

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 1160300901

班 级 1603009

学 生 孙月晴

指 导 教 师 吴锐

实 验 地 点 G712

实 验 日 期 2017-10-17

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）.....	- 9 -
第 4 章 汇编语言的位操作指令.....	- 10 -
4.1 逻辑运算(1 分).....	- 10 -
4.2 无符号数左右移（2 分）.....	- 10 -
4.3 有符号左右移（2 分）.....	- 11 -
4.4 循环移位（2 分）.....	- 11 -
4.5 带进位位的循环移位（2 分）.....	- 11 -
4.6 测试、位测试 BTx（2 分）.....	- 11 -
4.7 条件传送 CMOVxx（2 分）.....	- 12 -
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.....	- 14 -
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.....	- 15 -
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.....	- 16 -

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.....	- 17 -
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）	- 20 -
第 6 章 总结.....	- 21 -
10.1 请总结本次实验的收获.....	- 21 -
10.2 请给出对本次实验内容的建议.....	- 21 -
参考文献.....	- 22 -

第 1 章 实验基本信息

1.1 实验目的

- 1.熟练掌握计算机系统的数据表示与数据运算
- 2.通过 C 程序深入理解计算机运算器的底层实现与优化
- 3.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

1.2.2 软件环境

Windows10 64 位; Vmware 12; Ubuntu 16.04 LTS 64 位;

1.2.3 开发工具

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

1.3 实验预习

(1) 写出 C 语言下的位操作指令:

逻辑: ||、&&、!

无符号: 按位与&、或|、异或^、取反运算~、左移<<、逻辑右移>>

有符号: 按位与&、或|、异或^、取反运算~、左移<<、逻辑右移>>

(2) 写出汇编语言下的位操作指令:

逻辑运算: 逻辑与运算 AND、逻辑或运算 OR、逻辑非运算 NOT、逻辑异或运算 XOR、测试指令 TEST

无符号:

1. SHL k,D ; $D \leftarrow D \ll k$ 逻辑左移指令, 低位用 0 补齐
2. SHR k,D ; $D \leftarrow D \gg k$ 逻辑右移指令, 高位用 0 补齐

有符号:

- 1 SAL k,D ; $D \leftarrow D \ll k$ 算术左移指令, 低位用 0 补齐
- 2 SAR k,D ; $D \leftarrow D \gg k$ 算术右移指令, 高位和原来一样

测试、位测试 BTx:

- 1 BT: 把指定的二进制位传送给 CF;
- 2 BTC: 把指定的二进制位传送给 CF 之后, 还要使该位变反;
- 3 BTR: 把指定的二进制位传送给 CF 之后, 还要使该位变 0;
- 4 BTS: 把指定的二进制位传送给 CF 之后, 还要使该位变 1;

条件传送 CMOVxx

- | | | | | |
|------------|------------|-----------|-----------|------------|
| 1 cmovz | 2 cmovz | 3 cmovne | 4 cmovnz | 5 cmovs |
| 6 cmovns | 7 cmovg | 8 cmovnl | 9 cmovge | 10 cmovnl |
| 11 cmovl | 12 cmovnge | 13 cmovle | 14 cmovng | 15 cmova |
| 16 cmovnbe | 17 cmovae | 18 cmovnb | 19 cmovb | 20 cmovnae |
| 21 cmovbe | 22 cmovna | | | |

条件设置 SETCxx

- | | | | | |
|----------------------|----------------|----------------|--------------------|--------|
| 1 SETZ/SETE | 2 SETNZ/SETNE | 3 SETS | 4 SETNS | 5 SETO |
| 6 SETNO | 7 SETP/SETPE | 8 SETNP/SETPO | 9 SETC/SETB/SETNAE | |
| 10 SETNC/SETNB/SETAE | 11 SETNA/SETBE | 12 SETA/SETNBE | | |
| 13 SETL/SETNGE | 14 SETNL/SETGE | 15 SETLE/SETNG | | |
| 16 SETNLE/SETG | | | | |

进位位操作:

进位 CF 操作指令

清进位指令 CLC: $CF \leftarrow 0$

置进位指令 STC: $CF \leftarrow 1$

进位取反指令 CMC: $CF \leftarrow \text{not } CF$

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

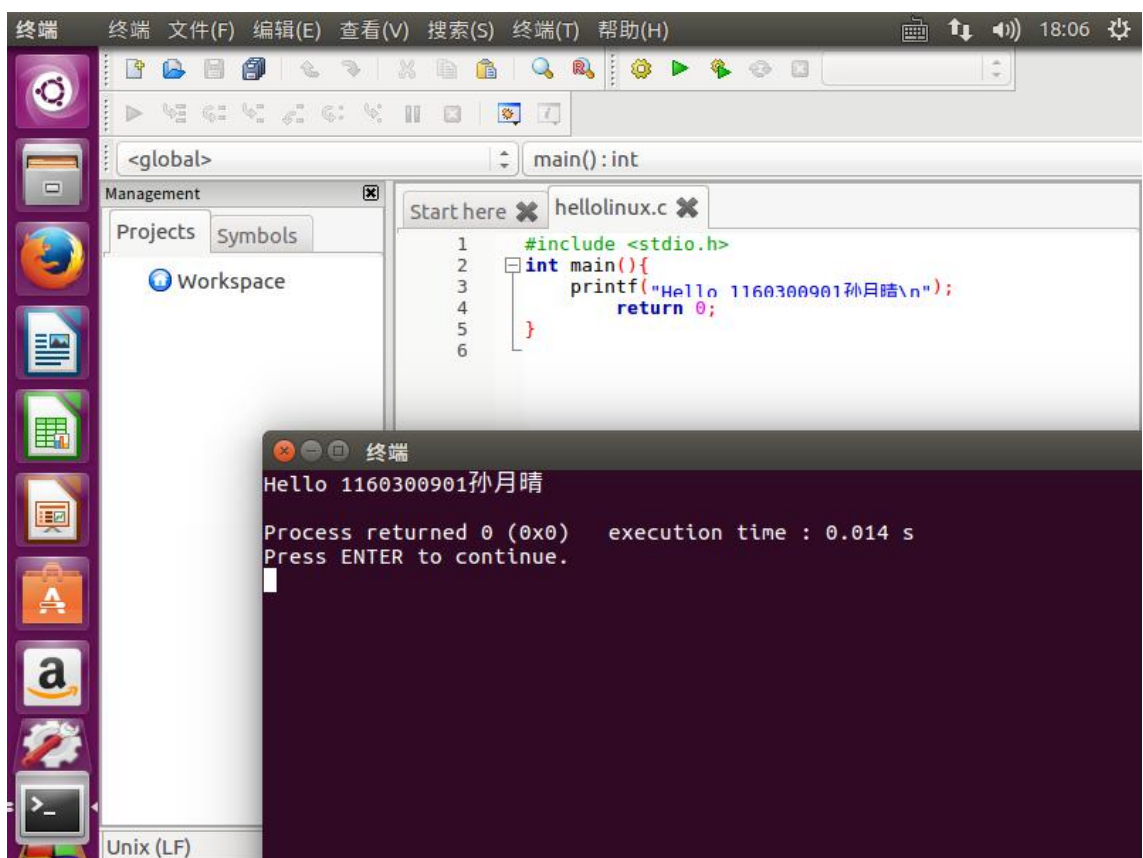


图 2-1 Ubuntu 下 CodeBlocks 截图

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。

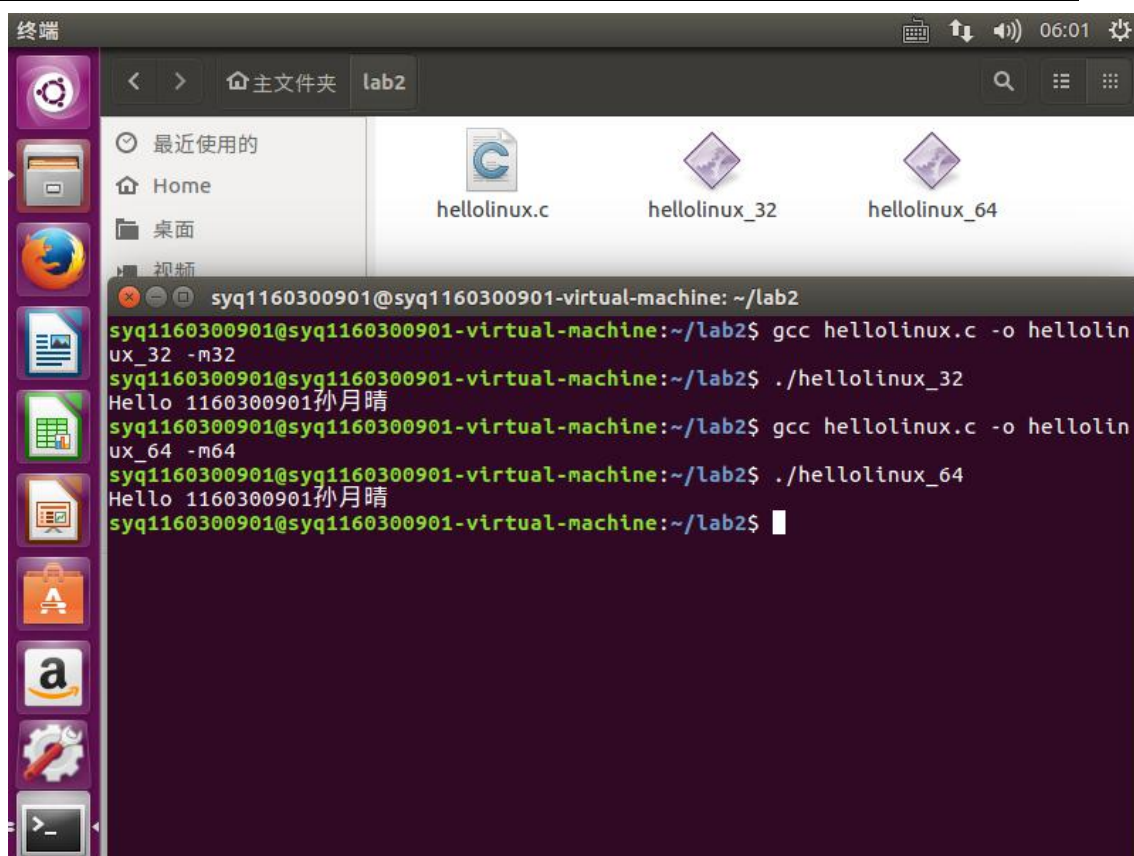


图 2-2 32 位运行环境建立

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)

C 语言提供了一组逻辑运算符`||`、`&&`、和`!`，分别对应命题逻辑的 OR、AND、和 NOT 运算。

例句：

表达式	结果
<code>!0x41</code>	<code>0x00</code>
<code>0x12&&0x20</code>	<code>0x01</code>
<code>0x44 0x23</code>	<code>0x01</code>

3.2 无符号数位操作 (2 分)

按位与`&`、或`|`、异或`^`、取反运算`~`、左移`<<`、逻辑右移`>>`、掩码运算，例子如表所示：

$A = 10101011_2$, $B = 10010110_2$

表达式	结果
<code>A&B</code>	<code>10000010</code>
<code>A B</code>	<code>10111111</code>
<code>A^B</code>	<code>00111101</code>
<code>~A</code>	<code>01010100</code>
<code>A<<4</code>	<code>10110000</code>
<code>A>>4</code>	<code>00001010</code>

掩码运算：取出 8~15 位

```
unsigned int a, b, mask = 0x0000ff00;
a = 0x12345678;
b = (a & mask) >> 8; /* 0x00000056 */
```


3.3 有符号数位操作（2分）

按位与、或、异或、取反运算、左移、逻辑右移、掩码运算，例子如表所示：

$A=10101011_2$, $B=10010110_2$

表达式	结果
$A \& B$	10000010
$A B$	10111111
$A \wedge B$	00111101
$\sim A$	01010100
$A \ll 4$	10110000
$A \gg 4$	00001010

掩码运算：取出 8~15 位

```
unsigned int a, b, mask = 0x0000ff00;  
a = 0x12345678;  
b = (a & mask) >> 8; /* 0x00000056 */
```

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)

1 逻辑与运算 AND SRC,DEST ; 将操作数相与, 返回给 DEST。CF,OF 是 0, 影响 ZF,SF,PF。

例: andl \$252645135, %edi

2 逻辑或运算 OR SRC,DEST ; 将操作数相或, 返回给 DEST。CF,OF 是 0, 影响 ZF,SF,PF。

例: orq %rsi, %rdi

3 逻辑非运算 NOT 操作数 ; 将操作数按位取反。不影响标志位。

例: notq %rdi

4 逻辑异或运算 XOR SRC, DEST ; 将操作数相异或, 返回给 DEST。CF,OF 是 0, 影响 ZF,SF,PF。

例: xorq %rsi, %rdi

5 测试指令 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: testw %di, %di

4.2 无符号数左右移 (2 分)

1 SHL k,D ; $D \leftarrow D \ll k$ 逻辑左移指令, 低位用 0 补齐

例: shlq \$4, %rax

2 SHR k,D ; $D \leftarrow D \gg k$ 逻辑右移指令, 高位用 0 补齐

例: shrq \$4, %rax

4.3 有符号左右移 (2 分)

1 SAL k,D ; $D \leftarrow D \ll k$ 算术左移指令, 低位用 0 补齐

例: salq \$4, %rax

2 SAR k,D ; $D \leftarrow D \gg k$ 算术右移指令, 高位和原来一样

例: sarq \$4, %rax

4.4 循环移位 (2 分)

1 ROL DEST, SRC ; 不带进位的循环左移指令, 移出的数进行循环

例: rol %ax, 1

2 ROR DEST, SRC ; 不带进位的循环右移指令, 移出的数进行循环

例: ror %ax, 2

3 RCL DEST, SRC ; 带进位的循环左移指令, 将 CF 顶进循环中

例: rcll %ax, 3

4 RCR DEST, SRC ; 带进位的循环右移指令, 将 CF 顶进循环中

例: rcr %ax, 1

4.5 带进位位的循环移位 (2 分)

1 RCL DEST, SRC ; 带进位的循环左移指令, 将 CF 顶进循环中

例: rcl %ax, 2

2 RCR DEST, SRC ; 带进位的循环右移指令, 将 CF 顶进循环中

例: rcr %ax, 4

4.6 测试、位测试 BTx (2 分)

1 BT: 把指定的二进制位传送给 CF;

例: bt %dx, 7

2 BTC: 把指定的二进制位传送给 CF 之后, 还要使该位变反;

例: btc %dx, 0

3 BTR: 把指定的二进制位传送给 CF 之后, 还要使该位变 0;

例: `btr %dx,7`

4 BTS: 把指定的二进制位传送给 CF 之后, 还要使该位变 1;

例: `bts %dx,6`

5 `testb`: 测试字节

例: `testb %rbx,%rax`

6 `testw` 测试字

例: `testw %rbx,%rax`

7 `testl` 测试双字

例: `testl %rbx,%rax`

8 `testq` 测试四字

例: `testq %rbx,%rax`

4.7 条件传送 `CMOVxx` (2 分)

1 `cmove/cmovz` S, D 等于 0 时传送 例: `cmove %rdx,%rax`

2 `cmovne/cmovnz` S, D 不等于 0 时传送 例: `cmovne %rdi,%rax`

3 `cmovs` S, D 负数时传送 例: `cmovs %rdi,%rax`

4 `cmovns` S, D 非负数时传送 例: `cmovns %rdi,%rax`

5 `cmovg/cmovnle` S, D 有符号大于时传送 例: `cmovg %rdi,%rax`

6 `cmovge/cmovnl` S, D 有符号大于等于时传送 例: `cmovge %rdx,%rax`

7 `cmovl/cmovnge` S, D 有符号小于时传送 例: `cmovl %rdi,%rax`

8 `cmovle/cmovng` S, D 有符号小于等于时传送 例: `cmovle %rdi,%rax`

9	cmova/cmovnbe	S, D	无符号大于时传送	例: cmova %rdi,%rax
10	cmovae/cmovnb	S, D	无符号大于等于时传送	例:cmovae %rdi,%rax
11	cmovb/cmovnae	S, D	无符号小于时传送	例:cmovb %rdi,%rax
12	cmovbe/cmovna	S, D	无符号小于等于时传送	例:cmovbe %rdi,%rax

4.8 条件设置 SETCxx (1 分)

指令助记符	操作数与检测条件之间的关系	例句
1 SETZ/SETE	reg/mem = ZF	例:sete %al
2 SETNZ/SETNE	reg/mem = not ZF	例:setne %al
3 SETS	reg/mem = SF	例:sets %dl
4 SETNS	reg/mem = not SF	例:sete %dl
9 SETC/SETB/SETNAE	reg/mem = CF	例:setc %al
10 SETNC/SETNB/SETAE	reg/mem = not CF	例:setae %al
11 SETNA/SETBE	reg/mem = (CF or ZF)	例:sete %dl
12 SETA/SETNBE	reg/mem = not (CF or ZF)	例:seta %al
13 SETL/SETNGE	reg/mem = (SF xor OF)	例:setl %al
14 SETNL/SETGE	reg/mem = not (SF xor OF)	例:setnl %al
15 SETLE/SETNG	reg/mem = (SF xor OF) or ZF	例:setle %al
16 SETNLE/SETG	reg/mem = not ((SF xor OF) or ZF)	例:setg %al

4.9 进位位操作 (1 分)

进位 CF 操作指令

清进位指令 CLC: $CF \leftarrow 0$

置进位指令 STC: $CF \leftarrow 1$

进位取反指令 CMC: $CF \leftarrow \text{not } CF$

Adc %rbx,%rax 将%rbx 与%rax 相加, 并且加上 CF

第 5 章 BITS 函数实验与分析

每题 8 分，总分不超过 80 分

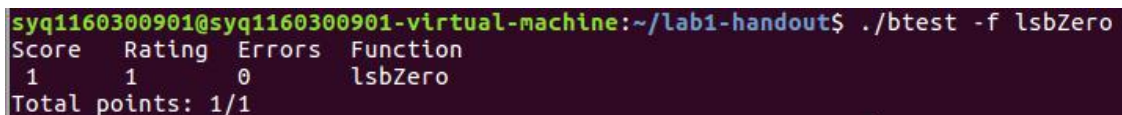
截图： `$. /btest -f 函数名`

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：



```
syq1160300901@syq1160300901-virtual-machine:~/lab1-handout$ ./btest -f lsbZero
Score Rating Errors Function
1      1      0      lsbZero
Total points: 1/1
```

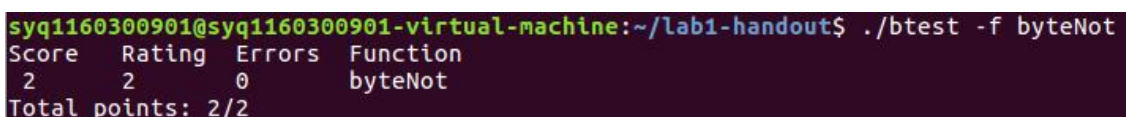
设计思想：我最开始想到的是 $x \wedge 0x1$ ，因为与 0 异或不变，与 1 异或取补，但是当最后一位是 0 时就不适用了，所以我改用了移位运算符，x 先右移一位去掉原来的最低位，再左移一位，使最低位为 0。

5.2 函数 byteNot 的实现及说明函数

程序如下：

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

btest 截图：



```
syq1160300901@syq1160300901-virtual-machine:~/lab1-handout$ ./btest -f byteNot
Score Rating Errors Function
2      2      0      byteNot
Total points: 2/2
```

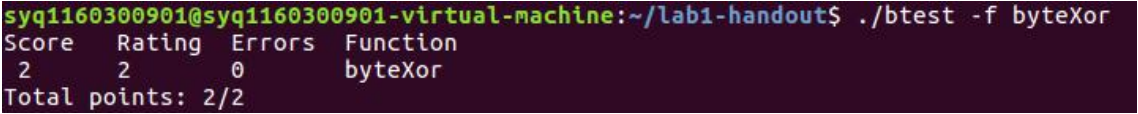
设计思想：这个函数的功能是指定字节取反。使用掩码运算，定义一个常量 `var=0xFF`，左移 `n*8` 位，得到 `0xFF00`，由于与 1 异或取补，与 0 异或不变，则达到了按位取反的目的。

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

btest 截图：



```
syq1160300901@syq1160300901-virtual-machine:~/lab1-handout$ ./btest -f byteXor
Score  Rating  Errors  Function
2      2        0      byteXor
Total points: 2/2
```

设计思想：函数的意思是 `x` 和 `y` 的指定字节相同返回 0，否则返回 1。首先想到的是 `x` 与 `y` 做异或，相同为 0，不同为 1，然后右移 `8*n` 位，再 `&0xFF`，把 `x` 和 `y` 的第 `n` 个字节取出来，再用两次 NOT 运算，转换为逻辑的 0 和 1，返回即可。

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：

```

syq1160300901@syq1160300901-virtual-machine:~/lab1-handout$ ./btest -f logicalAnd
Score  Rating  Errors  Function
3      3      0      logicalAnd
Total points: 3/3

```

设计思想：函数的功能是取 x 和 y 的逻辑与，即 C 语言中的 $\&\&$ 运算。当 $x \neq 0$ 时， $!!x=1$ ，当 $x=0$ 时， $!!x=0$ ，两次逻辑非运算把一个数转换成了逻辑的 0 和 1，然后再用布尔运算 $\&$ 即 $(!!x) \& (!!y)$ ，只有同时为 1 时返回 1，否则返回 0，实现了逻辑与运算。

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：

```

syq1160300901@syq1160300901-virtual-machine:~/lab1-handout$ ./btest -f logicalOr
Score  Rating  Errors  Function
3      3      0      logicalOr
Total points: 3/3

```

设计思想：与第 4 题逻辑与的操作类似， $(!!x) | (!!y)$ 用布尔运算 $|$ ，只有同时为 0 时返回 0，否则返回 1，实现了逻辑或运算。

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, temp, low;
    high = x << n;
    temp = ~(((1 << 31) >> 31) << n);
    low = (x >> (32 + (~n + 1))) & temp;
    return (high + low);
}

```


btest 截图：

```
syq1160300901@syq1160300901-virtual-machine:~/lab1-handout$ ./btest -f rotateLeft
t
Score   Rating  Errors  Function
3       3       0      rotateLeft
Total points: 3/3
```

设计思想：该函数的功能是把数 x 前 n 位移到 x 的后面。首先想到返回结果肯定是高位+低位的形式，于是 $high = x \ll n$ 取出高位， x 右移 $32-n$ 位得到低位，但这时还不能直接加，还需要把高 n 位置 0，即先构造低 n 位为 1，高 $(32-n)$ 位为 0 的数 $temp$ ， $low = (x \gg (32 - n)) \& temp$ ，返回 $high + low$ 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 >> 16) ^ x;
    x = (x >> 8) ^ x;
    x = (x >> 4) ^ x;
    x = (x >> 2) ^ x;
    x = (x >> 1) ^ x;
    return x & 0x1;
}
```

btest 截图：

```
syq1160300901@syq1160300901-virtual-machine:~/lab1-handout$ ./btest -f parityCheck
ck
Score   Rating  Errors  Function
4       4       0      parityCheck
Total points: 4/4
```

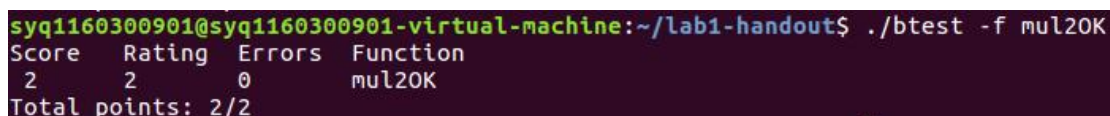
设计思想：函数的功能如果是如果 x 中有奇数个 1 则返回 1，否则返回 0。首先想到的是利用异或运算，相同为 0 不同为 1，然后想到把数逐渐折叠，会消去偶数个 1，即：把 32 位整数折叠成为 16 位整数进行运算，各对应 bit 若同时为 1 则结果为 0，其中一个为 1 则结果为 1，所以折叠的结果是会消去偶数个 1，而不改变 1 的个数的奇偶性，然后再折叠得到 8 位数、4 位数、2 位数，最后得到 1 位数就是奇偶校验位了，最后按位与运算，转化成逻辑的 0 或 1 并返回。

5.8 函数 mul20K 的实现及说明函数

程序如下：

```
int mul20K(int x)
{
    int m;
    m = ((x >> 31) & 0x1) ^ (((x<<1) >> 31) & 0x1);
    return m^0x1 ;
}
```

btest 截图：



```
syq1160300901@syq1160300901-virtual-machine:~/lab1-handout$ ./btest -f mul20K
Score Rating Errors Function
2      2      0      mul20K
Total points: 2/2
```

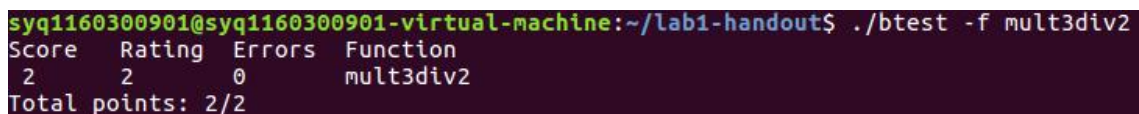
设计思想：该函数的功能是判断 $2*x$ 是否溢出，若溢出则返回 0，否则返回 1。首先只有当 x 与 $2*x$ 的符号位不相同时才能判断为溢出， x 的符号位为 $((x \gg 31) \& 0x1)$ ， $2*x$ 的符号位为 $((x \ll 1) \gg 31) \& 0x1$ ，两者做异或，相同为 0，不同为 1，由于溢出则返回 0，不溢出返回 1，则返回 $!m$ ，但题目中要求不能使用 $!$ ，所以返回 m^0x1 。

5.9 函数 mult3div2 的实现及说明函数

程序如下：

```
int mult3div2(int x)
{
    int y = (x << 1) + x;
    int temp = ((y >> 31) & 1) & (((y << 31) >> 31) & 1);
    y = (y >> 1) + temp;
    return y;
}
```

btest 截图：



```
syq1160300901@syq1160300901-virtual-machine:~/lab1-handout$ ./btest -f mult3div2
Score Rating Errors Function
2      2      0      mult3div2
Total points: 2/2
```

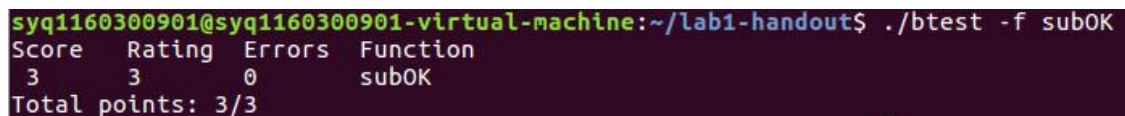
设计思想：该函数的功能是实现 $3*x/2$ 。首先令 $y = (x \ll 1) + x$ ，即 $3*x$ ，其次考虑 x 为负数的情况，补码的除法向上舍入，需要加上一个偏置量，即 $(1 \ll k) - 1$ 这里 $k=2$ ，这里当 x 为负数且 x 为奇数时才需加上偏置量，即 x 的最低位和最高位同时为 1，偏置量 $temp = ((y \gg 31) \& 1) \& (((y \ll 31) \gg 31) \& 1)$ 。

5.10 函数 subOK 的实现及说明函数

程序如下：

```
int subOK(int x, int y)
{
    int mask_x = (x >> 31) & 0x1;
    int mask_y = (y >> 31) & 0x1;
    int z=(mask_x ^ mask_y) & (mask_x ^ (((x + (~y + 1)) >> 31)) &
0x1);
    return !z;
}
```

btest 截图：



```
syq1160300901@syq1160300901-virtual-machine:~/lab1-handout$ ./btest -f subOK
Score  Rating  Errors  Function
3       3       0      subOK
Total points: 3/3
```

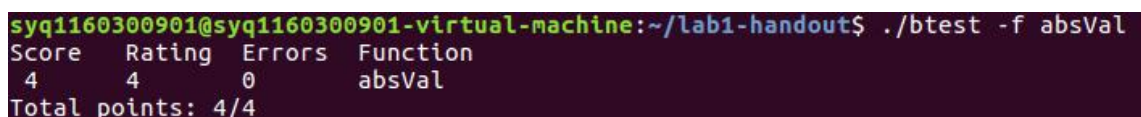
设计思想：这个函数的功能是判断 $x-y$ 是否溢出，溢出则返回 0，否则返回 1。只有当 x 的最高位和 y 的最高位不同且 x 的最高位和 $x-y$ 的最高位也不同时，才能判断为溢出。 $\text{mask_x} = (x \gg 31) \& 0x1$ 为取出 x 的符号位， $\text{mask_x} = (x \gg 31) \& 0x1$ 为取出 y 的符号位， $(x + (\sim y + 1)) = x - y$ ， $((x + (\sim y + 1)) \gg 31) \& 0x1$ 为取出 $x-y$ 的符号位，分别异或之后再与，再由溢出则返回 0，否则返回 1，返回 $!z$ 即可。

5.11 函数 absVal 的实现及说明函数

程序如下：

```
int absVal(int x)
{
    int sign_x = x >> 31;
    return (x ^ sign_x) + (1 + (~sign_x));
}
```

btest 截图：



```
syq1160300901@syq1160300901-virtual-machine:~/lab1-handout$ ./btest -f absVal
Score  Rating  Errors  Function
4       4       0      absVal
Total points: 4/4
```

设计思想：该函数的功能是求 x 得绝对值。首先想到的是先判断 x 是正数还是负数，于是将右移，若 x 为正数， $\text{sign_x}=0x00000000$ ，若 x 为负数，算术

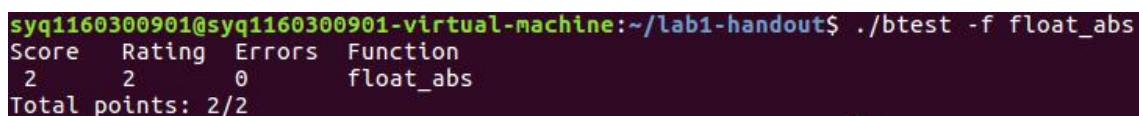
右移, 则 $\text{sign_x} = 0xFFFFFFFF$, 由于任何数与 0 异或不变, 与 1 异或取反, 当 x 为正数时, $x \wedge \text{sign_x} = x$, x 为负数时, $x \wedge \text{sign_x} = -x - 1$, 这时整数+0、负加 1 就可以直接输出了, 这个加 0 加 1 操作就由 $(\sim \text{sign_x}) + 1$ 完成, 当 x 为正数时, $(\sim \text{sign_x}) + 1 = 0$, 当 x 为负数时, $(\sim \text{sign_x}) + 1 = 1$ 。

5.12 函数 float_abs 的实现及说明函数

程序如下:

```
unsigned float_abs(unsigned uf)
{
    int temp = ~(1 << 31);
    int x = uf & temp;
    if (x > 0x7f800000)
        return uf;
    else
        return x;
}
```

btest 截图:



```
syq1160300901@syq1160300901-virtual-machine:~/lab1-handout$ ./btest -f float_abs
Score  Rating  Errors  Function
2       2        0      float_abs
Total points: 2/2
```

设计思想: 该函数的功能为返回浮点参数 f 的绝对值, 参数和结果都以 unsigned int 的形式传递, 但它们将被解释为单精度浮点值的位级表示, 当参数为 NaN 时, 返回参数。参数为 NaN 的分界线为 $01111111100000000000000000000000_2$ 即 $0x7f800000$, 最高位取 0 即 $x = uf \& (\sim(1 \ll 31))$, 当 $x > 0x7f800000$ 时返回参数 uf , 否则返回 x 。

5.13 函数 float_f2i 的实现及说明函数 (未做)

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

- 1.此次实验主要考查的是对数据的处理，对此需要掌握数据在机器中的表示，运用合理的位运算来实现相应的功能。
- 2.通过本次实验，我熟练掌握了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
- 3.熟悉掌握了 Linux 下 makefile 与 GDB 的使用

10.2 请给出对本次实验内容的建议

有些地方设置不合理，如图 3.2 和 3.3 位操作不区分有符号和无符号。

3.2 无符号数位操作 (2 分)

3.3 有符号数位操作 (2 分)



参考文献

- [1] 大卫 R.奥哈拉伦，兰德尔 E。布莱恩特. 深入理解计算机系统[M]. 机械工业出版社.2017.7
- [2] KANAMORI H. Shaking Without Quaking[J]. Science， 1998， 279（5359）： 2063-2064.
- [3] 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>.