计算机系统基础

Lab1 Data Lab

姓名：傅文杰

学号:22300240028

2023 年 9 月 27 日

目录

1 实验目的

1. 深入理解并运用位运算
2. 深入了解IEEE754规范下的浮点数表示

2 补全bits.c

2.1 int bitXor(int x, int y)

1. 目的：用与(&)和非(~)运算实现异或(^)
2. 证明:

$$\begin{aligned} A \oplus B &= \overline{A} \cdot B + A \cdot \overline{B} \\ &= \overline{\overline{\overline{A} \cdot B + A \cdot \overline{B}}} \\ &= \overline{(A + \overline{B}) \cdot (\overline{A} + B)} \\ &= \overline{A \cdot \overline{A} + A \cdot B + \overline{A} \cdot \overline{B} + B \cdot \overline{B}} \\ &= \overline{A \cdot B + \overline{A} \cdot \overline{B}} \\ &= \overline{A \cdot B} \cdot \overline{\overline{A} \cdot \overline{B}} \end{aligned}$$

3. 关键代码：

```
1 return ~(x & y) & ~(~x & ~y);
```

2.2 int tmin(void)

1. 目的：返回最小的32位二进制补码整数

2. n位二进制补码整数的取值范围是$[-2^{n-1}, 2^{n-1}-1]$
   当首位为0时，该整数为正；当首位为1时，该整数为负，并且后面的$n-1$位越小，负数的绝对值越大
   因此返回0x 8000 0000(1 << 32)即可

2.3 int isTmax(int x)

1. 目的：判断是否是最大的32位二进制补码整数，是则返回1，否则返回0

2. 最大的32位二进制补码整数是0x 7fff ffff, 位数太多, 由于我们无法使用大常数, 所以我们将它加1（或者取反）成为0x 8000 0000
   注意到这个数（最小的二进制补码整数）和0有一个特殊的性质：它们的相反数都是自身。因此我们只需要判断它取反加一是否等于自己并且排除0即可

3. 判断两数相等，我们用异或运算

4. 关键代码：

```
1 x = x + 1;
2 return !((~x+1) ^ x) & !!(x);
```

2.4 int allOddBits(int x)

1. 目的：如果所有的奇数位都是1，返回1

2. 我们只关心奇数位，所以**与**上一个奇数位全为1偶数位全为0的数即可

3. 但我们只能用0~0xFF的常数，因此在扩充位数时需要建立临时变量，利用C语言逻辑左移的特性和位或运算

4. 关键代码：

```
1 int mask = 0xAA;
2 int odd_bits = (mask << 8) | mask;
```

```
3 odd_bits = (odd_bits << 16) | odd_bits;
4 return !((x & odd_bits) ^ odd_bits);
```

2.5 int negate(int x)

1. 目的：获得相反数
2. 取反加1即可

2.6 int isAsciiDigit(int x)

1. 目的：如果是ASCII码数字，返回1，否则返回0
2. 如果$0x30 \leq x \leq 0x39$，则返回1，否则返回0
3. 减法用加相反数表示，判断正负用**与**上0x 8000 0000(1<<31)表示
4. 关键代码：

```
1 int flag = 1 << 31;
2 return !((x + (~0x30 + 1)) & flag) & !((0x39 +
    (~x + 1)) & flag);
```

2.7 int conditional(int x, int y, int z)

1. 目的：返回$x\ ?\ y : z$
2. 先将x转换成逻辑0或者1，利用0的补码全0、1的补码全1的特性，将想要的结果与上全1 or 或上全0
3. 关键代码：

```
1 int notx = !x;
2 int flag = ~notx + 1;
3 return (y & ~flag) + (z & flag);
```

2.8 int isLessOrEqual(int x, int y)

1. 目的：如果$x \leq y$，返回1，否则返回0

2. 首先想到diff $= y+ \sim x + 1$，diff的第1位为0就返回1。但是这仅仅对于x,y同号的情况下成立，如果异号，当负数减正数时，加法可能会溢出得到正数，需要根据两数的符号位特判

3. 关键代码：

```
int sign_x = (x >> 31);
int sign_y = (y >> 31);
int diff = y + (~x + 1);
return ((!(sign_x ^ sign_y)) & (!(diff >> 31)))
   | (sign_x & !sign_y);
```

2.9 int logicalNeg(int x)

1. 目的：实现逻辑非运算（0则返回1，否则返回0）

2. 0和最小数的补码为它们自身，其他数的补码都会改变第一位的值。而且，算术右移31位后0还是0，最小数变为全1，利用加法溢出的性质加1即可

3. 关键代码：

```
return ((x | (~x + 1)) >> 31)+1;
```

2.10 int howManyBits(int x)

1. 目的：表示一个整数至少需要几位二进制位

2. 一定会有一位符号位，最后加1即可，所以对于正数，需要忽略去掉符号位之后的前导0，我们将其保持不变；对于负数，需要忽略去掉符号位之后的前导1，我们将其取反，就可以和正数统一操作了

3. 二分查找：先查找后16位是否非0，如果是的话至少需要16位，否的话至少需要0位，记录下来，并将这16位或0位移掉；
然后查找后8位，后4位，后2位，后1位以及剩下的一位，重复相同的操作。最终把记录下来的需要的位数相加即可

4. 关键代码:

```
int b16, b8, b4, b2, b1, b0;
x = x ^ (x >> 31);
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;
return b16 + b8 + b4 + b2 + b1 + b0 + 1;
```

2.11 unsigned floatScale2(unsigned uf)

1. 目的：返回浮点数的2倍

2. 特殊处理：对于inf或者NaN（阶码全为1），返回它们本身

3. 对于规格数（阶码不全为0），阶码加1即可

4. 对于非规格数（阶码全为0），保持符号位不变，其他位左移1即可

5. 关键代码：

```
if(((uf>>23)&0xff)==0xff)        return uf;//nan
   or infinity
```

```
if (((uf>>23)&0xff)!=0)//normalized
    return uf+(1<<23);
return (uf&(~0x7fffff))|((uf&0x7fffff)<<1);//
   denormalized
```

2.12 int floatFloat2Int(unsigned uf)

1. 目的：浮点数取整
2. 分别提取出符号位sign，指数部分exp和小数部分frac
3. 32位二进制补码表示的整数的范围是$\{x|0 \le x \le 2^{31}-1, x \in \mathbb{Z}\}$
   $\because$ frac $\in [1,2)$
   $\therefore \exp < 0 \Rightarrow x < 2 \times 2^{-1} = 1 \Rightarrow \lfloor x \rfloor = 0$
   $\therefore \exp > 31 \Rightarrow x > 1 \times 2^{31} \Rightarrow \lfloor x \rfloor \ge 2^{31} \Rightarrow return\ 0x80000000u$
4. 当$23 < \exp \le 31$时，小数部分可全部被保留，注意要添加隐藏的1，并右移$\exp - 23$位
5. 当$0 < \exp \le 23$时，小数部分会被舍弃$23 - \exp$位，添加隐藏的1并右移即可
6. 最后要根据符号位定正负
7. 关键代码：

```
int sign = (uf >> 31) & 1;
int exp = ((uf >> 23) & 0xFF) - 127;
int frac = uf & 0x007FFFFF;
if (exp < 0) {
    return 0;
}
if (exp > 31) {
    return 0x80000000u;
} else if (exp > 23) {
```

```
10      return sign ? -((frac | 0x00800000) << (exp
            - 23)) : ((frac | 0x00800000) << (exp -
            23));
11  } else {
12      return sign ? -((frac | 0x00800000) >> (23 -
             exp)) : ((frac | 0x00800000) >> (23 -
            exp));
13  }
```

2.13 unsigned floatPower2(int x)

1. 目的：返回2^x

2. 指数范部分可以表示$2^{-126} \sim 2^{127}$，此时可以用规格数表示；小数部分可以表示$2^{-150} \sim 2^{-127}$，此时可以用非规格数表示

3. 关键代码：

```
1 if (x >= 128) return 0x7f800000;
2 if (x >= -126) return (x + 127) << 23;
3 if (x >= -150)
4     return 1 << (x + 150);
5 else
6     return 0;
```

3 总结

3.1 特殊的二进制补码数

1. 补码和自身相等的数：0 和 -1

2. 补码为全1的数：1

3.2 关于位运算

1. C语言位移运算的特性：
   (1)对于补码表示整数，逻辑左移、算术右移
   (2)对于无符号型整数，逻辑位移
   (3)对于带符号型整数，算术位移

2. 位移运算的作用：
   (1)左移给自己的尾巴补0
   (2)左移得到2的乘方
   (3)右移给自己的头部补1或者0，并舍弃尾部数字
   (4)右移除以2的乘方

3. 位与运算的作用：
   (1)有0则0的逻辑实现：排除某种特例
   (2)只关注一个数的某一位或者某几位，与上一个在那些位为1、其余位为0的数

4. 位或运算的作用：
   (1)有1则1的逻辑实现：允许某种特例
   (2)保留一个数的某一位或者某几位：或0

5. 异或运算的作用：
   (1)0和1之间的互相转换(判断同异号)
   (2)判断相等

3.3 关于构造

1. 构造全0：0

2. 构造全1：
   (1) $int\ x = \overline{1\cdots} \Rightarrow all_1 = x >> 31$
   (2) $\overline{1}$

3. 构造重复的数：假设$x = \overline{x_1x_1x_1\cdots}$，其中$x_1 = a_1a_2\cdots a_n$，
   则令$x = x_1$，循环$x = (x << n)|x_1$即可

4. 取符号位：

   (1)x >> 31得到0或者-1

   (2)x & 0x 8000 0000得到0或者最小数

5. 构造逻辑0或者1：!(x)或者!!(x)

3.4 关于IEEE浮点数

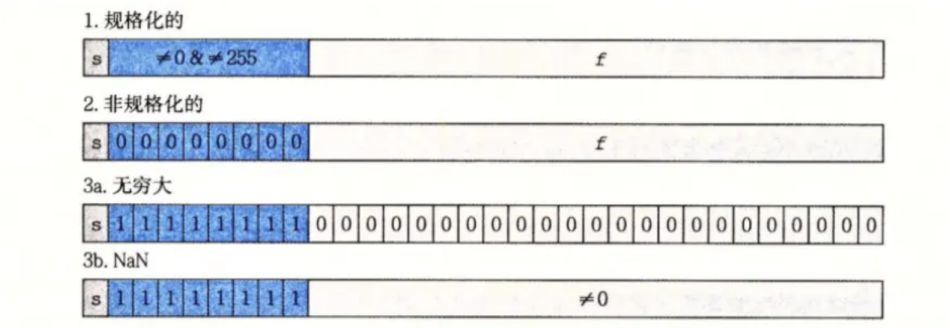

图 2-33 单精度浮点数值的分类(阶码的值决定了这个数是规格化的、非规格化的或特殊值)

深入理解计算机系统第三版

图 1: IEEE754浮点数标准

规格化数有隐含的1，非规格化数有隐含的0

Here are some examples of single-precision IEEE 754 representations:

Type	Sign	Actual Exponent	Exp (biased)	Exponent field	Fraction field	Value
Zero	0	−126	0	0000 0000	000 0000 0000 0000 0000 0000	0.0
Negative zero	1	−126	0	0000 0000	000 0000 0000 0000 0000 0000	−0.0
One	0	0	127	0111 1111	000 0000 0000 0000 0000 0000	1.0
Minus One	1	0	127	0111 1111	000 0000 0000 0000 0000 0000	−1.0
Smallest denormalized number	*	−126	0	0000 0000	000 0000 0000 0000 0000 0001	$\pm 2^{-23} \times 2^{-126} = \pm 2^{-149} \approx \pm 1.4 \times 10^{-45}$
"Middle" denormalized number	*	−126	0	0000 0000	100 0000 0000 0000 0000 0000	$\pm 2^{-1} \times 2^{-126} = \pm 2^{-127} \approx \pm 5.88 \times 10^{-39}$
Largest denormalized number	*	−126	0	0000 0000	111 1111 1111 1111 1111 1111	$\pm(1-2^{-23}) \times 2^{-126} \approx \pm 1.18 \times 10^{-38}$
Smallest normalized number	*	−126	1	0000 0001	000 0000 0000 0000 0000 0000	$\pm 2^{-126} \approx \pm 1.18 \times 10^{-38}$
Largest normalized number	*	127	254	1111 1110	111 1111 1111 1111 1111 1111	$\pm(2-2^{-23}) \times 2^{127} \approx \pm 3.4 \times 10^{38}$
Positive infinity	0	128	255	1111 1111	000 0000 0000 0000 0000 0000	+∞
Negative infinity	1	128	255	1111 1111	000 0000 0000 0000 0000 0000	−∞
Not a number	*	128	255	1111 1111	non zero	NaN

* Sign bit can be either 0 or 1 .

图 2: 特殊的IEEE浮点数