

# 哈尔滨工业大学

# 实验报告

## 实 验（二）

题 目 DataLab 数据表示

专 业 计算机科学与技术

学 号 1171000410

班 级 1703005

学 生 强文杰

指 导 教 师 吴锐

实 验 地 点 G712

实 验 日 期 2018.9.30

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） .....	- 6 -
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） .....	- 12 -
4.9 进位位操作（1 分） .....	- 13 -
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 .....	- 18 -
5.10 函数 SUBOK 的实现及说明函数 .....	- 19 -

5.11 函数 ABSVAL 的实现及说明函数.....	- 19 -
5.12 函数 FLOAT_ABS 的实现及说明函数.....	- 20 -
5.13 函数 FLOAT_F2I 的实现及说明函数 .....	- 20 -
5.14 函数 XXXX 的实现及说明函数（CMU 多出来的函数-不加分） .....	- 22 -
第 6 章 总结 .....	- 23 -
10.1 请总结本次实验的收获.....	- 23 -
10.2 请给出对本次实验内容的建议.....	- 23 -
参考文献 .....	- 24 -

## 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

X64CPU;

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 16.04LTS 64 位/  
优麒麟 64 位

#### 1.2.3 开发工具

Gcc , Codeblocks

### 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 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

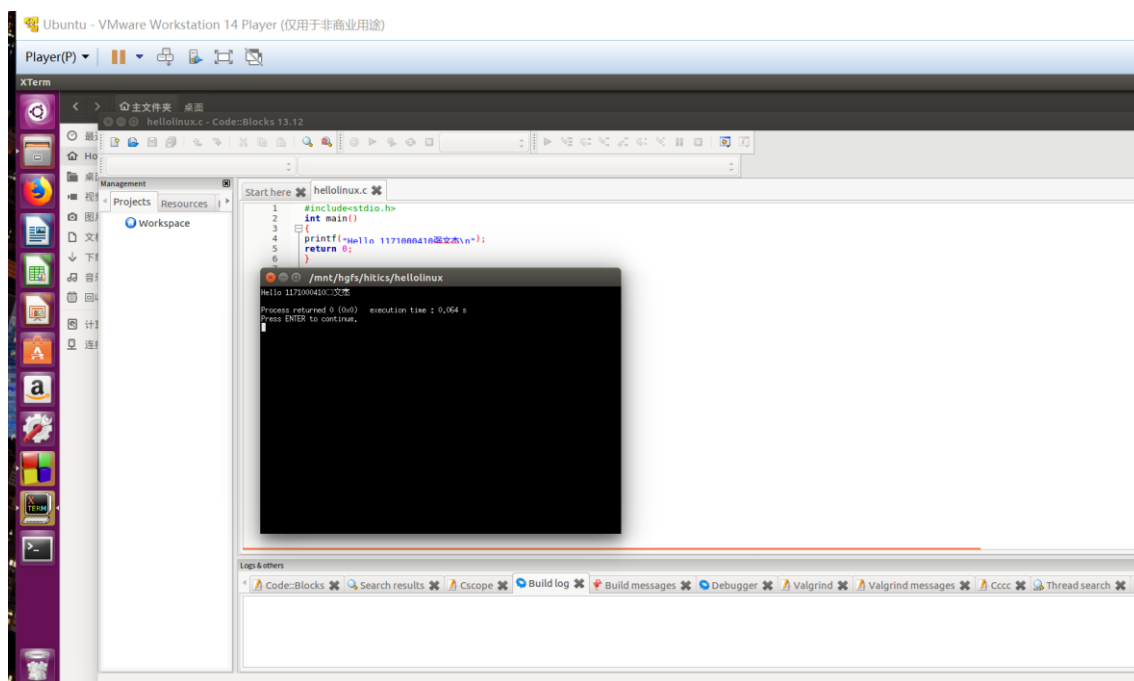


图 2-1 Ubuntu 下 CodeBlocks 截图

### 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。

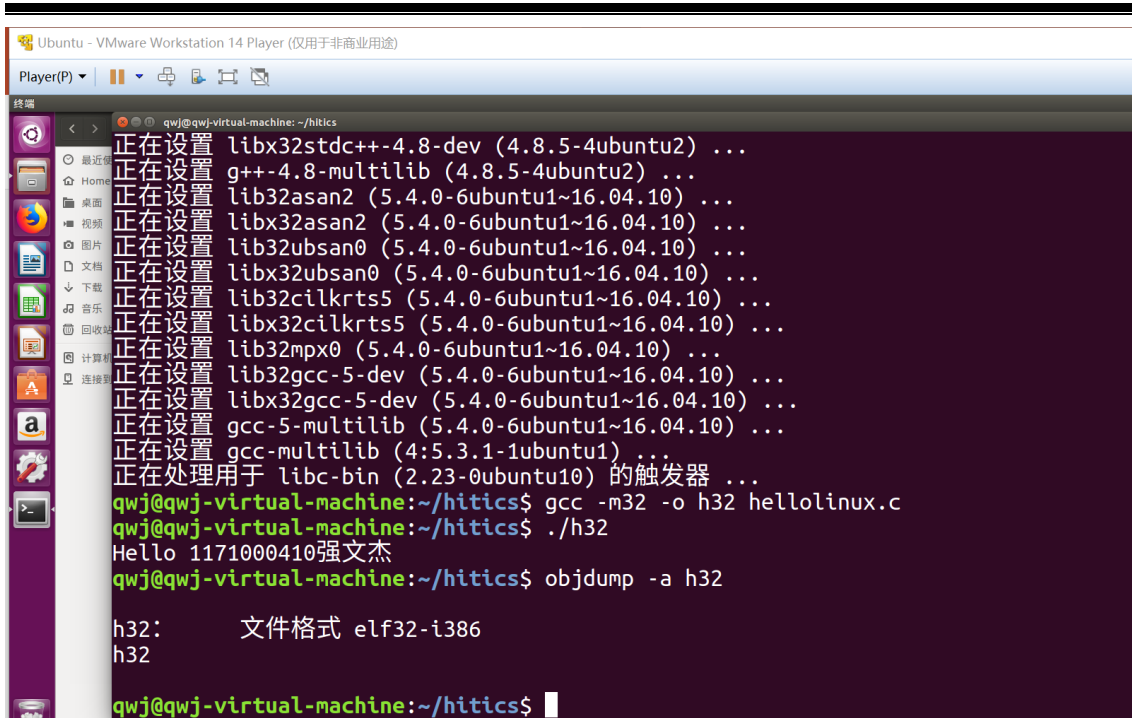


图 2-2 32 位运行环境建立

## 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）

||对应 OR:  $0x69 || 0x55 = 0x01$

&&对应 AND:  $0x69 \&\& 0x55 = 0x01$

! 对应 NOT:  $!0x41 = 0x00$

### 3.2 无符号数位操作（2 分）

取反:  $\sim(010000001) = 10111110$

与:  $(01101001) \& (01010101) = 01000001$

或:  $(01101001) | (01010101) = 01111101$

异或:  $01010101 \wedge 001100010 = 01101011$

移位: 对于无符号数, 左移右移都是逻辑移位, 即左移低位空出的补 0, 右移高位空出的补 0。

例: 无符号数  $x = 10010101$ ;

$x \gg 4 = 00001001$

### 3.3 有符号数位操作（2 分）

取反:  $\sim(010000001) = 10111110$

与:  $(01101001) \& (01010101) = 01000001$

或:  $(01101001) | (01010101) = 01111101$

异或:  $01010101 \wedge 001100010 = 01101011$

移位: 对于有符号数, 左移多出的会移进符号位, 右移是算术右移, 即空出的高位补符号位。

例: 有符号数  $x = 10010101$ ;

$x \gg 4 = 11111001$



## 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 SRC,DEST ; 将操作数相与, 返回 DEST

OR SRC,DEST ; 将操作数相或, 返回 DEST

NOT SRC; 将操作数 SRC 中每位取反

XOR SRC,DEST;将操作数相异或, 并返回给 DEST

TEST SRC,DEST;将操作数相与, 影响状态标志, 主要用于给数据转移指令传递状态标志。

例子: 假设寄存器%rax 的值为 x, %rdx 的值为 y

AND %rax, %rdx 指令执行后, %rdx 的值为 x&y

### 4.2 无符号数左右移 (2 分)

SHL k, DEST; 将操作数 DEST 左移 k 位

SHR k, DEST; 将操作数 DEST 右移 k 位

例如: 假设 AL= 01001101B

SHL 1, AL 指令执行后 AL=10011010B

### 4.3 有符号数左右移 (2 分)

SAL k, DEST; 将操作数 DEST 左移 k 位

SAR k, DEST; 将操作数 DEST 右移 k 位

例如 假设 AL=10011001B

SAR 1, AL 指令执行后 AL=11001100B

### 4.4 循环移位 (2 分)

ROL k, DEST; 把操作数的低位部分向高位方向循环移动 CL/imm 指定的位数,空出的低位部分由移出的高位部分来填充,同时,移出的高位部分仍然会存放在 CF 中;如果是循环左移 N 位,那么,就空出 N 个低位,移出 N 个高位,然后,把移出的这 N 个高位按照移出的顺序依次填入空出的 N 个低位中,同时,CF 中只保存最后一次移出的那一位的内容

ROR k, DEST; 把操作数的高位部分向低位方向循环移动 CL/imm 指定的位数,空出的高位部分由移出的低位部分来填充,同时,移出的低位部分仍然会存放在 CF 中;如果是循环右移 N 位,那么,就空出 N 个高位,移出 N 个低位,然后,把移出的这 N 个低位按照移出的顺序依次填入空出的 N 个高位中,同时,CF 中只保存最后一次移出

的那一位的内容

例如：假设当前，AL=01010011B，CF=1，则执行指令

ROL 1, AL 后，AL=10100110B，CF=0

## 4.5 带进位位的循环移位（2 分）

**RCL k, DEST**；把操作数的低位部分向高位方向循环移动 CL/imm 指定的位数，每向左移动一位，RCL 指令都会先把 CF 的原有值填充到空出的最低位上，再把移出的最高位存放到 CF 中；这样循环左移 N 位之后，CF 中保存的仍然是最后一次移出的那一位的内容

**RCR k, DEST**；把操作数的高位部分向低位方向循环移动 CL/imm 指定的位数，每向右移动一位，RCL 指令都会先把 CF 的原有值填充到空出的最高位上，再把移出的最低位存放到 CF 中；这样循环右移 N 位之后，CF 中保存的仍然是最后一次移出的那一位的内容

假设当前 AL=01010011B，CF=1

RCL 1, AL 后 RCL=10100111，CF=0

RCR 1, AL 后 RCR=10101001，CF=1

## 4.6 测试、位测试 BTx（2 分）

**BT（位测试）**

写法：BT REG16/MEM16,REG16/IMM8;或 BTREG32/MEM32,REG32/IMM8;

作用：将第一个操作数的第 n 位拷贝到进位标志 CF 中

**BTS（位测试并置位）**

写法：BTS REG16/MEM16,REG16/IMM8;或 BTSREG32/MEM32,REG32/IMM8;

作用：将第一个操作数的位 n 拷贝到进位标志中，同时 将位 n 置位

**BTR(位测试并复位)**

写法：BTR REG16/MEM16,REG16/IMM8;或 BTRREG32/MEM32,REG32/IMM8;

作用：将第一个操作数的位 n 拷贝到进位标志中，同时 将位 n 清零

**BTC(位测试并复位)**

写法：BTC REG16/MEM16,REG16/IMM8;或 BTCREG32/MEM32,REG32/IMM8;

作用：将第一个操作数的位 n 拷贝到进位标志中，同时 将位 n 取反

例子：假设(AX)=1234H

BT 2, AX; 指令执行后, CF=1, (AX)=1234H

## 4.7 条件传送 CMOV<sub>xx</sub> (2 分)

cmovcc src,dest

cc:表示条件

src: r16, r32, r64

dst: r/m16, r/m32, r/m64

### 无符号数的条件传送:

用 a、b、e、n、c 分别表示：大于、小于、等、否、进位

CMOVA/CMOVNB 大于/小于或不等于 (CF 或者 ZF)=0

CMOVAE/CMOVNB 大于或者等于/不小于 CF = 0

CMOVNC 无进位 CF = 0

CMOVNB/CMOVNAE 小于/不大于 CF = 1

CMOVC 进位 CF = 1

CMOVBE/CMOVNA 小于或者等于/不大于(CF 或 ZF) = 1

CMOVE/CMOVZ 等于/零 ZF = 1

CMOVNE/CMOVNZ 不等于/不为零 ZF = 0

CMOVP/CMOVPE 奇偶校验 PF = 1

### 有符号数的条件传送:

用 g、l、e、n、o 分别表示：大于、小于、等、否、溢出

CMOVG/CMOVNLE 大于 / 不小于等于 (ZF=0 and SF=OF)

CMOVGE/CMOVNL 大于等于 / 不小于 (SF 异域 OF) = 0

CMOVL/CMOVNGE 小于 / 不大于等于 (SF 民域 OF) = 1

CMOVLE/CMOVNG 小于等于 / 不大于 ((SF 异域 OF)或 ZF) =1

CMOVO 溢出 OF=1

CMOVNO 未溢出 OF=0

CMOVS 带符号 (负) SF=1

CMOVNS 无符号 (非负) SF=0

例子: `cmovge %r8, %r9`

`cmovgel %r9, %r10`

`cmovgl %r8d, %r10d`

`cmovll %r8d, %r10d`

例子: 假设%ecx 的值为 x, %edx 的值为 y, %ebx 的值为 y-x, %eax 的值为 x-y

`cmpl %edx, %ecx` //比较 x 和 y

`cmovl %ebx, %eax` //如果 x 小于 y, `eax=ebx=y-x`

## 4.8 条件设置 SETCxx (1 分)

指令	同义词	作用	设置条件
<code>sete</code>	<code>Setz</code>	ZF	相等 / 结果为 0
<code>setne</code>	<code>setnz</code>	$\sim$ ZF	不相等 / 结果不为 0
<code>sets</code>		SF	结果为负数
<code>setns</code>		$\sim$ SF	结果为非负数
<code>setl</code>	<code>setnge</code>	SF^OF	小于 (符号数)
<code>setle</code>	<code>setng</code>	(SF^OF) ZF	小于等于 (符号数)
<code>setg</code>	<code>setnle</code>	$\sim$ (SF^OF)& $\sim$ ZF	大于 (符号数)

setge	setnl	$\sim(\text{SF} \wedge \text{OF})$	大于等于 (符号数)
seta	setnbe	$\sim\text{CF} \& \sim\text{ZF}$	大于 (无符号数)
setae	setnb	$\sim\text{CF}$	大于等于 (无符号数)
setb	setnae	$\text{CF}$	小于 (无符号数)
setbe	setna	$\text{CF}   \text{ZF}$	小于等于 (无符号数)

具体操作如下：

SETcc DEST

Operation

IF condition THEN

DEST  $\leftarrow$  1;

ELSE

DEST  $\leftarrow$  0;

FI;

## 4.9 进位位操作（1 分）

adc src,dest 将 src 与 dest 相加，并且加上 CF，结果返回 dest  
例子：

```
mov 2, ax
mov 1, bx
sub ax, bx
adc 1, ax
```

执行后，(ax)=4.adc 执行时，相当于计算：(ax)+1+CF=2+1+1=4

## 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) {

    x=x>>1;

    x=x<<1;

    return x;

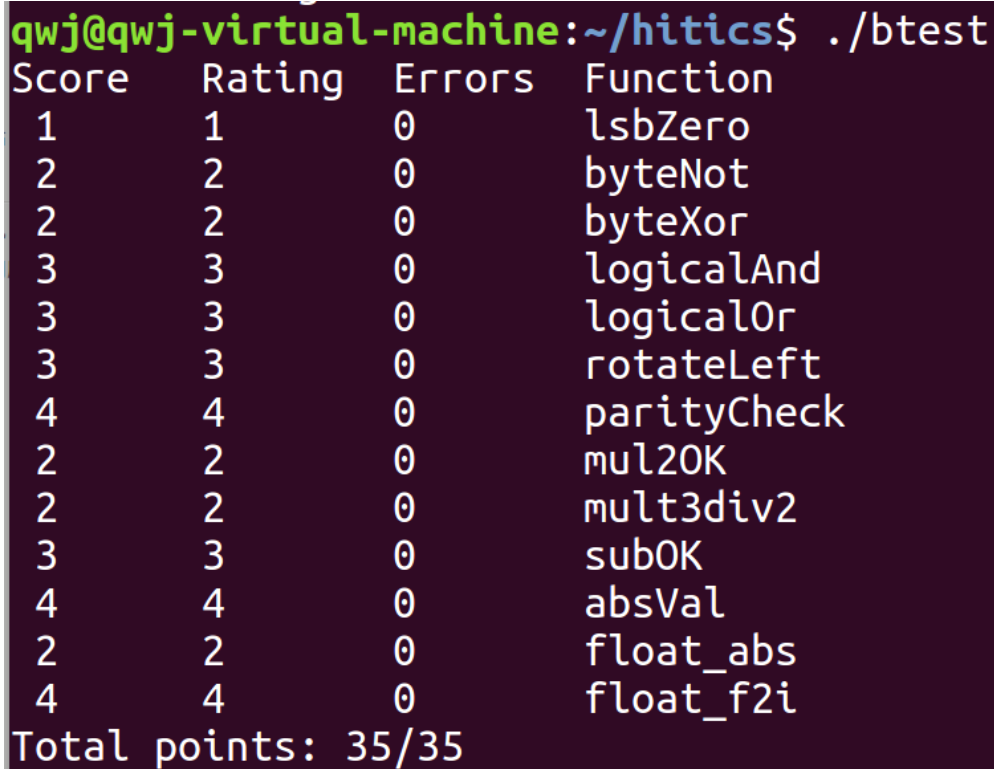
}

btest

截

图

:



```
qwj@qwj-virtual-machine:~/hitics$ ./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
Total points: 35/35
```

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

设计思想：先将  $x$  右移一位，再将  $x$  左移一位，实现将最后一位变为 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 y = 0xff;
```

```
    n=n<<3;
```

```
    y=y<<n;
```

```
    x=x^y;
```

```
    return x;
```

```
}
```

btest

截

图

:

```
qwj@qwj-virtual-machine:~/hitics$ ./btest -f byteNot
Score   Rating   Errors   Function
  2       2       0       byteNot
Total points: 2/2
```

设计思想： $A^0=A$ ,  $A^1 = \text{非 } A$ 。将  $x$  的第  $n$  个字节的每位都与 1 异或，实现取反。

## 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) {

```
    x=x>>n;
```

```
    y=y>>n;
```

```
    n=n<<3;
```

```
    x=x&0xff;
```

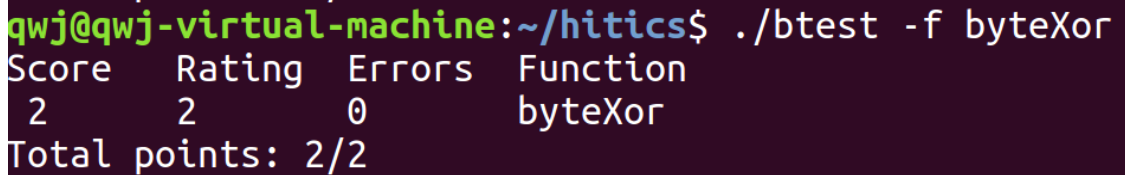
```
    y=y&0xff;
```

```

    return !(x^y);
}

```

btest 截图 图 :



```

qwj@qwj-virtual-machine:~/hitics$ ./btest -f byteXor
Score   Rating  Errors  Function
  2      2      0      byteXor
Total points: 2/2

```

设计思想：0^0=0, 1^1=0, 0^1=1。取出 x 和 y 的第 n 个字节进行异或，最后转化成逻辑上的 0 和 1 并返回。

## 5.4 函数 logicalAnd 的实现及说明函数

程序如下：int logicalAnd(int x, int y) {

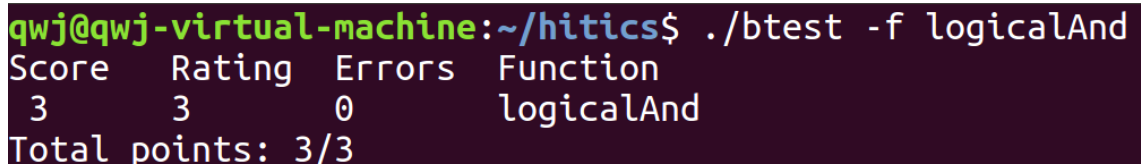
```

    x=!((!x)|(!y));

    return x;
}

```

btest 截图 图 :



```

qwj@qwj-virtual-machine:~/hitics$ ./btest -f logicalAnd
Score   Rating  Errors  Function
  3      3      0      logicalAnd
Total points: 3/3

```

设计思想：把 x 和 y 分别取 NOT，二者相或后再取 NOT，即可得到逻辑与。

## 5.5 函数 logicalOr 的实现及说明函数

程序如下：int logicalOr(int x, int y) {

```

    x=!(!x)|(!!y);

    return x;
}

```

btest 截图 图 :



```

qwj@qwj-virtual-machine:~/hitics$ ./btest -f logicalOr
Score   Rating  Errors  Function
  3      3      0      logicalOr
Total points: 3/3

```

设计思想：把 x 和 y 分别转化成逻辑的 0 和 1，在相或，实现逻辑或。

## 5.6 函数 rotateLeft 的实现及说明函数

程序如下：int rotateLeft(int x, int n) {

```

    int y;

    y=~((~0)<<n);

    x=(x<<n)+((x>>(32+(~n+1)))&y);

    return x;
}

```

btest                      截                      图                      :

```

qwj@qwj-virtual-machine:~/hitics$ ./btest -f rotateLeft
Score   Rating  Errors  Function
  3      3      0      rotateLeft
Total points: 3/3

```

设计思想：先构造 y 为高 (32-n) 位为 0 的 y，再与 x 右移 (32-n) 的 x 相与，相当于储存了 x 的高 n 位数，最后再与 x 左移 n 位相加即可。

## 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);

    x=x>>31;

```

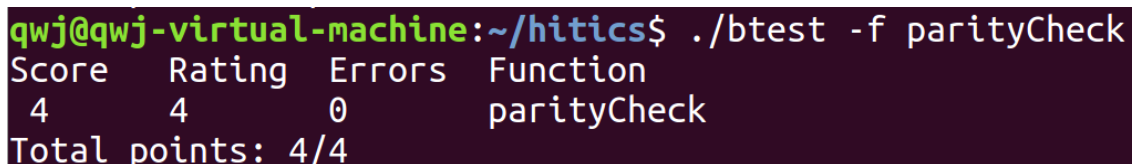
```

return !(!x);

}

```

btest 截图 图 :



```

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

```

设计思想：每次移位将  $x$  的低半位数与高半位数进行异或，实现得到的  $x$  位表示总是减去偶数个 1，并不影响  $x$  位表示中 1 个数的奇偶性。最后把得到的  $x$  右移 31 位，并变成逻辑 1 和 0。

## 5.8 函数 mul20K 的实现及说明函数

程序如下：int mul20K(int x) {

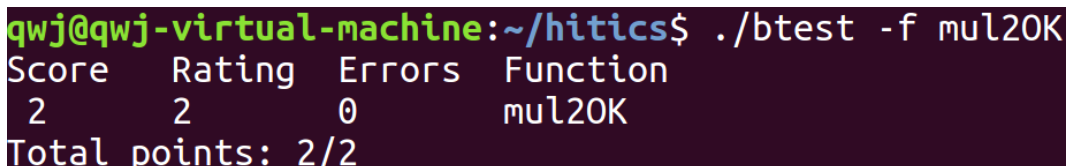
```

    return (((x>>31)&0x1)^((x>>30)&0x1))^0x1;

}

```

btest 截图 图 :



```

qwj@qwj-virtual-machine:~/hitics$ ./btest -f mul20K
Score Rating Errors Function
2      2      0      mul20K
Total points: 2/2

```

设计思想：若  $x$  的符号位为 1，则第 30 位为 0 时会溢出；若  $x$  符号位为 0，则第 30 位为 1 时会溢出；于是将  $x$  的 31 位和 30 位分别与 1 相与，再异或，再与 1 相与即可。

## 5.9 函数 mult3div2 的实现及说明函数

程序如下：int mult3div2(int x) {

```

    x=x+(x<<1);

    x=(x>>1)+(((x>>31)&0x1)&(((x<<31)>>31)&0x1));

    return x;

}

```



```

    return x;
}

```

btest 截图 图 :

```

qwj@qwj-virtual-machine:~/hitics$ ./btest -f absVal
Score  Rating  Errors  Function
  4      4      0      absVal
Total points: 4/4

```

设计思想：当  $x$  符号位为 0 时， $x$  绝对值就是  $x$ ；当  $x$  符号位为 1 时， $x$  绝对值是  $(\sim x + 1)$ 。

## 5.12 函数 float\_abs 的实现及说明函数

程序如下：unsigned float\_abs(unsigned uf) {

```
    int x=uf&0x7fffffff;
```

```
    if(x>0x7f800000)
```

```
        return uf;
```

```
    else
```

```
        return x;
```

```
}
```

btest 截图 图 :

```

qwj@qwj-virtual-machine:~/hitics$ ./btest -f float_abs
Score  Rating  Errors  Function
  2      2      0      float_abs
Total points: 2/2

```

设计思想：NaN 表示的数阶码全为 1，且小数域为非 0。将  $uf$  的符号位变为 0，得到的数若比  $0x7f800000$  大，即为 NaN。

## 5.13 函数 float\_f2i 的实现及说明函数

程序如下：int float\_f2i(unsigned uf) {

```
    int x,y;
```

```
    unsigned mini=0x80000000;
```

```
x=(uf>>23)&0xff;
y=(uf&0x007fffff)^0x00800000;
if(x>158)
{
    return mini;
}
if(x<127)
{
    return 0;
}
else if(((uf>>31)&0x1)==1)
{
    if(x>150)
        return ((~(y<<(x-150)))+1);
    else
        return ((~(y>>(150-x)))+1);
}
else
{
    if(x>150)
        return (y<<(x-150));
    else
        return (y>>(150-x));
}
}
```

btest

截

图

:

```
qwj@qwj-virtual-machine:~/hitics$ ./btest -f float_f2i
Score   Rating  Errors  Function
  4      4      0      float_f2i
Total points: 4/4
```

设计思想：将 `uf` 左移 23 位并与 `0xff` 相与来取出阶码域；设置 `y` 来取出小数域，并令其第 23 位为 1，为了规格化的值隐含的以 1 开头的表示。

当阶码的值 `E` 大于 31 时，表示 `float` 转化为 `int` 会溢出，或者出现阶码域全为 1 的情况；当阶码的值小于 0，返回 0；当 `uf` 最高位为 1 时，判断 `x` 与 150 的大小，决定移位的方向，并以补码形式将数值存储；当 `uf` 最高位为 0 时，判断 `x` 与 150 大小，决定移位的方向。

## 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

系统学习了汇编语言指令；  
熟练掌握位运算和逻辑运算来实现函数；  
通过 C 程序深入理解计算机底层实现与优化。

### 10.2 请给出对本次实验内容的建议

关于第三章写出 C 语言例句和第四章写出汇编语言例句，题目中未给出范例，因此关于例句的形式易产生困惑。

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>.