

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 1160300823

班 级 1603008

学 生 陈柯昊

指 导 教 师 吴锐

实 验 地 点 G712

实 验 日 期 2017.10.21

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)	- 8 -
第 3 章 C 语言的位操作指令	- 11 -
3.1 逻辑操作 (1 分)	- 11 -
3.2 无符号数位操作 (2 分)	- 11 -
3.3 有符号数位操作 (2 分)	- 12 -
第 4 章 汇编语言的位操作指令	- 13 -
4.1 逻辑运算(1 分)	- 13 -
4.2 无符号数左右移 (2 分)	- 13 -
4.3 有符号左右移 (2 分)	- 13 -
4.4 循环移位 (2 分)	- 13 -
4.5 带进位位的循环移位 (2 分)	- 13 -
4.6 测试、位测试 BTX (2 分)	- 13 -
4.7 条件传送 CMOVXX (2 分)	- 14 -
4.8 条件设置 SETCXX (1 分)	- 14 -
4.9 进位位操作 (1 分)	- 14 -
第 5 章 BITS 函数实验与分析	- 14 -
5.1 函数 LSBZERO 的实现及说明	- 14 -
5.2 函数 BYTENOT 的实现及说明函数	- 15 -
5.3 函数 BYTEXOR 的实现及说明函数	- 15 -
5.4 函数 LOGICALAND 的实现及说明函数	- 16 -
5.5 函数 LOGICALOR 的实现及说明函数	- 16 -
5.6 函数 ROTATELEFT 的实现及说明函数	- 17 -
5.7 函数 PARITYCHECK 的实现及说明函数	- 17 -
5.8 函数 MUL2OK 的实现及说明函数	- 18 -
5.9 函数 MULT3DIV2 的实现及说明函数	- 19 -
5.10 函数 SUBOK 的实现及说明函数	- 20 -

5.11 函数 ABSVAL 的实现及说明函数	- 20 -
5.12 函数 FLOAT_ABS 的实现及说明函数	- 21 -
5.13 函数 FLOAT_F2I 的实现及说明函数	- 21 -
5.14 函数 XXXX 的实现及说明函数（CMU 多出来的函数-不加分）	- 21 -
第 6 章 总结	- 22 -
10.1 请总结本次实验的收获	- 22 -
10.2 请给出对本次实验内容的建议	- 22 -
参考文献	- 23 -

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

X64 CPU; 2GHz; 2G RAM; 256GHD Disk 以上

1.2.2 软件环境

Windows7 64 位以上; VirtualBox/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 位以上; CodeBlocks; vi/vim/gpedit+gcc

1.3 实验预习

写出 C 语言下的位操作指令:

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

无符号

有符号

测试、位测试 BTx

条件传送 CMOVxx

条件设置 SETCxx

进位位操作

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

Ubuntu 64 位 - VMware Workstation

文件(E) 编辑(E) 查看(V) 虚拟机(M) 选项卡(T) 帮助(H)



库



在此处键入内容进行搜索

- 我的计算机
- Ubuntu 64 位
- 共享的虚拟机

主页

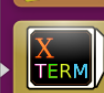


我的计算机



Ubuntu 64 位

XTerm



hello

```
Hello 1160300823 ChenKehao!  
Process returned 0 (0x0)   execution time : 0.017 s  
Press ENTER to continue.
```

Debug

```
stdio.h>  
stdlib.h>
```

```
("Hello 11  
0;
```

Logs & others

Code::Blocks

Search results

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。

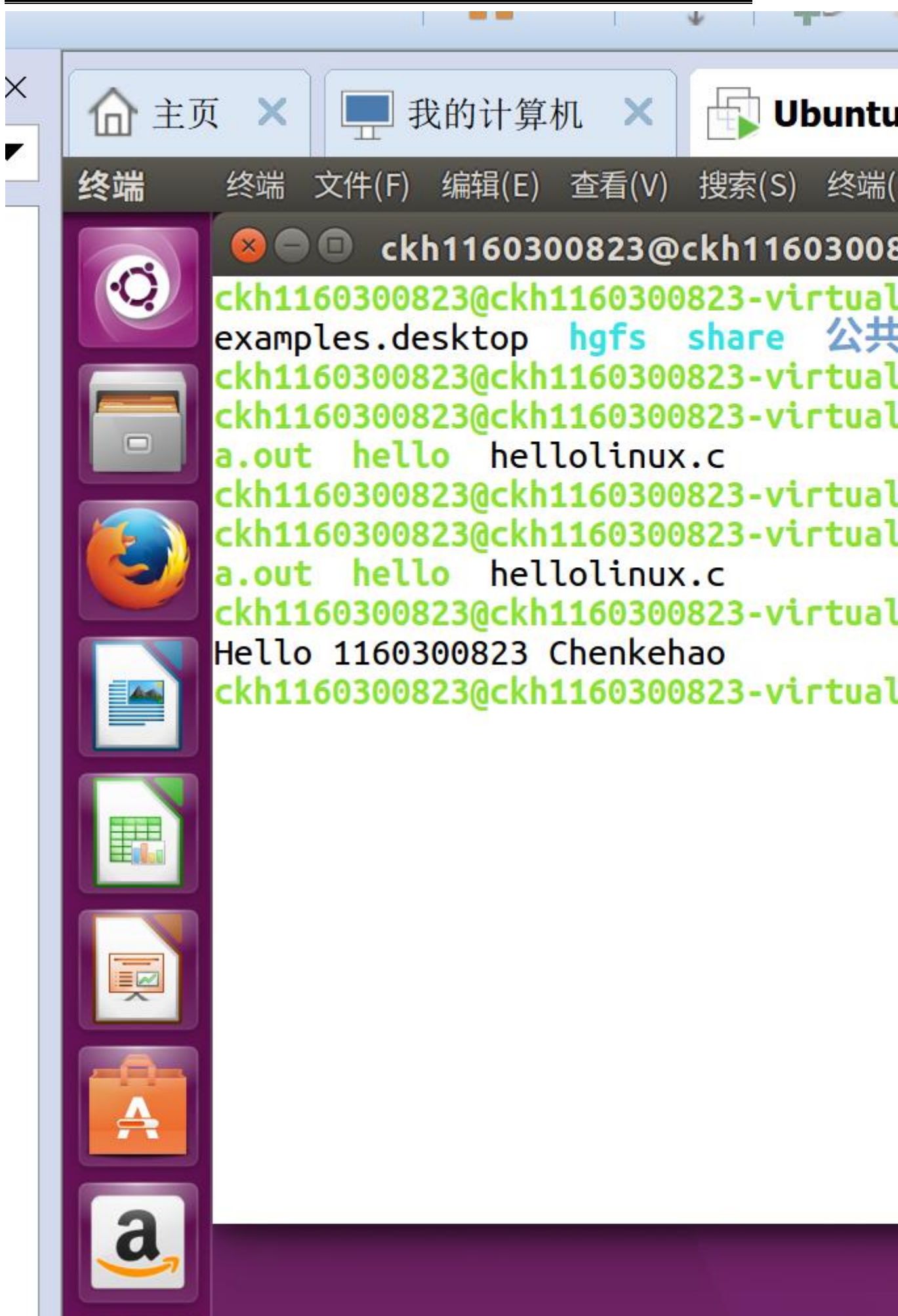


图 2-2 32 位运行环境建立

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)

& 按位与

```
int a = 3;
int b = 5;
printf("%d", a & b);
1
```

| 按位或

```
int a = 060;
int b = 017;
printf("%d", a | b);
63;
```

^ 按位异或

```
int a = 071;
int b = 052;
printf("%d", a ^ b);
19;
```

~ 取反

```
int a = 077;
printf("%d", ~a);
8;
```

<< 左移

>> 右移

3.2 无符号数位操作 (2 分)

```
unsigned x;
x << 1
x >> 2
```

3.3 有符号数位操作 (2 分)

```
int x;  
x<<1;  
x>>2;
```

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 DEST, SRC 逻辑与
OR DEST, SRC 逻辑或
XOR DEST, SRC 逻辑异或
NOT REG/MEM 逻辑非
TEST DEST, SRC 测试

4.2 无符号数左右移 (2 分)

SHL %rax;
SAL %rax;
SAR %rax

4.3 有符号左右移 (2 分)

SHL %rax;
SHR %rax;
SAR %rax;

4.4 循环移位 (2 分)

ROL %rax, 8;
ROR %rax, 6;

4.5 带进位位的循环移位 (2 分)

RCL %rax, 8;
RCR %rax, 8;

4.6 测试、位测试 BTx (2 分)

AX=0x1234;
BT AX, 2;
BTC AX, 6;
BTR AX, 10;
BTX AX, 14;

4.7 条件传送 CMOVxx (2 分)

```
CMOVE %rax, %rdx; //等于 0 时传送
CMOVNE %rax, %rdx; //不等于 0 时传送
CMOVS %rax, %rdx; //负数时传送
CMOVNS %rax, %rdx; //非负数时传送
CMOVG %rax, %rdx; //有符号大于时传送
CMOVGE %rax, %rdx; //有符号大于等于时传送
CMOVL %rax, %rdx; //有符号小于时传送
CMOVLE %rax, %rdx; //有符号小于等于时传送
CMOVA %rax, %rdx; //无符号大于时传送
CMOVAE %rax, %rdx; //无符号小于时传送
CMOVB %rax, %rdx; //无符号小于时传送
CMOVBE %rax, %rdx; //无符号小于等于时传送
```

4.8 条件设置 SETCxx (1 分)

```
SETCE %rax; //等于 0 时传送

SETCNE %rax; //不等于 0 时传送
```

4.9 进位位操作 (1 分)

```
setb D
jb
```

第 5 章 BITS 函数实验与分析

每题 8 分，总分不超过 80 分
截图： \$./btest -f 函数名

5.1 函数 lsbZero 的实现及说明

程序如下：

```
int lsbZero(int x) {
    return (x>>1)<<1;
}
```

btest 截图：

```
ckh1160300823@ckh1160300823-virtual-machine:~/lab1-handout$ ./btest -f lsbZero
Score   Rating   Errors   Function
1       1         0       lsbZero
Total points: 1/1
```

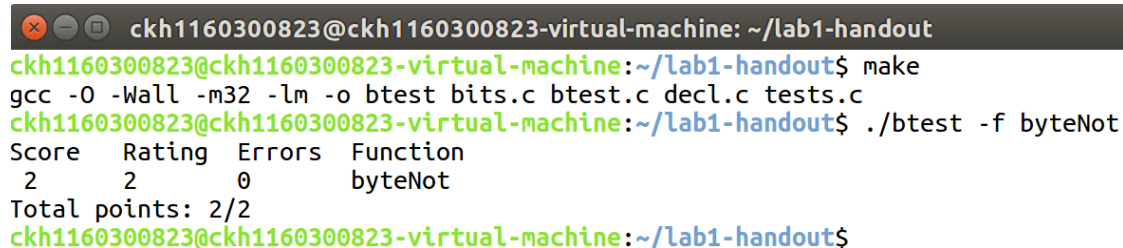
设计思想：通过右移后再左移，自动使得最后一位为 0，即最低有效位为 0。

5.2 函数 byteNot 的实现及说明函数

程序如下：

```
int byteNot(int x, int n) {
    int temp = 0XFF;
    temp = 0XFF<<(n<<3);
    return x^temp;
}
```

btest 截图：



```
ckh1160300823@ckh1160300823-virtual-machine: ~/lab1-handout
ckh1160300823@ckh1160300823-virtual-machine:~/lab1-handout$ make
gcc -O -Wall -m32 -lm -o btest bits.c btest.c decl.c tests.c
ckh1160300823@ckh1160300823-virtual-machine:~/lab1-handout$ ./btest -f byteNot
Score   Rating   Errors   Function
2       2         0       byteNot
Total points: 2/2
ckh1160300823@ckh1160300823-virtual-machine:~/lab1-handout$
```

设计思想：由题意，要对 x 的第 n 个字节进行操作。所以设置一个 $temp = 0XFF$ ，将其左移 $8*n$ 位，这样就保证了除了第 n 个字节全为 1，其余都为 0。此时进行异或运算，即可达到目的。

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) {
    int temp = (x^y)>>(n<<3);
    temp = temp & 0xFF;
    return !!temp;
}
```

btest 截图:

```
ckh1160300823@ckh1160300823-virtual-machine: ~/lab1-handout
ckh1160300823@ckh1160300823-virtual-machine:~/lab1-handout$ make
gcc -O -Wall -m32 -lm -o btest bits.c btest.c decl.c tests.c
ckh1160300823@ckh1160300823-virtual-machine:~/lab1-handout$ ./btest -f byteXor
Score  Rating  Errors  Function
  2      2      0      byteXor
Total points: 2/2
ckh1160300823@ckh1160300823-virtual-machine:~/lab1-handout$
```

设计思想:

与上一题(byteNot)有类似之处。首先根据题意对 x 与 y 进行异或运算。随后为了保证输出是 0 或 1, 对其进行向右移位的操作。最后利用两次逻辑非运算, 确保结果非 1 即 0, 满足结果要求。

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);
}
```

btest 截图:

```
ckh1160300823@ckh1160300823-virtual-machine:~/lab1-handout$ make
gcc -O -Wall -m32 -lm -o btest bits.c btest.c decl.c tests.c
ckh1160300823@ckh1160300823-virtual-machine:~/lab1-handout$ ./btest -f logicalAnd
Score  Rating  Errors  Function
  3      3      0      logicalAnd
Total points: 3/3
```

设计思想:

!!x, 若 $x = 0$, !!x 为 0, 若 $x \neq 0$, !!x 为 1。

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);
}
```



```
}

```

btest 截图:

```
ckh1160300823@ckh1160300823-virtual-machine:~/lab1-handout$ ./btest -f logicalOr
Score  Rating  Errors  Function
  3      3      0      logicalOr
Total points: 3/3
```

设计思想:

!!x, 若 $x = 0$, !!x 为 0, 若 $x \neq 0$, !!x 为 1。

5.6 函数 rotateLeft 的实现及说明函数

程序如下:

```
int rotateLeft(int x, int n) {
    int High = x << n;
    int Low = (x >> (32 + ~n + 1)) & (~((~0) << n));
    return High | Low;
}
```

btest 截图:

```
ckh1160300823@ckh1160300823-virtual-machine:~/lab1-handout$ ./btest -f rotateLeft
Score  Rating  Errors  Function
  3      3      0      rotateLeft
Total points: 3/3
```

设计思想: 首先将处于低位的元素取出, 存储在高位中。再从 x 中设法取出高位元素, 存储在低位, 并使得其他位为 0。将二者进行或运算, 即可得到转化后的值。

5.7 函数 parityCheck 的实现及说明函数

程序如下:

```
int parityCheck(int x) {
    x = x ^ (x >> 16);
    x = x ^ (x >> 8);
}
```

```

    x = x^(x>>4);

    x = x^(x>>2);

    x = x^(x>>1);

    return x&0x1;

}

```

btest 截图:

```

ckh1160300823@ckh1160300823-virtual-machine:~/lab1-handout$ ./btest -f parityCheck
Score   Rating  Errors  Function
   4       4       0      parityCheck
Total points: 4/4
ckh1160300823@ckh1160300823-virtual-machine:~/lab1-handout$

```

设计思想:

不断对其进行对半的异或运算，只有一位为 0 一位为 1 才可运算出 1。由此，可以筛选出不成对的 1，最终可以筛出是否存在单独的 1。若有，即存在奇数个 1。

5.8 函数 mul2OK 的实现及说明函数

程序如下:

```

int mul2OK(int x) {

    int a = (x>>31)&0x1;

    int b = (x>>30)&0x1;

    return (a^b)^0x1;

}

```

btest 截图:

```

ckh1160300823@ckh1160300823-virtual-machine:~/lab1-handout$ ./btest -f mul2OK
Score   Rating  Errors  Function
   2       2       0      mul2OK
Total points: 2/2

```

设计思想:

int 类型在 C 语言中占 4 个字节，即 32 个二进制位。

当表示正数时，最高位为符号位（符号位为 0），最大的正数是 0111 1111 1111 1111 1111 1111 1111 1111 即 $2^{31} - 1 = 2147483647$ 。

所以此时，如果第 31 位为 1，则 $x*2$ 必然大于 2^{31} ，导致溢出。

当表示负数时，最高位为符号位（符号位为 1），最小的负数是 1000 0000 0000 0000 0000 0000 0000 0000 而在计算机中是以补码的形式存储的，C 语言规定 1000 0000 0000 0000 0000 0000 0000 0000 的补码为 -2147483648。

与正数类似，若第 31 为 0，则 $x*2$ 会溢出。

综上，问题转换到判断第 32 位（符号位）与第 31 位的关系上。用两个异或判断二者是否相同，相同返回 1，满足题意。

5.9 函数 mult3div2 的实现及说明函数

程序如下：

```
int mult3div2(int x) {
    int th = (x<<1) +x;
    int lsb = th&0x1;
    int s = (th>>31)&0x1;
    int result = th>>1;
    return (lsb&s)+result;
}
```

btest 截图：

```
ckh1160300823@ckh1160300823-virtual-machine:~/lab1-handout$ ./btest -f mult3div2
Score   Rating  Errors  Function
  2       2       0      mult3div2
Total points: 2/2
```

设计思想：

首先取出符号为和最低有效位。当且仅当这二者同时为 1 时，需要额外对运算结果+1。

5.10 函数 subOK 的实现及说明函数

程序如下：

```
int subOK(int x, int y) {
    int ans = x + (~y + 1); //ans = x - y;
    return !(((ans ^ x) & (x ^ y)) >> 31);
}
```

btest 截图：

```
Total points: 3/3
ckh1160300823@ckh1160300823-virtual-machine:~/lab1-handout$ make
gcc -O -Wall -m32 -lm -o btest bits.c btest.c decl.c tests.c
ckh1160300823@ckh1160300823-virtual-machine:~/lab1-handout$ ./btest -f subOK
Score   Rating  Errors  Function
   3       3       0      subOK
Total points: 3/3
```

设计思想：

当 x 的最高有效位和 y 的最高有效位不同，且 x 和 $(x-y)$ 的最高有效位不同时，才能判断为溢出。为节省运算符号，进行一定的简化。

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;
    x = (sign & (~x + 1)) + ((~sign) & x);
    return x;
}
```

btest 截图：

```
ckh1160300823@ckh1160300823-virtual-machine:~/lab1-handout$ ./btest -f absVal
Score   Rating  Errors  Function
   4       4       0      absVal
Total points: 4/4
ckh1160300823@ckh1160300823-virtual-machine:~/lab1-handout$
```

设计思想：

根据补码编码规则：若 x 最高位为 0，即 x 为非负数， $\text{absVal}=x$ ；若 x 最高位为 1，则 x 为负数， $\text{absVal}=-x$ ，取反再加 1。

5.12 函数 `float_abs` 的实现及说明函数

程序如下：

btest 截图：

设计思想：

5.13 函数 `float_f2i` 的实现及说明函数

程序如下：

btest 截图：

设计思想：

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

提高了对于数据表达部分的理解，同时通过自己动手操作，注意到了许多思考方法上面的细节，比如：

符号！符号！符号！

10.2 请给出对本次实验内容的建议

预习部分的 C 语言部分希望更加严谨。

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>.