哈爾濱工業大學

实验报告

实 验（二）

题　　目 DataLab 数据表示

专　　业 计算机类

学　　号 1170500913

班　　级 1703002

学　　生 熊健羽

指 导 教 师 史先俊

实 验 地 点 G712

实 验 日 期 2018.09.25

计算机科学与技术学院

目 录

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

CPU：Intel(R) Core(TM) i5-7200U @ 2.50GHz (64 位)

GPU：Intel(R) HD Graphics 620
　　Nvidia GeForce 940MX

物理内存：8.00GB

磁盘：1TB HDD
　　128GB SSD

1.2.2 软件环境

Windows10 64 位；
Vmware 14.11；
Ubuntu 18.04 64 位；

1.2.3 开发工具

Visual Studio 2010 64 位；

Code::Blocks；

gedit，gcc，notepad++；

1.3 实验预习

1) 写出 C 语言下的位操作指令：

逻辑运算：逻辑或||、逻辑与&&、逻辑非!
无符号：按位与&、按位或|、按位异或^、按位取反~、左移<<、逻辑右移>>
有符号：按位与&、按位或|、按位异或^、按位取反~、左移<<、算数右移>>

2) 写出汇编语言下的位操作指令：
逻辑运算：与运算 AND，或运算 OR，非运算 NOT，异或运算 XOR

无符号：逻辑左移 SHL ，逻辑右移 SHR

有符号：算术左移 SAL，算术右移 SAR

测试、位测试 BTx：BT，BTC，BTR，BTS，TEST

条件传送 CMOVxx：
cmove/cmovz
cmovne/cmovnz
cmovs
cmovns
cmovg/cmovnle
cmovge/cmovnl
cmovl/cmovnge
cmovle/cmovng
cmova/cmovnbe
cmovae/cmovnb
cmovb/cmovnae
cmovbe/cmovna
条件设置 SETCxx：
sete
setne
sets
setns
setg
setge
setl
setle
seta
setae
setb
setbe
进位位操作：
清进位指令 CLC，置进位指令 STC，进位取反指令 CMC, 带进位的加法指令 ADC

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

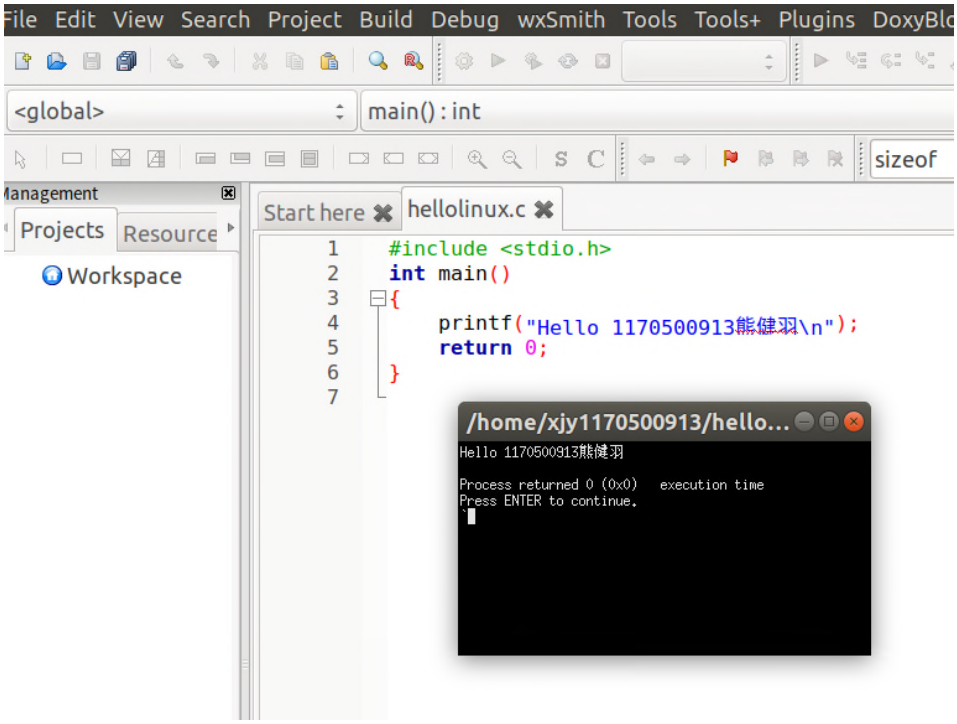

图 2-1 Ubuntu 下 CodeBlocks 截图

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。

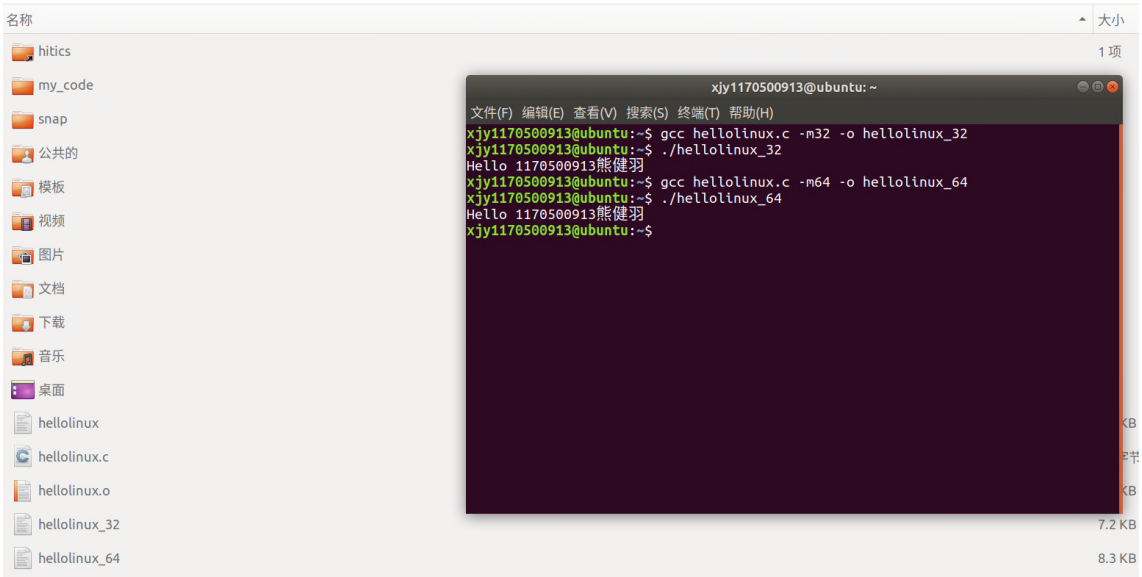

图 2-2　Ubuntu 与 Windows 共享目录截图

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）

逻辑或||、逻辑与&&、逻辑非!

例句：

表达式	结果
!0xff	0
0x0&&0x20	0
0xa||0x22	1

3.2 无符号数位操作（2 分）

按位与&、按位或|、按位异或^、按位取反~、左移<<、逻辑右移>>

例句：

设: x = 11110000_2 , y = 00111100_2

表达式	结果
x & y	00110000
x | y	11111100
x ^ y	11001100
~x	00001111
x >> 4	00001111
y << 4	11000000

3.3 有符号数位操作（2 分）

按位与&、按位或|、按位异或^、按位取反~、左移<<、算数右移>>

例句：

设: x = 11110000_2 , y = 00111100_2

表达式	结果
x & y	00110000
x | y	11111100
x ^ y	11001100
~x	00001111
x >> 4	11111111
y << 4	11000000

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. 与运算 AND S,D ；将操作数 S 和 D 按位相与，结果返回给 D。CF,OF 置为 0，影响 ZF,SF,PF。

   例：andl $666666, %eax

2. 或运算 OR S,D ；将操作数 S 和 D 按位相或，结果返回给 D。CF,OF 置为 0，影响 ZF,SF,PF。

   例：orq %rsi, %rdi

3. 非运算 NOT D ；将操作数 D 按位取反。不影响标志位。

   例：notq %rdi

4. 异或运算 XOR S，D ； 将操作数 S 和 D 相异或，返回给 D。CF,OF 置为 0，影响 ZF,SF,PF。

   例：xorq %rsi, %rdi

4.2 无符号数左右移（2 分）

1SHL k,D ；D←D<<k 逻辑左移指令，低位用 0 补齐
例：shlw $66, %ax

2SHR k,D ；D←D>>k 逻辑右移指令，高位用 0 补齐
例：shrq $88, %rax

4.3 有符号左右移（2 分）

1. SAL k,D ；D←D<<k 算术左移指令，低位用 0 补齐
例：salq $4, %rax
2. SAR k,D ；D←D>>k 算术右移指令，高位用符号位补齐
例：sarq $4, %rax

4.4 循环移位（2 分）

1ROL　S,D　;不带进位的循环左移指令，移出的数进行循环

例：rolw　$1,%ax

2ROR　S,D　;不带进位的循环右移指令，移出的数进行循环

例：rolq　$4,%rax

4.5 带进位位的循环移位（2 分）

1RCL　S,D　；带进位的循环左移指令。每移一位，将 CF 顶进右侧空缺的位中，然后 CF 置为高位溢出的一位。

例：rcll　$3,%eax

2RCR　S,D　；带进位的循环右移指令。每移一位，将 CF 顶进左侧空缺的位中，然后 CF 置为低位溢出的一位。

例：rcrw　$2,%ax

4.6 测试、位测试 BTx（2 分）

1BT:　把指定的二进制位传送给 CF;

例：btw　$7, %dx

2BTC: 把指定的二进制位传送给 CF 之后，还要使该位变反;

例：btcw　$0, %dx

3BTR: 把指定的二进制位传送给 CF 之后，还要使该位变 0;

例：btrw　$7, %dx

4BTS: 把指定的二进制位传送给 CF 之后，还要使该位变 1;

例：btsw　$3, %dx

5TEST S1, S2 ;基于 S1 & S2，但只设置条件码，不改变目的寄存器的值

testb:测试字节 testw:测试字 testl:测试双字 testq:测试四字
例：testw %ax,%ax

4.7 条件传送 CMOVxx（2 分）

cmove/cmovz S, R 等于 0 时传送 例：cmove %rdx,%rax
cmovne/cmovnz S, R 不等于 0 时传送 例：cmovne %rdi,%rax
cmovs S, R 负数时传送 例：cmovs %rdi,%rax
cmovns S, R 非负数时传送 例：cmovns %rdi,%rax
cmovg/cmovnle S, R 有符号大于时传送 例：cmovg %rdi,%rax
cmovge/cmovnl S, R 有符号大于等于时传送 例：cmovge %rdx,%rax
cmovl/cmovnge S, R 有符号小于时传送 例：cmovl %rdi,%rax
cmovle/cmovng S, R 有符号小于等于时传送 例：cmovle %rdi,%rax
cmova/cmovnbe S, R 无符号大于时传送 例：cmova %rdi,%rax
cmovae/ cmovnb S, R 无符号大于等于时传送 例:cmovae %rdi,%rax
cmovb/cmovnae S, R 无符号小于时传送 例:cmovb %rdi,%rax
cmovbe/cmovna S, R 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	效果	设置条件	例子
sete D	D←ZF	相等/零	sete %ax
setne D	D←~ZF	不等/非零	setne %ax
sets D	D←SF	负数	sets %ax
setns D	D←~SF	非负数	setns %ax
setg D	D←~(SF^OF)&~ZF	大于（有符号）	setg %ax
setge D	D←~(SF^OF)	大于等于（有符号）	setge %ax
setl D	D← SF^OF	小于（有符号）	setl %ax
setle D	D←(SF^OF)|ZF	小于等于（有符号）	setle %ax
seta D	D←~CF&~ZF	超过（无符号）	seta %ax
setae D	D←~CF	超过或相等（无符号）	setae %ax
setb D	D←CF	低于（无符号）	setb %ax
setbe D	D←CF|ZF	低于或相等（无符号）	setbe %ax

4.9 进位位操作（1 分）

清进位指令 CLC：CF←0　例：clc
置进位指令 STC：CF←1　例：stc
进位取反指令 CMC：CF←～CF　例：cmc
带进位的加法指令 ADC：　adc S,D　将 S 与 D 相加，并且加上 CF,结果赋给 D
例：adc　%edx, %eax

第 5 章 BITS 函数实验与分析

每题 8 分，总分不超过 80 分

btest 截图：

```
xjy1170500913@ubuntu:~/桌面/hitics/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       mul2OK
 2      2       0       mult3div2
 3      3       0       subOK
 4      4       0       absVal
 2      2       0       float_abs
 4      4       0       float_f2i
Total points: 35/35
```

dlc 截图：

```
xjy1170500913@ubuntu:~/桌面/hitics/lab1-handout$ ./dlc -e bits.c
dlc:bits.c:178:lsbZero: 2 operators
dlc:bits.c:191:byteNot: 3 operators
dlc:bits.c:207:byteXor: 9 operators
dlc:bits.c:218:logicalAnd: 5 operators
dlc:bits.c:229:logicalOr: 5 operators
dlc:bits.c:244:rotateLeft: 10 operators
dlc:bits.c:261:parityCheck: 11 operators
dlc:bits.c:282:mul2OK: 6 operators
dlc:bits.c:301:mult3div2: 6 operators
dlc:bits.c:320:subOK: 16 operators
dlc:bits.c:335:absVal: 6 operators
dlc:bits.c:357:float_abs: 9 operators
dlc:bits.c:393:float_f2i: 16 operators
```

5.1 函数 lsbZero 的实现及说明

程序如下：

```
1.  int lsbZero(int x)
2.  {
3.      int ans = x & (~1);
4.      return ans;
```

```
}
```

设计思想：要使最低位为 0，其他位不变，可利用 a & 1 = a, a & 0 = 0 (a = 0 或 1)的性质，即让 x 与 0xfffffffe 按位且。而 0xfffffffe 为 ~1。

5.2 函数 byteNot 的实现及说明函数

程序如下：

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

设计思想：要使第 n 个字节取反，可利用 a ^ 1 = ~a , a ^ 0 = a (a = 0 或 1)的性质，让 x 与目标字节为 1，其他字节为 0 的数按位异或。要确定这个数，即让 0xff 左移 n 个字节，即 8n 位，而 8n 可用 n << 3 表示。

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

设计思想：先利用按位且的特性，让 x，y 分别与目标字节为 1,其他字节为 0 的数按位且。要确定这个数，即让 0xff 左移 n 个字节，即 8n 位，而 8n 可用 n << 3 表示。如此得到的 x，y，非目标字节均置为 0。然后 x 与 y 按位异或，若目标字节相同，结果为 0，否则为非 0，利用两次逻辑非运算，0 转化为 0，非 0 转化为 1，即得到结果。

5.4 函数 logicalAnd 的实现及说明函数

程序如下：

```
int logicalAnd(int x, int y)
{
    int ans = (!!x) & (!!y);
```

```
    return ans;
}
```

设计思想：两次逻辑非运算，0 转化为 0，非 0 转化为 1。只有最低位有效，其他位均为 0，此时逻辑与和按位与等效，故得到结果。

5.5 函数 logicalOr 的实现及说明函数

程序如下：

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

设计思想：思想同上一个函数。两次逻辑非运算，0 转化为 0，非 0 转化为 1。只有最低位有效，其他位均为 0，此时逻辑或和按位或等效，故得到结果。

5.6 函数 rotateLeft 的实现及说明函数

程序如下：

```
int rotateLeft(int x, int n)
{
    int temp1 = x >> (32 + (~n + 1));
    int temp2 = ~(~0 << n);
    x = x << n;
    return x + (temp1 & temp2);
}
```

设计思想：

要实现循环左移，关键是将正常左移溢出的位保存下来。左移了 n 位，则需要保存的位数为高 n 位。具体思路为：

1. 将 x 右移 32 – n 位，此时高 n 位移到了低 n 位（– n 利用补码的性质，表示为~n + 1）；
2. 为消除第一步逻辑右移与算数右移的差别，将第一步中的结果做掩码运算，即把高 32 – n 位置 0，掩码即为 0xffffffff 左移 n 位，而 0xffffffff 可表示为~0；
3. 将 x 正常左移，此时低 n 位全零；

4. 将第三步结果与第二步结果相加，即让保存下来的低 n 位加到了的新 x 的右侧，得到结果。

5.7 函数 parityCheck 的实现及说明函数

程序如下：

```
1.  int parityCheck(int x)
2.  {
3.      x = x ^ (x >> 16);  //将高 16 位与低 16 位异或，结果保存于低 16 位
4.      x = x ^ (x >> 8);   //将上述 16 位的高 8 位与低 8 位异或，结果保存于低 8 位
5.      x = x ^ (x >> 4);   //将上述 8 位的高 4 位与低 4 位异或，结果保存于低 4 位
6.      x = x ^ (x >> 2);   //将上述 4 位的高 2 位与低 2 位异或，结果保存于低 2 位
7.      x = x ^ (x >> 1);   //将上述 2 位的高 1 位与低 1 位异或，结果保存于低 1 位
8.      x = x & 1;          //将无效位置零
9.      return x;
10. }
```

设计思想：

要在及其有限的步骤（且不能用循环）中数出 1 的个数的奇偶性，显然不能一位一位地判断。但注意到按位异或的性质：

两个操作位的 1 的个数之和为奇：0 ^ 1 = 1 → 结果 1 的个数也为奇；

两个操作位的 1 的个数之和为偶：0 ^ 0 = 0, 1 ^ 1 = 0 →结果 1 的个数为偶（即 0）

结论：异或不改变两个操作数的 1 个数之和的奇偶性。

故想到让 x 本身的两部分相互异或，得到的有效位，1 的个数的奇偶性与原 x 相同。最简单的即为高 16 位与低 16 位异或，得到 16 位有效位，1 的个数奇偶不变。再让有效 16 位的高 8 位与低 8 位异或，得到 8 位有效位。1 的个数奇偶不变……以此类推，最终得到的 1 位有效位，1 的个数的奇偶性与原 x 也相同，最后通过掩码运算，将无效位置为 0，若结果为 1，则原 x 的 1 的个数为奇，反之为偶，符合题意。

5.8 函数 mul2OK 的实现及说明函数

程序如下：

```
1.  int mul2OK(int x)
2.  {
3.      int val_bits = x;
4.      int sign_bits;
5.      int ans;
6.      val_bits <<= 1;
7.      val_bits >>= 31;
8.      sign_bits = x >> 31;
9.      ans = sign_bits ^ val_bits;
10.     ans = ~ans & 0x1;
11.     return ans;
12. }
```

设计思想：

int 范围为：$-2^{31} \sim 2^{31} - 1$

需考虑两种情况：

1. x 为非负数：x 小于 2^{30} 时不会溢出，而 2^{30} 表示为：

   0x40000000(最高字节为 0100)

2. x 为负数：x 大于等于-2^{30} 时不会溢出，而-2^{30} 表示为：

   0xc0000000(最高字节为 1100)

因此，当符号位为 0 时，只要最高数值位为 1，则会溢出；当符号位为 1 时，只要最高数值位为 1，则不会溢出。画出真值表：

符号位	最高有效位	mul2OK
0	1	0
0	0	1
1	1	1
1	0	0

由上表可看出，符号位与最高有效位 同或（异或之后再取反） 即得到结果。

可利用有符号整型的右移运算是算数右移的性质，确定符号位与最高数值位：

1. 确定符号位：算数右移 31 位，若符号位为 1，则结果全 1；反之结果全 0；

2. 确定最高数值位：先左移 1 位，再算数右移 31 位，若最高数值位为 1，则结果全 1；反之结果全 0。

将上述两个结果按位异或，再取反，最后通过掩码计算将最低位之外的位全置为 0，即得到最后结果。

5.9 函数 mult3div2 的实现及说明函数

程序如下：

```
1.  int mult3div2(int x)
2.  {
3.      int ans, bias;
4.      x = (x << 1) + x;
5.      bias = (x >> 31) & 0x1;
6.      ans = (x + bias) >> 1;
7.      return ans;
8.  }
```

设计思想：

乘除均可用左移和算数右移实现，但关键是实现最终结果的向 0 舍入。

向 0 舍入，即正数向下舍入，负数向上舍入。正数算数右移时，无需额外操作，即为向下舍入；而负数不符合要求，算数右移时，结果为向下舍入。参考课本中的方法：**负数除以 2^k 时，给原数加上一个 $2^k - 1$ 的偏移量，使负数除法向上舍入。**这里 k = 1，即若符号位为 1，则加上一个 bias = 1 的偏移量。

具体步骤：

1. 同上题，x >> 31 确定符号位，所得结果与 1 按位且，确定 bias = 1 或 0：若符号位为 1，bias = 1；若符号位为 0，bias = 0，符合要求。
2. 先让 x 乘以 3，即 x = (x << 1) + x；
3. x 加上 bias，算数右移一位，实现除以 2，即得到结果。

5.10 函数 subOK 的实现及说明函数

程序如下：

```
1.  int subOK(int x, int y)
2.  {
3.      int sub_ans = x + (~y + 1);
```

```
    int sign_x = x >> 31;
    int sign_y = y >> 31;
    int sign_ans = sub_ans >> 31;
    int overflow1 = sign_x | ~sign_y | ~sign_ans;    //x > 0, y < 0, ans < 0
    int overflow2 = ~sign_x | sign_y | sign_ans;     //x < 0, y > 0, ans > 0
      //全 0 代表溢出
    return !!(overflow1 & overflow2);
}
```

设计思想：x – y 溢出，无非两种情况：x > 0 , y < 0 , 结果却小于 0；x < 0 , y > 0，结果却大于零。确定符号的方法同上述几题：>> 31 确定符号位。overflow1 = sign_x | ~sign_y | ~sign_ans，为第一种情况，overflow2 = ~sign_x | sign_y | sign_ans，为第二种情况。由于 overflow 全零代表溢出，则用 overflow1 & overflow2 表示最终结果，即若其中一个为 0（溢出），结果即为 0（溢出）。否则为非 0，通过两个！运算转化为 1，即得到正确结果。

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;
    int opx = ~x + 1;
    int bias = (opx & sign_x) << 1;
    return (x + bias);
}
```

设计思想：

取绝对值，分为两种情况：x 为非负数，输出本身；x 为正数，输出本身的相反数。两种情况可通过偏移量 bias 表示：结果为 x + bias，若 x 为非负数，则 bias = 0；若 x 为负数，bias = 2*（-x）。具体步骤如下：

1. 确定符号的方法同上述几题，sign_x = x >> 31 确定符号位；
2. x 的相反数补码表示为 opx = ~x + 1，opx 与 sign_x 按位与，如果 x 符号位为 1，则结果值为-x；反之结果为 0；
3. 将所得结果左移 1 位，相当于-2x，即得到偏移量 bias；
4. bias 与 x 相加，即得到正确结果。

5.12 函数 float_abs 的实现及说明函数

程序如下：

```
1.  unsigned float_abs(unsigned uf)
2.  {
3.      unsigned e = uf & 0x7f800000;                   //取阶码数位
4.      unsigned isNANorINF = !(e ^ 0x7f800000);        //是否为 NAN 或 INF，是则为 1，否则
    为 0
5.      unsigned eAndm = uf & 0x7fffffff;               //取阶码数位和尾数数位
6.      unsigned isINF = !(eAndm ^ 0x7f800000);         //是否为 INF，是则为 1，否则为 0
7.      if(isNANorINF & !isINF)                         //如果为 NAN
8.          return uf;                                  //返回原数
9.      else
10.         return (uf & 0x7fffffff);                   //否则返回 符号位 置 0 之后的原
    数
11. }
```

设计思想：

符号位置 0 非常简单，关键在于判断 NAN，若为 NAN，返回本身；否则将符号位置零后返回。

NAN：阶码全 1，尾数不全为 0；

INF：阶码全 1，尾数全为 0.

利用掩码计算取出只含阶码数位的 e 与同时含阶码和尾数数位的 eAndm，

阶码数位 e 与 0x7f800000 异或，判断是否为 INF 或 NAN；阶码数位和尾数数位 eAndm 与 0x7f800000 异或，判断是否为 INF，两者结合，可判断是否为 NAN。若是，返回原数；若不是，和 0x7fffffff 按位与，将符号位置零之后返回。

5.13 函数 float_f2i 的实现及说明函数

程序如下：

```
1.  int float_f2i(unsigned uf)
2.  {
3.      unsigned e = (uf & 0x7f800000) >> 23;           //阶码
4.      unsigned E = e + 0xffffff81;                    //实际指数 = 阶码 - 127
5.      unsigned above_one = (e + 1) >> 7;              //绝对值是否大于 1
6.      unsigned below_zero = uf & 0x80000000;          //符号位是否为 1
7.      unsigned overflow = E >> 5;                     //是否超出 int 型的范围
8.      unsigned m = (uf & 0x007fffff) + 0x800000;      //尾数
9.      unsigned abs_ans = m >> (23 + (~E + 1));        //uf 的绝对值强转 int 的结果
10.     if(!above_one)                                  //若绝对值小于 1
11.         return 0;
12.     else
```

```
13.     {
14.         if(overflow)                //若超出 TMIN~TMAX 的范围，或是 NAN 或 INF
15.             return 0x80000000u;
16.         else
17.         {
18.             if(below_zero)                          //若小于零,返回其相反数的补码
19.                 return (~abs_ans + 1);
20.             else
21.                 return abs_ans;
22.         }
23.     }
24. }
```

设计思想：

关键在于分类讨论：

1. 非规格化的值：由于绝对值小于 1，强转为 int 后值为 0；
2. 规格化的值：
   a) 绝对值小于 1，强转为 int 后值为 0；
   b) 绝对值大于等于 1，小于 2^{31} 的：即在 int 表示范围内的；
   c) 绝对值大于等于 2^{31} 的，即在 int 表示范围外的，返回 0x80000000u；
3. INF，返回 0x80000000u
4. NAN，返回 0x80000000u

具体思路：

1. 先考虑特殊的：
   a) 阶码数位 e 只要小于 127 的，above_one = (e + 1) >> 7 为 0，绝对值即小于 1，返回 0；
   b) 由于非规格化数的情况已经在步骤一中考虑。故实际指数 E = e – 127。若 E 大于等于 31，即 overflow = E >> 5 为非 0，则表示超出 int 范围，返回 0x80000000u；
   c) 由于 INF，NAN 的阶码全 1，故也包括在步骤二中。
2. 再考虑正常情况：
   a) m = (uf & 0x007fffff) + 0x800000 得到尾数 m（其中+ 0x800000 意在把尾数存储中省略的整数位 1 加上）

b) 把 m 右移 23 - E 位，得到 uf 的绝对值强转 int 的结果 abs_ans；

c) below_zero = uf & 0x80000000 判断符号位是否为 1：若是，返回 abs_ans 的相反数，即~abs_ans + 1；若否，返回 abs_ans。

每步详细说明见程序注释。

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. 对数据的机器级存储有了更深的理解，如无符号、有符号整数，以及 IEEE754 标准的浮点数存储；
2. 在上次实验的基础上，增加了应用环节。通过使用 C 语言中的指定操作，编写指定功能的函数，让我体验到了计算机底层的操作，同时也懂得了高级语言的局限性，体会到了学习汇编语言的必要性；
3. 总结了汇编语言的重要操作指令，让我对其有了总体的认识。

10.2 请给出对本次实验内容的建议

暂无。实验内容充实而丰富，我获益匪浅。

注：本章为酌情加分项。

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.2018.4