中等专业学校计算机专业教材

微型计算机原理入门

周华英 王景玉 陆 海 编著

科学出版社

中等专业学校计算机专业教材

微型计算机原理入门

周华英　王景玉　陆　海 编著

科 学 出 版 社

2002

内 容 简 介

本书以 Intel 8086 CPU 作为典型机型，介绍微型计算机的原理。内容包括：计算机基础知识、Intel 8086 微处理机、8086 CPU 的指令系统、宏汇编程序设计、内存储器、输入输出系统和接口技术。全书共分七章，配有习题。

本书可供各类中等专业学校计算机专业作为教材，亦可供有关专业师生、工程技术人员和其他读者学习、参考。

图书在版编目(CIP)数据

微型计算机原理入门/周华英等编著.-北京：科学出版社，2000
（中等专业学校计算机专业教材）
ISBN 7-03-008279-6

Ⅰ.微… Ⅱ.周… Ⅲ.微型计算机-基本知识-专业学校-教材
Ⅳ.TP36

中国版本图书馆 CIP 数据核字（2000）第 60220 号

科学出版社 出版
北京东黄城根北街 16 号
邮政编码：100717
北京双青印刷厂 印刷
科学出版社发行 各地新华书店经销
*
2000 年 8 月第 一 版 开本：787×1092 1/16
2002 年 1 月第二次印刷 印张：17
印数：4 501—6 500 字数：397 000
定价：25.00 元
(如有印装质量问题,我社负责调换<环伟>)

中等专业学校计算机专业教材

编　委　会

出版说明

在计算机科学技术飞速发展、广泛应用、深入普及的今天，计算机科学技术图书的出版发行轰轰烈烈、规模空前。但是，在浩瀚的计算机出版物中，人们很难寻觅到适合当前中等专业技术学校使用的计算机专业及专业基础教材，更难发现适合具有高中文化程度的计算机爱好者、“发烧友”系统学习计算机基础知识的图书。因此，我们组织了部分具有丰富中等专业技术教育经验的优秀教师和部分计算机技术专家，编写了这套中等专业学校计算机专业教材。

“中等专业学校计算机专业教材”大致可以分为以下三个模块：

（1）专业基础知识模块，包括：

- 计算机应用基础
- 计算机数学初步
- 模拟和数字电路基础
- 计算机英语阅读
- 数据结构与算法入门

（2）专业知识模块，包括：

- PASCAL 程序设计实践
- 数据库应用基础
- 计算机操作系统基础
- 数据通信基础
- C 语言程序设计
- 微型计算机原理入门
- 计算机网络技术基础
- 多媒体技术基础

（3）实用技术模块，包括：

- 视窗系统及办公应用软件
- 微型计算机系统维护与故障诊断
- 因特网应用入门
- 计算机绘图

我们衷心感谢南京市中等专业（走读）学校的教师在教材编撰过程中所给予的大力支持和关心指导，衷心感谢中国人民解放军理工大学计算机与指挥自动化学院专家、教授们的精心组织和辛勤工作，衷心感谢南京同创计算机学校各级领导和老师们，感谢他们的通力协作和在部分书稿试用阶段的卓有成效的实践。

中等专业学校计算机专业教材

编委会

1999 年 7 月

目　录

第一章 概 述

1.1 计算机数据表示

1.1.1 计算机中数值数据的表示

电子计算机主要是由只能识别和处理两个稳定状态的器件组成的。例如，计算机中用一些只有两个稳定状态的器件组成的寄存器或存储器来存储数据。因此，进入计算机的数字、汉字、符号、图象以及声音等信息都必须转换成由0、1两个数字组合成的数据形式才能被计算机接受、存储和处理。但是当用计算机处理实际数值时将碰到更多的情况，比如正数与负数、带小数点的数等。也就是说，在计算机中表示一个数值型数据，需要解决三个问题：确定数的长度、确定数的符号和小数点的位置。如何在计算机中准确且明白地表示各种不同的数据，并使之更加适合于计算机的运算，就成了学习计算机时必须掌握的基础知识。计算机中的数据分为数值数据和非数值数据两大类。我们把能够进行算术运算并有明确数值概念的数据称为数值数据，其余的数据称为非数值数据。下面，我们把计算机中的数值数据分为有符号数、无符号数、小数以及二进制表示的十进制数（简称“二-十进制数”）等几种情况分别进行讨论。

1. 无符号数的表示

在有些情况下不需要考虑数的正负符号，比如对学校的一个班级学生进行的编号，就可以用无符号数的方式表示它，这种表示法与二进制数的基本形式完全一样。以八位二进制数为例，它最小值为“00000000”，而最大值为“11111111”，即它所表示的数值范围从0到255，共256个数。一般来说，一个n位的二进制数，它表示的无符号数范围为$0 \sim (2^n-1)$。

当两个无符号数相加时，如果相加的结果超出原定二进制数的位长，比如10001010＋11000001，其结果将用九位表示，即1010001011。这种超出原定数位的现象统称为“溢出”。习惯上就把无符号数加法造成的溢出叫作“进位”。而把“溢出”这个词专门留给有符号数运算时用。

2. 有符号数的表示

所谓有符号数是指需要区分正负的数。我们用加在数字前面的“＋”和“－”符号来表示一个数的正负，在计算机中如用“0”和“1”分别代表这两种符号，这样就可以用计算机来表示有符号数了。也就是说数的符号数码化了，为了区分有符号数的符号与数值部分，需要给符号规定一个固定的位置，即设定一个“符号位”。比如，在一个规定

长度为八位的二进制数中它的最高位为符号位，它的表示形式就成了：（正数）0×××××××；（负数）1×××××××。请注意：这里只是举出一种微机中最常见的表示例子而已，实际上各种不同的系统可以自行规定不同的表示方法。

除了符号表示以外，由于数值部分的不同表示方法，又可以将有符号数分成原码，反码和补码三种不同的表示方式。

（1）原码

这是一种除符号位外利用各二进制位表示一个数的绝对值的表示法。比如，用八位二进制数原码表示+127 和-127 两个有符号数时，最高位用来表示符号，后面的七位用来表示数值。两个数的绝对值都是 127，可以用 1111111 表示。所以这两个数分别可以用“01111111”和“11111111”来表示。可以写成：

$$[+127]_{原} = 01111111$$

$$[-127]_{原} = 11111111$$

把+127 和-127 称为真值(当然也包括+1111111 和-1111111)，而把 01111111 与 11111111 称为机器数。

对于一个用八位二进制数表示的原码，可以用下面的一般形式表示：

$$[X]_{原} = \begin{cases} X & 0 \leq X \leq (2^7 - 1) \\ 2^7 + |X| & -(2^7 - 1) \leq X \leq 0 \end{cases}$$

按这样的表示方法，当真值为 0 时，其符号可以是“0”或“1”中的任意一个，即它既可以是正零，也可以是负零。换句话讲，“00000000”与“10000000”表示相同的零。八位二进制数从全“0”到“1”，全部组合共 256 种，即可以表示 256 个数。但是由于用两个组合表示同一个零，所以实际只能表示 255 个数。它所表示的范围为：$-127 \leqslant X \leqslant +127$。把八位二进制数改为任意位长 n，就可以把原码的定义改写成下面的形式：

$$[X]_{原} = \begin{cases} X & 0 \leq X \leq 2^{n-1} - 1 \\ 2^{n-1} + |X| = 2^{n-1} - X & -(2^{n-1} - 1) \leq X \leq 0 \end{cases}$$

这样我们就把真值和机器码两者联系在一起了。在规定的表示范围内，除零以外，对每一个真值都有一个唯一的机器码与之对应；同样，每一个机器码也都有一个唯一的真值对应。

原码表示法比较简单，也比较直观，但如果用它进行数学运算就很不方便。比如，两个用原码表示的数进行加法运算，计算机首先要判断两个数的符号，若是同号才相加，且和的符号应不变；若是异号则应相减，而且以绝对值大的去减小的，其结果符号与绝对值大的数相同。

例如，有两个数：$X=+3$；$Y=-4$，进行加法运算，先用四位原码表示它们，可写成：

$$[X]_{原} = 0011; [Y]_{原} = 1100$$

如果不顾符号直接相加，结果是 1111，在原码中它的真值为“-7”，这个结果显然是错误的。正确的做法是用绝对值大的 100 去减 011，结果为 001，添上绝对值大的那个数的符号“1”，就成了 1001，其真值就是-1。可见，用原码进行加减运算时，运算规则复杂，运算时间长。因为计算机大量的数据处理工作是加减运算，采用原码表示是很不方便的。

(2) 补码

补码是一种可以避免原码缺点的机器数表示法。还可以把加法和减法运算统一成只做加法，从而大大地提高了计算机的运行效率。

① 补码的概念和定义

在介绍补码之前，我们先介绍模和同余的概念。然后再导出补码的概念和讨论补码的定义及性质。先让我们以钟表为例说明模和同余的概念。假定时钟现在正好指在四点钟，那末以现在时刻为基点，再过12，24……个小时，它将再次指在四点钟。当然，如果倒退12，24……小时，它也是指在四点钟的。这个现象说明，对钟表来说4点钟与16点钟、28点钟的表针指示是一样的。时钟最多能计量到12点。“模”是指一个计量系统的测量范围，12就是计时系统的模。用数学的方法描述同余的概念如下：

如果有两个整数 a 和 b，当用某一正整数 M 去除，所得余数相等，则称 a 和 b 对模 M 是同余的，或称 a，b 在以 M 为模时是相等的，记作：

$$a = b \pmod{M}$$

例如，$a=15$，$b=25$，若模 $M=10$，则 a，b 除以10余数均为5。所以15和25在以10为模时是同余的，即：

$$15 = 25 \pmod{10}$$

由同余的概念不难得出：

$$M + a = a \pmod{M}$$

因此，当 a 为负数时，如 $a=-4$，在以10为模时，有：

$$10 + (-4) = -4 \pmod{10}$$

$$6 = -4 \pmod{10}$$

这样，在以10为模时，负数（−4）就可以转化为正数（+6）了。我们说，当以10为模时，（−4）的补码为（+6），同理，（−2）的补码为（+8）。

例：求−5、+5相对于模式12的补码。

解：（−5）的补码为（−5）+12=7，即，−5=7 (mod 12)

（+5）的补码为（+5）+12=5 (mod 12)

由上述可知，正数的补码是正数本身；求负数的补码只要用模加上该负数就可以得到。即：

$$[X]_{补} = \begin{cases} X & 0 \le X < M/2 \\ M - |X| = M + X & -M/2 \le X < 0 \end{cases}$$

这个定义把一个处在从$-M/2$到$+M/2$之间的有符号数与一个在0～M之间的机器数 $[X]_{补}$ 对应了起来。在补码表示中，由某个 X 的真值，可以得到该值唯一的一个补码机器数；同样地也可以从确定的补码机器数求出它的真值。

在计算机中，如果每个存数单元的长度为二进制 n 位，则对于有符号纯小数来说，模是2；对于有符号整数来说，则模是 2^n。

当 X 是定点小数时，二进制数补码定义如下：

$$[X]_{补} = \begin{cases} X & 0 \le X < 1 \\ 2 + X & -1 \le X \le 0 \end{cases} \pmod{2}$$

当 X 是定点整数时，二进制数补码定义如下：

$$[X]_{补}=\begin{cases}X & 0\leq X<2^{n-1}\\ 2^n+X & -2^{n-1}\leq X\leq 0\end{cases}\quad(\text{mod } 2^n)$$

为了加深对补码概念的理解，让我们再以时钟对时为例进行说明。如果时钟快了三个小时，当前正指在6点钟。为了校正它，你可以倒拨3小时（数学上可以用－3表示）。但是你也可以顺时针向前拨9小时，其效果是一样的（数学上用9替代）。在时钟这个例子中，它的模为12。显然倒拨的－3可以用12＋（－3）＝9这个补码来表示。

从概念和例子中都可以看出，求一个数补码的关键是确定它的“模”。

② 补码的求法

从定义出发，可以方便地求出一个有符号数的补码表示。

若$X>0$，补码表示法与原码一样，$[X]_{补}=[X]_{原}$。

若$X<0$，如果已知它的原码，可以将原码除符号位以外的各位取反，然后末位加“1”，并保持符号不变。如果已知的是X真值的绝对值，则在将绝对值用$n-1$位二进制数表示后，对各位取反，末位加“1”，符号位等于“1”。

仍以真值为（－127）的表示为例，前面已经知道它的原码为“11111111”，它的补码应先把后七位取反，得“0000000”，然后再加“1”，得到“0000001”，添上符号位后就得到：$[-127]_{补}=10000001$。

用第二种方法，先写出（－127）的绝对值的七位二进制编码为：“1111111”。全部取反后，成为：“0000000”，再在末位加上“1”，再加上最前面的符号位“1”，最后也得到：“10000001”的结果。

利用上面所讲的概念和方法，可以看出，补码并不是一种非常直观的数的表示方法，即不能由真值马上就得到相应的机器码，在真值与机器码之间总是要通过运算才能建立起唯一的联系。好在这一切都是交给计算机去做的，而计算机在完成这种使各位取反、加“1”之类运算时是非常方便的，从这个意义上讲补码就是为计算机运算而采用的表示法。

为了对原码、补码和无符号数的表示有更清晰的概念，我们把几种数的表示法列出在表1.1中，以便进行比较。表中最左边一列是按二进制绝对数值大小排列的，最右边一列则是对应的十六进制数，它们是完全等同的，中间各列是该二进制数分别在无符号数、原码、补码表示方法下所代表的数值。

表1.1　二进制、原码与补码对照表

二进制数	无符号数	原　码	补　码	十六进制
00000000	0	＋0	＋0	00H
00000001	1	＋1	＋1	01H
00000010	2	＋2	＋2	02H
⋮	⋮	⋮	⋮	⋮
01111110	126	＋126	＋126	7EH
01111111	127	＋127	＋127	7FH
10000000	128	－0	－128	80H
10000001	129	－1	－127	81H
⋮	⋮	⋮	⋮	⋮
11111100	252	－124	－4	FCH
11111101	253	－125	－3	FDH
11111110	254	－126	－2	FEH
11111111	255	－127	－1	FFH

如表 1.1 所列，八位二进制数用作补码表示时，它表示的数值范围为：－128～＋127。同时请读者注意表中二进制补码与十六进制数的对应关系。

③ 补码的运算

计算机中使用补码，并不是因为它可以方便地将一个真值转变为补码表示，而是因为它对于简化计算机的算术运算十分有利。

先让我们看利用补码表示的数进行两数相加的运算。其规则是：将两数的补码包括符号位在内，直接相加，丢掉模值（即符号位向前的进位），所得即为两数和的补码。用简明的式子表示，就是：

$$[X]_{补} + [Y]_{补} = [X + Y]_{补} \qquad (\text{mod } M)$$

下面的例子以八位二进制数进行运算，故 $n=8$，$M=256$。

例 1.1 计算 50＋40＝？

因为 $[50]_{补}=00110010$；$[40]_{补}=00101000$。

$$\begin{array}{rr} & 00110010 \\ +) & 00101000 \\ \hline & 01011010 \end{array}$$

这个结果恰好是＋90 的补码表示。

例 1.2 计算 50＋（－40）＝？

因为 $[50]_{补}=00110010$；$[-40]_{补}=11011000$。

$$\begin{array}{rr} & 00110010 \\ +) & 11011000 \\ \hline \text{丢弃进位}\cdots\cdots & \boxed{1}\ 00001010 \end{array}$$

所得结果是＋10 的补码。

有了上面的两个例子，我们当然会想到如果是要进行两数相减的运算，也可以使运算变成加上减数的负值的运算，其方法与第二个例子相同，这里不再重复。换句话说，由于运用补码表示有符号数，使计算机可以用加法代替减法，简化了机器的设计。

④ 溢出

利用补码进行运算中，有时也会出现错误，请看下面的例子：

例 1.3 计算 68＋75＝？

$$\begin{array}{rr} & 01000100 \\ +) & 01001011 \\ \hline & 10001111\cdots\cdots -113 \text{ 的补码} \end{array}$$

两个正数相加，结果却成了负数，显然这个运算出现了错误。出现这种错误的原因是运算的结果超出了八位补码的表示范围，例中就是正数超过了 127。实际上当负数超过－128 时也会出现类似的错误。我们把这种错误称为“溢出”，这当然是计算机运算中所不希望的。事实是，计算机在连续高速运转中，许多数据是难以预料的，所以只能在计算机中设定自动检测机构，对加法运算进行监测，一旦发生溢出就用程序的方法或者通知操作者给出处理。那末如何进行检测呢？还是让我们先看下面的几个例子：

为了讨论方便，先把八位二进制数的各位从左到右分别标作 D_7，D_6，…，D_1 和 D_0。把运算中从第六位向最高位的进位标作 C_S，而把最高位（即第七位）向上的进位记作 C_P。

50+40=?	68+75=?	(−50)+(−40)=?	(−68)+(−75)=?
00110010	01000100	11001110	10111100
+) 00101000	+) 01001011	+) 11011000	+) 10110101
01011010	10001111	110100110	101110001
正确	出错	正确	出错
D_6，D_7 无进位；$C_S=C_P=0$；	D_6 进位 D_7 无进位；$C_S=1$，$C_P=0$；	D_6，D_7 均进位；$C_S=C_P=1$；	D_6 无进位，D_7 进位 $C_S=0$，$C_P=1$

从四个例子中可以看出，凡是无溢出的运算，它的 C_S 必然与 C_P 相等，而不管它们是“0”还是“1”；凡是有溢出时，C_S 与 C_P 必不相等，也不管它们谁是“1”，谁是“0”。在计算机中就是将这两个进位标记作异或运算，即：$C_S \oplus C_P$。根据异或运算的规则，只有当两个数不等时才使结果为“1”，这个“1”就作为溢出的标志。

(3) 反码

反码又被称为 1 的补码。仍用“0”作正数的符号位，“1”为负数的符号位，正数的数值就是它本身的编码，而负数则将它的绝对值编码后各位全部取反。

例如：$[+50]_{反}=00110010$，与原码和补码一样。而 $[-50]=11001101$，则正好与正数各位相反。

3. 十进制数的表示

计算机在与人交互时，为了便于使用，还是经常要采用十进制数表示法。但是由于在机器内部只有二进制数的“0”和“1”，于是人们专门为之规定了一组用二进制数表示的十进制数，称为二—十进制数，简称为 BCD 码。

在这种编码中规定用四位二进制编码代表一位十进制数，当有多位数时，按十进制数的次序将各位的 BCD 码排列起来。按照这样的规定，一个八位长的二进制码可以表示两位十进制数。比如十进制数 39 的 BCD 码就是 0011 (3) 1001 (9)，反过来，BCD 码 10000100 表示的是十进制数 84。BCD 码与十进制数之间的关系如表 1.2 中列出的一样。可以看出二进制编码的最大值是 1001，因为这是十进制数位的最大值。

表 1.2　十以内数的二进制和十进制表示

十进制	0	1	2	3	4	5	6	7	8	9
二进制	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001

既然计算机以二进制为基础设计的，那末它在运算时所遵循的当然是二进制数的运算规律，可以想到用这样的规律去运算 BCD 码必然会带来一些问题。比如下面的两组 BCD 码相加的结果都出现了错误：

01001000+00111001=10000001；48+39 成了 81，显然答案错了。

00110110+01011000=10001110；低四位在 BCD 码中为非法表示。

在第一个例子中，正确的答案应是 87，两者相差 6。考察加法的过程，当相加时 D_3

向 D_4 有一个进位（注意：最低位称为 D_0），由于它正处在八位的中间，我们专门叫它作半进位。按照权的概念，这个半进位相当进了 $2^4=16$，也就是说比正常十进制运算的进位多进了 6，于是造成了在个位上少 6 这种现象。解决的办法是：当有半进位时，在低位的 BCD 码上再加上一个 0110（6）进行调整。在上例中用 10000001＋00000110 进行调整，结果得到 10000111 这个正确的结果。

第二个例子中之所以出现非法表示的 1110，是因为在二进制运算中这个数还不足以引起进位，这同时也造成高位 BCD 码因没得到进位而出错。解决的办法是：当低位 BCD 码大于 9 时，给它加上 0110（6）加以调整。在例子中 10001110＋00000110＝10010100，调整后得到 94 这个正确的结果。

4. 小数的表示

前面所介绍的是各种整数在计算机中的表示法，由于它们没有小数部分，所以不需要考虑小数点的位置。但是在实际使用中，小数是一个经常会碰到的运算对象，所以必须有一个可行的表示方法。计算机中的小数点有固定位置和不固定位置两种表示方法，前者称为定点数，而后者则称为浮点数。

（1）定点数

所谓定点数，是指把小数点固定在某一位置上不变的数表示法。计算机中存储数的是一些寄存器或存储器，它们通常都以八个二进制位联合在一起作为一个存储单元。当它表示有符号数时，可以把小数点看作在数值位的最高位之前符号位之后，这样整个数就是一个小数，即小于 1。反之若把小数点看作放在最低位之后，这个数就是整数。这两种情况表示在图 1.1 中。

在计算机内部存储数据的每一位只能存放“1”或“0”，并没有一个物理的装置来表示小数点。所以像图上所示的小数点表示，仅仅是一种约定的方法。

（2）浮点数

小数点位置可以根据表示的需要而改变是浮点数的最主要特点。比如，$N=0.00101$ 是一个小数。它也可以表示成：$N=0.101\times2^{-2}$。这种表示方法可以一般化为：

$$N=S\times2^{p}$$

式中的 S 称为尾数，它表示这个数的有效数字部分，一般用定点小数表示；p 被称为阶码，它表示所表示的数中小数点的位置，是有符号整数；式中，2 是底数，在机器中是约定的。图 1.2 中画出了一个用以存储浮点数的单元中各部分的安排。

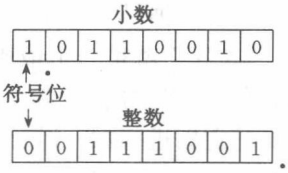

图 1.1 定点数表示法

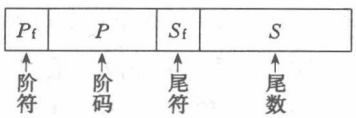

图 1.2 小数的浮点表示法

按这种结构方式用一个八位的存储单元如何存 $N=0.101\times2^{-2}$ 呢？用最高的三位表示它的阶码，其中最高位作为阶符，它应为“1”，而剩余的两位用 10 表示阶码的值。低的五位用作尾数，其中也用一位表示数的符号，这里应为“0”，数值部分用四位表示。于

是我们得到这个浮点数的八位是：11001010。

注意：这里的阶码和尾数都是有符号数，所以也存在着用什么形式表示的问题。为了便于以后计算，建议采用补码表示。

当用浮点数表示小数时，通常都要将尾数的有效位充分地表示出来，以获得最高的表示精度，这个过程称为规格化。比如有一个浮点数为：11000101，在尾数的四个有效位里最高位无效，现将这四位左移一位，成为1010，使 最高位为“1”。为了使整个值不变，在尾数扩大了两倍的情况下，应给阶码减“1”，使之缩小两倍。规格化的情况如图1.3所示。

图1.3　浮点数规格化示意图

对于原码尾数，无论是正数还是小数，尾数的最高数位是“1”时是规格化形式。如果尾数是补码，当 N 是正数时，尾数的最高数位是“1”，N 是负数时，尾数的最高数位是“0”才是规格化形式。可以发现，尾数是补码时的规格化形式的特点是：尾数最高位与符号位相反，这也是尾数用补码表示时判断浮点数是否为规格化数的标志。

浮点运算过程中，运算结果出现非规格化形式，通过移动尾数，将其转换为规格化的形式，与此同时，阶码相应有所加或减，保证 N 的值不变。

最后还应该提及的是关于浮点数的运算问题。正如我们知道的，当进行两个数相加时，总是先要把位权相同的各位对齐，然后才能实施加法。这种方法的另一种说法就是把两个数的小数点对齐。在用浮点表示的小数中所谓对齐小数点，就是使两个数的阶码相同。其方法是使阶码较小的数向较大的那个数看齐，每当阶码增加1，尾数部分相应向右移一位，直到两者的阶码相等为止。运算时只作尾数相加，阶码不能相加，最后再将结果规格化。

计算机中采用浮点数表示小数，使在同样位数的情况下扩大了数的表示范围，也相应地提高了数的表示精度。关于浮点数的深入讨论超出了本书的范围，这里不再赘述。

1.1.2　计算机中非数值数据的表示

上一小节讨论了计算机中数值数据，即一种具有数值大小的数据是如何在计算机中表示的。实际上，数值计算只是现代计算机应用工作中并不很大的一部分，从某种意义上说，计算机之所以能如此迅猛地发展和广泛的普及，其原因就在于除了计算外它还具有很强的非数值数据的处理能力。从概念上讲，所谓非数值数据是一类不具有“值”大小的数据。比如，计算机作文字处理时的处理对象——“文字”就是一个最典型的例子。“大”和“小”这两个字并没有值的大小，你也不可能想象把它们加在一起。所以在这类对象中，各处理对象之间没有值的大小之分，也没必要对它们进行加、减、乘、除等的算术运算。既然要处理它们，首先要在计算机中表示它们，下面首先介绍字符编码。

1. 字符编码

因为计算机中只能存储二进制码，所以非数值的字符也只能用二进制编码形式表示，才能被计算机所接受。ASCII码是目前在计算机中用得最普遍的一种字符编码。它用七位

二进制码进行编码，用以代表各种不同的可见和不可见字符，包括大小写字母、数字符号、标点符号、特殊符号以及一些控制符等。ASCII 码是 American Standard Code for Information Interchange（美国信息交换标准代码）的缩写。它的编码情况如表 1.3 所示。

表 1.3 ASCII 码表

低 四 位		高 三 位							
16 进制	二进制	000	001	010	011	100	101	110	111
0	0000	NUL	DLE	SP	0	@	P	′	p
1	0001	SOH	DC1	!	1	A	Q	a	q
2	0010	STX	DC2	″	2	B	R	b	r
3	0011	ETX	DC3	#	3	C	S	c	s
4	0100	EOT	DC4	$	4	D	T	d	t
5	0101	ENQ	NAK	%	5	E	U	e	u
6	0110	ACK	SYN	&	6	F	V	f	v
7	0111	BEL	ETB	‘	7	G	W	g	w
8	1000	BS	CAN	(	8	H	X	h	x
9	1001	HT	EM	)	9	I	Y	I	y
A	1010	LF	SUB	*	:	J	Z	j	z
B	1011	VT	ESC	+	;	K	[	k	{
C	1100	FF	FS	,	<	L	\	l	|
D	1101	CR	GS	–	=	M	]	m	}
E	1110	SO	RS	.	>	N	^	n	~
F	1111	SI	US	/	?	O	—	o	DEL

注：NUL NULl　　DLE Data Link Escape
SOH Start Of Heading　　DC1 Device Control 1
STX Start of TeXt　　DC2 Device Control 2
ETX End of TeXt　　DC3 Device Control 3
EOT End Of Transmit　　DC4 Device Control 4
ENQ ENQuiry　　NAK Neg. AcKnowledge
ACK ACKnowledge　　SYN SYNchronous idle
BEL Audible BELl　　ETB End Trans. Block
BS BackSpace　　CAN CANcel
HT Horizontal　　EM End of Medium
LF Line Feed　　SUB SUBstitution
VT Vertical Tab　　ESC ESCape
FF Form Feed　　FS Figures Shift
CR Carriage Return　　GS Group Separator
SO Shift Out　　RS Record Separator
SI Shift In　　US Unit Separator

ASCII 码表中只用了七位二进制进行编码，而不是平时所熟悉的八个二进制位组成的一个存储单元。表中编码的高三位在“010”以上的各列分别表示了大小写字母、数字以及标点符号等一些可显示的符号。表中高三位为“000”和“001”的两列为控制代码，这些代码是一些不可显示的代码，它们主要用在通信过程的控制和信息交换的起止中。表