**课 程 实 验 报 告**

**课程名称： 计算机组成原理**

**专业班级： 软件工程1809**

**学 号： 8002118240**

**姓 名： 杨孟衡**

**指导教师： 胡勇**

**报告日期： 2019.9.18 – 10 .09**

# 实验1： 数据表示

## 1.1 实验概述

实验目的：更好地熟悉和掌握计算机中整数和浮点数的二进制编码表示。

实验目标：加深对数据二进制编码表示的了解。

实验要求：使用有限类型和数量的运算操作实现一组给定功能的函数。

实验语言：c

实验环境：linux、gcc

实验资料：datalab-handout.tar.gz

实验中,你需要解开一系列编程“难题”——**使用有限类型和数量的运算操作实现一组给定功能的函数**。

可用的运算符：“! ~ & ^ | + << >>”

|  |  |  |
| --- | --- | --- |
| 运算符 | 描述 | 实例 |
| ! | 逻辑非运算符。用来逆转操作数的逻辑状态。如果条件为真则逻辑非运算符将使其为假。 | A && B 为假，则  !(A && B) 为真。 |
| ~ | 按位取反运算符。是一元运算符，具有"翻转"位效果，即0变成1，1变成0。 | ~60 为 -61 |
| & | 按位与运算符。如果同时存在于两个操作数中，二进制 AND 运算符复制一位到结果中。 | 60 & 13 为 12 |
| ^ | 按位异或运算符。如果存在于其中一个操作数中但不同时存在于两个操作数中，二进制异或运算符复制一位到结果中。 | 60 ^ 13 为 49 |
| | | 按位或运算符。如果存在于任一操作数中，二进制 OR 运算符复制一位到结果中。 | 60 | 13 为 61 |
| + | 把两个操作数相加 |  |
| << | 二进制左移运算符。左操作数的值向左移动右操作数指定的位数。 | 60 << 2 为240 |
| >> | 二进制右移运算符。左操作数的值向右移动右操作数指定的位数。 | 60 >> 2 为15 |

## 1.2 实验内容

需要完成**实验资料**datalab-handout.tar.gz中bits.c 的函数功能,具体分为三大类:位操作、补码运算和浮点数操作。

1）位操作

表1列出了bits.c中一组操作和测试位组的函数。其中，“级别”栏指出各函数的难度等级（对应于该函数的实验分值），“功能”栏给出函数应实现的输出（即功能），“约束条件”栏指出你的函数实现必须满足的编码规则（具体请查看bits.c中相应函数注释），“最多操作符数量”指出你的函数实现中允许使用的操作符的最大数量。

表1 位操作题目列表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 级别 | 函数名 | 功能 | 约束条件 | 最多操作符数 |
| 1 | lsbZero | 将x的最低有效位（LSB）清零 | 仅能使用! ~ & ^ | + << >> | 5 |
| 2 | byteNot | 将x的第n个字节取反（字节从LSB开始到MSB依次编号为0-3） | 仅能使用! ~ & ^ | + << >> | 6 |
| 2 | byteXor | 比较x和y的第n个字节（字节从LSB开始到MSB依次编号为0-3），若不同，则返回1；若相同，则返回0 | 仅能使用! ~ & ^ | + << >> | 20 |
| 3 | logicalAnd | x&& y | 仅能使用! ~ & ^ | + << >> | 20 |
| 3 | logicalOr | x|| y | 仅能使用! ~ & ^ | + << >> | 20 |
| 3 | rotateLeft | 将x循环左移n位 | 仅能使用! ~ & ^ | + << >> | 25 |
| 4 | parityCheck | 若x有奇数个1，则返回1；否则，返回0 | 仅能使用! ~ & ^ | + << >> | 20 |

2）补码运算

表2列出了bits.c中一组使用整数的补码表示的函数。可参考bits.c中注释说明和tests.c中对应的测试函数了解其更多具体信息。

表**2** 补码运算题目列表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 级别 | 函数名 | 功能 | 约束条件 | 最多操作符数 |
| 2 | mul2OK | 计算2\*x，如果不溢出，则返回1，否则，返回0 | 仅能使用~ & ^ | + << >> | 20 |
| 2 | mult3div2 | 计算(x\*3)/2，朝零方向取整 | 仅能使用! ~ & ^ | + << >> | 12 |
| 3 | subOK | 计算x –y，如果不溢出，则返回1，否则，返回0 | 仅能使用! ~ & ^ | + << >> | 20 |
| 4 | absVal | 求x的绝对值 | 仅能使用! ~ & ^ | + << >> | 10 |

3）浮点数操作

表3列出了bits.c中一组浮点数二进制表示的操作函数。可参考bits.c中注释说明和tests.c中对应的测试函数了解其更多具体信息。注意float\_abs的输入参数和返回结果（以及float\_f2i函数的输入参数）均为unsigned int类型，但应作为单精度浮点数解释其32 bit二进制表示对应的值。

表3 浮点数操作题目列表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 级别 | 函数名 | 功能 | 约束条件 | 最多操作符数 |
| 2 | float\_abs | 返回浮点数‘|f|’的二进制表示，当输入参数是NaN时，返回NaN | 仅能使用任何整型/无符号整型操作，包括||，&&以及if，while控制结构 | 10 |
| 4 | float\_f2i | 返回浮点数‘f’的强制整型转换“(int)f”表示 | 仅能使用任何整型/无符号整型操作，包括||，&&以及if，while控制结构 | 30 |

## 1.3 实验设计

根据题目要求，选择合适的操作符来编写代码。

注意事项：

1、实验资料包中包含以下文件：

Makefile - Makes btest, fshow, and ishow

README - This file

**bits.c - The file you will be modifying and handing in**

bits.h - Header file

btest.c - The main btest program

btest.h - Used to build btest

decl.c - Used to build btest

tests.c - Used to build btest

tests-header.c- Used to build btest

dlc\* - Rule checking compiler binary (data lab compiler)

driver.pl\* - Driver program that uses btest and dlc to autograde bits.c

Driverhdrs.pm - Header file for optional "Beat the Prof" contest

fshow.c - Utility for examining floating-point representations

ishow.c - Utility for examining integer representations

2、bits.c编程要求：

将每个函数中的“返回”语句替换为一个或者更多的C代码行来实现函数。你的代码必须符合以下风格：

int Funct(arg1, arg2, ...) {

/\* brief description of how your implementation works \*/

int var1 = Expr1;

...

int varM = ExprM;

varJ = ExprJ;

...

varN = ExprN;

return ExprR;

}

比如：

/\*

\* pow2plus1 - returns 2^x + 1, where 0 <= x <= 31

\*/

int pow2plus1(int x) {

/\* exploit ability of shifts to compute powers of 2 \*/

return (1 << x) + 1;

}

/\*

\* pow2plus4 - returns 2^x + 4, where 0 <= x <= 31

\*/

int pow2plus4(int x) {

/\* exploit ability of shifts to compute powers of 2 \*/

int result = (1 << x);

result += 4;

return result;

}

其他要求：

Each "Expr" is an expression using ONLY the following:

1. Integer constants 0 through 255 (0xFF), inclusive. You are

not allowed to use big constants such as 0xffffffff.

2. Function arguments and local variables (no global variables).

3. Unary integer operations ! ~

4. Binary integer operations & ^ | + << >>

Some of the problems restrict the set of allowed operators even further.

Each "Expr" may consist of multiple operators. You are not restricted to

one operator per line.

You are expressly forbidden to:

1. Use any control constructs such as if, do, while, for, switch, etc.

2. Define or use any macros.

3. Define any additional functions in this file.

4. Call any functions.

5. Use any other operations, such as &&, ||, -, or ?:

6. Use any form of casting.

7. Use any data type other than int. This implies that you cannot use arrays, structs, or unions.

3、在完成函数时，可以用以下方式检查语法及使用的操作符数量：

**检查语法：**

unix> ./dlc bits.c

如果没有返回，表示没有语法错误，否则会提示错误位置及相关信息。

**统计运算符数量：**

unix> ./dlc -e bits.c

4、如果需要检查程序运行的合规，可以通过运行btest来检查。

在每次修改bits.c后，都需要重新编译btest

unix> make btest

unix> ./btest [optional cmd line args]

如果需要完全重新编译，可以先清除，当然如果文件有修改，一般会自动判断，不需要单独清除。

unit>make clean

btest程序的用法：

unix> ./btest -h

Usage: ./btest [-hg] [-r <n>] [-f <name> [-1|-2|-3 <val>]\*] [-T <time limit>]

-1 <val> Specify first function argument

-2 <val> Specify second function argument

-3 <val> Specify third function argument

-f <name> Test only the named function

-g Format output for autograding with no error messages

-h Print this message

-r <n> Give uniform weight of n for all problems

-T <lim> Set timeout limit to lim

Examples:

Test all functions for correctness and print out error messages:

unix> ./btest

Test all functions in a compact form with no error messages:

unix> ./btest -g

Test function foo for correctness:

unix> ./btest -f foo

Test function foo for correctness with specific arguments:

unix> ./btest -f foo -1 27 -2 0xf

5、另外，资料包中提供了两个工具：ishow和fshow。

unix> ./ishow 0x27

Hex = 0x00000027, Signed = 39, Unsigned = 39

unix> ./ishow 27

Hex = 0x0000001b, Signed = 27, Unsigned = 27

unix> ./fshow 0x15213243

Floating point value 3.255334057e-26

Bit Representation 0x15213243, sign = 0, exponent = 0x2a, fraction = 0x213243

Normalized. +1.2593463659 X 2^(-85)

linux> ./fshow 15213243

Floating point value 2.131829405e-38

Bit Representation 0x00e822bb, sign = 0, exponent = 0x01, fraction = 0x6822bb

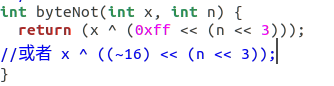
Normalized. +1.8135598898 X 2^(-126)

## 实验过程

lsbZero代码：先将真值1按位取反得到一个除最低位为0的机器数，再用这个机器数和x做与运算即可实现最低位置零的目的；



byteNot代码：第一种方式是第二种方式的优化版本，本关要点：通过对一个二进制数按位异或可以实现求反的效果，所以先根据n的值取到相应的字节位置，然后再按位异或；



byteXor代码：根据第二题的方法先行定位比较的字节，再通过按位与的方式，使机器数除了比较的位数通通为0，目的是排除其他位对比较的影响（因为之后使用按位异或比较，如果其他位不为零会影响最终结果），最后确定比较字节位置的x和y按位异或，若相同则结果为0，然后！两次可得到0，若不相同，！一次为0，再！一次为1；



logicalAnd代码：利用！运算符的特殊性，任何非零值（真）求！后都为0，而0求！后为1，所以利用这个特性，若x，y都是非零值那么求！后为0，所以按位或以后为0，求！后返回1，表示真。若x，y中只有一个不为0，那么另一个数为0求！后为1，所以按位或运算后为1，求！后返回0，表示假。同理，两个数都为0时，求！后都为1，再按位或操作得1，求！后返回0表示假；



logicalOr代码：利用！特性，与上题逻辑与基本相似，只不过！x与！y之间采用与运算符即可达到逻辑或的效果；



rotateLeft代码：分析如何实现循坏左移，循环左移需要保留左移出去那几位数据，所以首先需要实现的就是如何保留测试数据给出的左移位数（n）左移出去的那几位数据，保留数据之后，让这个数据的高（32-n）位置0，又根据左移的特点，左移出去后低位通通补0，所以保留了左移出去的数据之后再按位异或就可以实现循环左移了！

代码解释：左侧括号是机器数左移对应的n位，第一个异或右侧的大括号表示保留左移数据的过程，具体如下。

首先将x右移（32 – n）位，这里需要注意的就是计算机中的减法问题，可转换为加法(算法原理：计算机内部机器数是补码表示，所以一个数的求补操作相当于求它的相反值），具体公式：a – b = a + (~b + 1) = a + 1 + ~b;所以32 – n 相当于 32 + 1 + ～n；右移后低n位使我们左移出去的数据段，接着最后一步需要让这个数据的高32 – n 位置0，采用按位与的方法，既然是与，所以需要准备一个与操作的机器数，这时候就想到一个高32-n位全为0，低n位全为1的机器数，实现起来就是用0的求反操作，0求反一次为全1.然后利用左移的特性（左移多少位，低位补多少0），左移n位，这样就可以得到一个高32 – n位全为1，低n位全为0的机器数，再求反一次，即可得到前文提到的位与操作准备的机器数。

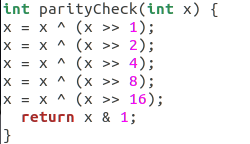
总结，第一个异或括号左侧是左移后的所求机器数副本，右侧是保留左移出去的n位数据的操作，所以两者按位异或即可实现循环左移。

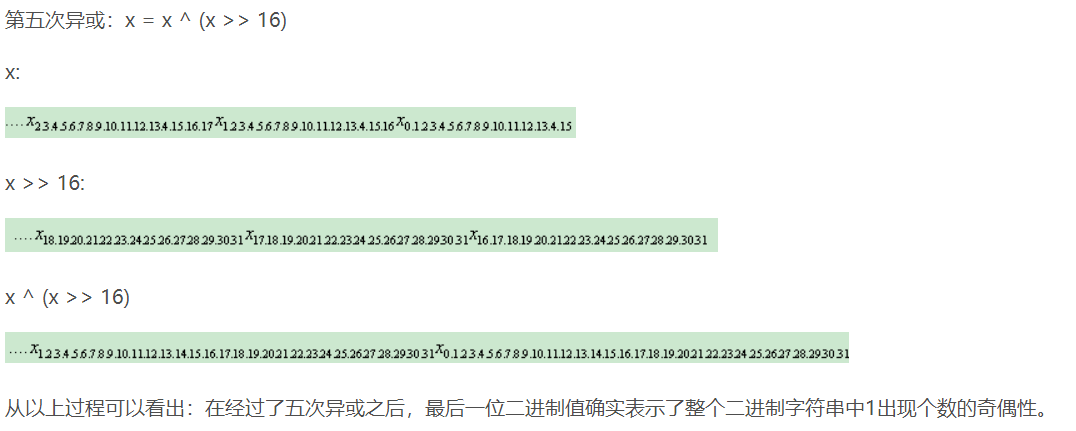
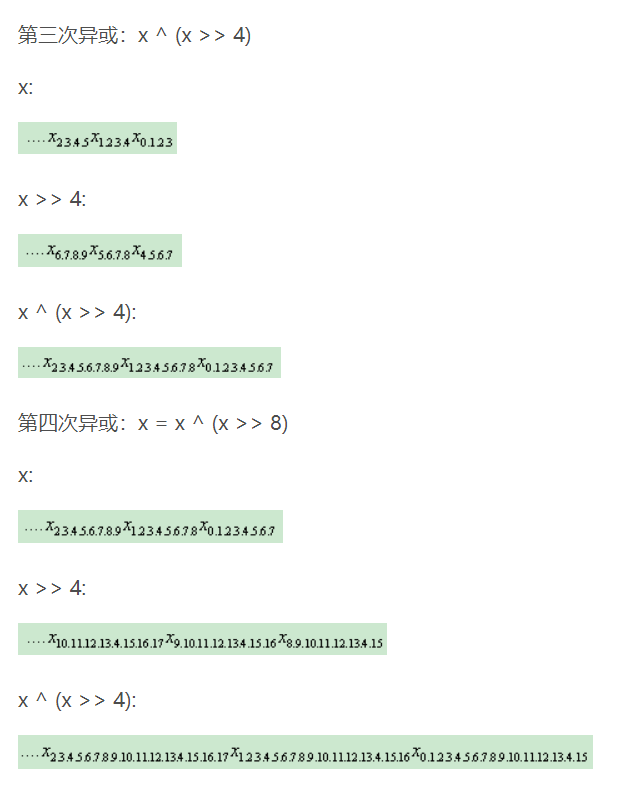
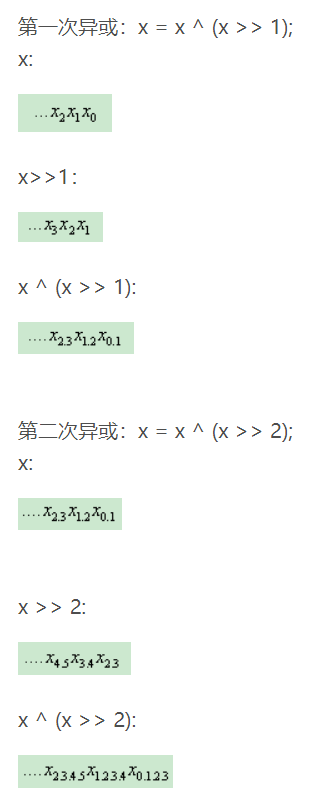


parityCheck代码：利用异或操作的特性，若两个机器数异或操作时同一位中有奇数个1则结果为1，若不是则为0的特点进行设计实现。意思就是说两个机器数异或一次的结果，第i位其实代表了之前两个机器数第i位和第i+1位上奇数个1的结果，若是奇数个位，1不是则为0.

具体代码实现详细阐述：判断一个机器数有无奇数个1，只需要不断右移1位， 2 位， 4位，…..直到机器数的总位数的一半.每次右移之后都要和原来的机器数进行按位异或操作，

这样当进行到最后一次右移异或操作后，最低位就是我们需要用到的最终判别数据位，所以将它与1按位与操作，然后返回即可，若为奇数个1，那么返回1，不是返回0.

parityCheck代码实例如下：



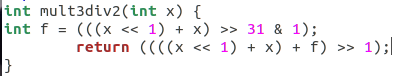
mul2OK代码：判断乘以二后是否溢出就是判断乘以二后的数据，与原数据相比符号位是否变化了，若符号位不同则说明溢出，若相同则说明未溢出，举例：正数×2得负数，正溢出，负数×2得正数，负溢出。

首先保留数据的符号位，将数据右移31为后，再异或1即可保留符号位。之后再比较两个数据的符号位，也就是将两个数据异或，异或之后的结果与题目要求的返回值刚好相反，所以再与1异或一次即可求反，得到与题目相匹配的返回值。



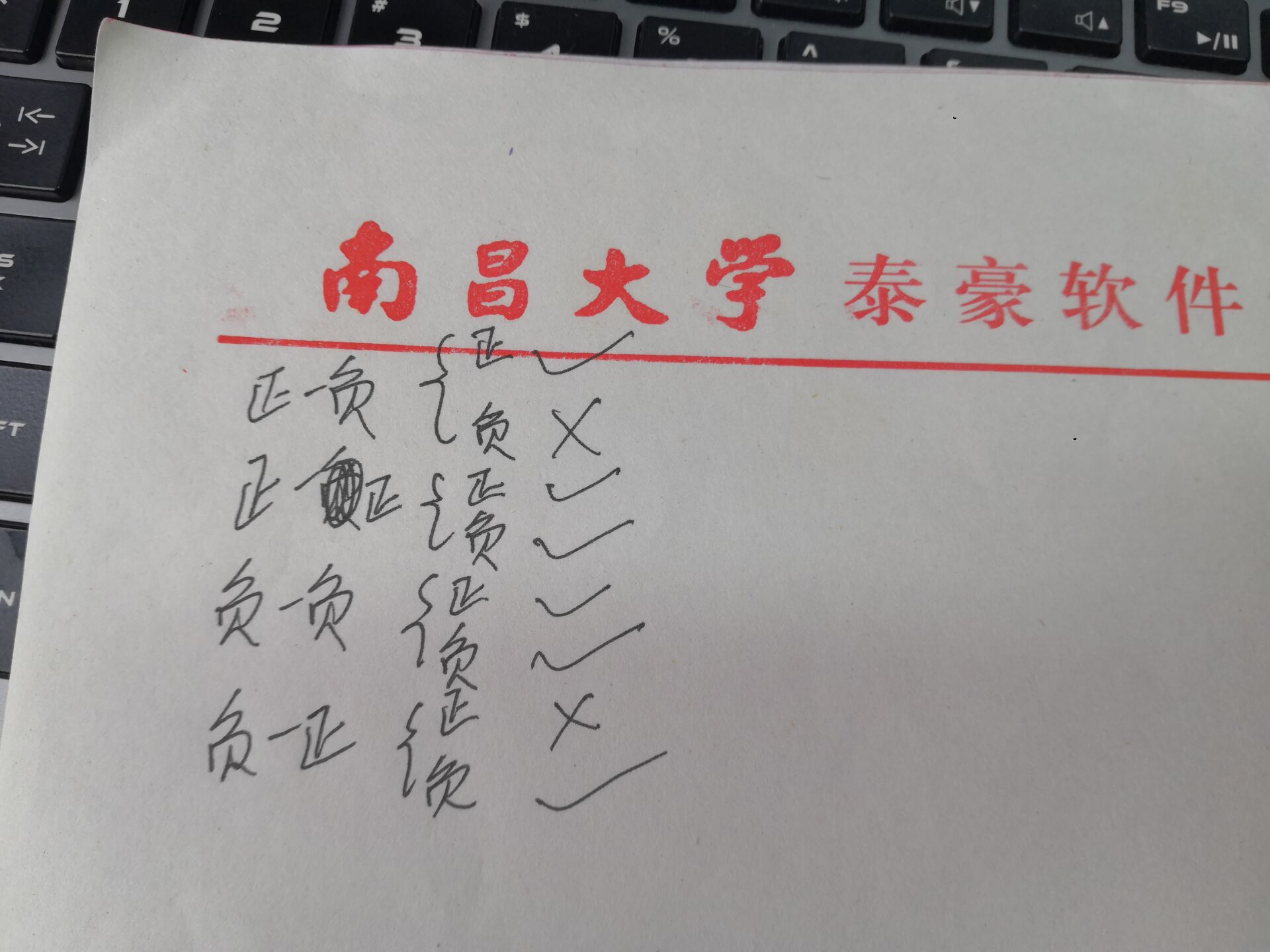
mult3div2代码：首先分析题目要求，一个乘除法向零取整！计算机底部实现除法比较复杂，需要很多指令，所以一般都是采取左移右移的方式简化乘除法。但是针对负数的除法有一些小问题需要注意，就是一个带符号整数负数，除以某个数后，小数部分不为0的话，计算机底部是直接舍去的，对于这个负数的商来说变小了（小数点的部分被舍去了），所以我们得到了一个向负无穷取整的机器数，所以这时候就需要仔细看书了，书上教了一种方法，就是加偏移量（2的k次方 – 1，k是右移的位数），采用这种思想设计了自己的代码，最后成功通过！

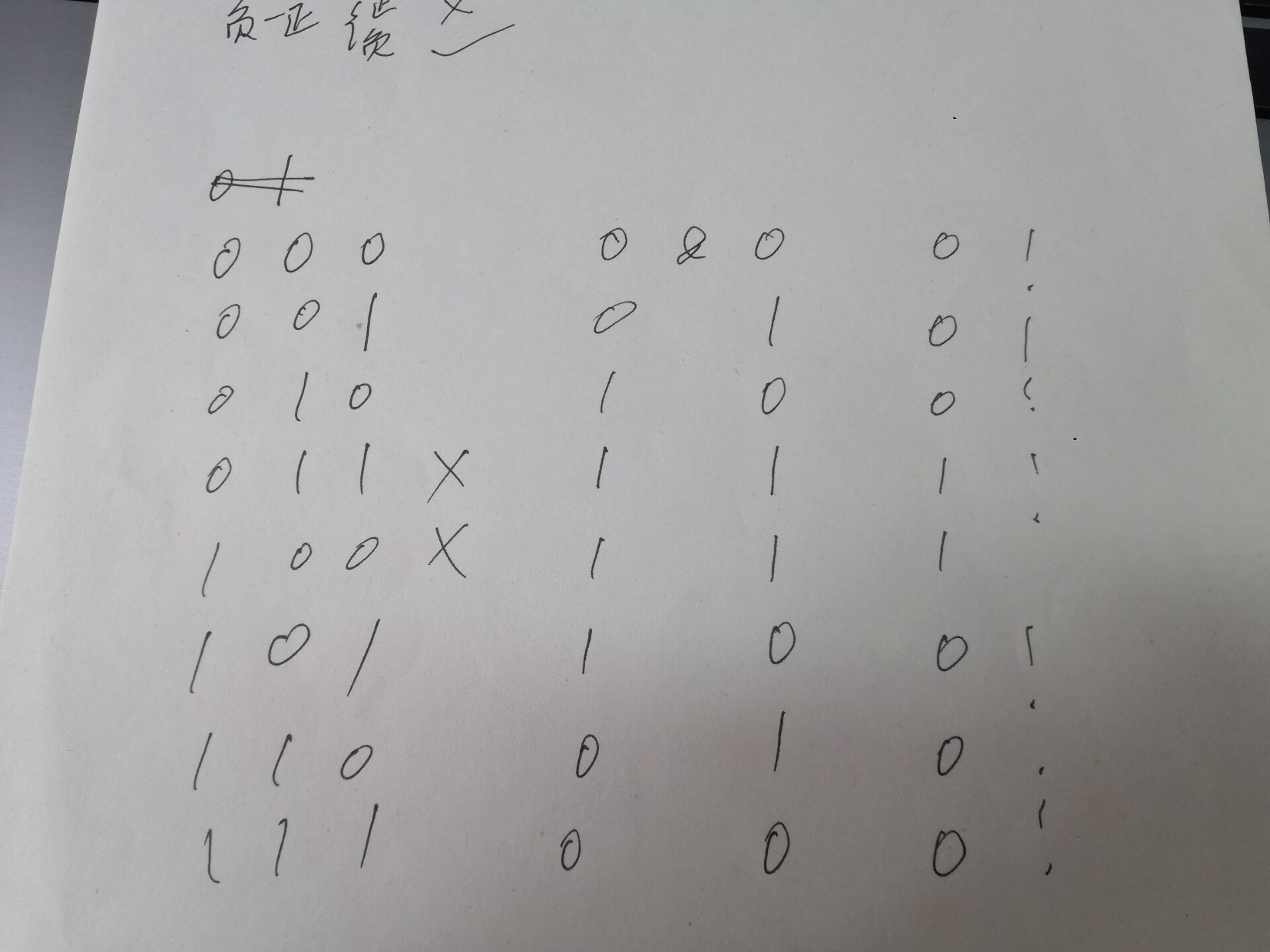
分析代码，先用一个临时变量f存储乘3除以2以后的机器数的符号位，然后再用一次原机器数乘3除以2，加上f，这时的f由于经过前面的处理，只可能是0或者1，乘3除以2后是正数那么是0，否则是1，这就避免了向负无穷取整，之后再右移一位即可。



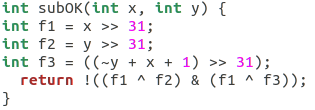
subOK代码：分析实现手段，把所有的可能x-y符号可能性都写出来方便分析，如下第一张图所示，接着可以根据数据的符号位把所有的组合写出来分析（如第二张图所示），可分析得到011（正-负=负），100（负-正=正）这两种组合是溢出的，所以我们只需要设计针对这两种情况的溢出就可以了。

具体代码解析放在第三张图上方！

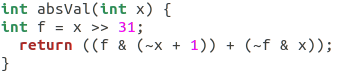




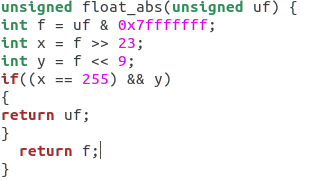
代码解析：首先用f1保留x的符号位，f2保留y的符号位，f3保留x，y相减以后的符号位。然后根据前面的分析可以发现，f1与f2相同时不可能溢出，同时f1与f3不同时不可能溢出，所以满足这两个条件的只有溢出的那两种情况，所以先将f1与f2异或，f1与f3异或，再按位与操作，经过实际计算发现返回值正好与我们题目的返回值相反，这时再求反即可实现题目要求！



absVal代码：分析题目要求，实现求绝对值，那么我们需要对正数和负数有不同的处理，但又不能使用选择结构，那怎么办呢？后来我仔细琢磨发现可以根据符号位来做这种类似的选择！就是用一个值先保留符号位，然后两边分别是本身，本身的求反，也就是说，当这个数是0时，只有一边计算结果不为0，另一边计算结果为0，两边计算结果相加相当于只算了一边，所以成功实现本题需要选择处理数据的目的！

代码分析：用f保留符号位，左边是负数的处理，若是负数，那么f为全1，所以按位与操作后左边括号就是这个负数数据的求反加一（绝对值），同时右侧由于～f为全0所以不管x是什么，右侧括号都为0，两边括号相加即为负数的绝对值。同理可得当x是正数时，f为全0，所以左侧括号为0，右侧括号等于x，即实现了正数的补码表示是本身，负数的绝对值是负数的补码求反加一。

float\_abs代码：分析题目要求求单精度浮点数的绝对值，只需要将符号位置0即可，但需要处理特殊情况（NaN），所以后续需要分离出阶码和尾数，再行判断。

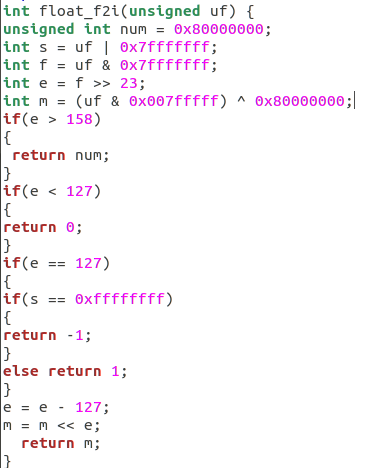
代码详解：用f存储置0后的32位机器数，然后将f右移23位取得解码值赋给x，将f左移9位取得尾数值赋给y。若是解码全为1，同时尾数不为0则说明这是一个NaN，所以直接返回uf即可。若不是，返回将符号位置0的f即可。

float\_f2i代码：题目实现分析，无特殊说明，现在的机器几乎都是IEEE754标准，所以我们默认题目是IEEE754标准下的实现。IEEE754标准下浮点数有三个特殊情况，第一种特殊情况就是0的表示，阶码全0，尾数也为0说明是-0或者是0，第二种特殊情况就是无穷大的表示，阶码全为1，尾数全为0，说明是正或负无穷大，第三种特殊情况就是非数的表示(NaN)，阶码全为1，尾数不为0，说明是NaN。针对这三种情况，题目已经给出了返回值的要求，若是0，返回0，若是NaN,或者是无穷大等特殊情况，返回0x8000 0000u。也就是说实现这个题目，只需要前面针对这三种特殊情况处理，之后统一按照规格化浮点数处理即可！（非规格化数因为过于小，在书上有提到，小于最小阶的数都看作0，所以非规格化数其实处理0时就已经顺带一起被处理了，并不会影响后面的规格化数处理）。

针对规格化数，又有一种特殊情况，就是1和-1的表示，我们知道IEEE754标准中规格化数隐藏了一位1所以最后我们需要还原这个1，所以若是1和-1的话，实际上它的解码为0，需要单独处理。

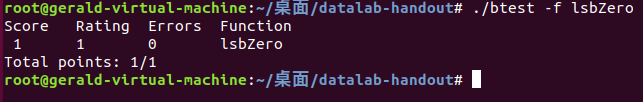
代码详解：

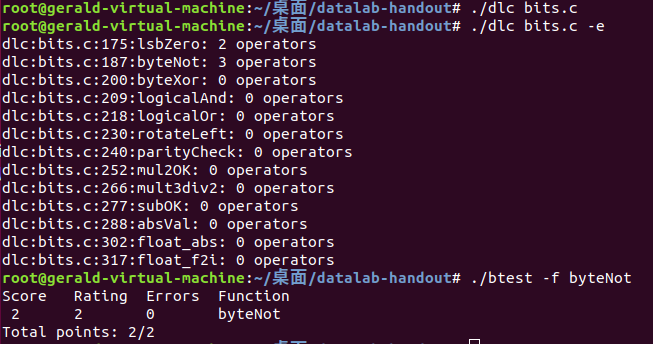
用num代表非数那些特殊情况的返回值，s保留浮点数的符号位，便于后面浮点数符号位的判断，f将符号位置0，e是阶码，用f右移23位即可实现保留阶码值，m是尾数，将uf高9位置0，这样得到了小数部分的尾数，再和0x8000 0000异或，这样就得到了把隐藏位1还原后的实际浮点小数，第一步首先判断超出了整数的表示范围，整数的存储大小是四个字节，也就是32位，所以若阶码值大于158（阶码真值= 阶码值 – 127），也就是阶码真值大于31，浮点数转换为整数时，小数部分直接截掉，所以若是阶码真值大于31，那么后面的部分都被截掉了，也是我们不需要的，也即题目要求特殊情况返回num。第二步判断阶码是否小于127，若是小于127则代表阶码真值小于0，也就是非规格化数，所以相当于就是0了，直接返回0即可。再进一步处理规格化数的判断，若是阶码等于127，说明不是1就是-1，在判断符号位s即可。最后就是处理规格化数了，非常简单，阶码真值e = 阶码值- 127，尾数部分m左移阶码真值，最后再返回这个处理过的浮点数值m即可。



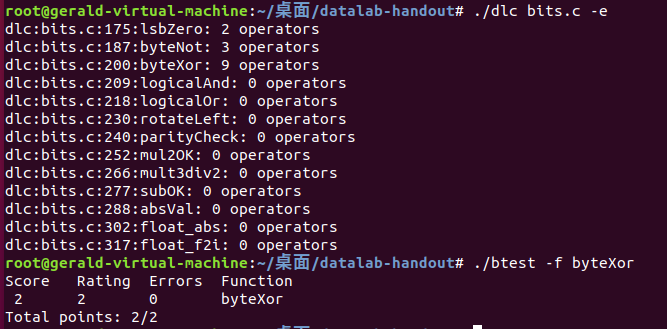
## 1.5实验结果

lsbZero测试结果：

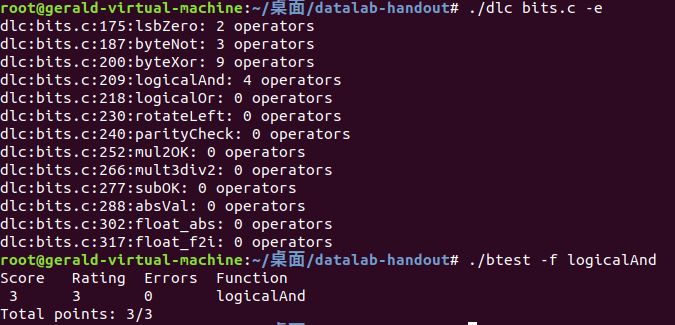
byteNot测试结果：



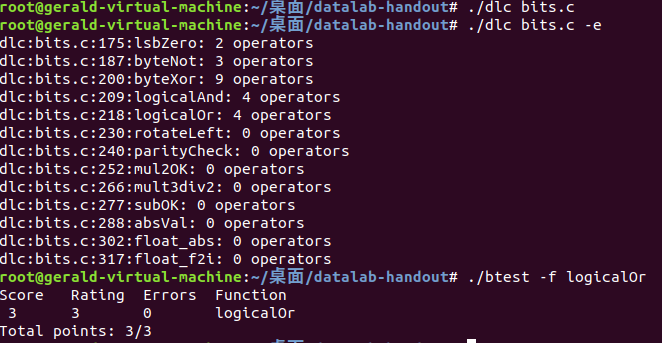
byteXor测试结果：



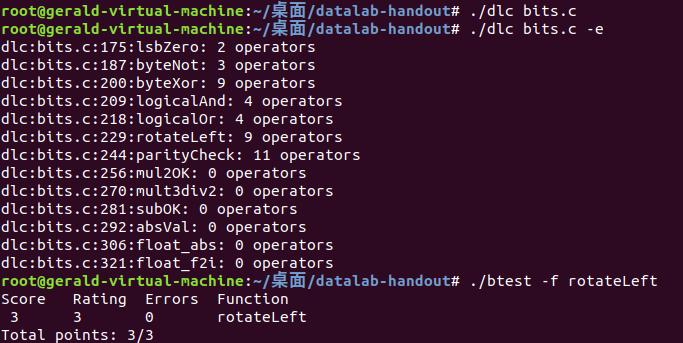
logicalAnd测试结果：



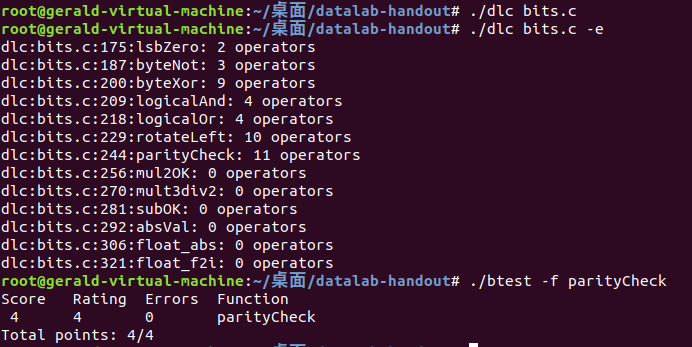
logicalOr测试结果：



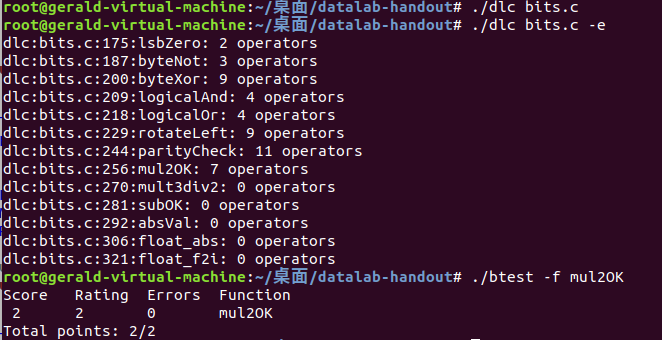
rotateLeft测试结果：



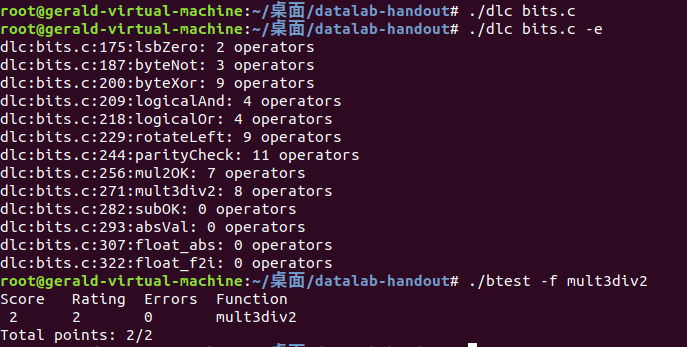
parityCheck测试结果:

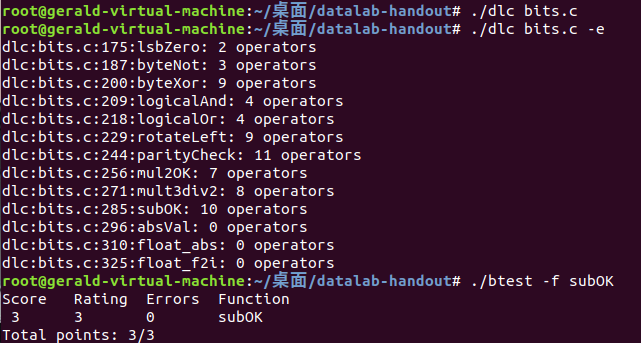


mul2OK测试结果:

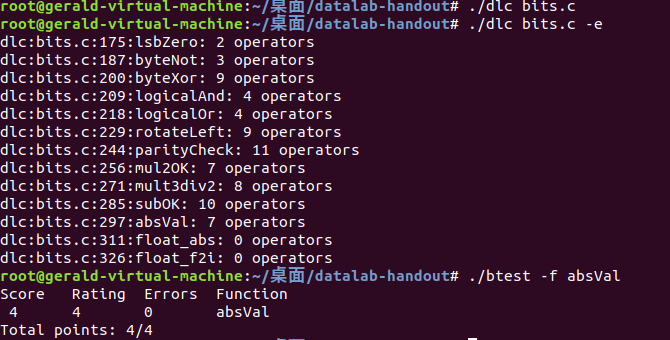


mul3div2测试结果：

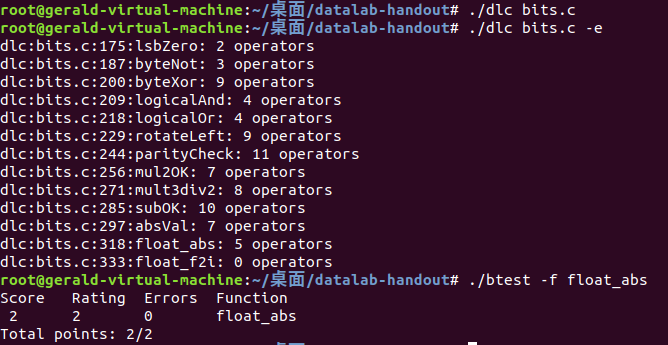
subOK测试结果：



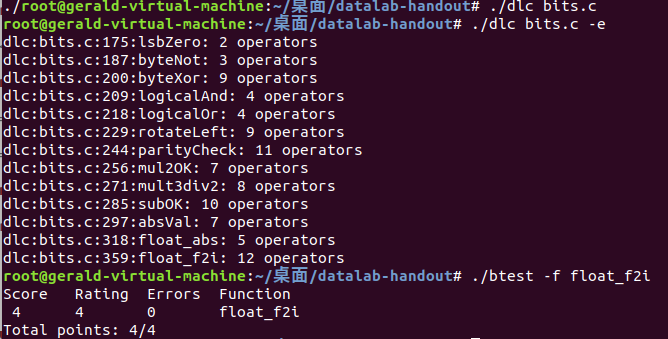
absVal测试结果：



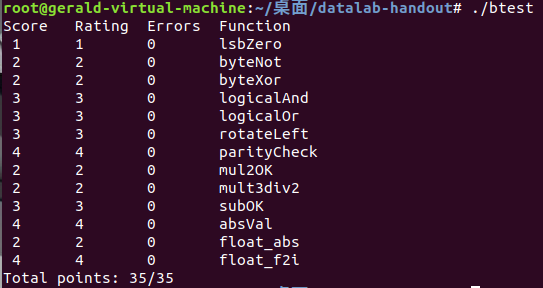
float\_abs测试结果：



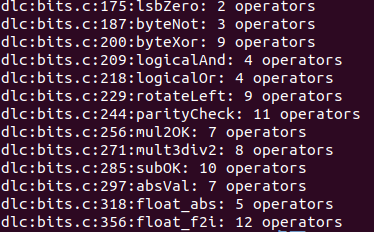
float\_f2i测试结果：



最后来一张全家福：



运算符全家福：



## 1.6实验小结

本次实验非常有意义，从中我学到了很多。从课程一开始就感受到硬件的魅力，虽然是很枯燥的东西，但是我很喜欢硬件，所以也一直在琢磨这些实验题目，就像老师说的，有关硬件的东西不自己去琢磨琢磨是学不好的，一开始，不相信，现在我是信的。

学习硬件最大的乐趣就在于了解一个微体系结构怎么去实现人的思维的解决方法，当你真正的自己一个人解决一个问题了，你会很有成就感，不仅仅是作业完成的释然，更多的是一种自己能行的感觉，这次实验真的很棒，让我体验到了机器硬件的底层，感受到了很多有趣的地方。

本次实验过程历时一个月左右，让我加深了对机器底部的表示方式的认识，有关补码表示，移码表示，还有原码之间的种种关系，运算，换算方法等等，对于我学习计算机组成原理这门课有极大的帮助，对于以后设计软件，硬件等也有了很好的基础铺垫。虽然在过程中，也遇到了很多困难，比如如何利用有限的操作符实现特定功能，实现选择结构等等，这真的很有趣，也很有挑战性，难点就是在于对机器数的表示方式的认识，以及对于操作符的利用，理解的体会。

最后我想说，这样的实验真的很棒，谢谢老师！