2.信息的表示和处理

2.1 信息存储

2.1.2 字数据大小

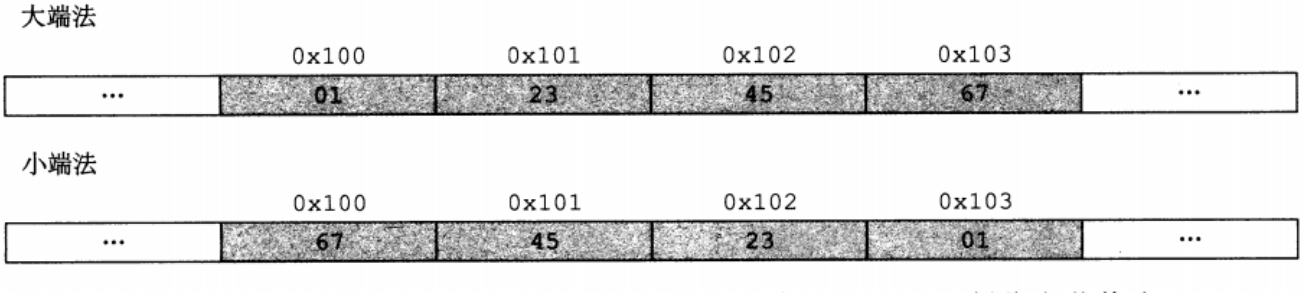
C声明		字节数	
有符号	无符号	32位	64位
[signed] char	unsigned char	1	1
short	unsigned short	2	2
int	unsigned	4	4
long	unsigned long	4	8
int32_t	uint32_t	4	4
int64_t	uint64_t	8	8
char *		4	8
float		4	4
double		8	8

图 2-3 基本 C 数据类型的典型大小(以字节为单位)。分配的字节数受程序是如何编译的影响而变化。本图给出的是 32 位和 64 位程序的典型值

大多数视为有符号数。

2.1.3 寻址和字节顺序

小端法：最低有效字节在最前面（最前面就是指内存地址小的地方）
大端法：最高有效字节在最前面
（但是字符串统一使用大端法，注意！）
大多数intel兼容机都使用小端模式。



2.1.7 C语言中的位级运算

自己对自己求异或就是0（说明每个元素是他自身的加法逆元）
自己对0异或，还是自己。

2.1.8 C语言中的逻辑运算

逻辑运算与位级运算的一个区别就是，如果对第一个参数求值就能确定表达式的结果，那么逻辑运算就不会对第二个参数求值（其实可以从汇编语言的角度看出来）。

2.1.9 C语言中的移位运算

算术右移SAR，在左侧补K个最高有效位的值，有符号数使用算术右移；无符号数必须使用逻辑右移SNL。

2.2 整数表示

C和C++都支持有符号）默认）和无符号数，Java只支持有符号数。

无符号数：$$B2U_w(\vec{x}) \doteq \sum_{i=0}^{w-1} x_i 2^i$$

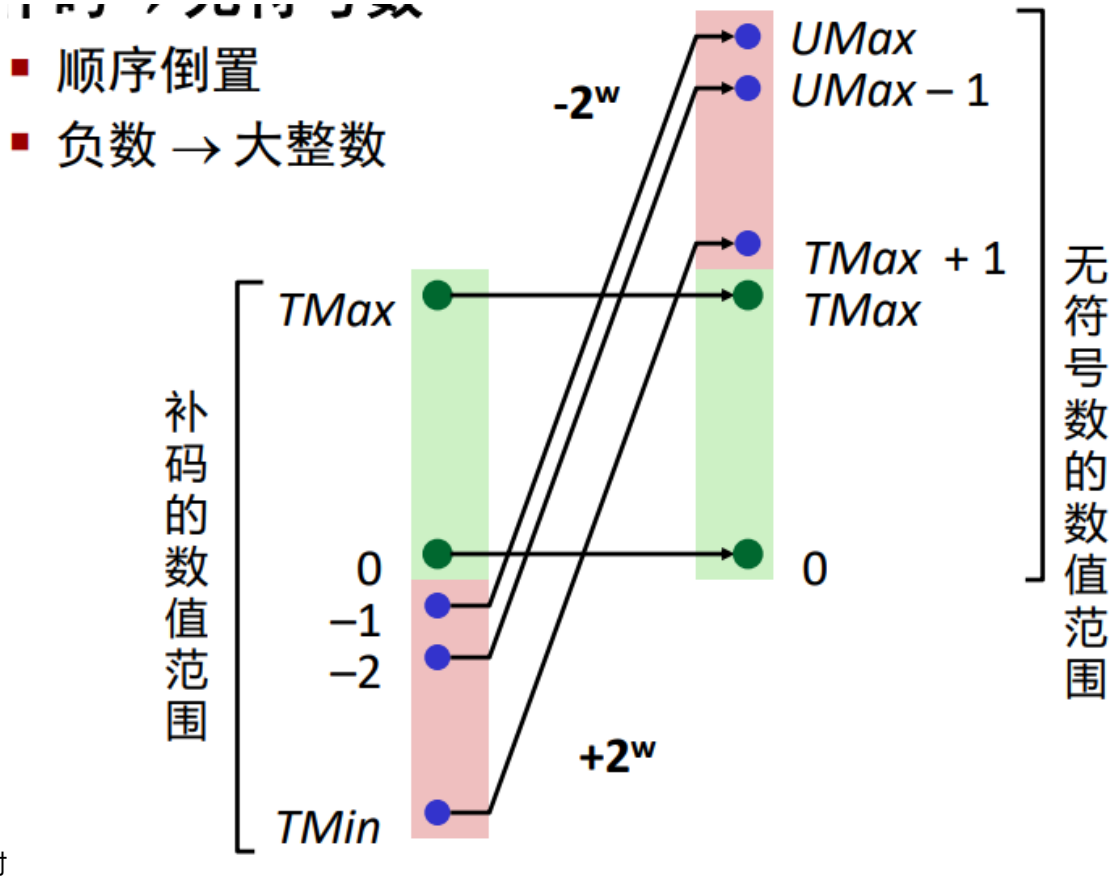
最小值为0，最大值为 $2^w - 1$

有符号数：$$B2T_w(\vec{x}) \doteq - x_{w-1} 2^{w-1} + \sum_{i=0}^{w-2} x_i 2^i$$

（就是用后面K-1位的和减去第K位的值）

最小值为 -2^{w-1} ，最大值为 $2^{w-1} - 1$

2.2.4 有符号数和无符号数之间的转换



就是映射

有符号数小于0的部分，加上 2^w 映射的无符号大于TMAX（有符号数能表示的最大值）的部分，无符号数大于TMAX的部分减去 2^w 映射到有符号数的负数部分；其余部分保持不变。

关键是要算出TMAX，**即** $2^{w-1} - 1$ **）**

2.2.5 C语言中有符号数与无符号数

当执行一个运算时，如果它的一个运算数是有符号而另一个是无符号的，那么C语言会隐式地将有符号参数强制类型转换为无符号数。

2.2.6 扩展一个数字的位表示

无符号左侧补零；
补码数符号位扩展。

2.2.7 截断数字

无符号数截断后的数值 $x' = x \bmod 2^k$
有符号截断后的数值相当于先将截断前的二进制字符串看成无符号数，利用求模算出阶段后地无符号数值，再利用无符号转成有符号映射成有符号数。

2.2.8 关于有符号数和无符号数地建议

* 无符号0-1后-1，转换城无符号数实际要加上2的n次方倍，会导致许多意想不到的错误。所以建议不使用无符号数。*

2.3 整数运算

2.3.1 无符号加法

原理：无符号数加法

对满足 $0 \leqslant x, y < 2^w$ 的 x 和 y 有：

$$x +_w^u y = \begin{cases} x+y, & x+y < 2^w \quad \text{正常} \\ x+y-2^w, & 2^w \leqslant x+y < 2^{w+1} \quad \text{溢出} \end{cases}$$

s=x+y，**当**s < x **(或等价的**s< y)**时发生了溢出**

2.3.2 补码加法

原理：补码加法

对满足 $-2^{w-1} \leqslant x, y \leqslant 2^{w-1}-1$ 的整数 x 和 y，有：

$$x +_w^t y = \begin{cases} x+y-2^w, & 2^{w-1} \leqslant x+y \quad \text{正溢出} \\ x+y, & -2^{w-1} \leqslant x+y < 2^{w-1} \quad \text{正常} \\ x+y+2^w, & x+y < -2^{w-1} \quad \text{负溢出} \end{cases}$$

（向下溢出就加2n，向上溢出就-2n）
（因为无符号与补码加法具有相同的位级表示，也可以先当作无符号加法，然后再映射）

2.3.3 补码的非

原理：补码的非

对满足 $TMin_w \leqslant x \leqslant TMax_w$ 的 x，其补码的非 $-_w^t x$ 由下式给出

$$-_w^t x = \begin{cases} TMin_w, & x = TMin_w \\ -x, & x > TMin_w \end{cases} \tag{2.15}$$

也就是说，对 w 位的补码加法来说，$TMin_w$ 是自己的加法的逆，而对其他任何数值 x 都有 $-x$ 作为其加法的逆。

十六进制表示上就是两个数加在一起是进制（所以对于十六进制，就是5+B=16，所以互为补码）
十进制就是加符号。

2.3.4 乘法

无符号乘法定义为产生w位的值，就是从2w位的整数乘积的低w位表示的值。如果用十进制运算就是乘后结果模2的w次方。

$$x *_w^u y = (x \cdot y) \bmod 2^w$$

补码乘法就是先用位级的无符号乘法，再映射。

2.3.6 乘以常数

乘以2的幂：就是左移幂位。

形式 A：(x<<n)+(x<<(n−1))+⋯+(x<<m)

形式 B：(x<<(n+1))−(x<<m)

2.3.7 除以2的幂

就是右移k位。
无符号是逻辑右移，结果是向下舍入。
有符号是算术右移，不对原数进行操作结果是向下舍入（但不太准确）；在移位前加上“偏置”值（2的位移-1），结果就是向零舍入。

2.4 浮点数

2.4.2 IEEE浮点值表示

所以阶码=指数+偏移（偏移为2的exp方-1）

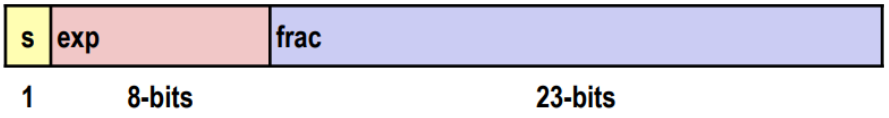
规则化浮点数：

$$+/-1.xxxx\ldots x * 2^{exp}$$

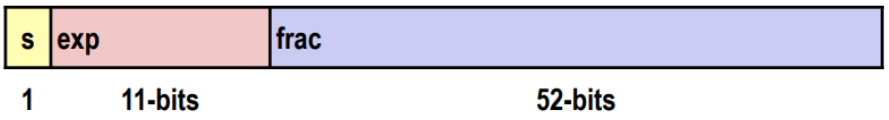
规格化尾数：小数点的前一位总是1，所以我们的M部分只需要表示小数点后的数，所以单精度23位可以表示24位的二进制小数，双精度52位可以表示53位的二进制小数。
规格化阶码：
指数 = 阶码 - 偏置
单精度阶码：0000 0001 ~ 1111 1110 用来表示的指数范围：-126 ~ 127。阶码是无符号数。**全0和全1的阶码用来做特殊的表示。**
单精度偏置：127，双精度偏置：1023
规格化浮点数：

$$(-1)^s * (1 + frac) * 2^{阶码}$$

单精度: 32 bits



双精度: 64 bits



描述	位表示	指数			小数		值		
		e	E	2^E	f	M	$2^E \times M$	V	十进制
0	0 0000 000	0	−6	$\frac{1}{64}$	$\frac{0}{8}$	$\frac{0}{8}$	$\frac{0}{512}$	0	0.0
最小的非规格化数	0 0000 001	0	−6	$\frac{1}{64}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{512}$	$\frac{1}{512}$	0.001953
	0 0000 010	0	−6	$\frac{1}{64}$	$\frac{2}{8}$	$\frac{2}{8}$	$\frac{2}{512}$	$\frac{1}{256}$	0.003906
	0 0000 011	0	−6	$\frac{1}{64}$	$\frac{3}{8}$	$\frac{3}{8}$	$\frac{3}{512}$	$\frac{3}{512}$	0.005859
	⋮								
最大的非规格化数	0 0000 111	0	−6	$\frac{1}{64}$	$\frac{7}{8}$	$\frac{7}{8}$	$\frac{7}{512}$	$\frac{7}{512}$	0.013672
最小的规格化数	0 0001 000	1	−6	$\frac{1}{64}$	$\frac{0}{8}$	$\frac{8}{8}$	$\frac{8}{512}$	$\frac{1}{64}$	0.015625
	0 0001 001	1	−6	$\frac{1}{64}$	$\frac{1}{8}$	$\frac{9}{8}$	$\frac{9}{512}$	$\frac{9}{512}$	0.017578
	⋮								
	0 0110 110	6	−1	$\frac{1}{2}$	$\frac{6}{8}$	$\frac{14}{8}$	$\frac{14}{16}$	$\frac{7}{8}$	0.875
	0 0110 111	6	−1	$\frac{1}{2}$	$\frac{7}{8}$	$\frac{15}{8}$	$\frac{15}{16}$	$\frac{15}{16}$	0.9375
1	0 0111 000	7	0	1	$\frac{0}{8}$	$\frac{8}{8}$	$\frac{8}{8}$	1	1.0
	0 0111 001	7	0	1	$\frac{1}{8}$	$\frac{9}{8}$	$\frac{9}{8}$	$\frac{9}{8}$	1.125
	0 0111 010	7	0	1	$\frac{2}{8}$	$\frac{10}{8}$	$\frac{10}{8}$	$\frac{5}{4}$	1.25
	⋮								
	0 1110 110	14	7	128	$\frac{6}{8}$	$\frac{14}{8}$	$\frac{1792}{8}$	224	224.0
最大的规格化数	0 1110 111	14	7	128	$\frac{7}{8}$	$\frac{15}{8}$	$\frac{1920}{8}$	240	240.0
无穷大	0 1111 000	—	—	—	—	—	—	∞	—

图 2-35　8 位浮点格式的非负值示例($k=4$ 的阶码位的和 $n=3$ 的小数位。偏置量是 7)

注意，虽然阶码为0，但是指数实际上为1-7=-6，而不是0-7=-7，同时阶码全0时整数位不含1。且规格化数最大到7次方，因为全1111用来表示非规格化数无穷

实例：（重点在于阶码的计算，不要把阶码和指数搞混了）
先将十进制数转换为二进制数，改写为二进制的科学计数法，求尾数，求阶码。

- **数值: `float F = 15213.0`**
  - $15213_{10} = 11101101101101_2$
    $= 1.1101101101101_2 \times 2^{13}$
- **尾数(Significand)**

M = 1.1011011011101_2

frac = $11011011011010000000000_2$

■ 阶码(Exponent)

E = 13

$Bias$ = 127

Exp = 140 = 10001100_2

■ 编码结果:

0	10001100	11011011011010000000000
s	exp	frac

根据浮点数求真值:

已知float型变量x的机器数为BEE00000H，求x的值是多少？

1	011 11101	110 0000 0000 0000 0000 0000

$(-1)^S \times (1 + Significand) \times 2^{(Exponent-127)}$

- 数符: 1（负数）
- 阶（指数）:
  - 阶码: 0111 1101B = 125
  - 阶码的值: 125 - 127 = -2

为避免混淆，用阶码表示阶的编码，用阶或指数表示阶码的值

- 尾数数值部分:

$1 + 1\times2^{-1} + 1\times2^{-2} + 0\times2^{-3} + 0\times2^{-4} + 0\times2^{-5} + \ldots$

$=1+2^{-1}+2^{-2} = 1+0.5+0.25 = 1.75$

- 真值: $-1.75\times2^{-2} = -0.4375$

阶码的值指代不同类型的数:

（1）阶码非全0和非全1，为规格化数

（2）阶码全0，为非规格化数

这里的规定相对于上面，全都变了！！！！！

此时的指数值为：**指数 = 1 - 偏置 = -126**（这里不是 0 - 偏置！！！）

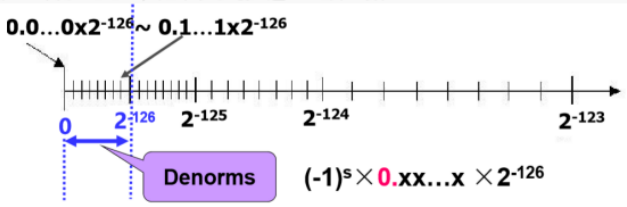
尾数值为：**0.frac**，这里没有前导1！！！！！

当frac全0的时候，此时的浮点数代表0，分为正0和负0:

+0: 0 00000000 00000000000000000000000

-0: 1 00000000 00000000000000000000000

当frac不为0的时候，用来表示最接近0.0的那些数



（3）阶码全1，为特殊数值

当阶码全1，frac全0的时候，用来表示无穷。

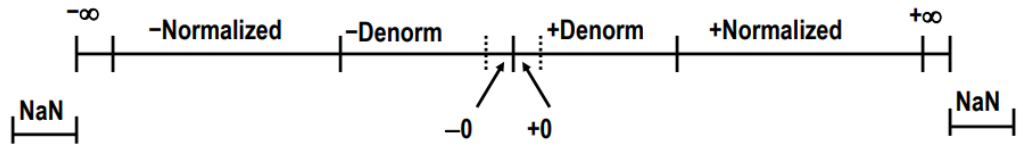
注意：浮点数除0会得到无穷，整数除0抛出异常。

+∞ : 0 11111111 00000000000000000000000

-∞ : 1 11111111 00000000000000000000000

当阶码全1，frac不为0的时候，用来表示NaN(not a number)，可以理解为比无穷还大的数被视为非数值，用来表示非法无数值结果的，主要用来调试代码。

划分图:



2.4.4 舍入

向偶数舍入：小数时，最低有效位后面的数，如果是中间的数（即1000），则将最后有效位向偶数舍入（0为偶数，1为奇数）。