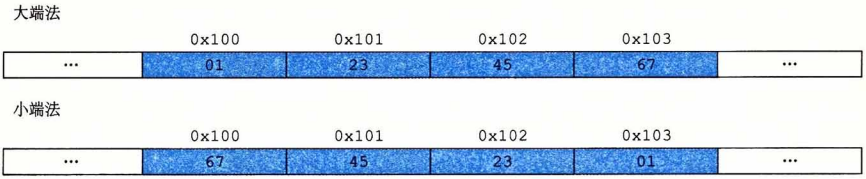
2.1.3　寻址和字节顺序

对于跨越多字节的程序对象，我们必须建立两个规则：这个对象的地址是什么，以及在内存中如何排列这些字节。在几乎所有的机器上，多字节对象都被存储为连续的字节序列，对象的地址为所使用字节中最小的地址。例如，假设一个类型为 int 的变量 x 的地址为 0x100，也就是说，地址表达式 &x 的值为 0x100。那么，(假设数据类型 int 为 32 位表示)x 的 4 个字节将被存储在内存的 0x100、0x101、0x102 和 0x103 位置。

排列表示一个对象的字节有两个通用的规则。考虑一个 w 位的整数，其位表示为$[x_{w-1}, x_{w-2}, \cdots, x_1, x_0]$，其中 x_{w-1}是最高有效位，而 x_0 是最低有效位。假设 w 是 8 的倍数，这些位就能被分组成为字节，其中最高有效字节包含位$[x_{w-1}, x_{w-2}, \cdots, x_{w-8}]$，而最低有效字节包含位$[x_7, x_6, \cdots, x_0]$，其他字节包含中间的位。某些机器选择在内存中按照从最低有效字节到最高有效字节的顺序存储对象，而另一些机器则按照从最高有效字节到最低有效字节的顺序存储。前一种规则——最低有效字节在最前面的方式，称为*小端法*(little endian)。后一种规则——最高有效字节在最前面的方式，称为*大端法*(big endian)。

假设变量 x 的类型为 int，位于地址 0x100 处，它的十六进制值为 0x01234567。地址范围 0x100～0x103 的字节顺序依赖于机器的类型：



注意，在字 0x01234567 中，高位字节的十六进制值为 0x01，而低位字节值为 0x67。

大多数 Intel 兼容机都只用小端模式。另一方面，IBM 和 Oracle(从其 2010 年收购 Sun Microsystems 开始)的大多数机器则是按大端模式操作。注意我们说的是“大多数”。这些规则并没有严格按照企业界限来划分。比如，IBM 和 Oracle 制造的个人计算机使用的是 Intel 兼容的处理器，因此使用小端法。许多比较新的微处理器是*双端法*(bi-endian)，也就是说可以把它们配置成作为大端或者小端的机器运行。然而，实际情况是：一旦选择了特定操作系统，那么字节顺序也就固定下来。比如，用于许多移动电话的 ARM 微处理器，其硬件可以按小端或大端两种模式操作，但是这些芯片上最常见的两种操作系统——Android(来自 Google)和 IOS(来自 Apple)——却只能运行于小端模式。

令人吃惊的是，在哪种字节顺序是合适的这个问题上，人们表现得非常情绪化。实际上，术语“little endian(小端)”和“big endian(大端)”出自 Jonathan Swift 的《格利佛游记》(Gulliver's Travels)一书，其中交战的两个派别无法就应该从哪一端(小端还是大端)打开一个半熟的鸡蛋达成一致。就像鸡蛋的问题一样，选择何种字节顺序没有技术上的理由，因此争论沦为关于社会政治论题的争论。只要选择了一种规则并且始终如一地坚持，对于哪种字节排序的选择都是任意的。

旁注　“端”的起源

以下是 Jonathan Swift 在 1726 年关于大小端之争历史的描述：