**课 程 实 验 报 告**

**课程名称： 计算机组成原理**

**专业班级：**

**学 号：**

**姓 名：**

**指导教师：**

**报告日期：**

# 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需要用到的文件：

Makefile - make命令的配置文件，用于生成btest、fshow和ishow

README - 说明文件

**bits.c - 这个是需要进行修改，增加内容的源代码文件**

bits.h - 头文件

btest.c - btest的主程序源文件

btest.h - 用于构建btest的头文件

decl.c - 用于构建btest的源文件

tests.c - 用于构建btest的源文件

tests-header.c - 用于构建btest的源文件

dlc - 规则检查应用程序

fshow.c - 用于检查浮点表示的实用程序

ishow.c - 用于检查整数表示的实用程序

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：

每个“Expr”需要遵守以下规则（）：

1. 从0到255(0xFF)的整型常量。不能使用大的整型常量，如0xffffffff

2. 函数参数、本地局部变量（不能使用全局变量）

3. 一元整型运算符：! ~

4. 二进制整形运算符：& ^ | + << >>

有的题目对可用的运算符有进一步的限制。

**禁止**：

1. 使用诸如：if, do, while, for, switch, 等语句

2. 定义或使用宏

3. 在文件中定义任何其他函数

4. 调用任何函数

5. 使用超出题目要求的运算符，如 &&, ||, -, or ?:

6. 使用任何形式的类型转换

7. 使用int以外的其他数据类型，包括数组、结构体和联合体

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

如果没有输出信息，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、全面检查程序运行是否合规，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:

测试所有函数的正确性并打印错误信息：

unix> ./btest

以简洁的方式测试所有函数，不输出错误信息：

unix> ./btest -g

测试指定的函数“foo”的正确性：

unix> ./btest -f foo

指定函数参数来测试指定的函数“foo”的正确性：

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。这两个工具在make时会自动生成，对应的源程序分别为ishow.c和fshow.c。

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

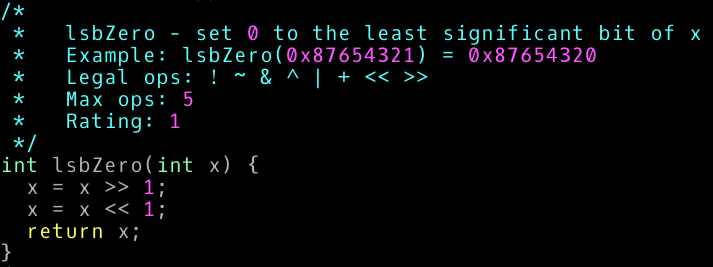
Floating point value 2.131829405e-38

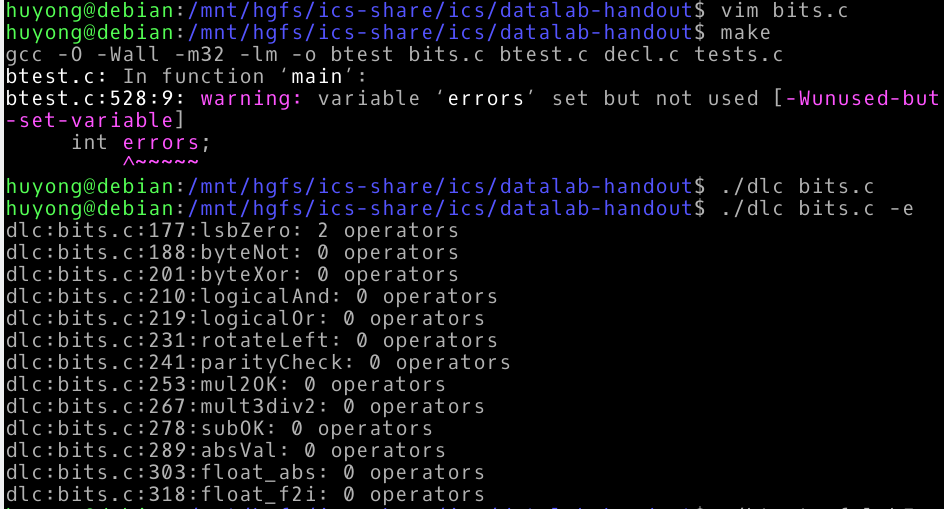
Bit Representation 0x00e822bb, sign = 0, exponent = 0x01, fraction = 0x6822bb

Normalized. +1.8135598898 X 2^(-126)

## 实验过程

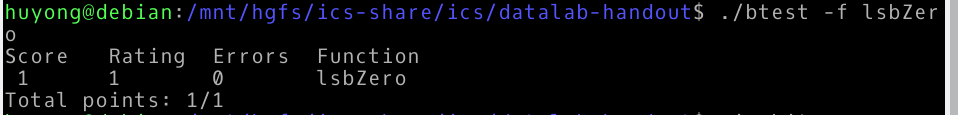
*下面以lsbZero函数为例以截屏方式，展示实验过程。同学们上交时，替换为自己的！尽可能考虑多种解决方法（优秀成绩的必要条件）*





## 1.5实验结果

*[btest检测结果]* *同学们上交时，替换为自己的！*



## 1.6实验小结