用来删除名为 MYLETTER.TXT 的文件，而运行下面的命令：

```
ERA  *.TXT
```

则所有的文本文件都被删除。一旦删除文件，此文件的目录项及其所占用的磁盘空间都将被释放。

REN 也是一个常用命令，它是 Rename 的缩写，此命令可以改变文件名。如果想显示文本文件的内容，可以使用 TYPE 命令，这条命令的功能还不仅如此，要知道文本文件中仅包含 ASCII 码，所以屏幕上文件的内容也可以通过这条命令来浏览，例如：

```
TYPE  MYLETER.TXT
```

表示查看 MYLETER.TXT 文件的内容。SAVE 命令用来保存文件，它可把临时存储区域中的一个或多个 256 字节的内存块保存到磁盘中，并且给这个内存块指定一个名字。

当然，上述所介绍的都是 CP/M 可识别的命令，如果你输入一个不能被 CP/M 识别的命令，CP/M 就会默认为输入的是保存在磁盘上的一个程序名。而程序通常是以文件形式存储的，其文件类型为 COM，代表着命令。控制台命令处理程序负责在磁盘上查找此文件，如果找到，此文件会被 CP/M 从磁盘加载到临时程序区域，该区域的地址从 0100h 开始，一旦文件被调入内存即可运行。上面从流程的角度介绍了磁盘中的文件是如何被运行的，下面举个例子来说明。假如在 CP/M 命令提示符后面输入：

```
CALC
```

如果在磁盘中存在名为 CALC.COM 的文件，则该文件会被控制台命令处理程序调入到以地址 0100h 开始的内存中，接着控制台命令处理程序会转到该地址并执行这段程序。

前面介绍了如何将机器码指令插入到内存空间的任意位置并执行。但是存储在磁盘上的 CP/M 程序并不能被随意存放到内存中的任意位置，它必须被加载到指定的位置，在这里是以 0100h 开始的内存空间。

CP/M 由一些实用的程序组成，如 PIP（Peripheral Interchange Program），即外设交换程序，通过它可以复制文件。ED 是文本编辑器，可以创建和修改文本文件。在 CP/M 系统中，有很多像 PIP 和 ED 一样的程序，它们很小但可以完成简单的事务处理，这类程序称为实用（utility）程序。仅仅有这些小的简单程序是远远不够的，还有一些大的商业化

应用程序（application），比如字处理软件或计算机电子报表软件等，在 CP/M 系统中，这些软件的使用会给你的工作带来很大方便。当然，如果你是一个程序开发高手，也可以自己动手编写这些软件，这些程序的存储类型都是 COM 文件类型。

在 CP/M（跟许多操作系统一样）中，我们了解了很多内容，比如如何利用命令和实用程序对文件进行基本操作，如何将程序加载到内存中并运行等。操作系统的功能远不止如此，下面将介绍它的第三个重要功能。

前面提到过，把输出信息写到视频显示器上，或者从键盘读取输入的内容，或者读/写磁盘中的文件，这些都是运行在 CP/M 下的程序常常要做的操作。这就需要 CP/M 程序能够直接向视频显示器的内存写入输出内容，也需要 CP/M 程序能够访问键盘硬件来捕获所输入的内容，还需要 CP/M 程序能够访问磁盘驱动器来读/写磁盘扇区，然而在通常情况下程序本身是很难直接做到的。

那么有没有别的方法来实现上述的要求呢？答案是肯定的，这些常用事务由 CP/M 中的子程序集来完成，在 CP/M 下运行的程序通过调用这些子程序即可完成相应的操作。这些子程序都是专门设计的，计算机中的所有硬件都可以很容易地通过它们来访问，如视频显示器、键盘、磁盘驱动器等，而程序员无须关心这些外设实际是如何连接的。更重要的是，像磁道、扇区这类信息，程序没有必要知道，这些工作都是由 CP/M 来完成的，它可以负责读/写磁盘上的文件。

操作系统提供的第三个主要功能是让程序能够方便地访问计算机的硬件，操作系统提供的这种访问操作称为 API（Application Programming Interface），即应用程序接口。

那么如何设置和使用 API 呢？在 CP/M 下运行的程序，可以通过将寄存器 C 设置为特定的值（称为功能值），并且运行如下指令：

```
CALL 5
```

来使用 API。例如，你从键盘上按下了一个键，程序会通过执行下面的指令来获取此键对应的 ASCII 码：

```
MVI C,01h
CALL 5
```

并且将这个键的 ASCII 码值保存在累加器 A 中。类似的，运行这条命令：

```
MVI C,02h
CALL 5
```

在视频显示器上当前的光标位置将显示累加器 A 中的 ASCII 码字符，然后光标移到下一个位置。

如果程序要新建一个文件，它首先将文件名所在区域的地址保存在寄存器对 DE 中，接着执行如下代码：

```
MVI  C,16h
CALL 5
```

执行此命令，CP/M 会在磁盘上新建一个空文件。程序可以利用 CP/M 提供的其他功能来向空文件中写入内容，最后关闭（close）文件，关闭文件意味着文件使用完毕，不需要在对该文件执行任何操作了。当然，该文件可以再次被此程序和其他程序打开（open）并读取内容。

CALL 5 指令到底如何工作呢？CP/M 在内存中地址为 0005h 处设置了一条 JMP（Jump）指令，它跳转到 CP/M 基本磁盘操作系统（Basic Disk Operating System，BDOS）中的某个位置。这个区域包含许多小程序，CP/M 的每一项功能都可由它们完成。BDOS，顾名思义，它的主要功能是维护磁盘上的文件系统。文件系统经常要与终端设备打交道，所以 BDOS 经常要调用 CP/M 基本输入/输出系统（Basic Input And Output System，BIOS）中的一些子程序。这里顺便提一下，BIOS 可以对硬件进行访问，比如键盘、视频显示器和磁盘驱动器等。实际上，BIOS 是 CP/M 中唯一需要了解计算机中硬件的程序，其他一些对硬件的操作都可通过调用 BIOS 中的子程序来实现。控制台命令处理程序通过调用 BDOS 的子程序来实现自己所有的功能，而在 CP/M 中运行的程序也是这样。

对于计算机硬件来说，API 是一个与设备无关（device-independent）的接口。换言之，对于特定的机器上键盘的工作机制、视频显示器的工作机制以及磁盘扇区的读/写机制，在 CP/M 下编写的程序不需要知道也没有必要知道。程序使用 CP/M 提供的功能便可完成对键盘、视频显示器和磁盘驱动器操作，简言之，API 屏蔽了硬件之间的差异。有了 API，尽管不同计算机硬件差别很大，其访问外设的方式也不尽相同，但 CP/M 程序都可以在上面运行，从而实现了 CP/M 程序的跨平台（这里要求所有 CP/M 程序必须运行在 Intel 8080 微处理器上，或者运行在能执行 Intel 8080 指令的处理器上，例如 Intel 8085 或 Zilog Z-80

处理器）。所以，在使用 CP/M 系统的计算机中，程序对硬件访问是通过 CP/M 来实现的。如果没有标准的 API，程序必须根据不同型号的计算机进行相应的修改后，才能访问硬件。

CP/M 是 8080 中非常流行的操作系统，曾经辉煌一时，至今仍有重要的历史意义。16 位操作系统 QDOS（Quick and Dirty Operating System）的开发在很多方面都借鉴了 CP/M 的思想。QDOS 当初是专为英特尔公司的 16 位 8086 和 8088 芯片而设计的，它出自西雅图计算机产品公司（Seattle computer product）的提姆·帕特森之手。QDOS 系统被微软公司注册后更名为 86-DOS。1981 年，随着 IBM 第一代 PC 诞生，微软公司也将 86-DOS 更名为 MS-DOS（Microsoft Disk Operating System），并以此名授权给 IBM 公司用作第一代个人计算机的操作系统。这就是著名的 MS-DOS 系统，在现在的计算机中也会经常看到它的身影。虽然 16 位版本的 CP/M（称为 CP/M-86）也可用于 IBM PC，但由于 MS-DOS 的影响力更大，其很快成为了标准。其他生产 IBM PC 兼容机的厂商也被授权使用 MS-DOS（在 IBM 计算机上称为 PC-DOS）系统。

CP/M 的文件系统在 MS-DOS 没有被继续使用，在 MS-DOS 中，文件系统是以文件分配表（FAT，File Allocation Table 的简写）的形式来组织的，Microsoft 公司早在 1977 年就开始使用 FAT 了。FAT 的基本思想是：将磁盘空间分成簇（cluster）——簇的大小由磁盘空间的大小来决定——从 512 字节到 16 K 字节不等，每个文件占用若干簇。文件的目录项只记录文件起始（starting）簇的位置，而磁盘上每一簇的下一簇的位置由 FAT 来记录。

每个目录项在 MS-DOS 磁盘上占用 32 字节，其命名形式跟 CP/M 上的 8.3 形式一样，只是使用的术语有些区别：最后的三个字符称做文件扩展名，而非 CP/M 中的文件类型。MS-DOS 目录项中没有包含分配块列表，它主要包含如下所示的有用信息：文件的最后修改的日期、时间和文件的大小等。

实际上，MS-DOS 的早期版本与 CP/M 在结构上很类似，只是 IBM PC 本身在 ROM 中已经包含了一套完整的 BIOS 了，所以在 MS-DOS 中不再需要 BIOS。在 MS-DOS 中，命令处理器是一个命名为 COMMAND.COM 的文件。MS-DOS 有两种运行程序：一是以 COM 为扩展名的文件，其大小不能超过 64 KB，二是更大一些的程序，以 EXE（意思是可以被执行）为文件的扩展名，表明文件本身是可执行的。

尽管 MS-DOS 起初支持 API 函数的 CALL 5 接口，但不久 Microsoft 为新的程序重新

设计了一款新的接口。这个新的接口使用了 8086 的一个称为软件中断（software interrupt）的功能，说起软件中断，它与子程序调用很类似，只不过程序不必知道它所调用的子程序的确切地址。通过执行 INT 21h 这条指令，程序可以调用 MS-DOS 的 API 功能。

从理论上讲，应用程序并不能直接访问计算机的硬件，如果它要访问计算机的硬件，可以通过操作系统提供的接口来进行。然而实际上，在 20 世纪 70 年代末和 80 年代初，由于计算机上运行的都是小型操作系统，许多应用程序往往都绕过它们，这种情况在处理有关视频显示器任务的时候显得特别明显。为什么会这样呢？因为如果程序把字节直接写入视频显示存储器，它的执行速度要远快于不直接写入视频显示存储器。事实上，对于一些程序——比如要将图形显示在视频显示器上——操作系统就显得“心有余而力不足”，这就需要在操作系统之外另作处理。也许正是因为 MS-DOS 系统卓越的“反传统性”，使得大多数程序员都很热衷于它，它可以让程序员编写的程序最大限度地达到硬件的最快速度，更加充分地发挥硬件的性能。

恰恰是这个原因，运行在 IBM PC 上的流行软件通常依赖于 IBM PC 这个硬件平台，换句话说，就是这些软件是根据 IBM PC 的硬件特点编制的。为了和 IBM PC 竞争，其他机器制造商不得不沿袭这些特点。如果不这样做的话，再流行的软件也将流行不起来，因为它不能在这些机器上运行。所以在软件的显著位置往往会出现“与 IBM PC 百分之百兼容”的字样，这同时体现了软件对硬件的要求。

微软于 1983 年 3 月发布了 MS-DOS 2.0 版本，与最开始的版本相比，它加强了对硬盘驱动器的管理。虽然当时的硬盘容量很小（按今天的标准），但是发展很快，没经过多久，硬盘的容量变得越来越大。这带来了不少问题，硬盘容量越大存储的文件也就越多，存储的文件越多，当然，查找某个指定的文件或组织文件也就越困难。

为了解决上述问题，MS-DOS 2.0 引入了层次文件系统（hierarchical file system），它只是在原有 MS-DOS 的文件系统上做了一些小的改动。前面介绍过，目录存储在磁盘中特定区域，它是一个文件列表，包含了文件存储在磁盘位置的信息。在层次文件系统中，有些文件其本身可能就是目录，也就是说这些文件包含其他文件，其中的一些可能还是目录。在磁盘中，目录的称法也是有讲究的，常规的目录称为根目录（root directory），子目录（subdirectories）是包含在其他目录里的目录。有了目录（有时称文件夹，folder），就可以很方便地对相关文件进行分组，所以目录在磁盘文件的管理中起着非常重要的作用。

讲到操作系统，我们不能不提到著名的 UNIX 系统，它在操作系统的发展史上有着举足轻重的地位。实际上，层次文件系统和 MS-DOS 2.0 的其他很多功能都是从 UNIX 操作系统借鉴而来的。UNIX 操作系统是 20 世纪 70 年代初在贝尔实验室开发的，肯·汤普森（Ken Thompson，生于 1943 年）和丹尼斯·里奇（Dennis Ritchie，生于 1941 年）是此系统的主要开发者。UNIX 系统（包括名字本身）的发展历程在这里要提一下，UNIX 系统来源于早期的 Multics（Multiplex Information and Computing Services，表示多路复用信息和计算业务），Multics 是贝尔实验室和 MIT（麻省理工大学）和 GE（通用电气公司）合作开发的一个项目，开发初期由于 Multics 没能达到预定的目的而且进度缓慢，加之昂贵的开发代价，1969 年贝尔实验室退出了该项目，但肯·汤普森和丹尼斯·里奇继续对此进行了开发，为了区别于 Multics，取名为 UNIX。

对于计算机核心程序开发的精英们来说，UNIX 无疑是最受欢迎的操作系统。以前大部分操作系统是针对特定的计算机硬件平台开发的，但 UNIX 打破了常规，它不针对具体的计算机硬件平台，具有很好的可移植性（portable），也就意味着它在各种机器上都可以运行。

在开发 UNIX 的时候，贝尔实验室是电信业巨头 AT&T（American Telephone & Telegraph，美国电话电报公司）旗下的一员，为了限制 AT&T 在电话业务的垄断地位，法院裁定禁止 AT&T 销售 UNIX 系统，AT&T 被迫将它授权给别人。因此从 1973 年初，很多大学、公司和政府机构被授权使用和研究 UNIX，这在一定程度上促进了 UNIX 的发展。1983 年，AT&T 获准重返计算机业务并发布了它自己的 UNIX 版本。

由于 UNIX 的广泛授权导致了许多不同版本共存的情况，它们使用着不同的名字，运行在不同的计算机上，并由不同的经销商销售。由于 UNIX 的影响力和开源性，使得很多人将精力投入到 UNIX 中并推动着它的不断发展。当人们向 UNIX 中添加新的组件时，无形之中流行着一种不成文的“UNIX 思想”。在这种思想的指导下，人们都使用文本文件作为公用文件形式。许多 UNIX 实用程序读取文本文件，利用它提供的一些功能做些处理，然后将其写入另一个文本文件。因此可以用链的形式来组织 UNIX 实用程序，然后对这些文本文件做相应的处理。

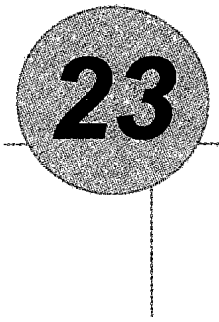
在 20 世纪 60-70 年代，计算机不仅体积庞大而且价格昂贵，仅仅为一个人服务显然不太现实，为了能够让多个人同时使用计算机，必须有相应操作系统来支持，这也就是

开发 UNIX 系统的最初目的。使用 UNIX 系统的计算机通过时分复用（time sharing）技术——这种技术允许多个用户同时与计算机进行交互——来达到这个目的。计算机连接多个配备了显示器和键盘的终端（terminals），每个用户通过这些终端访问计算机。通过在所有终端间的快速切换，使用户感觉这台计算机似乎只为自己工作，而其实计算机同时在为多个用户服务。

如果在一个操作系统上可以同时运行多个程序，则称此系统为多任务（multitasking）操作系统。显然，与 CP/M 和 MS-DOS 这样的单任务的系统相比，这种操作系统要复杂得多。正是由于支持多任务这种功能，文件系统变得很复杂，因为同一个文件可能被多个用户同时访问。程序的运行需要占用内存空间，多任务系统就要考虑内存的分配问题，也就是说需要进行内存管理（memory management）。也许你会有疑问，多道程序并行运行需要占用大量内存，如果内存不够怎么办？为此，操作系统引入了虚拟内存（virtual memory）技术。虚拟内存是指，在磁盘上划出部分空间用做保存临时文件，程序把暂时不需要用的内存块放到临时文件里，待需要时再把它调入内存。。

UNIX 能够存在并发展到现在是无数人共同努力的结晶，如今 FSF（Free Software Foundation，自由软件基金会）和 GNU 项目为推动 UNIX 的发展注入了新的活力，它们都是由理查德·斯塔门（Richard Stallman）创建的。GNU 意味着："GNU 与 UNIX，既要划清界限又相辅相成"。GNU 项目的宗旨是：创建一个与 UNIX 系统兼容，但不受私有权限制的操作系统和开发环境。在这个项目的推动下，涌现出了许多和 UNIX 兼容的实用程序和工具，其中最著名的要算 Linux 系统。Linux 系统的内核（Core 与 Kernel 都可以表示内核的意思）和 UNIX 的是完全兼容的，它的大部分程序是由芬兰的李纳斯·托沃兹（Linus Torvalds）完成的。由于 Linux 系统的开源性，近年来已成为非常流行的操作系统。

从 20 世纪 80 年代中期开始，开发大型的、复杂度更高的系统成为了操作系统发展的一大趋势，例如苹果公司的 Macintosh 系统和微软的 Windows 系统，它们融合了图形和高级可视化视频显示技术，使应用程序变得更易于使用。关于图形和可视化方面的发展趋势，我们将在本书的最后一章进一步介绍。



定点数和浮点数

数字就是数字，整数、分数以及百分数等各种类型的数字与我们形影不离，它几乎出现在我们生活的所有角落。例如你加班 2.75 小时，而公司按正常工作时间的 1.5 倍支付你工资，你用这些钱买了半盒鸡蛋并交了 8.25%的销售税。就算你不是数字研究方面的专家，也可能对这种“数字生活”非常熟悉。我们还经常听到类似这样的统计信息“美国每个家庭的平均人口是 2.6 人”，但谁也不会为了满足这个数字表达的真实含义而把人拆解掉（这种事情想起来都觉得恐怖）。

在计算机存储器中，整数和分数之间的转换并不是这么随意。现在我们应该清楚，计算机中的一切数据都是以位的形式存储的，这就意味着所有的数都表示为二进制形式。但另一个不可否认的事实是，某些类型的数比其他类型更容易用位的形式来表示。

我们将从整数的二进制表示开始，这里的整数被数学家称做“自然数”（positive whole numbers），即计算机程序员口中的“正整数”（positive integers），之后将介绍如何利用 2 的补数来表示“负整数”（negative integers），该方法可以让正数和负数的相加变得非常简单。下面的表格列出了 8 位、16 位、32 位二进制数所能表示的正整数及其 2 的补数的范围。

数的位数	正整数的范围	整数的 2 的补数范围
8	0～255	–128～127
16	0～65,535	–32,768～32,767
32	0～4,294,967,295	–2,147,483,648～2,147,483,647

我们所要介绍的整数部分就是这些。除此之外，数学家还定义了用两个整数的比值表示的一类数，称做有理数（rational number）或分数（fraction）。例如，3/4 是一个有理数，因为它是整数 3 和 4 的比。我们也可以把 3/4 表示成十进制小数的形式，即 0.75。尽管我们可以把它写成十进制数的形式，但它实际上代表一个分数，即 75/100。

如第 7 章所述，在十进制数字系统中，小数点左边的数的每一位都和 10 的正整数次幂相关，而其右边的数的每一位都和 10 的负整数次幂相关。我们在第 7 章用到了 42705.684 这个实例，现在，我们把它表示为以下等价形式：

$$
\begin{aligned}
&4 \times 10000 + \\
&2 \times 1000 + \\
&7 \times 100 + \\
&0 \times 10 + \\
&5 \times 1 + \\
&6 \div 10 + \\
&8 \div 100 + \\
&4 \div 1000
\end{aligned}
$$

上面的除号表达了该位置的数与 10 的负整数次幂相关的情况，用下面的表达方式可以不用除号：

$$
\begin{aligned}
&4 \times 10000 + \\
&2 \times 1000 + \\
&7 \times 100 + \\
&0 \times 10 + \\
&5 \times 1 + \\
&6 \times 0.1 + \\
&8 \times 0.01 +
\end{aligned}
$$

$$4 \times 0.001$$

最后，我们将该数表示为 10 的幂的形式：

$$\begin{aligned}
&4\times10^{4}+\\
&2\times10^{3}+\\
&7\times10^{2}+\\
&0\times10^{1}+\\
&5\times10^{0}+\\
&6\times10^{-1}+\\
&8\times10^{-2}+\\
&4\times10^{-3}
\end{aligned}$$

有一些有理数很难表示成小数，最明显的一个例子是 1/3，如果用 3 来除 1，会得到这样的结果：

$$0.33333333333333333333333333333333333333333333333333333333333333333...$$

在小数点的后面将会有无数个 3。通过在第一个 3 的上面加一条短横线，可以将这个无限循环的数简单表示为：

$$0.\overline{3}$$

尽管如此，把 1/3 表示为小数还不是很方便，但它毕竟是一个有理数，因为在本质上它是两个整数的比。类似的，1/7 可以表示为：

$$0.142857142857142857142857142857142857142857142857142857...$$

或

$$0.\overline{142857}$$

无理数（irrational number）是一些更加奇特的数，如 2 的平方根等。它们不能表示为两个整数的比，这就意味着其小数部分是无穷的，而且毫无规律，没有循环，例如：

$$\sqrt{2}\approx1.41421356237309504880168872420969807856967187537695...$$

下面这个数学方程式的正数解就是 2 的平方根：

$$x^2-2=0$$

如果某个数不是任何以整数为系数的代数方程的解，那么这个数称做超越数（transcendental，所有的超越数都是无理数，但是反之不成立）。π就是一个典型的超越数，它是圆的周长与其直径的比值，可以近似的表示为：

3.14159265358979328462643383279502884197169399375 11...

e是另一个典型的超越数，它是数学表达式：

$$(1+\frac{1}{n})^n$$

当n趋向无穷大时的值，近似的值为：

2.71828182845904523536028747135266249775724709369996...

目前我们所讨论过的所有数——有理数和无理数——统称为实数（real numbers）。使用实数定义它们的目的是为了将其与虚数（imaginary numbers）区别开来，虚数是负数的平方根。实数和虚数一起构成了复数（complex numbers）。不管名称如何，它们都有重要的作用，例如，虚数确实存在于现实世界，它在解决电子学的某些高级问题中有重要应用。

我们习惯于把数字看做连续（continuous）的，任意给出两个有理数，都可以找出一个位于它们之间的数。实际上，只需要取这两个数的平均值即可。但是，数字计算机对连续数据却无能为力，因为二进制中的每一位非0即1，两者之间没有任何数。这一特点决定了数字计算机只能处理离散（discrete）数据。二进制数的位数直接决定了所能表示的离散数值的个数。例如，如果你选择的二进制位数是32，则所能表示的自然数的范围是0～4,294,967,295。如果想要在计算机中存储4.5这个数，则需要选择新的方法并做一些其他方面的改进。

小数也可以表示为二进制数吗？当然可以，最简单的方法可能就是使用BCD码（二进制编码的十进制数）。如第19章所述，BCD码是将十进制数以二进制的形式进行编码，0～9之间的每一个数都需要用4位来表示，如下表所示。

十进制数字	BCD 代码
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

BCD 编码在程序处理用美元和美分表示的钱款、账户时特别有用。银行和保险公司是非常典型的两类整日与钱打交道的机构，这些机构所使用的计算机程序中，大多数小数所占用的存储空间仅仅相当于两个十进制数所占用的位数。

通常把两个 BCD 数字存放在一个字节，这种方式称为压缩 BCD（packed BCD）。由于 2 的补数不和 BCD 数一起使用，因此压缩 BCD 通常需要增加 1 位用来标识数的正负，该位被称做符号位（sign bit）。用一整个字节保存某个特定的 BCD 数是很方便的，但要为这个短小的符号位牺牲 4 位或 8 位的存储空间。

让我们来看一个例子，假设计算机程序所要处理的钱款数目在+/−100 万美元之间，这就意味着，需要表示的钱的数目的范围是 –9,999,999.99～99,999,999.99，因此保存在存储器中的每一笔钱的金额都需要 5 个字节。因此，—4,325,120.25 可以表示为下面 5 个字节：

00010100　00110010　01010001　00100000　00100101

将每个字节转换成十六进制数，上面的数可以等价地表示成：

14h　　32h　　51h　　20h　　25h

注意，最左边的半个字节所构成的 1 用来指明该数是负数，这个 1 即符号位。如果这半个字节所构成的数是 0，则说明该数是正数。组成该数的每一个数字都需要用 4 位来表示，从十六进制的表示形式中可以很直观地看到这一点。

如果要表示的数的范围扩大到 –99 ,999,999.99～99 ,999,999.99，则我们需要用 6 个字

节来实现，其中 5 个字节用来表示 10 个数字，另一个字节整个用来做符号位。

这种基于二进制的存储和标记方式也被称作定点格式(fixed-point format),所谓的“定点”是指小数点的位置总是在数的某个特定位置——在本例中，它位于两位小数之前。值得注意的是，有关小数点位置的计数信息并没有与整个数字一起存储。所以，使用定点小数的程序必须知道小数点的位置。你可以设计有任意小数位的定点小数，并且可以在程序中混合使用它们，但程序中对这些数进行算术运算的部分都需要知道小数点的位置，这样才能正确地对其做各种运算处理。

如果可以确定程序用到的数字不会大到超过预定的存储空间，并且这些数的小数位不会很多，那么使用定点格式的小数将是一个很好的选择。在表示非常大或非常小的数时，使用定点格式数是绝对不合适的。假设需要保留一块内存空间用来存放以英尺为单位的距离数据，可能存在的问题是某些距离的长度可能超出范围。地球与太阳之间的距离是 490,000,000,000 英尺，而氢原子的半径只有 0.000,000,000,26 英尺，如果采用定点格式的存储方案，为了存储这些极大或极小的数需要 12 个字节。

科学家和工程师们喜欢使用一种称为“科学计数法”(scientific notation) 的方法来记录这类较大或较小的数，利用这种计数系统可以更好地在计算机中存储这些数。科学计数法把每个数表示成有效位与 10 的幂的乘积的形式，这样就可以避免写一长串的 0，因此这种计数方式特别适合表示极大或极小的数。采用科学计数法，下面这个数：

$$490,000,000,000$$

可以记为：

$$4.9\times10^{11}$$

而

$$0.00000000026$$

可以表示为：

$$2.6\times10^{-10}$$

在上面的两个例子中，4.9 和 2.6 被称做小数部分或者首数 (characteristic)，有时候

也被称作尾数（mantissa，这个词通常与对数运算一起使用）。在计算机术语中这一部分被称做有效数（significand），因此为了保持一致，这里把科学计数法表示形式中的这一部分也称作有效数。

采用科学计数法表示的数可以分为两部分，其中指数（exponent）部分用来表示 10 的几次幂。在第一个例子中，指数是 11，第二个例子中指数是−10。指数可以表明小数点相对于有效数移动的距离。

为了便于操作，一般规定有效数的取值范围是大于或等于 1 而小于 10。尽管下面列出的各种写法表示的都是同一个数：

$$4.9\times10^{11}=49\times10^{10}=490\times10^{9}=0.49\times10^{12}=0.049\times10^{13}$$

但是上面等式中的第一种写法是最恰当的。这种写法有时被称做科学计数法的规范化式（normalized）。

这里需要说明，指数的正负性只是表明了数的大小，它并不能指明数本身的正负性。下面列出两个负数的科学计数法表示形式：

$$-5.8125\times10^{7}$$

与−58,125,000 相等。而

$$-5.8125\times10^{-7}$$

则与

$$-0.000,000,058,125$$

相等。

在计算机中，对于小数的存储方式，除了定点格式外还有另外一种选择，它被称做浮点格式（floating-point notation）。因为浮点格式是基于科学计数法的，所以它是存储极大或极小数的理想方式。但计算机中的浮点格式是借助二进制数实现的科学计数法形式，因此我们首先要了解如何用二进制表示小数。

实际操作起来比预想的要简单。在十进制数中，小数点右边的数字与 10 的负整数次

幂相关联；而在二进制数中，二进制小数点（就是一个简单的句点，看起来同十进制小数点一样）右边的数字和 2 的负整数次幂相关。例如，下面这个二进制数：

101.1101

可以用如下方式转换为十进制数：

$$
\begin{aligned}
&1\times 4+\\
&0\times 2+\\
&1\times 1+\\
&1\div 2+\\
&1\div 4+\\
&0\div 8+\\
&1\div 16
\end{aligned}
$$

将乘数和除数用 2 的整数次幂替换，就可以替换除号：

$$
\begin{aligned}
&1\times 2^{2}+\\
&0\times 2^{1}+\\
&1\times 2^{0}+\\
&1\times 2^{-1}+\\
&1\times 2^{-2}+\\
&0\times 2^{-3}+\\
&1\times 2^{-4}
\end{aligned}
$$

2 的负整数次幂等于从 1 开始反复除以 2，上式可以改写为：

$$
\begin{aligned}
&1\times 4+\\
&0\times 2+\\
&1\times 1+\\
&1\times 0.5+\\
&1\times 0.25+\\
&0\times 0.125+\\
&1\times 0.0625
\end{aligned}
$$

经过这种计算，101.1101 与十进制数 5.8125 是相等的。

在十进制的科学计数法中，规范化式的有效数应该大于或等于 1 且小于 10；类似的，