

哈尔滨工业大学

实验报告

实验（二）

题 目 DataLab 数据表示

专 业 计算机类

学 号 1170301027

班 级 1703010

学 生 冯帅

指 导 教 师 史先俊

实 验 地 点 G712

实 验 日 期 20180930

计算机科学与技术学院

目 录

第 1 章 实验基本信息	- 4 -
1.1 实验目的.....	- 4 -
1.2 实验环境与工具.....	- 4 -
1.2.1 硬件环境.....	- 4 -
1.2.2 软件环境.....	- 4 -
1.2.3 开发工具.....	- 4 -
1.3 实验预习.....	- 4 -
第 2 章 实验环境建立	- 6 -
2.1 UBUNTU 下 CODEBLOCKS 安装（5 分）	- 6 -
2.2 64 位 UBUNTU 下 32 位运行环境建立（5 分）	- 7 -
第 3 章 C 语言的位操作指令	- 8 -
3.1 逻辑操作（1 分）	- 8 -
3.2 无符号数位操作（2 分）	- 8 -
3.3 有符号数位操作（2 分）	- 8 -
第 4 章 汇编语言的位操作指令	- 9 -
4.1 逻辑运算(1 分).....	- 9 -
4.2 无符号数左右移（2 分）	- 9 -
4.3 有符号左右移（2 分）	- 9 -
4.4 循环移位（2 分）	- 9 -
4.5 带进位位的循环移位（2 分）	- 10 -
4.6 测试、位测试 BTX（2 分）	- 10 -
4.7 条件传送 CMOVXX（2 分）	- 10 -
4.8 条件设置 SETCXX（1 分）	- 10 -
4.9 进位位操作（1 分）	- 10 -
第 5 章 BITS 函数实验与分析	- 11 -
5.1 函数 LSBZERO 的实现及说明.....	- 11 -
5.2 函数 BYTENOT 的实现及说明函数.....	- 11 -
5.3 函数 BYTEXOR 的实现及说明函数.....	- 12 -
5.4 函数 LOGICALAND 的实现及说明函数.....	- 12 -
5.5 函数 LOGICALOR 的实现及说明函数.....	- 12 -
5.6 函数 ROTATELEFT 的实现及说明函数.....	- 13 -
5.7 函数 PARITYCHECK 的实现及说明函数.....	- 13 -
5.8 函数 MUL2OK 的实现及说明函数.....	- 14 -
5.9 函数 MULT3DIV2 的实现及说明函数.....	- 14 -
5.10 函数 SUBOK 的实现及说明函数.....	- 15 -

5.11 函数 ABSVAL 的实现及说明函数.....	- 15 -
5.12 函数 FLOAT_ABS 的实现及说明函数.....	- 16 -
5.13 函数 FLOAT_F2I 的实现及说明函数.....	- 16 -
5.14 函数 XXXX 的实现及说明函数（CMU 多出来的函数-不加分）	- 18 -
第 6 章 总结.....	- 19 -
10.1 请总结本次实验的收获.....	- 19 -
10.2 请给出对本次实验内容的建议.....	- 19 -
参考文献.....	- 21 -

第 1 章 实验基本信息

1.1 实验目的

熟练掌握计算机系统的数据表示与数据运算
通过 C 程序深入理解计算机运算器的底层实现与优化
掌握 Linux 下 makefile 与 GDB 的使用

1.2 实验环境与工具

1.2.1 硬件环境

Intel Core i7-7700HQ 2.81GHz, 8GB RAM, 128GB SSD

1.2.2 软件环境

Windows10 64 位以上; Vmware 11 以上; Ubuntu 16.04 LTS 64 位/优麒麟 64 位

1.2.3 开发工具

Visual Studio 2010 64 位; CodeBlocks 64 位; vi/vim/gedit+gcc

1.3 实验预习

上实验课前, 必须认真预习实验指导书 (PPT 或 PDF)

了解实验的目的、实验环境与软硬件工具、实验操作步骤, 复习与实验有关的理论知识。

写出 C 语言下的位操作指令:

- 逻辑 :逻辑&& || !
- 无符号 :& | ~ ^ << >>
- 有符号 :& | ~ ^ << >>

写出汇编语言下的位操作指令：

▪ 逻辑运算：

NOT:求反；

AND : 逻辑乘；

OR: 逻辑加；

XOR: 异或；

TEST: 测试位；

SHL、SHR、SAL、SAR、SHLD、SHRD: 左/右移位；

ROL、ROR、RCL、RCR: 左/右循环移位。

无符号:SHL、SHR、SAL、SAR、SHLD、SHRD: 左/右

移位；

ROL、ROR、RCL、RCR: 左/右循环移位。

有符号:SHL、SHR、SAL、SAR、SHLD、SHRD: 左/右

移位；

ROL、ROR、RCL、RCR: 左/右循环移位。

▪ 测试、位测试: TEST、BT、BTC (位 n 取反)、BTR (清零)、
BTS (置位)

▪ 条件传送: CMOVcc src,dst (cc 表示条件, src: r16,r32,r64,
dst:r/m16,r/m32,r/m64

▪ 条件设置: SETCxx (xx 为条件, 成立 dest = 1)

▪ 进位位操作:CLC, CMC, STC

第 2 章 实验环境建立

2.1 Ubuntu 下 CodeBlocks 安装 (5 分)

CodeBlocks 运行界面截图：编译、运行 hellolinux.c

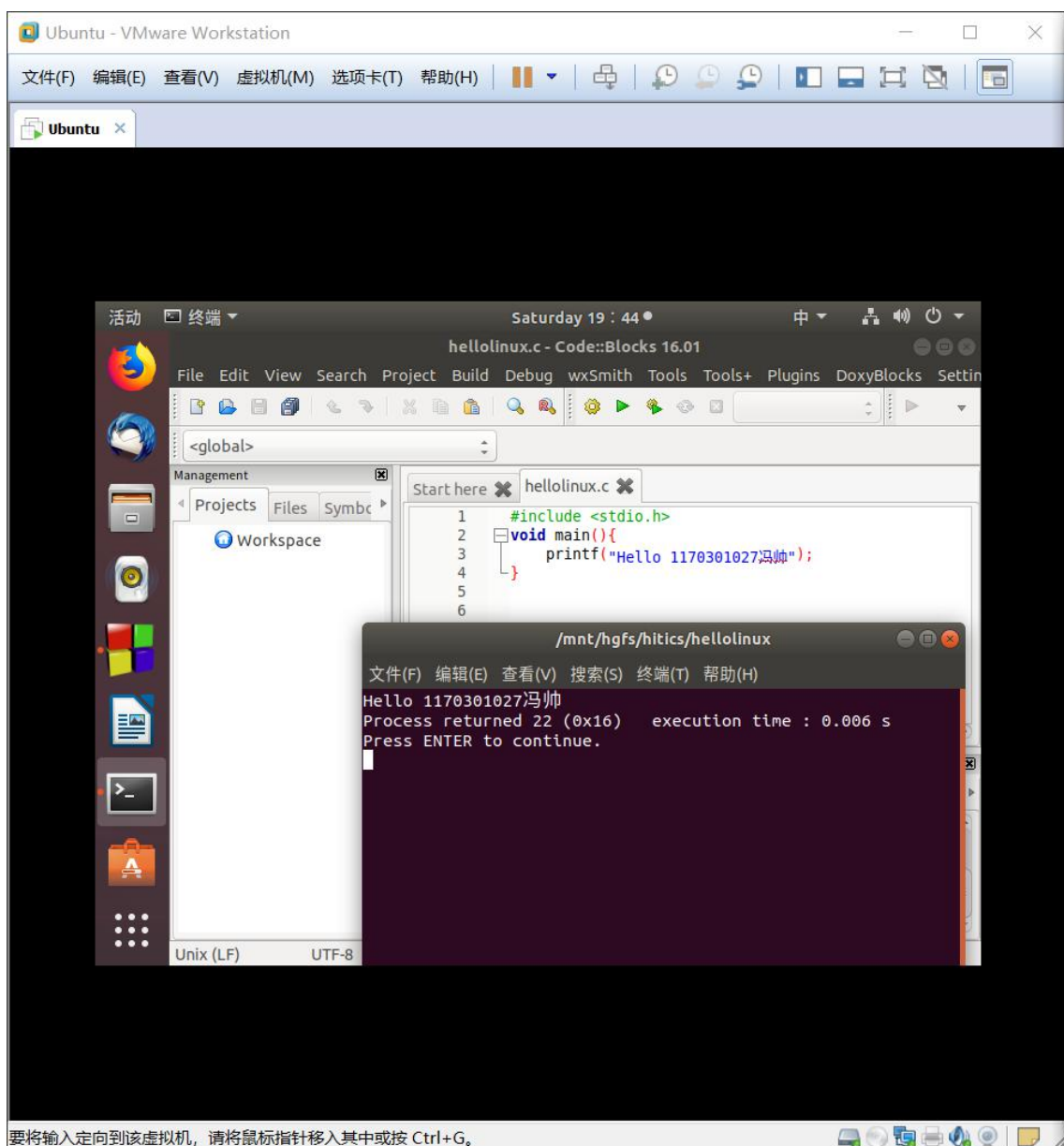
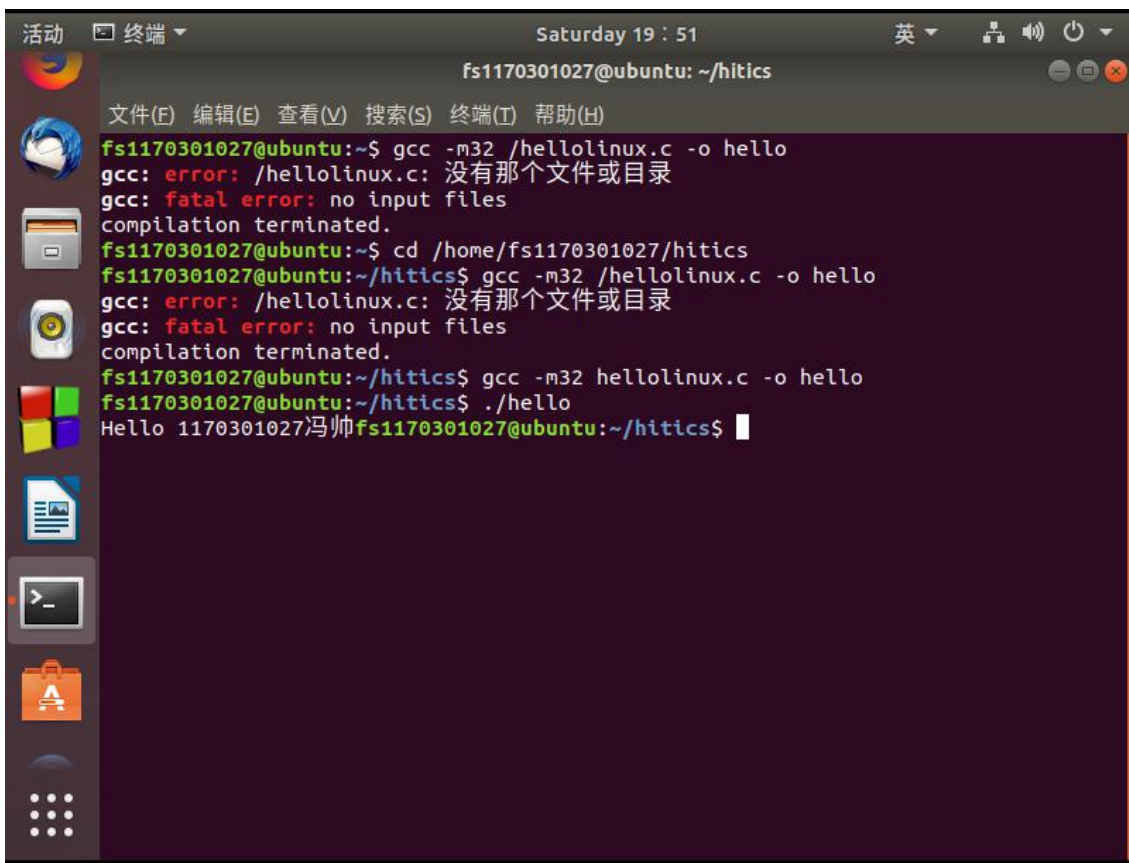


图 2-1 Ubuntu 下 CodeBlocks 截图

2.2 64 位 Ubuntu 下 32 位运行环境建立 (5 分)

在终端下，用 gcc 的 32 位模式编译生成 hellolinux.c。执行此文件。

Linux 及终端的截图。



```
活动 终端 Saturday 19:51 英 终端(T) 帮助(H)
fs1170301027@ubuntu: ~/hitics
文件(E) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
fs1170301027@ubuntu:~$ gcc -m32 /hellolinux.c -o hello
gcc: error: /hellolinux.c: 没有那个文件或目录
gcc: fatal error: no input files
compilation terminated.
fs1170301027@ubuntu:~$ cd /home/fs1170301027/hitics
fs1170301027@ubuntu:~/hitics$ gcc -m32 /hellolinux.c -o hello
gcc: error: /hellolinux.c: 没有那个文件或目录
gcc: fatal error: no input files
compilation terminated.
fs1170301027@ubuntu:~/hitics$ gcc -m32 hellolinux.c -o hello
fs1170301027@ubuntu:~/hitics$ ./hello
Hello 1170301027冯帅fs1170301027@ubuntu:~/hitics$
```

图 2-2 Ubuntu 与 Windows 共享目录截图

第 3 章 C 语言的位操作指令

写出 C 语言例句

3.1 逻辑操作 (1 分)

`Conditon1 && condition2 || (!condition3)`

3.2 无符号数位操作 (2 分)

`~x | !x & x^1 | x>>3 & x<<3`

3.3 有符号数位操作 (2 分)

`~x | !x & x^1 | x>>3 & x<<3`

第 4 章 汇编语言的位操作指令

写出汇编语言例句

4.1 逻辑运算 (1 分)

and \$0x4a, %al

or \$0x61, %al

xor \$0x37, %al

not %al

4.2 无符号数左右移 (2 分)

shl \$1, %bl

shr \$1, %al

SAL \$1, %BL

SAR \$1, %AL

4.3 有符号左右移 (2 分)

shl \$1, %bl

shr \$1, %al

SAL \$1, %BL

SAR \$1, %AL

4.4 循环移位 (2 分)

```
ROL $4, %AL  
ROR $1, %AL
```

4.5 带进位位的循环移位 (2 分)

```
RCL $4, %AL  
RCR $1, %AL
```

4.6 测试、位测试 BTx (2 分)

TEST BTTEST

```
test $0x2e, %al
```

```
bt $7, %al
```

```
mov $0x80, %al
```

```
btsl $7, %eax
```

```
btc $7, %eax
```

```
btr $7, %eax
```

4.7 条件传送 CMOVxx (2 分)

```
cmovl %ebx, %eax
```

4.8 条件设置 SETCxx (1 分)

```
setc %al
```

4.9 进位位操作 (1 分)

```
clc, cmc, stc
```

```
clc # CF=0
```

```
cmc
```

```
stc    # CF=1
```

第 5 章 BITS 函数实验与分析

每题 8 分，总分不超过 80 分

5.1 函数 lsbZero 的实现及说明

程序如下：

```
int lsbZero(int x) {  
    return x&(~1);  
}
```

设计思想：就是要让 x 与上 0xffffffffe, 将前 n-1 位保持不变，最后一位置零，由于可用常数范围在 0~255 之间，所以想到对 1 按位取反操作可得到 0xffffffffe, 因而可行。

5.2 函数 byteNot 的实现及说明函数

程序如下：

```
int byteNot(int x, int n) {  
  
    return x ^ (255<<(n<<3));  
}
```

设计思想：异或 1 相当于对该位取反。因为要将对应的字节取反，就是要将 32 位看成 4 位去操作，因此想到用八位 1 也就是 0xff (255) 的相应左移右移

操作去与之相异或；又因为操作符限定，且每移位 n 相当于移位字节 n ，即对应 32 位二进制来说移位 $n*8$ 即 $n<<3$ ，然后 0xff 相应移位使得程序正确。

5.3 函数 byteXor 的实现及说明函数

程序如下：

```
int byteXor(int x, int y, int n) {  
    return !((((255<<(n<<3))&x)>>(n<<3)) ^ (((255<<(n<<3))&y)>>(n<<3))));  
}
```

设计思想：刚开始把 byte 看成 bit，反复试验都不正确，看了好多个测试用例才知道是对字节进行操作，有点迷。很快就改过来了，整体思想还是和上一题差不多，就是用 0xff 去刷新相应字节，然后移位到 0 字节处相异或，若结果为零则为零不作操作，若结果不为零取非再取非之后就把非零数变成 1 了，符合题目要求的返回值。

5.4 函数 logicalAnd 的实现及说明函数

程序如下：

```
int logicalAnd(int x, int y) {  
    return (!x) & (!y);  
}
```

设计思想：我们知道两次取非操作可把非零数值置 1，而按位与操作相当于 0 和 1 之间的逻辑与；那么逻辑与的实现只要将两操作数通过重复取非运算变成 0，1 之间的操作即可，相对简单

5.5 函数 logicalOr 的实现及说明函数

程序如下：

```
int logicalOr(int x, int y) {  
    return (!x) | (!y);  
}
```

设计思想：同上一题逻辑，逻辑或的实现只要将两操作数取非取非变成 0，1 之间的操作然后位或即可

5.6 函数 rotateLeft 的实现及说明函数

程序如下：

```
int rotateLeft(int x, int n) {
    return (x<<n) + ((x>>(32-~n+1)) & ((1<<n)+~1+1));
}
```

设计思想：由于 C 语言是算术右移，所以花了好长时间去分析如何把符号位的影响消掉，而且由于没有减法，所以将被减数取非加一能得到一样的效果。消符号位就构造 00000（32-n 个）1111（n 个）这样的操作数即可。因此表达式表示操作数执行逻辑左移后加上操作数经处理的逻辑右移结果，因而成立。

附加：((1<<n)+~1+1)该操作是想求(1<<n)-1,因为没有减法所以对原数按位取反加一实现减法操作，屡试不爽。

5.7 函数 parityCheck 的实现及说明函数

程序如下：

```
int parityCheck(int x) {
    x ^= (x >> 16);
    x ^= (x >> 8);
    x ^= (x >> 4);
    x ^= (x >> 2);
    x ^= (x >> 1);
    return x&1;
}
```

设计思想：异或相当于半加法，只不过没有进位而已，而且异或不会改变其中 1 的奇偶性；开始前十六位与后十六位异或，若对应位上同一，则变为零，若不同为 1 则变为 1，相当于偶数个一变为零，奇数个 1 保留，往后一次用剩

下的十六位折半异或，再折半异或，到最后一次赋值实际上是因为比较所有位之后，偶数个 1 都消下去之后的结果，此时最后一位代表 1 的个数奇偶性，1 则为奇，0 则为偶；

5.8 函数 mul2OK 的实现及说明函数

程序如下：

//原来程序

```
int mul2OK(int x) {
    return !((x<<1)>>31) ^ (x>>31);
}
```

//原来程序

```
int mul2OK(int x) {
    return (((x<<1)>>31)&1) ^ ((x>>31)&1) + 1 & 1;
}
```

设计思想：判断数据乘二有无溢出，即相当于判断数据左移一位之后符号位是否改变，我将两个数据符号位右移到最低位之后只有四种情况，分析之后知道异或取非（即同或）可行，再根据题目要求就出来了。

以上是之前的想法，但是检查时候发现不让使用“！”，想到是要符号位判断就行，所以&1（只留最后一位当符号位），异或变同或就想法子把 1 变成 0 就 ok 了。

5.9 函数 mult3div2 的实现及说明函数

程序如下：

```
int mult3div2(int x) {
    int y = x+x+x;
    int signy = (y>>31)&1;
    int y0 = ((y<<31)>>31)&1;
```

```

    return (y>>1) + (signy & y0);
}

```

设计思想：题本意为乘三除以二，我以为和乘以二分之三一样，绕了很多弯，后来知道之后如上，y 为 x 乘 3 的结果，如果乘三之后为正数则直接>>1 即可，如果 y 是负数且为奇数则需要加一用来消除负数近似四舍五入的取整形式。

5.10 函数 subOK 的实现及说明函数

程序如下：

```

int subOK(int x, int y) {
    int signx = (x>>31) & 1;
    int signy = (y>>31) & 1;
    int signxy = ((x+(~y)+1)>>31) & 1;
    return  !(!(signx^signxy) | !(signx^signy));
}

```

设计思想：光看符号位的话不溢出可以有如下情况：

被减数	减数	差
1	0	1
0	1	0
1	1	1、0
0	0	1、0

因而只要判断相应符号位即可，即当 xy 异号时，x 是否和 x-y 异号，最后只要保证返回结果对应是 0, 1 即可，方法不唯一，程序中 return 是我想出来的。

5.11 函数 absVal 的实现及说明函数

程序如下：

```

int absVal(int x) {

```

```

    int sign = (x>>31);

    return ((~sign) & x) | ((sign) & ((~x) + 1));
}

```

设计思想：C 语言中的右移是算数右移，所以取 32 个符号进行不同的与运算，同时用符号来决定输出不输出。负数以补码形式存在，按位取反加一直接消除了符号位，相当于取补时候将符号位变成了零，直接是对应的绝对值，有些投机取巧。

5.12 函数 float_abs 的实现及说明函数

程序如下：

```

unsigned float_abs(unsigned uf) {
    if((uf & 0x7fffffff)>0x7f800000)
        return uf;
    else
        return uf & 0x7fffffff;
}

```

设计思想：因为把 unsigned 看作单精度浮点数（1：8：23-->符号位：指数位+127：尾数位），所以 0x7f800000（exp=11..111,frac = 00...00）代表 infinity，比这个大的为 nan，参数在这个范围内要返回参数值，不在范围内返回 abs 值（即符号位置零）

5.13 函数 float_f2i 的实现及说明函数

程序如下：

```

int float_f2i(unsigned uf) {
    unsigned E = ((uf>>23)&0xff);
    unsigned sign = (uf>>31)<<31;
    unsigned flag = (((uf & 0x00ffffff) | (0x00800000))>>(150-E));
    if((uf & 0x7fffffff)>0x7f800000 || E>150+7)

```



```

        return 0x80000000u;

    else
    {
        if(E>=127 && E<=150 && !sign){
            return flag;
        }
        else if(E>=127 && E<=150 && sign){
            return ~flag + 1;
        }
        else if(E>150)
        {
            return (((uf & 0x00ffffff) | (0x00800000))<<(E-150) );
        }
        else
            return 0;
    }
}

```

设计思想：emmmm，我自己都没想到过了，思路就是，E 为 exp，sign 为符号判断正负的，flag 是我定义的一个标志，意味着在 23 位尾数前一位加一个 1 前面归零，构成整个二进制小数的部分，相当于整数化为二进制小数之后的小数部分右移了 23 位；

然后是判断部分，如果 $\text{exp} > 127 + 23 + 7$ (阶码移位+127；尾数部分 23；除小数部分 23 位，整数部分 1 位，符号位 1 位，32 位剩下 7 位)，代表数据超范围，同时如果数据为 nan 和 infinity 用前一个条件判断；

Else 里面是多种情况，一种是 $\text{exp} < 127 + 23$ 时，如果直接右移（flag）是补码形式（但是最高位全为零），这时需要 sign 来判断正负，如果正数则直接返

回，如果为负数，则对 `flag` 取反加一就得到正常负数的补码形式了；

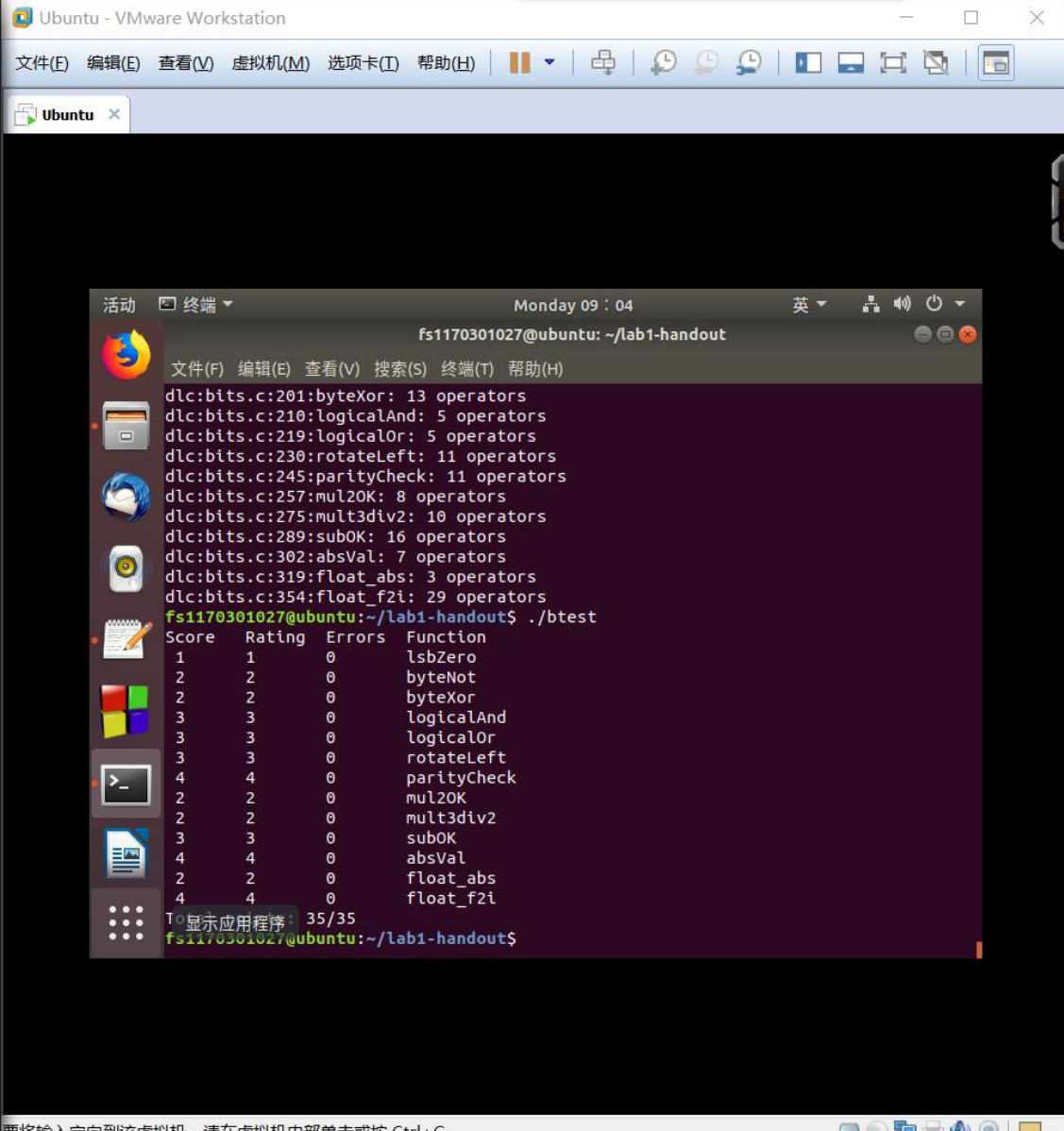
第二种是 $127+23 < \text{exp} < 127+23+7$ 此时相当于把小数部分 23 位补齐还要乘 $2^{(\text{exp}-127-23)}$ 即对填上整数部分的数据左移这么多位即可；

其余情况则均是大于零小于一的，直接返回 0 即可。

5.14 函数 XXXX 的实现及说明函数（CMU 多出来的函数-不加分）

第 6 章 总结

10.1 请总结本次实验的收获



```
fs1170301027@ubuntu: ~/lab1-handout
d1c:bits.c:201:byteXor: 13 operators
d1c:bits.c:210:logicalAnd: 5 operators
d1c:bits.c:219:logicalOr: 5 operators
d1c:bits.c:230:rotateLeft: 11 operators
d1c:bits.c:245:parityCheck: 11 operators
d1c:bits.c:257:mul20K: 8 operators
d1c:bits.c:275:mult3div2: 10 operators
d1c:bits.c:289:sub0K: 16 operators
d1c:bits.c:302:absVal: 7 operators
d1c:bits.c:319:float_abs: 3 operators
d1c:bits.c:354:float_f2i: 29 operators
fs1170301027@ubuntu:~/lab1-handout$ ./btest
Score  Rating  Errors  Function
1      1        0      lsbZero
2      2        0      byteNot
2      2        0      byteXor
3      3        0      logicalAnd
3      3        0      logicalOr
3      3        0      rotateLeft
4      4        0      parityCheck
2      2        0      mul20K
2      2        0      mult3div2
3      3        0      sub0K
4      4        0      absVal
2      2        0      float_abs
4      4        0      float_f2i
To display application: 35/35
fs1170301027@ubuntu:~/lab1-handout$
```

收获颇多啊，看到这个的一刹那有点想哭，感觉对位操作理解加深了好多好多

10.2 请给出对本次实验内容的建议

觉得挺好啊。。。

注：本章为酌情加分项。

参考文献

为完成本次实验你翻阅的书籍与网站等

- [1] 林来兴. 空间控制技术[M]. 北京: 中国宇航出版社, 1992: 25-42.
- [2] 辛希孟. 信息技术与信息服务国际研讨会论文集: A 集[C]. 北京: 中国科学出版社, 1999.
- [3] 赵耀东. 新时代的工业工程师[M/OL]. 台北: 天下文化出版社, 1998 [1998-09-26]. <http://www.ie.nthu.edu.tw/info/ie.newie.htm> (Big5) .
- [4] 谌颖. 空间交会控制理论与方法研究[D]. 哈尔滨: 哈尔滨工业大学, 1992: 8-13.
- [5] KANAMORI H. Shaking Without Quaking[J]. Science, 1998, 279 (5359) : 2063-2064.
- [6] CHRISTINE M. Plant Physiology: Plant Biology in the Genome Era[J/OL]. Science , 1998 , 281 : 331-332[1998-09-23]. <http://www.sciencemag.org/cgi/collection/anatmorp>.