

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 1170300817

班 级 1703008

学 生 林之浩

指 导 教 师 郑贵滨

实 验 地 点 G712

实 验 日 期 2018.10.08

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

5.11 函数 ABSVAL 的实现及说明函数	- 14 -
5.12 函数 FLOAT_ABS 的实现及说明函数	- 15 -
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）	- 16 -
第 6 章 总结	- 17 -
10.1 请总结本次实验的收获	- 17 -
10.2 请给出对本次实验内容的建议	- 17 -
参考文献	- 18 -

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

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、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 SETxx、进位位(CF)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

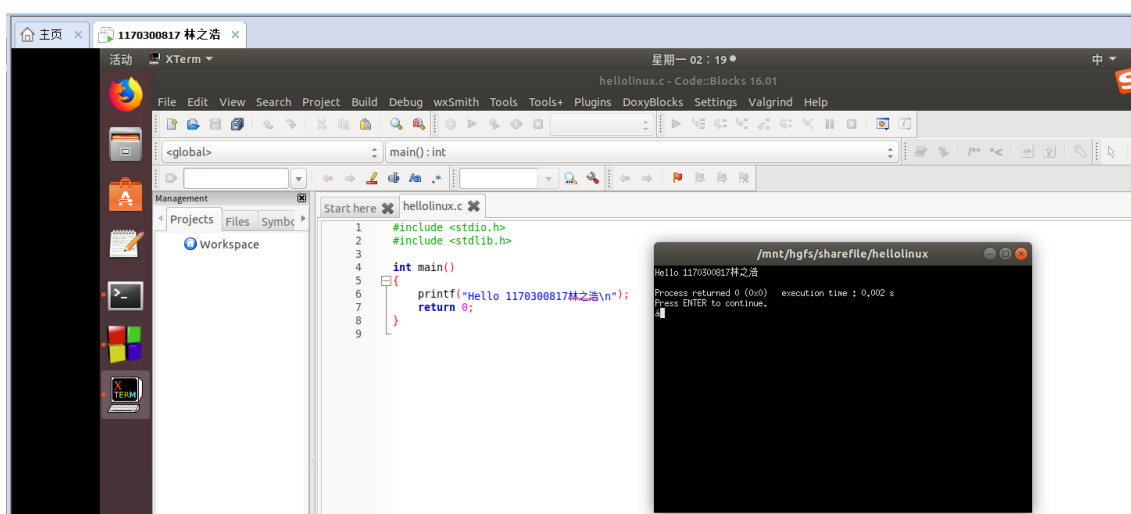


图 2-1 Ubuntu 下 CodeBlocks 截图

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。

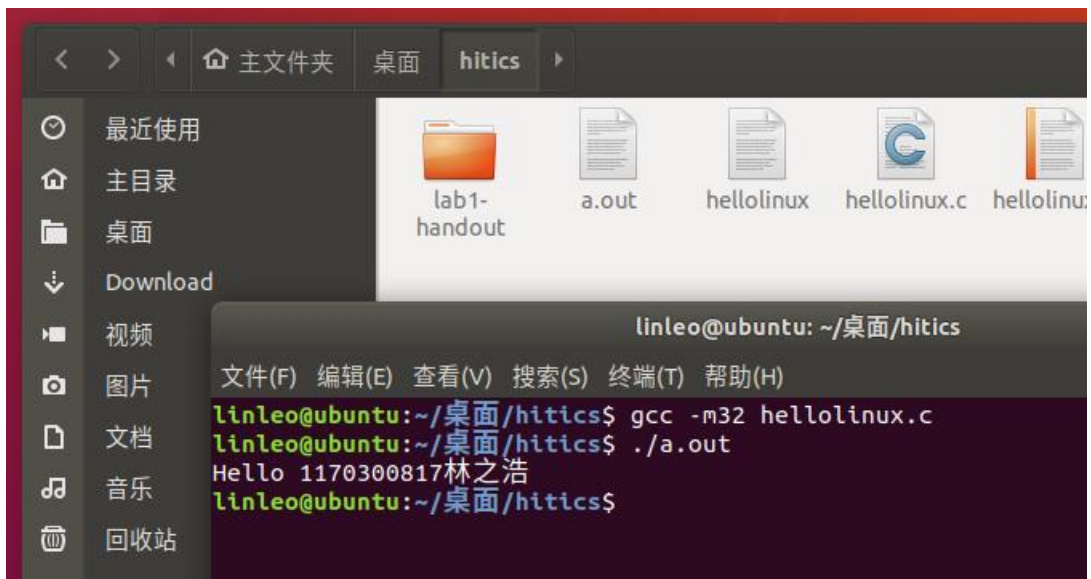


图 2-2 32 位运行环境建立

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)

位与: $x \& y$ 位或: $x | y$ 位取反: $\sim x$ 位异或: $x \wedge y$

3.2 无符号数位操作 (2 分)

位与: $x \& y$ 位或: $x | y$ 位取反: $\sim x$ 位异或: $x \wedge y$
左移: $x \ll 1$ 右移: $x \gg 1$

3.3 有符号数位操作 (2 分)

位与: $x \& y$ 位或: $x | y$ 位取反: $\sim x$ 位异或: $x \wedge y$
左移: $x \ll 1$ 右移: $x \gg 1$

对于有符号数: 左移的时候右侧补 0; 右移的时候左侧补符号位 (正数符号位为 0, 补 0; 负数符号位为 1, 补 1)

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 %eax,%ebx
OR %eax,%ebx
XOR %eax,%ebx
NOT %eax
```

4.2 无符号数左右移 (2 分)

SHL %eax
SHR %eax

4.3 有符号左右移 (2 分)

SAL %eax
SAR %eax

4.4 循环移位 (2 分)

ROL %eax
ROR %eax

4.5 带进位位的循环移位 (2 分)

RCL %eax
RCR %eax

4.6 测试、位测试 BT_x (2 分)

TEST %eax,%ebx
BTC r16/imm8,r/m16
BTR r16/imm8,r/m16
BTS r16/imm8,r/m16

4.7 条件传送 CMOV_{xx} (2 分)

CMOVG %eax,%ebx
CMOVNLE %eax,%ebx
CMOVGE %eax,%ebx
CMOVNGE %eax,%ebx
CMOVL %eax,%ebx
CMOVLE %eax,%ebx
CMOVNG %eax,%ebx
CMOVNO %eax,%ebx
CMOVS %eax,%ebx
CMOVNS %eax,%ebx

4.8 条件设置 SET_{xx} (1 分)

SETE %eax
SETNE %eax
SETS %eax
SETNS %eax
SETG %eax


```
SETGE %eax
SETL %eax
SETLE %eax
SETA %eax
SETAE %eax
SETB %eax
SETBE %eax
```

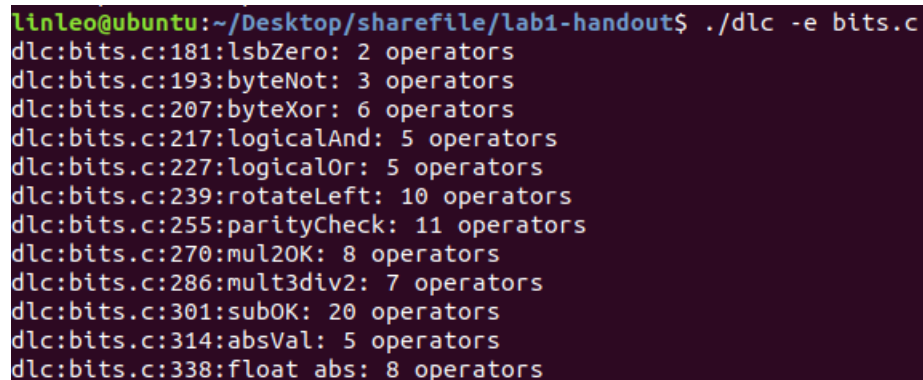
4.9 进位位操作 (1 分)

```
STC
CLC
```

第 5 章 BITS 函数实验与分析

每题 8 分，总分不超过 80 分

语法检查命令 `./dlc -e bits.c` 的结果截图：



```
linleo@ubuntu:~/Desktop/sharefile/lab1-handout$ ./dlc -e bits.c
dlc:bits.c:181:lsbZero: 2 operators
dlc:bits.c:193:byteNot: 3 operators
dlc:bits.c:207:byteXor: 6 operators
dlc:bits.c:217:logicalAnd: 5 operators
dlc:bits.c:227:logicalOr: 5 operators
dlc:bits.c:239:rotateLeft: 10 operators
dlc:bits.c:255:parityCheck: 11 operators
dlc:bits.c:270:mul20K: 8 operators
dlc:bits.c:286:mult3div2: 7 operators
dlc:bits.c:301:subOK: 20 operators
dlc:bits.c:314:absVal: 5 operators
dlc:bits.c:338:float_abs: 8 operators
```

要求：每个函数不可以有非法运算符、函数调用等，否则相应函数会被扣分

5.1 函数 `lsbZero` 的实现及说明

程序如下：

```
int lsbZero(int x) {
    return (~0x1)&x;
}
```

btest (命令 ./btest -f lsbZero) 的结果截图:

```
linleo@ubuntu:~/Desktop/sharefile/lab1-handout$ ./btest -f lsbZero
Score  Rating  Errors  Function
1      1        0      lsbZero
Total points: 1/1
```

设计思想: 通过 0x1 按位取反得到 0xffffffff 与原数相与达到将最低位置 0 的效果

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^(0xff<<(n<<3));
}
```

btest 截图:

```
linleo@ubuntu:~/Desktop/sharefile/lab1-handout$ ./btest -f byteNot
Score  Rating  Errors  Function
2      2        0      byteNot
Total points: 2/2
```

设计思想: 将需要转变的字节与 0xff 取异或达到按位取反的效果

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 !((x^y)&(0xff<<(n<<3)));
}
```

btest 截图:

```
linleo@ubuntu:~/Desktop/sharefile/lab1-handout$ ./btest -f byteXor
Score  Rating  Errors  Function
2      2        0      byteXor
Total points: 2/2
```

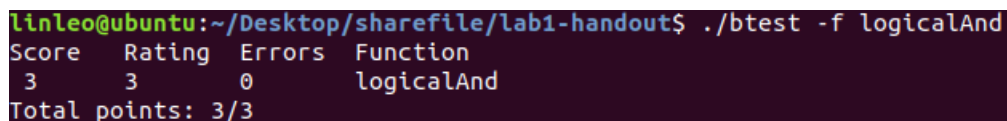
设计思想：如果 xy 的第 n 字节相等，异或后返回全 0 与移到第 n 位的 0xff 相与还是全 0，反之如果不为全 0 则二次取反返回 1

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：



```
linleo@ubuntu:~/Desktop/sharefile/lab1-handout$ ./btest -f logicalAnd
Score  Rating  Errors  Function
3      3      0      logicalAnd
Total points: 3/3
```

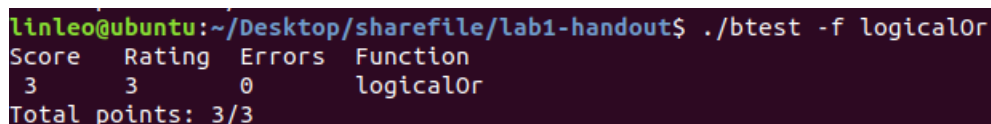
设计思想：如果 x 或 y 不是全 0，则 !! x 或 !! y 将返回 1，与运算后得到一与&&一致。如果其中一者为全 0，则二次取反后得 0，最后结果也将为 0。

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：



```
linleo@ubuntu:~/Desktop/sharefile/lab1-handout$ ./btest -f logicalOr
Score  Rating  Errors  Function
3      3      0      logicalOr
Total points: 3/3
```

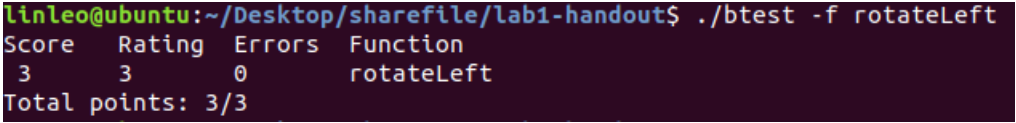
设计思想：如果 x, y 有一者不是全 0，则 !! x 或 !! y 将返回 1，或运算后得到 1，与||结果一致，如果 xy 全是全 0，则 !! x 和 !! y 都返回 0，取或还是 0。

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<<n)));
}
```

btest 截图：



```
linleo@ubuntu:~/Desktop/sharefile/lab1-handout$ ./btest -f rotateLeft
Score  Rating  Errors  Function
3      3      0      rotateLeft
Total points: 3/3
```

设计思想：(x>>(32-n))的后 n 位是溢出的 n 位，利用&(~(-1<<n)))，就是将其和前 32-n 位 0 后位 1 的数相与，达到清零前 32-n 位的效果。最后再和(x<<n)相或得到答案。

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 & 1;
}
```

btest 截图：

```
linleo@ubuntu:~/Desktop/sharefile/lab1-handout$ ./btest -f parityCheck
Score   Rating  Errors  Function
  4       4      0     parityCheck
Total points: 4/4
```

设计思想：参考书本课后习题 2.65 的巧妙解法，两个数异或过程不会改变 1 的个数的奇偶，例如两个 1 异或得 0，1 的个数一直是偶数个，0 和 1 异或运算后得 1，过程中 1 的个数一直是奇数个，利用此规律每次将原数的前一半与后一半按位异或，最后得到的 1 的个数的奇偶与原数一致。

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<<1) >> 31)&0x1;
    return ~(a^b)&0x1;
}
```

btest 截图：

```
linleo@ubuntu:~/Desktop/sharefile/lab1-handout$ ./btest -f mul2OK
Score   Rating  Errors  Function
  2       2      0     mul2OK
Total points: 2/2
```

设计思想：a 得到原数的符号位，b 得到原数的第二位，假如原数是正数则 a=0，此时如 b=1，则该数已经超出了正数表示上限的一半，将要溢出。同理，当 a=1，b=0 时这个负数也超出了负数表示范围的一半，乘以两倍后，将要溢出。

5.9 函数 mult3div2 的实现及说明函数

程序如下：

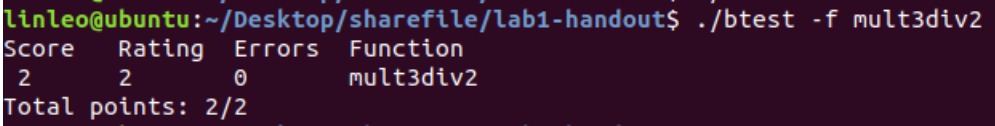
```
int mult3div2(int x)
{
    int m=x +(x << 1);
```

```

    return (m >> 1) + (m & 0x1 & (m >> 31) & 0x1);
}

```

btest 截图：



```

linleo@ubuntu:~/Desktop/sharefile/lab1-handout$ ./btest -f mult3div2
Score  Rating  Errors  Function
  2      2      0      mult3div2
Total points: 2/2

```

设计思想：先计算 x 的 $3/2$ ，然后通过 m 的符号判断是否需要舍入，达成向 0 舍入的目的。

5.10 函数 subOK 的实现及说明函数

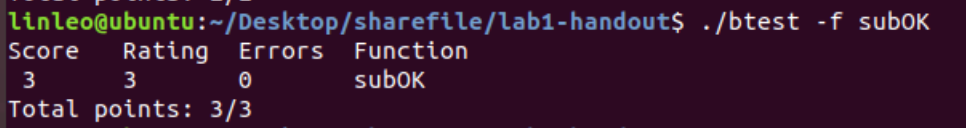
程序如下：

```

int subOK(int x, int y)
{
    int f11 = (x >> 31) & 0x1;
    int f12 = (y >> 31) & 0x1;
    int f13 = ((x + (~y + 1)) >> 31) & 0x1;
    return ((f11 & f12) | (f12 ^ f13)) | ((!f11 & !f12) | (!f12 ^ !f13));
}

```

btest 截图：



```

linleo@ubuntu:~/Desktop/sharefile/lab1-handout$ ./btest -f subOK
Score  Rating  Errors  Function
  3      3      0      subOK
Total points: 3/3

```

设计思想：f11 表示 x 的符号位，f12 表示 y 的符号位，f13 表示 $x-y$ 的符号位，不难发现，如果 x 正 y 负且得数为负，或者 x 负 y 正得数为正，则发生了溢出。

5.11 函数 absVal 的实现及说明函数

程序如下：

```

int absVal(int x)

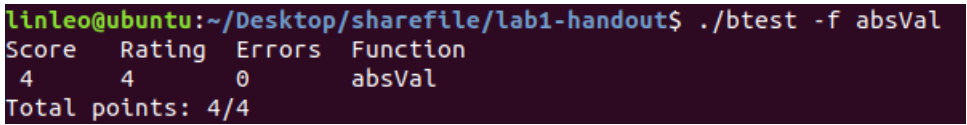
```

```

{
    int m=(x>>31)^x;
    return m+((x>>31)&1);
}

```

btest 截图：



```

linleo@ubuntu:~/Desktop/sharefile/lab1-handout$ ./btest -f absVal
Score  Rating  Errors  Function
  4      4      0      absVal
Total points: 4/4

```

设计思想：如 x 为正则 $m=x$ ，如 x 为负，则 m 为 x 取反，返回时判断 x 的符号位，如果是负数则对 m 加一得到绝对值，正数不操作。

5.12 函数 float_abs 的实现及说明函数

程序如下：

```

unsigned float_abs(unsigned uf)
{
    unsigned exp=(uf>>23)&0xff;
    unsigned frac=uf&0x7fffff;
    if (exp==0xff&&frac!=0)
    {
        return uf;
    }
    else
    {
        return frac|exp<<23;
    }
}

```

btest 截图：

```
linleo@ubuntu:~/Desktop/sharefile/lab1-handout$ ./btest -f float_abs
Score  Rating  Errors  Function
2      2      0      float_abs
Total points: 2/2
```

设计思想：先得到位表示的 `exp` 部分和 `frac` 部分，然后进行判断，当 `exp=11111111`，`frac` 不为 0 时，原数为 `nan`，返回原值，否则返回 `frac` 与上 `exp`。

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

复习了 linux 常用指令

掌握了计算机系统的数据表示与数据运算

10.2 请给出对本次实验内容的建议

增加实验的趣味性，例如实验三的做法。

注：本章为酌情加分项。

参考文献

- [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] 湛颖. 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>.