实验二

1 实验内容

本实验中，你需要利用整型及浮点数的位表达形式，来解开一些谜题，一共有15个需要补充的函数，除了完成代码进行在线评测之外，还需要提交实验报告。

2 完成情况

2.1 bitXor(x, y)

① 功能描述

只用 & 和 ~ 实现 x^y

② 思路与实现

根据离散数学的相关知识，有：$x \oplus y = x\overline{y} + \overline{x}y$

但是题目要求只能使用&和~，由摩根定律可知：

$$x\overline{y} + \overline{x}y = \overline{\overline{x\overline{y}} \cdot \overline{\overline{x}y}} \tag{1}$$

由此，仅用与和取非操作就可以实现异或

2.2 getByte(x, n)

① 功能描述

从字x中取出第n个字节

② 思路与实现

采取课上所说的掩码操作，构造一个这样的掩码0xff：最低有效位的byte是1，其余byte均为0。

之后将n右移三位，也就是乘以8，因为一个byte占用8位，取第n个字节就相当于将原数字左移$8 \times n$位

最后将右移$8n$后的原数与0xff相与就可以得到想要的结果。

2.3 logicalShift(x, n)

① 功能描述

逻辑右移

② 思路与实现

这个问题的难点在于int型的数字如果右移的话会执行算术右移，所以要想办法把右移之后填充的位置为0。

不禁想到异或运算$\oplus$的性质：$a \oplus a = 0$，　$a \oplus 0 = a$，由此得到如下的想法：构造一个数，这个数的最高几位与右移之后的数的填充位相同，剩下的低位全为0。将这个数与算术右移之后的数作异或操作就可以得到逻辑右移的结果。

下面是构造这个数的过程，先将原数左移31位，之后再将这个数左移（32-n）位，因为左移永远只会补0，所以保证了构造出的这个数高n位与原数的符号位相同，而低32-n位全为0

2.4 bitCount(x)

① 功能描述

计算x中1的个数

② 思路与实现

此题采用“分治”的思想。主要步骤如下：

1. 将原数划分为8组，每一组4位，先计算每一组中1的个数1。计算的做法如下：

    a. 构造一个掩码mask：0x11111111，与原数相与

    b. 将原数左移一位之后与mask相与，并与上一次得到的结果相加

    c. 重复b操作3次

    源代码如下：

```
m = (m << 16) + m;
m2 = 0x0f;
m2 = (m2 << 8) + m2;
a = x & m;
a += (x >> 1) & m;
a += (x >> 2) & m;
a += (x >> 3) & m;
a = a + (a >> 16);
```

    这里不用担心计算的结果会溢出，因为4个bit可以表示的数的范围是0~15，而四位中最多只有4个1

2. 分完开始合并：将原数的高16位与低16位相加，相当于把原来划分的合并为4组，这里同样的不用担心计算结果会溢出，因为8位最多只有8个1，而4个bit能表示的范围是0~15。示意图如下：

$$0x\,n_8n_7n_6n_5n_4n_3n_2n_1 \rightarrow 0x\,n_8 + n_4\,n_7 + n_3\,n_6 + n_2\,n_5 + n_1 \quad (2)$$

    记得到的中间结果为A。

3. 这一步很关键，不能简单地把2中得到的结果的前8位与后8位相加，因为16位中最多可能有16个1，超出了4个bit的表示范围，所以因该这样操作：

    a. 构造掩码mask2：0x0f0f，将A和mask2相与

    b. 将A左移4位之后再与mask2相与，然后与上一步中得到的结果相加

    这个操作相当于两个型如0x0_0_的十六进制数相加，这样就留出了加法的进位空间

    源代码如下：

```
b = a & m2;
b = b + ((a >> 4) & m2);
```

记得到的中间结果为B。

4. 最后将B左移8位之后再与B相加，取最后结果的低6位，因为最多可能有32个1

2.5 conditional(x,y,z)

① 功能描述

类似于C语言中的 x ? y : z

② 思路与实现

最终结果要从y和z中选一个，由此想到数字逻辑课上讲的硬件：选择器。表式如下

$$x = \overline{s}a + sb \tag{3}$$

由于s的值要么为1，要么为0，所以最终结果必然只可能是a或者b中的一个。

所以代码可以这样写：

```
int a = ~(!!x) + 1;
return (a & y) | (~a & z);
```

对x连续取两次非，这样如果x为0那么!!x也为0，x为1，!!x也为1。之后再进行取反加一是为了将x的所有位都置为0或者1，这样的掩码相与之后才能获得原数。

2.6 tmin()

① 功能描述

返回最小的补码

② 思路与实现

最小的补码除了符号位是1，其余位均为0，因此可以将0x1右移31位得到

2.7 fitsBits(x,n)

① 功能描述

x的补码是否可以表示成n位

② 思路与实现

对于一个正数而言，由于符号位是0，所以可以将它左移n-1位，看得到的结果是不是等于0就可以n位可不可以表示x。之所以是n-1，是因为还要留一位给符号位！

对于一个负数可以采取同样的思路，将负数取反就可以得到同样的效果。

但是，负数右移是算术右移，补的是1，而且对正负数的处理得到一样的效果，所以需要将x（假设已经右移n-1位）左移31位得到m，然后将m与x异或，这样的话，如果x为正数，那么m全为0，异或之后x值不变；如果x是负数，m就全为1，异或之后补的符号位全为0，剩下的有效位取反，这样就等价于一个正数。

源代码如下：

```
int s = x >> 31;
x = s ^ x; // 这样操作之后，符号位一定是0，肯定执行的是算术右移
n = n + (~1 + 1);
x = x >> n;
return !x;
```

2.8 dividePower2(x,n)

① 功能描述

计算x/(2n)

② 思路与实现

注意审题，题目要求了要向0舍入

对于正负数有不同的操作：

1. 正数，直接左移n位就可以
2. 负数，直接左移n位得到的结果是向下取整的，这里需要我们向上取整，根据公式$\lceil x/m \rceil = \lfloor (x+m-1)/m \rfloor$，所以需要先将x加上一个$2^n - 1$

至于如何选择不同的操作，用一个选择器就可以实现，代码如下：

```
int a = x >> 31;
int bias = (1 << n) + (~1) + 1;
int b = a & bias;
return (x + b) >> n;
```

2.9 negate(x)

① 功能描述

不用负号得到

② 思路与实现

对于补码而言，除了最大的负数，任何一个数的相反数都可以通过如下操作实现：

- 先按位取反，然后加1

2.10 howManyBits(x)

① 功能描述

计算表达 x 所需的最少位数

② 思路与实现

首先，有符号数最少都有一个符号位。因此可以先展示将符号位抛到一边不考虑。

一个很显而易见的事实：一个数丢掉符号位之后，它的最高位一定是1。因此，一个数需要多少个bit表示也就是找这个数最高位的1在从左往右数的第几位。对于负数，它的最高有效位一定是0，所以只需要对负数取反，就可以将其转换为找最高位的1。

```
x = (x & ~a) | (~x & a);
```

现在问题就变为了怎样找最高位的1？这里采用二分的思路（假设原数为w位）：

1. 将原数划分为两半
2. 通过连续取两次非（！）的操作来判断最高位的1是在前一半还是后一半，$w = \frac{w}{2}$，并将得到的结果b左移$\log_2(w)$位。对于一个这样的话如果最高位的1在前一半，那么$b = 1 \times \frac{w}{2}$；如果b在后一半，那么b=0。这里的b相当于是一个权重，记录了1所在的大概位置，b的值需要记录下来
3. 将原数a左移b位，相当于在有1的那一半里继续寻找1的更加精确的位置
4. 重复上述操作，直到范围缩小到1

上述操作的代码实现如下：

```
b16 = !!(x >> 16) << 4;
x = x >> b16;
b8 = !!(x >> 8) << 3;
x = x >> b8;
b4 = !!(x >> 4) << 2;
x = x >> b4;
b2 = !!(x >> 2) << 1;
x = x >> b2;
b1 = !!(x >> 1);
x = x >> b1;
b0 = x;
```

最后将二分过程每一步计算到的b全部加起来，然后再加上一位符号位。

2.11 isLessOrEqual(x,y)

① 功能描述

x <= y?

② 思路与实现

判断一个数小于另一个数比较自然的想法是直接将两个数作减法，根据得到的结果的符号来比较二者的大小。

但是上述做法存在问题，因为两个符号不同的数相减可能会溢出，以致得出错误的结论。

解决这个问题的思路是：分情况讨论

1. 第一种情况：两个数如果符号相同，那么可以直接相减，此时不会溢出
2. 第二种情况：两个数如果符号不同，那么负数一定比正数小

由此可以写出如下的代码：

```
int sx = x >> 31;
int sy = y >> 31;
int s = sx ^ sy;
int c;
c = ~x + y + 1;
c = c >> 31;
return !!((~s & !c) | (s & sx));
```

先将两个数分别左移31位，让符号位填充满所有的位，之后将处理后的两个数作异或操作，以此来判断二者的符号是否相同。

至于选择哪一种比较大小的方法，用一个选择器实现

2.12 intLog2(x)

① 功能描述

计算[log2(x)]（向下取整）

② 思路与实现

对于一个十进制数，求$\lfloor \log_{10} x \rfloor$相当于x的最高位的1后面有多少位。

同样的道理，对于一个二进制数而言，求$\lfloor \log_2 x \rfloor$相当于求最高位的1后面有多少位，这就转换为了2.10之中的问题，所以只需要将2.10中的结果减2就可以得到本题的结果。

此外，由于x必须大于0，所以不需要考虑负数

2.13 floatAbsVal(uf)

① 功能描述

计算f的绝对值的位级表示

② 思路与实现

根据IEEE754标准，浮点数的最高位为符号位，要求一个浮点数的绝对值，只需要保证其最高位是0就可以了，因此可以将原操作数和掩码0x7fffffff相与。

源代码如下：

```
int mask = 0x7fffffff;
if (!(exp ^ 0xff) && !!(f))
  return uf;
return uf & mask;
```

注意，要排除NaN和无穷大的情况

2.14 floatScale1d2(uf)

① 功能描述

计算0.5*f的位级表示

② 思路与实现

这个问题的关键在于不清楚给定的浮点数是否是规格化的，所以要分两种情况讨论：

1. 规格化数：乘以0.5相当于指数部分减1

   - 这里有一种情况需要注意，如果指数减完1之后变为0，意味着这个数由规格化数变为了非规格化数，由于二者的指数部分是相同的，需要将尾数部分左移一位，并且将尾数的最高位置为1，相当于把隐藏的1“移出来了”

源代码：

```
    exp = exp - 1;
    // 指数部分变为0之后就表示非规格化数
    // 最大的非规格化数与最小的规格化数之间的过渡是平稳的，也就是说，指数部分的解释不变
    // 因此，需要将尾数部分整体右移1位，包括隐藏的1
    if (!exp)
    {
      if (!((f & 0x3) ^ 0x3))
      {
        f = (f >> 1) + 1;
      }
      else
        f = f >> 1;
      f = f | (1 << 22);
    }
```

2. 非规格化数：将尾数部分右移一位，并且采用向偶数舍入的方法：

   a. 如果左移之前最后两位是11，那么需要向上舍入

   b. 如果左移之前最后两位是01或者00或者10，那么向下舍入

源代码：

```
    if (!((f & 0x3) ^ 0x3))
    {
      f = (f >> 1) + 1;
    }
    else
      f = f >> 1;2.15 floatFloat2Int(uf)

```

2.15 floatFloat2Int(uf)

① 功能与实现

计算(int)f的位级表示

② 思路与实现

此题也要分情况讨论，关键点在于指数的大小。

1. 情况1，阶码小于127（也就是指数小于0），此时表示的数小于1，转换为整型一定为0

```
1   if (iexp < 127)
2     return 0;
```

2. 情况2，阶码太大，包括正负无穷的情况。int型数据只有32位，除去符号位之后只剩31位，所以指数部分最多只能为30（先不考虑最小的负数），由于此题规定超限的情况直接返回0x80000000，也就是最小的负数，所以可以将最小负数归到这一类情况中；

```
1   else if (iexp >= 158)
2     return u;
```

3. 情况3，阶码介于上述二者之间。此时的关键在于取多少尾数的问题。先将尾数移到32的最左端，然后根据指数iexp的大小将尾数向左移32-iexp位，相当于取了尾数的前iexp位，假如尾数不够就用0补

```
1   iexp = iexp - 127;
2   result = result << iexp;
3   result = result | f >> (31 - iexp) >> 1;
4   if (sign)
5     result = ~result + 1;
```

由于iexp可能为0，此时f需要左移32位，编译器会对32取模，由此得到错误的结果，因此先左移31-iexp位，然后再左移一位。

注意别忘了隐藏的1，而且如果负数的话还需要取反加1变为补码