第 2 章

C H A P T E R 2

信息的表示和处理

现代计算机存储和处理的信息以二值信号表示。这些微不足道的二进制数字，或者称为位(bit)，形成了数字革命的基础。大家熟悉并使用了 1000 多年的十进制(以 10 为基数)起源于印度，在 12 世纪被阿拉伯数学家改进，并在 13 世纪被意大利数学家 Leonardo Pisano(大约公元 1170—1250，更为大家所熟知的名字是 Fibonacci)带到西方。对于有 10 个手指的人类来说，使用十进制表示法是很自然的事情，但是当构造存储和处理信息的机器时，二进制值工作得更好。二值信号能够很容易地被表示、存储和传输，例如，可以表示为穿孔卡片上有洞或无洞、导线上的高电压或低电压，或者顺时针或逆时针的磁场。对二值信号进行存储和执行计算的电子电路非常简单和可靠，制造商能够在一个单独的硅片上集成数百万甚至数十亿个这样的电路。

孤立地讲，单个的位不是非常有用。然而，当把位组合在一起，再加上某种解释(interpretation)，即赋予不同的可能位模式以含意，我们就能够表示任何有限集合的元素。比如，使用一个二进制数字系统，我们能够用位组来编码非负数。通过使用标准的字符码，我们能够对文档中的字母和符号进行编码。在本章中，我们将讨论这两种编码，以及负数表示和实数近似值的编码。

我们研究三种最重要的数字表示。*无符号*(unsigned)编码基于传统的二进制表示法，表示大于或者等于零的数字。*补码*(two's-complement)编码是表示有符号整数的最常见的方式，有符号整数就是可以为正或者为负的数字。*浮点数*(floating-point)编码是表示实数的科学记数法的以 2 为基数的版本。计算机用这些不同的表示方法实现算术运算，例如加法和乘法，类似于对应的整数和实数运算。

计算机的表示法是用有限数量的位来对一个数字编码，因此，当结果太大以至不能表示时，某些运算就会*溢出*(overflow)。溢出会导致某些令人吃惊的后果。例如，在今天的大多数计算机上(使用 32 位来表示数据类型 int)，计算表达式 200*300*400*500 会得出结果 −884 901 888。这违背了整数运算的特性，计算一组正数的乘积不应产生一个负的结果。

另一方面，整数的计算机运算满足人们所熟知的真正整数运算的许多性质。例如，利用乘法的结合律和交换律，计算下面任何一个 C 表达式，都会得出结果 −884 901 888：

```
(500  * 400) * (300 * 200)
((500 * 400) * 300) * 200
((200 * 500) * 300) * 400
400   * (200 * (300 * 500))
```

计算机可能没有产生期望的结果，但是至少它是一致的！

浮点运算有完全不同的数学属性。虽然溢出会产生特殊的值 $+\infty$，但是一组正数的乘积总是正的。由于表示的精度有限，浮点运算是不可结合的。例如，在大多数机器上，C 表达式 (3.14+1e20)-1e20 求得的值会是 0.0，而 3.14+(1e20-1e20) 求得的值会是 3.14。整数运算和浮点数运算会有不同的数学属性是因为它们处理数字表示有限性的方式不同——整数的表示虽然只能编码一个相对较小的数值范围，但是这种表示是精确的；而浮