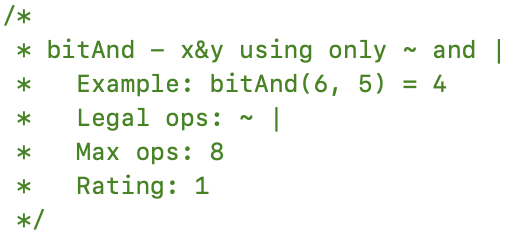
# **实验任务**

## **实验任务A**

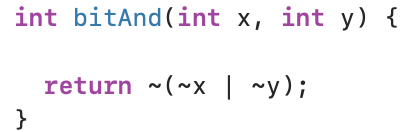
**任务名称：**修改bits.c，使其在不违反任何编码指南的情况下通过btest中的所有测试。

1. ****int bitAnd (int x, int y)****

**函数要求：只使用~（按位取反） 和 | （按位或）两个操作符实现按位与操作**

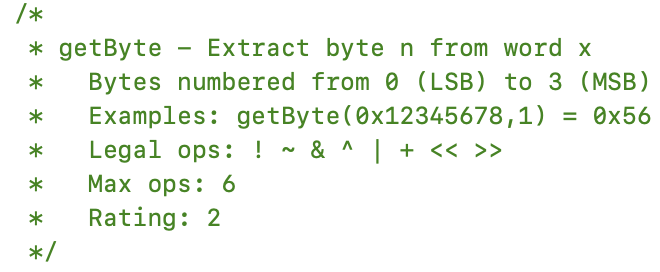
****

**算法思路：根据离散数学中德摩根律将 &（与） 转化为 |（或）和～（按位取反）**

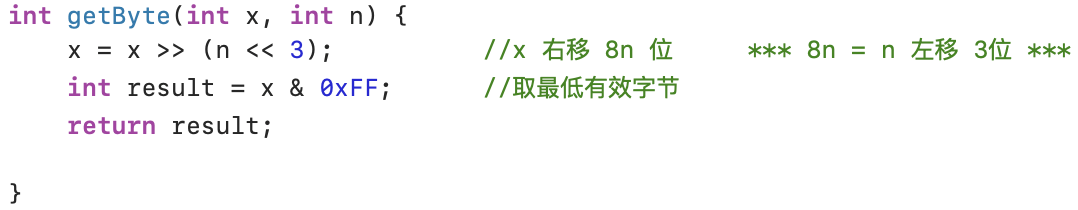
****

1. ****int getByte (int x, int n)****

**函数要求：从x中提取n个字节，x为32位二进制数，共四个字节，从高位到低位是3~0号字节**

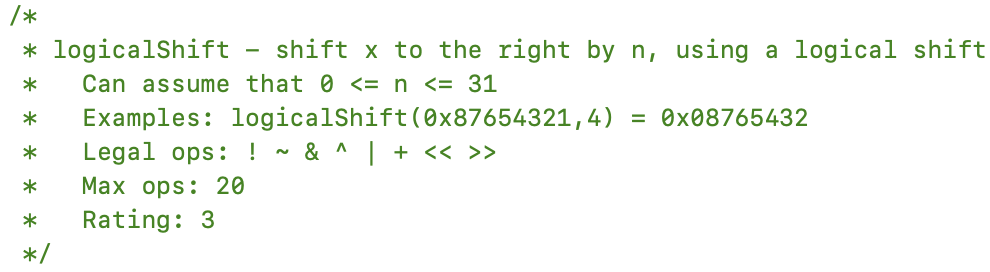
********

**算法思路：用右移操作将目的的字节移到最低字节，再用0xFF取得最低有效字节。右移是按一个字节一个字节移位，即移8\*n位。**

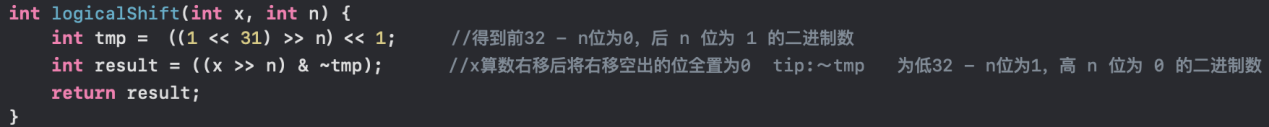
****

1. ****int logicalShift (int x, int n)****

**函数要求：实现逻辑右移**

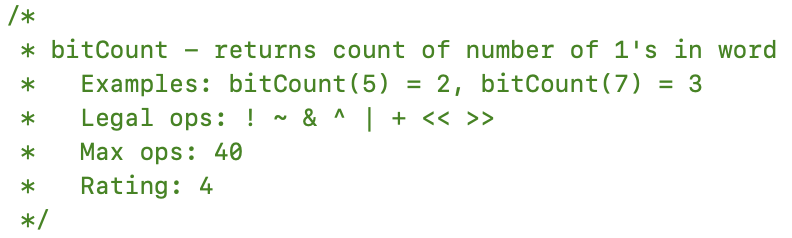
****

**算法思路：逻辑右移与符号位无关，移位后都用0补移位后空出的位，所以逻辑右移n位后需要把高n位与0相与，其余32-n位与1相与保持不变。即将>> (算数右移) 后得到的数与，0...011...11（高n位全为0，其余32-n为1）相与。**

****

1. ****int bitCount (int x)****

**函数要求：返回x二进制中1的个数**

****

**算法思路：将x的32位分成2位为一组，计算每两相邻两位1的个数，计算方法是与0x55555555相与，以8位二进制举例：**

**1101 0100**

**& 0101 0101**

**0101 0100**

**此时每相邻两位中先计算了低1位1的个数，将x右移，计算相邻两位高1位1的个数，二者相加就是相邻两位1的个数：**

**0110 1010**

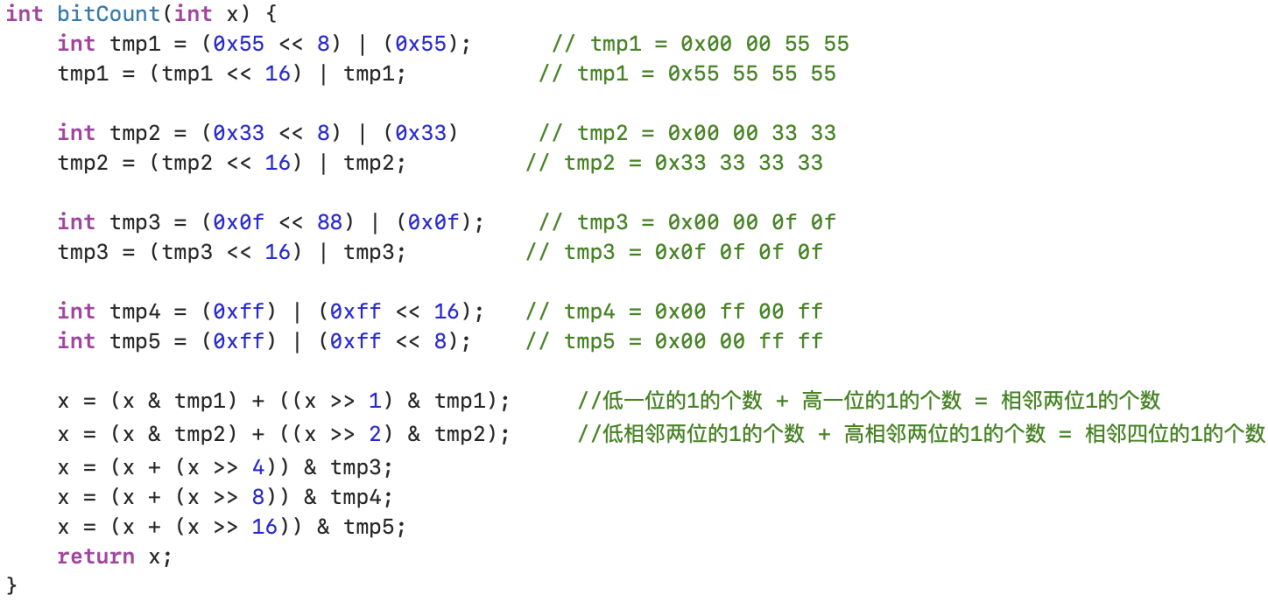
**& 0101 0101**

**0100 0000**

**+ 0101 0100**

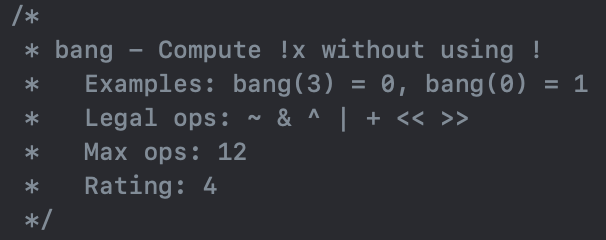
**1001 0100 （每两位表示对应相邻两位1的个数）**

**以此类推，计算每相邻4、8、16位1的个数，最终的结果就是32位中1的个数**

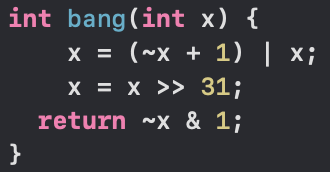
****

1. ****int bang (int x)****

**函数要求：不使用 ! 运算符计算 !x**

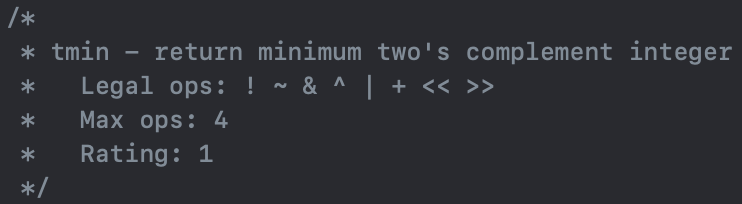
****

**算法思路：只有当x=0时，~x为全1，~x + 1为全0(逸出)，x|(~x+1)的最高位为0。只要x不为0，x和 ~x+1中一定有一个的最高位为1， 可以用反证法证明这一点。因此，~(x | (~x + 1))的最高位即为结果。只需要将~(x | (~x + 1))右移31位，与1按位与即可。**

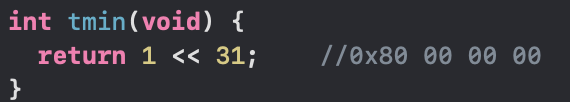
****

1. ****int tmin (int x)****

**函数要求：返回二进制补码的最小值**

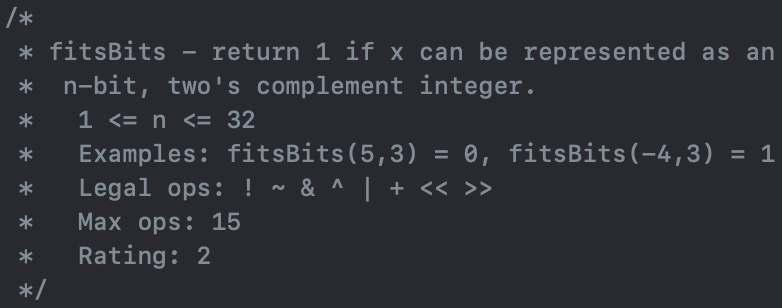
********

**算法思路：二进制补码表示的最小值为1000 0000 0000 0000 0000 0000 0000 0000，所以只要将1左移31位即可**

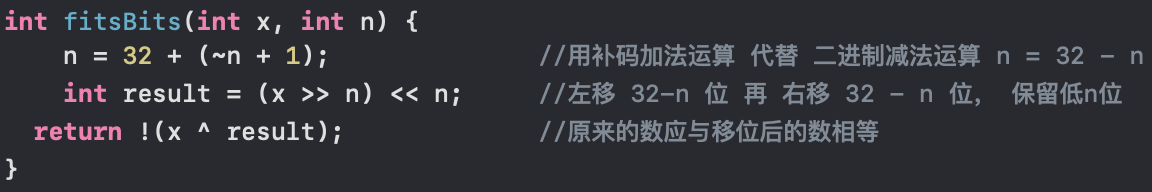
****

1. ****int fitsBits(int x, int n)****

**函数要求：如果x能表示成n位二进制补码就返回1，否则返回0**

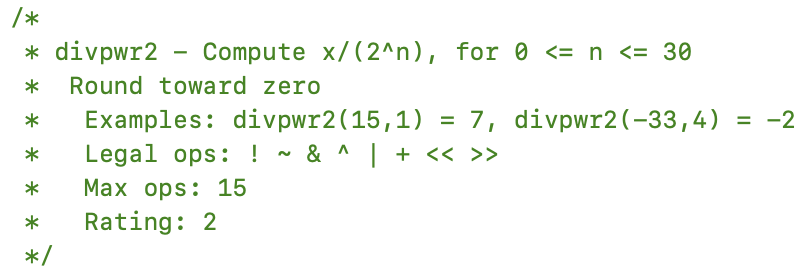
********

**算法思路：如果一个数能表示为n位二进制补码，说明这个数高32-n位是无意义的，所以这个数左移32位再右移32位后保留了低n位，应与没移位之前相同，所以最后异或取非，即可得知x是否超过了n位二进制补码所能表示的范围。**

********

1. ****int divpwr2(int x, int n)****

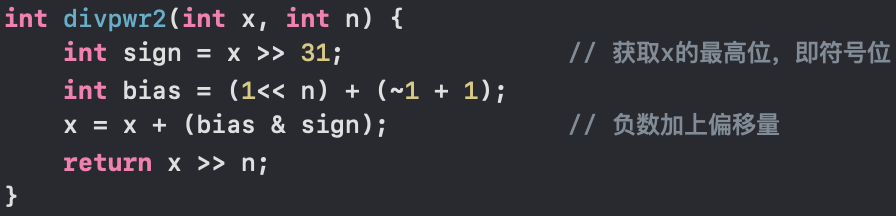
**函数要求：计算x / (2^n)，结果向0舍入。**

****

**算法思路： 对于x >= 0，只需要将x直接向右移n位即可；**

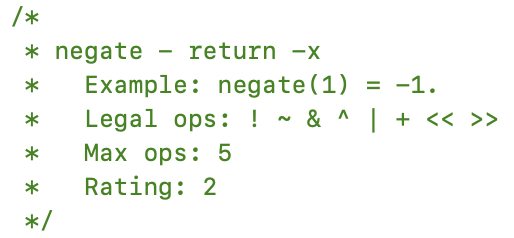
**对于x < 0, 如果直接右移是向负无穷舍入，所以需要加上偏移量2^n - 1 向0修正再右移n位。**

**所以首先要获取x的符号位判断它是正数还是负数决定是否加上偏移量。**

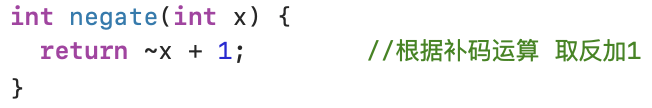
****

1. ****int negate(int x)****

**函数要求：不使用’-’操作符返回-x**

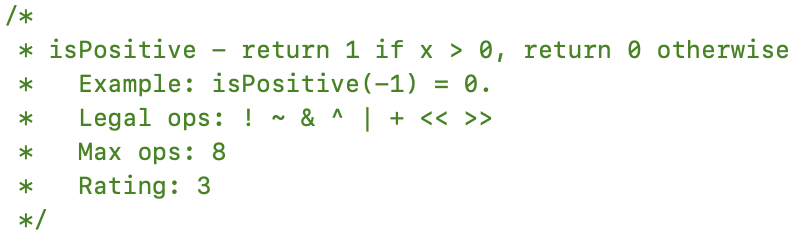
********

**算法思路：根据补码规则，-x就是x按位取反再+1。**

****

1. ****int isPositive(int x)****

**函数要求：判断x是否是正数，即如果x>0返回1，否则返回0。**

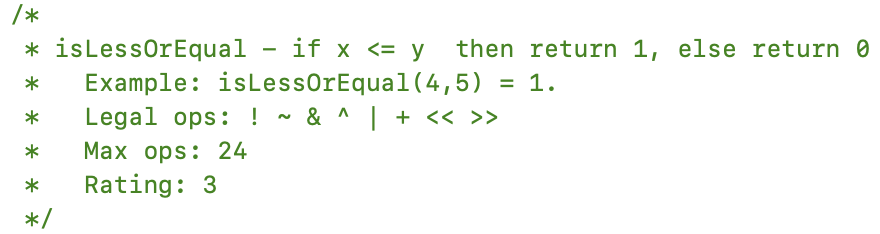
****

**算法思路：直接判断x的最高位符号位，正数符号位为0，负数为1，所以直接取非返回即可，但对0要特殊处理，因为0的最高位也是0，思路为两次取非，因为只有0两次取非后还是0。**

****

1. ****int isLessOrEqual(int x, int y)****

**函数要求：不使用比较运算符判断两个参数x和y的大小，如果x <= y 则返回1，否则返回0。**

****

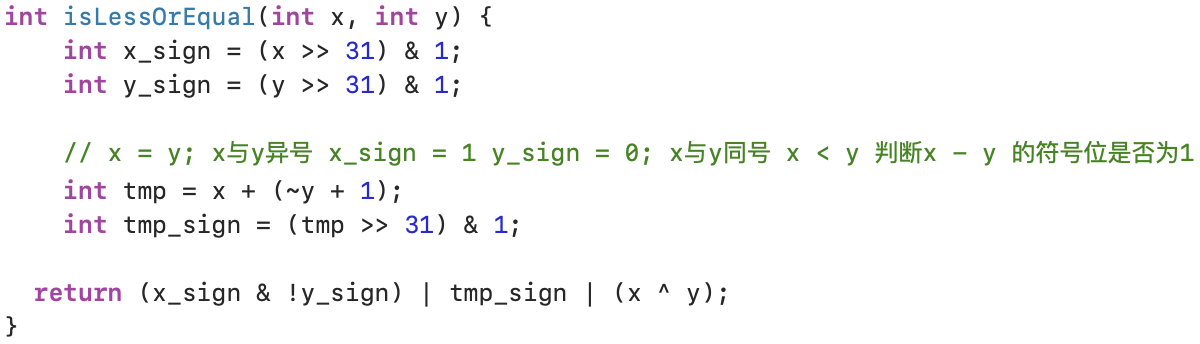
**算法思路：**

**x与y二者有两种情况：**

1. **x与y异号：计算y-x，此时应返回结果(正数)的最高位符号位取非；**
2. **x与y同号：此时返回0，即返回二者符号位相与；**

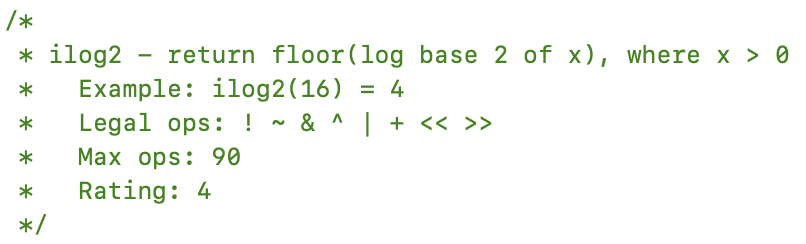
**最后处理二者相等的情况，此时应返回1，所以直接返回二者按位异或的结果**

**最后返回两种情况的结果相或即可。**

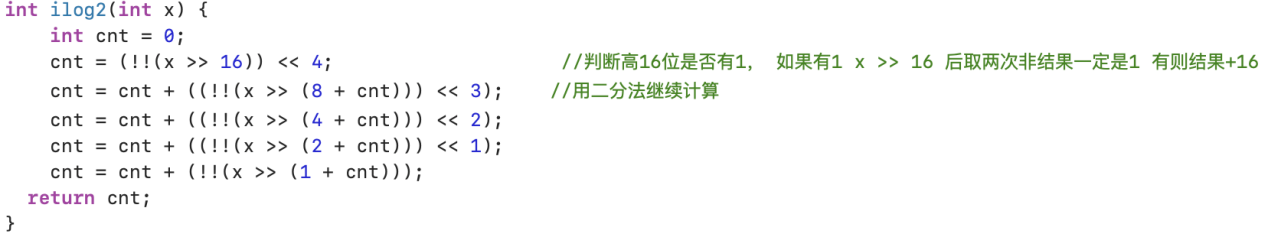
****

1. ****int ilog2(int x)****

**函数要求：计算返回log2x,输入保证x > 0。**

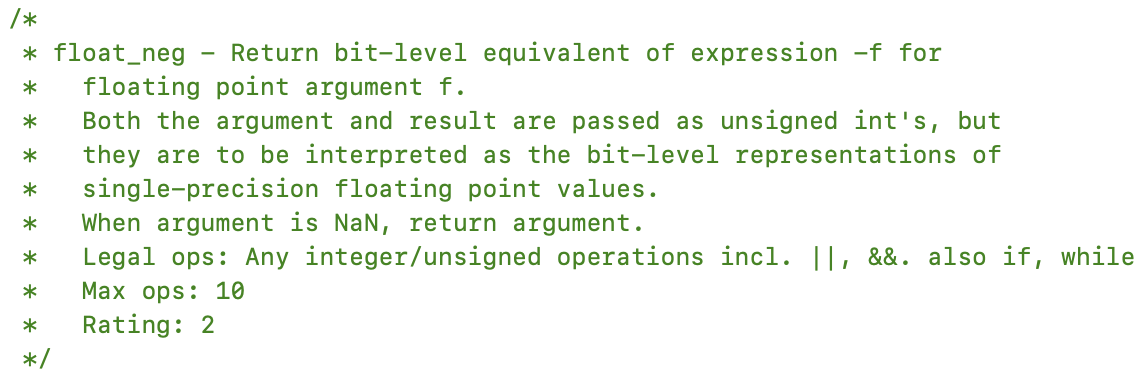
****

**算法思路：log2x的结果就是x二进制表示中最高位的1的位置，所以目标在于找到最高位的1，可采用二分法计算，先判断高16位中是否有1，有则最高位的1从16位开始，有则继续从高8位判断，没有则从低16位判断，以此类推。**

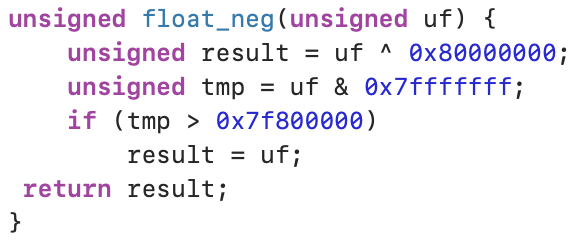
****

1. ****unsigned float\_neg(unsigned uf)****

**函数要求：返回浮点数-uf**

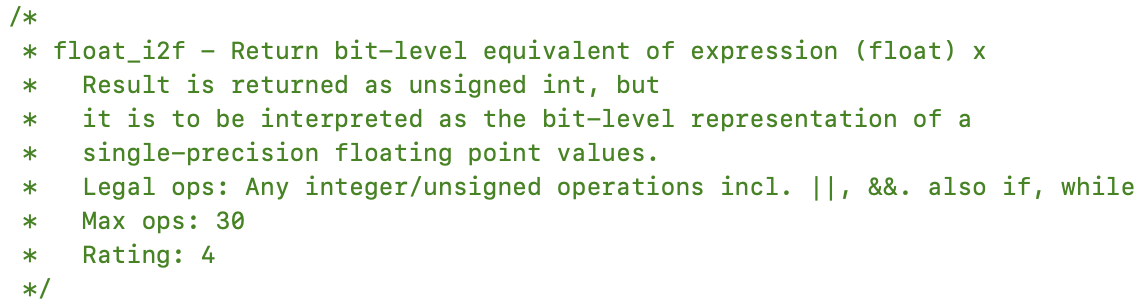
****

**算法思路：浮点数的二进制最高位也是符号位，只需将符号位按位取反即可，即用最高位与1按位异或，其他位与0按位异或保持不变。但需要特殊处理传入的参数是NaN，此时直接返回即可。**

****

1. ****unsigned float\_i2f(int x)****

**函数要求：将整型二进制转换为浮点数二进制表示。**

****

**算法思路：浮点数二进制有三个部分：符号位、exp、尾数位**

**符号位可通过x的符号位确定；**

**exp = E + 127，其中E是阶码，可通过右移找到最高位的1确定；**

**尾数位是剩下23位；**

**①首先判断符号位，如果x是负数则转换成正数；**

**②然后循环左移找到最高位的1，记录左移的次数shiftLeft，最后退出循环时得到的是32位尾数**

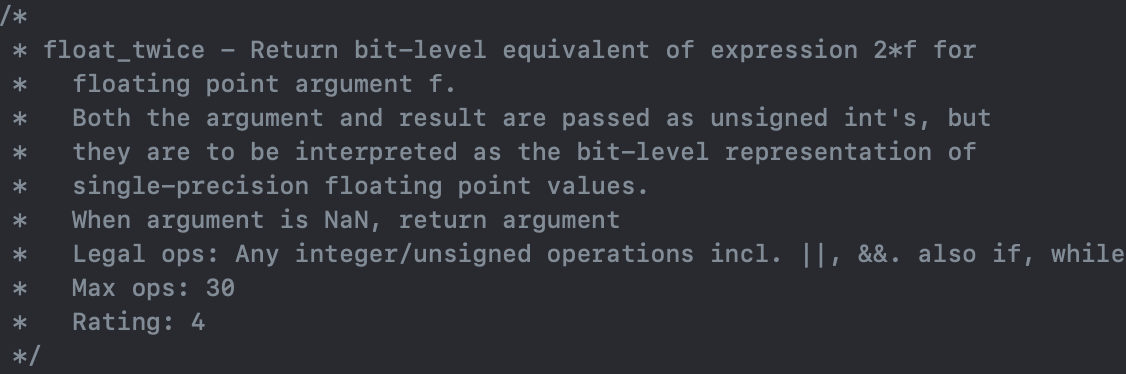
**③最后符号位sign+（127 + （32 - shiftLeft)) + frac就是结果**

**④但要注意的是对尾数的向偶数舍入，因为最后结果尾数右移了9位空出给符号位和exp，会缺失部分精度。**

****

1. ****unsigned float\_twice(unsigned uf)****

**函数要求：返回2\*f，f是浮点数**

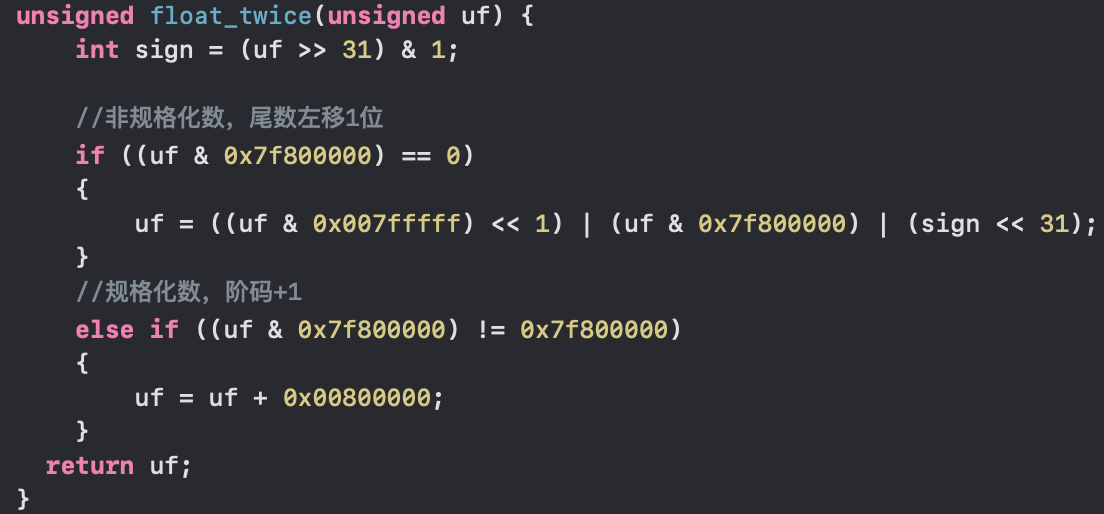
****

**算法思路：对于规格化数，阶码+1即可；**

**对于非规格化数，尾数左移1位。**

**其中非规格化数exp = 0000 0000**

**规格化数 exp != 0000 0000 且 exp != 1111 1111**

****

# **总结**

## 实验中出现的问题

1. 使用x86\_64位的Ubuntu运行dlc时，会出现“No such file or dictionary”的报错；

解决办法：用file dlc查看文件属性可知dlc时32位elf，64位系统没有32位兼容库所以不能直接运行，最终解决办法是执行以下命令安装32位兼容库

sudo dpkg --add-architecture i386

sudo apt-get update

sudo apt-get install libc6:i386 libncurses5:i386 libstdc++6:i386

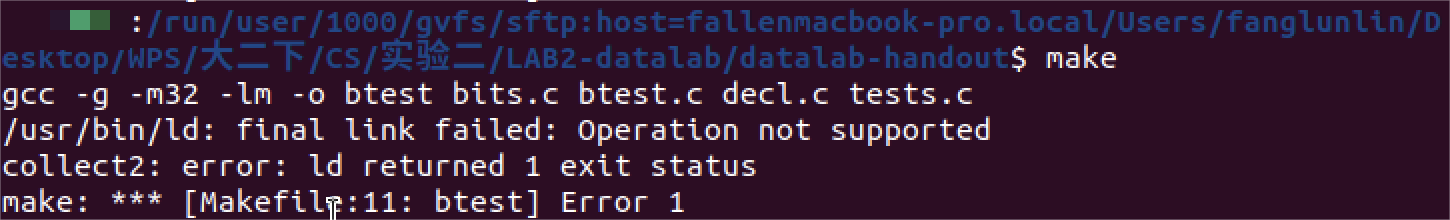
sudo apt install lib32z1

1. 运行./dlc bits.c时出现多处声明变量的语法错误，根据bits.c里的编码风格样例比较没有发现问题。



解决办法：dlc要求将报错的变量声明写在函数声明前。另外有些变量在函数体内声明也不会出现类似错误，请教老师了解到是由于dlc本身有问题。

1. 最终make btest时链接失败，报错如下：



报错显示操作不支持。

解决办法：虚拟机已经安装了32位兼容库，所以排除系统架构不匹配的原因。上网查询得到类似情况，可能是因为直接在虚拟机连接到服务器后用虚拟机本地终端打开共享文件夹链接会失败，所以最终直接拷贝一份文件到虚拟机再链接即可。

https://stackoverflow.com/questions/43312441/cant-use-gcccompile-on-files-in-network-drive

## 心得体会

1. 熟悉了位的基本操作，更进一步熟悉了数的机器表达形式以及特点。熟悉了异或取反判等，移位和0x1与得符号位等基本的位操作，通过使用一些数的特性，算数右移左移操作特点进一步熟练了整数和浮点数在位级的表示与一些运算方法。
2. 遇到问题学会自己分析并上网浏览资料解决，像是链接失败的问题。一些函数实现参考了网上的思路，自己认真学习思考后印象也更深刻。
3. 完成实验二后真正体会到计算机系统的难度，如果不自己动手去学习和实践的话很多问题仅靠看解答是不能理解的，且只有自己上手了印象才更加深刻，像本次实验用到的许多位操作和一些算法，一定要动手将思路实践了才能更好记忆。