第3章

信息的表示与存储

计算机科学研究内容主要包括信息的采集、存储、处理和传输，而这些都与信息的量化与表示密切相关。本章将从信息的定义出发，对信息的衡量方法、表示方法、存储方法和压缩方法进行讨论。

3.1 信息和信息的表示

关于信息的定义，有多个角度。

一般意义上的信息是指现实世界中事物的存在方式或运动状态的反映，具有可感知、可存储、可加工、可传递和可再生等自然属性。

目前认为对信息的比较科学的定义是从概率统计的观点给出的，即信息是“对事物运动状态或存在方式的不确定性”的描述。

在信息通信的理论中，信息被认为是用以消除通信双方知识上的“不确定性”的东西。

【课堂提问 3-1】 请大家说说下列哪些是信息？哪些不是？它们的不同是什么？

①图书馆，②图书馆有很多书，③火车，④现在的火车速度很快了。

3.1.1 计算机中的信息和信息的表示形式

【课堂提问 3-2】 大家先想一下，生活中，有哪些表示和传递信息的方法？

1. 计算机中的信息

计算机科学中的信息通常被认为是能够用计算机处理的任何有意义的内容或消息，它们以数据的形式出现，如数字、字符、文本、图像、视频和声音等。数据是信息的载体。

2. 信息的表示

在计算机科学中，信息的表示形式有信号表示、符号表示和机器表示等。

(1) 符号表示

符号表示是对信息的抽象描述，符号主要是文字符号，如将温度用数字表示，数字越高表示越温暖，数字越低表示越凉爽或寒冷，符号是信息在数学层面的表达。符号表示的特点是“抽象”、“一般”，可以根据具体对象赋予符号不同的内容。例如，假如有个小教室

有 4 行 8 列的座位(真是小了点),如果座位上有人,就用 1 表示,如果没人就用 0 表示,则下列形式的数据就很容易地表示这个教室的空闲情况。

1011 1100
0101 1110
0110 0110
0011 1000

(2) 机器表示

为了可以使用计算机来存储和处理信息,必须将符号表示的信息变成一种计算机能够“理解”的“数据”,这就是**信息的机器表示**。实际上,信息在计算机中是以电子器件的稳定物理状态来表示的。很多常用物理器件具有两种稳定的状态,如磁铁的 N 极、S 极,电路的有电、无电,高电平和低电平,开关的断开和闭合、晶体管的导通和截止、电容有电荷和无电荷、光线的强弱等。两种不同的状态可以分别用来表示符号 0 和 1,多个不同状态的组合就可以表示不同的信息。例如,要表示常用的交通工具,可以用 00 表示自行车,01 表示汽车,10 表示飞机,11 表示轮船。用物理器件的两个状态表示 0 和 1,就把数据记录了下来。

(3) 信号表示

信息需要传递(或传输)。生活中的信息传递方式(如喊话、灯光、鸣笛、旗语等),其传输距离和速度都是非常有限的,邮政的传输距离可以很远但速度却很慢,比较快捷的传输方式是使用电信号。无论是计算机内的各个部件(如键盘、鼠标、硬盘等),还是计算机所控制的对象(如电冰箱中的温度传感器、压缩机启停控制电路等),硬件电路之间传输的信息都以**电信号**的形式进行表示。

3.1.2　数的表示

我们看到一个数如 313,自然认为是三百一十三。这种个位上的 1 代表 1,十位上的 1 代表十,百位上的 1 代表一百的计数方法称为**十进制记数法**。实际上,十位上的 1 也可以不代表十,这就是不同的记数方法。计算机中常用的记数方法包括二进制、八进制、十六进制和十进制。为了不同的目的,计算机中还经常需要将一个数在不同的数制间进行转换。

1. 进位记数制

十进制数使用 0～9 共 10 个符号,数量每增加 1,个位数字就加 1,当加到十的时候,个位数字写为 0,十位数字加 1,称为**进位**。这种使用有限数量的符号,采用进位方法记数的格式就是**进位记数制**。日常生活中采用的是**“逢十进一”**的记数规则,就是**十进制**。

(1) 进位记数制的一般形式

进位记数制一般数的表示形式是

$$d_n d_{n-1} \ldots d_3 d_2 d_1 d_0 . d_{-1} d_{-2} d_{-3} \ldots d_m$$

它表示的数对应的十进制数是(注意,d_0后面有一个小数点)

$$\sum_{i=0}^{n} d_i N^i + \sum_{j=-1}^{m} d_j N^j$$

一种进位记数制由一组符号和三个**要素**组成。

- **符号**：组成数的数字，如 0,1,2,3,4,5,6,7,8,9 等。
- **基数**：或基，就是上面式子中的 N，也是所用符号的个数。
- **数位**：数字所处的位置，如上面表示形式中的下标。
- **权**：不同位置的数字 1 代表的数量，是基数的幂 N^i。

N 不同时，就是不同的进位记数制表示法，称为 ***N* 进制**。计算机科学中常用数制是二进制、八进制和十六进制。

(2) 十进制

十进制数采用的符号是 0,1,2,3,4,5,6,7,8,9 这 10 个数字，基数是 10，从小数点向左的数位依次叫个位、十位、百位、千位、万位等，也可以按下标叫第 0 位，第 1 位，第 2 位等。从小数点向右的数位依次叫十分位、百分位、千分位等。对整数，左边的称**高位**，右边的称**低位**。十进制的位权是 10^i，$i=n,n-1,\cdots,2,1,0,-1,-2,\cdots,m$。

(3) 二进制

二进制采用的符号是 0 和 1，基数是 2，数位的称呼可以采用十进制的称呼，但要注意它的权是不同的。避免混淆的称呼是第 0 位，第 1 位等。二进制的位权是 2^i，$i=n,n-1,\cdots,2,1,0,-1,-2,\cdots,m$。例如二进制的 11，对应的十进制数是 $1\times2^1+1\times2^0=2+1=3$。读非十进制的数时，不能读成几百几十几，应只读数字。如二进制的 11，不能读成“十一”，应读成“幺幺”或“一一”。在书写上，为了和十进制区分，其他进制的数采用下标或加后缀方法。二进制下标是 2，后缀是 B，如二进制 11 记为$(11)_2$或 11B。十进制的下标是 10，后缀是 D，常省略。不写后缀和下标的默认是十进制。

(4) 八进制

八进制采用的符号是 0,1,2,3,4,5,6,7，基数是 8，位权是 8^i，$i=n,n-1,\cdots,2,1,0,-1,-2,\cdots,m$。例如八进制的 11，对应的十进制数是 $1\times8^1+1\times8^0=8+1=9$。八进制下标是 8，后缀是 Q 或 O(大写字母 O)，如八进制 11 记为$(11)_8$，11Q。注意，如果大写字母 O 和数字 0 易混时，应写下标 8 或使用字母 Q。

(5) 十六进制

十六进制采用的符号是数字 0～9 以及字母 A～F，字母分别表示数量的 10,11,12,13,14 和 15。字母的大小写均可，但一般一个数中大小写应一致。十六进制数的基数是 16，位权是 16^i，$i=n,n-1,\cdots,2,1,0,-1,-2,\cdots,m$。例如十六进制的 11，对应的十进制数是 $1\times16^1+1\times16^0=16+1=17$，十六进制 AB2 对应的十进制数为 $10\times16^2+11\times16^1+2\times16^0=10\times256+11\times16+2=2738$。十六进制下标是 16，后缀是 H，如十六进制 11 记为$(11)_{16}$，11H。有些程序设计语言中也使用前缀 0X 或 0x，如 0X11 表示 11 是十六进制数。

除二进制、八进制、十六进制外，如果需要，也可以将数表示为三进制、七进制和二十进制等。

某种进制的数，按照各位数字和位权计算出对应的十进制数，称为**按权展开**。

2. 不同数制数的转换

在生活和计算机科学中，为了方便，在不同的场合需要采用不同进制的数，所以，经常需要在不同进制的数之间转换。其他进制转十进制的方法就是按权展开，所以不再介绍。

(1) 十进制转其他进制

一个十进制数转换为 N 进制数，基本的方法是：整数部分，除 N 取余，直到商为 0，再将得到的余数从低位到高位依次排列即得到相应的二进制整数；小数部分，乘 N 取整，直到小数部分为 0，将得到的整数从高位到低位排列，形成小数部分。下面以十进制转换为二进制为例介绍具体的转换算法。

【例 3-1】 将十进制 124.625 转换为二进制数。

解：计算步骤如下：

整数部分的转换过程

除数	被除数	余数
2	124	0
2	62(商，新被除数)	0
2	31	1
2	15	1
2	7	1
2	3	1
2	1	1
	0——商	

小数部分的转换过程

0.625		
× 2		取整
1.250	……	1
× 2		
0.50	……	0
×2		
1.0	……	1

整数部分，最先得到的余数是最低位，最后得到的是最高位，所以整数部分是 111 1100B。习惯上从小数点向两边，每 4 位加空格隔开，这是为易于阅读。

小数部分，最先得到的是高位，所以，小数部分是 0.101B。

两部分合起来就是 124.625D＝111 1100.101B＝$(111\ 1100.101)_2$。

注意，在十进制转换为其他进制时，会出现小数部分无限循环的情况，这时，可以根据需要取若干位小数，比如 4 位或 8 位等。

用计算机求解十进制整数转二进制整数。设要转换的十进制整数为 d。b[1,…,n]为数组，用于存放转换后的二进制数的每一位，算法如下：

```
k=1
若 d=0:                          #d 为 0 时,不需计算
    b[k]=0
否则:                            #d 不为 0 时,需要计算
    当 d//2>0:                   #商不为 0 时转换,继续转换。//表示 d 除以 2 的商,也称整除
        b[k]=d%2                 #求余,存入 b[k]
        d=d//2                   #求商
        k=k+1                    #位数加 1
    b[k]=d                       #最后的余数存入 b[k]
循环,i 从 k,到 1:
```

```
    显示 b[i]
```

其中，%号表示求余，d//2 表示 d 除以 2 的商，即整数部分。不同的计算机语言中，取整的运算是不同的。

如果要将十进制小数转换为二进制，设要转换的十进制小数为 f，h[1，…，n]为数组，用于存放转换后的二进制数的每一位，则算法如下：

```
k=1，
若 f=0.0：
    h[k]=0
    k=k+1
否则：
    当 f>0 且 k<9：              #k<9 是限制小数位数不超过 8 位，避免无限循环
        h[k]=[f * 2]             #右边的[]表示对其中的运算结果取整
        f=f * 2-h[k]             #乘 2 后减去整数部分
        k=k+1
循环，i 从 1，到 k-1：
    显示 h[i]
```

请读者使用 Python 语言将上面两个算法写成程序。数组在 Python 的对应项是列表。取整的对应项是 int()函数。对于整数，也可以用//符号。

十进制转八进制、十六进制以及其他进制，都使用相同的方法，只是基和使用的数字符号不同，特别是十六进制，当余数为 10，11，…，15 时，显示的符号应是 A，B，…，F。

（2）二进制和八进制的转换

二进制数转换为八进制数，从小数点向两边，每 3 位隔开，不够 3 位补 0，将每 3 位的二进制按权展开，再写成一位的八进制数，就得到与二进制数在数量上相等的八进制数。

【例 3-2】 将二进制数 111 1100.1011B 转换为八进制数。

解：转换过程如下：

将二进制数每 3 位隔开，不够 3 位补 0，得到 001 111 100.101 100B。

每 3 位二进制按权展开：1 7 4 .5 4

八进制数：174.54Q＝$(174.54)_8$

即：111 1100.1011B＝174.54Q＝174.54O＝$(174.54)_8$

特别注意，小数点后，不够 3 位的一定补 0，否则转换容易出错。

八进制到二进制的转换，将每一位的八进制，转换为 3 位二进制。

【例 3-3】 将$(1567.123)_8$转换为二进制。

解：将每一位的八进制，转换为 3 位二进制。

$(1567.123)_8$＝001 101 110 111.001 010 011B＝11 0111 0111.0010 1001 1B

（3）二进制和十六进制的转换

二进制数转换为十六进制数，从小数点向两边，每 4 位隔开，不够 4 位补 0，将每 4 位的二进制按权展开，再写成一位的十六进制数，就得到与二进制数在数量上相等的十六进制数。

【例 3-4】 将二进制数 111 1100.10111B 转换为十六进制数。

解：转换过程如下：

将二进制数每 4 位隔开，不够 4 位补 0，得到 0111 1100.1011 1000 B

每 4 位二进制按权展开：7 12 .11 8

十六进制数：7C.B8H＝$(7C.B8)_{16}$

即：111 1100.10111B ＝7C.B8H＝$(7C.B8)_{16}$

十六进制到二进制的转换，将每一位的十六进制数，转换为 4 位二进制。

【例 3-5】 将$(1DB1.A11)_{16}$转换为二进制。

解：将每一位的十六进制，转换为 4 位二进制。

$(1DB1.A11)_{16}$＝ 0001 1101 1011 0001.1010 0001 0001B

【课堂提问 3-3】 请总结十进制、二进制和十六进制数字符号的对应关系，填写表 3-1。

表 3-1　十进制、二进制、八进制和十六进制基本数字的对应关系

十进制	二进制	八进制	十六进制	十进制	二进制	八进制	十六进制
0				8			
1				9			
2				10			
3				11			
4				12			
5				13			
6				14			
7				15			

3. 数的四则运算

四则运算是数的基本运算，对于非十进制数，加、减、乘、除的计算方法和十进制是相同的，只是，加法逢 N 进 1；减法借 1 当 N；对于乘法，本位是积$\%N$，进位是积$//N$。这里不再举例。

4. 整数在计算机中的表示

数，在数学上是无限的，可以无限大，可以无限小，位数也可以无限长。而实际表示时，就要受到存储装置，显示装置的限制，比如，算盘能表示的数就是有限位的，在纸上也不可能写出无限长的数。

(1) 无符号数

不考虑数的符号，或者说只考虑正数，就是无符号数。如果存储装置只能存储最长 8 位的二进制数，那么能表示的最小整数就是 0000 0000B，十进制就是 0；能表示的最大

整数就是1111 1111B，按权展开，十进制就是255，也就是1 0000 0000B－1B＝2^8-1＝255。所以，一个k位的无符号数，能表示的数的范围是0～2^k-1。要表示更大的数，就需要更多的二进制位，16位的二进制数，能表示的无符号数的范围是0～65 535。32位二进制能表示的最大无符号数是4 294 967 295，这也是许多程序设计语言中表示整数采用的位数和范围。

(2) 有符号数的原码表示

数学上，用“＋”表示正号，“－”表示负号，一个数前面加上正号或负号，就是正数或负数。正号通常可以省略。

考虑正负的数，就是**有符号数**或**带符号数**。一个二进制数，前面加上符号，称为**二进制真值**，简称**真值**，如＋1101 0111B，－1000 1010B等。也可以写十进制的真值，如＋215，－138等。

计算机中，所有的数据都需要用物理器件的状态表示，也就是0或1，或它们的序列。正号和负号也要用0或1表示。一般用有限位二进制数的最高位表示符号，0表示正，1表示负，其他数位表示数的绝对值，这样的二进制数称为**原码**。用数码0和1表示符号的二进制数称为**机器数**。如果是8位二进制有符号数，1101 0111B对应的十进制真值就是－87，0110 1010B的十进制真值就是＋106。反过来，1101 0111B是－87的原码，0110 1010B是＋106的原码，记为

$$[-87]_{原}=1101\ 0111B$$
$$[+106]_{原}=0110\ 1010B$$

由于有一位要用来表示符号，所以，8位原码表示的绝对值最大的正数是0111 1111，即十进制＋127。绝对值最大的负数是1111 1111，十进制真值是－127。如果是16位，能表示的数的范围是－32767～＋32767。如果是n位，能表示的数的范围是$-(2^{n-1}-1)$～$+(2^{n-1}-1)$。

设X是一个整数，用n位二进制表示，X的原码定义为

$$[X]_{原}=\begin{cases}X, & 0\leqslant X<2^{n-1}\\ 2^{n-1}-X=2^{n-1}+|X|, & -2^{n-1}<X\leqslant 0\end{cases}$$

方括号中的X常写为十进制真值，等号右边的按二进制无符号数写。例如，用8位表示：

$$[+58]_{原}=(0011\ 1010)_2$$
$$[-121]_{原}=2^{8-1}-(-121)=128+121=249=1111\ 1001B$$

0的原码有两种形式：

$$[+0]_{原}=0000\ 0000\ B$$
$$[-0]_{原}=1000\ 0000\ B$$

分别称为正0和负0。

原码表示直观易懂，机器数、真值转换容易，实现乘除运算简单，但实现加减运算不方便。例如，－1＋1，按原码计算

$$\begin{array}{rl}1000\ 0001 & ——-1\\ +0000\ 0001 & ——\ \ 1\\ \hline 1000\ 0010 & ——-2\end{array}$$

对应位相加，得到的是－2 而不是 0。当然，通过判断符号，特殊处理也可以得到正确结果，但通用性差。

(3) 有符号数的补码表示

在校对指针式的钟表时，如果当前钟表的指示时间是 3 点(见图 3-1)，而实际时间是 1 点，对表的方法一是将时针向后拨 2 格，即 3 减 2 变成 1；另一种方法是将时针向前拨 10 格，即 3 加 10 变成 1。相当于在能表示 12 个小时的表盘上，3 减 2 等于 1，3 加 10 也“等于”1。钟表对时，**减法可以转换为加法**。

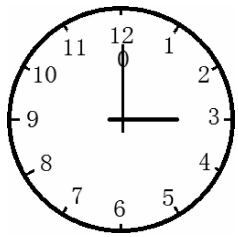

图 3-1　对时方法

为什么 3＋10 也等于 1 呢？根本原因是在表盘上，当数大于 12 时，自动丢掉了 12。

计算机中，存储一个数总是使用有限的数位，那么它能表示的数的范围就是有限的，当超出最大范围时，就称为**溢出**，最小的溢出量称为**模**或**模数**。例如，8 位的二进制数，最大数是 1111 1111B，表示 255，如果再加 1，就变成 0000 0000B，就是 0 了，多出的进位无法表示，被舍掉。溢出的数量是 255＋1＝256，所以 256 就是 8 位二进制数的模数。若机器能表示的数是二进制的 n 位数，则能表示的数是 $0\sim2^n-1$，它的模是 2^n。

算术运算中，自动舍弃溢出量的运算称为**“模运算”**。在模运算中，若 A,B,M 满足：

$$A = B + kM(k \text{ 为整数})$$

则称 A,B 在模 M 下**同余**，即 A 和 B 除以 M 得到的余数是相同的，记为

$$A \equiv B(\bmod M)$$

有时也写为

$$A = B(\bmod M)$$

例如，$13=1+1\times12$，则 $1\equiv13(\bmod 12)$；$-2=10-1\times12$，则 $-2\equiv10(\bmod 12)$。即在模 12 下，1 和 13 同余，－2 和 10 同余。

同余具有如下性质：

① 反身性：$a\equiv a(\bmod M)$。

② 对称性：若 $a\equiv b(\bmod M)$，则 $b\equiv a(\bmod M)$。

③ 传递性：若 $a\equiv b(\bmod M)$，$b\equiv c(\bmod M)$，则 $a\equiv c(\bmod M)$。

④ 同余式相加：若 $a\equiv b(\bmod M)$，$c\equiv d(\bmod M)$，则 $a\pm c\equiv b\pm d(\bmod M)$。

⑤ 同余式相乘：若 $a\equiv b(\bmod M)$，$c\equiv d(\bmod M)$，则 $ac\equiv bd(\bmod M)$。

在模 12 下，－2 和 10 同余。由同余性质④，$a-2\equiv a+10(\bmod 12)$，即一个数减 2 和加 10 在模 12 意义下，得到的值是相同的。这样就将减法转换为了加法。10 称为－2 在模 12 下的**补数**或**补码**。这就是为什么对表时，向后拨 2 小时和向前拨 10 小时得到的结果是一样的。

设 $M>x>0$，则：

$$[-x]_{补} = M - x$$

计算机科学中定义，在 n 位二进制能表示的数的范围内，正数的补码还是这个数，负数的补码是模数加这个数或模数减这个数的绝对值，表示为

$$[X]_{补}=\begin{cases}X & 0\leqslant X<2^{n-1}\\ 2^n+X=2^n-|X| & -2^{n-1}\leqslant X<0\end{cases}$$

例如，$n=8$，则 $2^n=256$。

$[1]_{补}=1=0000\ 0001B$

$[2]_{补}=2=0000\ 0010B$

$[127]_{补}=127=0111\ 1111B$

$[-1]_{补}=256-1=255=1111\ 1111B$

$[-2]_{补}=256-2=254=1111\ 1110B$

$[-128]_{补}=256-128=128=1000\ 0000B$

$[-127]_{补}=256-127=129=1000\ 0001B$

$[0]_{补}=0000\ 0000B$

注意，正数的补码最高位为 0；负数的补码最高位为 1。

原码表示中，1000 0000B 是表示－0 的，而补码表示中它表示－128，所以，补码的表示范围是－128～127。如果是 n 位二进制数，补码能表示的数的范围是 $-2^{n-1}\sim+(2^{n-1}-1)$。

一个负数从原码转换为补码，除使用定义外，还可以使用一种简便方法，就是自低位向高位，连续的 0 及第一个 1 不变，再向左，按位取反（即 0 变为 1，1 变为 0），符号位不变。例如，8 位有符号数，－98 的原码是 1110 0010B，则补码是 1001 1110B，注意，末两位没变，向左除符号位外，0 变为 1，1 变为了 0。但这种方法不适合－0 和－128 的转换，请特别注意。

【例 3-6】 若一个 8 位有符号数的补码是 1011 0101B，那么这个数是多少？原码是多少？

解：这个数的最高位为 1，它是一个负数的补码。按权展开：

$$1011\ 0101B=181,\quad X=[X]_{补}-2^n=181-256=-75$$

所以，它应该是－75 的补码，即 $[-75]_{补}=256-75=181=1011\ 0101B$，其原码为

$$[-75]_{原}=1100\ 1011B$$

简单的求原码的方法和求补码的方法相同，就是：从右向左，连续 0 和第一个 1 不变，其他位取反，符号位不变。比较－75 的原码和补码的差别。

（4）有符号数的补码运算

使用补码的定义，可以证明：

$$[X]_{补}\pm[Y]_{补}=[X\pm Y]_{补}(\mathrm{mod}\ 2^n)$$

$$[[X]_{补}]_{补}=X(\mathrm{mod}\ 2^n)$$

这样，就可以将减法和负数的运算全部转换为补码运算，符号位也一同参加运算。例如，－1＋1：－1 的 8 位补码为 1111 1111B，1 的补码为 0000 0001B，则

$$[-1]_{补}+[1]_{补}=0000\ 0000B(\mathrm{mod}\ 256)$$

0000 0000B 就是 0 的补码，所以－1＋1 的结果为 0。注意，本来有一个进位，在（mod 256）下，被舍去。

上例中，最高位有进位，自然舍去得到了正确结果。但情况并不都是这样，例如，

－127－2，－127 的补码是 1000 0001B，－2 的补码是 1111 1110B，1000 0001B＋1111 1110B＝1 0111 1111B。舍去进位 1，0111 1111B 应是 127 的补码，但－127－2 应是－129 而不是 127。

(5) 溢出的判断

前面介绍过溢出，溢出的本质是运算结果超出了机器能表示的数的范围。但进位并不代表真正的溢出。实际上，**判断真溢出的方法**是：①正数和负数相加不会产生溢出；②正数相加，如果符号位变为 1，则为溢出，否则不溢出；③负数相加，如果符号为变为 0，则为溢出，否则不溢出。有进位而不溢出的情况称为**假溢出**。

【例 3-7】 下列式子中的数字是 8 位二进制补码，计算下列式子的值并判断是否溢出。

① 1100 0001＋1100 0001

② 1000 1001＋1100 0001

③ 1100 0001＋0100 1010

④ 0100 0011＋0011 1110

⑤ 0100 1111＋0010 0011

解：

① 1100 0001＋1100 0001＝1 1000 0010，负数相加，最高位有进位，**符号不变，不溢出**。

1100 0001＋1100 0001＝1000 0010 ＝$[-126]_{补}$(mod 256)，所以结果的十进制是－126。

② 1000 1001＋1100 0001＝1 0100 1010，负数相加，最高位有进位，**符号改变，溢出**。

③ 1100 0001＋0100 1010＝1 0000 1011，**负数和正数相加，不会溢出**。

1100 0001＋0100 1010＝0000 1011＝$[11]_{补}$(mod 256)，所以结果的十进制是 11。

④ 0100 0011＋0011 1110＝1000 0001 正数相加，**没进位，但符号位改变，溢出**。

⑤ 0100 1111＋0010 0011＝0111 0010 正数相加，**符号位不变，不溢出**，0111 0010＝$[114]_{补}$，所以结果是 114。

(6) 有符号数的反码表示

为了计算有符号数的补码，常利用一种中间状态——反码。**正数的反码是它本身，负数的反码是除符号位外，其他位按位取反**。

带符号数的反码定义为

$$[X]_{反}=\begin{cases}X & 0\leqslant X<2^{n-1}\\(2^n-1)+X & -2^{n-1}<X\leqslant 0\end{cases}$$

例如：

$[0011\ 1010]_{反}$＝0011 1010(正数的反码)

$[1011\ 1010]_{反}$＝1100 0101

$[1000\ 0000]_{反}$＝1111 1111(－0 的反码)

$[0000\ 0000]_{反}$＝0000 0000(＋0 的反码)

$[1111\ 1111]_{反}$＝1000 0000

n 位二进制数的反码的表示范围是$-(2^{n-1}-1)\sim+(2^{n-1}-1)$。**0 的反码表示不**

唯一。

补码可以通过反码求得，方法是：**正数的补码等于反码；负数的补码等于反码加 1；0 的补码是 0**。例如，求 27、－27 的补码：

$$[+27]_{原}=0001\ 1011$$
$$[+27]_{反}=0001\ 1011$$
$$[+27]_{补}=0001\ 1011$$
$$[-27]_{原}=1001\ 1011$$
$$[-27]_{反}=1110\ 0100$$
$$[-27]_{补}=1110\ 0100+1=1110\ 0101$$

注意，这种方法不适合求 -2^{n-1} 的补码。可以永远记住，－1 的补码是全 1，-2^{n-1} 的补码是 1 后面全 0（n－1 个）。

小数或实数也可以用原码、反码和补码表示，但它们在计算机中实际采用的是另一种表示方法——浮点表示。

5. 数的二进制浮点表示

与浮点对应的概念是定点。所谓**定点表示**就是事先约定小数点的位置，一旦约定，不能改变。前面的整数的补码表示，就是定点表示，默认约定小数点在最右端。如果约定在最前面或中间某位置，就是可以表示小数或实数了。但小数点的固定，限制了数的表示范围和精度。拿十进制来说，例如约定数位为 8 位，小数点后保留两位。像 12.18，1840.28 这样的数容易表示，而 0.0000128，128000000000 这样的数，就无法表示。

科学计算中对于很小或很大的数，常用的一种表示方法是科学记数法，0.0000128 表示为 1.28×10^{-5}，128000000000 表示为 1.28×10^{11}，简洁、明了而不失精度。

计算机中，实数也采用这种方法，只不过针对的是二进制实数。小数点是根据记法的改变，改变了位置，称为**浮点表示**，这样表示的数就是**浮点数**。若约定，整数位保留一位，则 10101110B 和 0.0010111B 可以分别表示为

$$10101110B=1.010111B\times2^{7}$$
$$0.0010111B=1.0111B\times2^{-3}$$

一个二进制实数，总可以表示为

$$\pm1.f\times2^{e}$$

的形式，称为**规格化的形式**（整数位为 1）。要保存这个数，只须保存符号、f 和 e，而确定的整数部分 1 和阶码 e 的底 2 不需要保存。所以，计算机中浮点表示的数具有如下结构：

符号	阶码 E(指数)	尾数(f)F

符号位用 1 位表示，阶码位的多少决定了数的表示范围，尾数位的多少决定了数的精度。阶码和尾数可以用原码或补码表示。

为了数据交换的需要，大家需要遵循统一的表示格式。计算机中有两种浮点表示的约定，称为 IEEE 754 国际标准。一种是使用 4 字节即 32 位表示实数，1 位符号位，8 位阶

码位，23 位尾数位，称为**单精度浮点格式**；另一种使用 8 字节即 64 位表示实数，1 位符号位，11 位阶码位，52 位尾数位，称为**双精度浮点格式**。显然双精度比单精度数的范围要大，精度要高。

单精度格式的符号位用 0 表示正数，1 表示负数。阶码使用移码表示。所谓**移码**，就是在原数之上加上(或减去)一个数，加的这个数叫**偏移量**或**偏置值**。单精度使用的偏移量是 127。阶码(E)＝实际指数(e)＋偏移量(127)。阶码位的取值为 0～255。0 和 255 有特别意义，其他 1～254，减去偏移量，实际的指数范围是－126～127。尾数保存的是小数部分，原码，不含符号位。这样，**单精度规格化形式**能表示的**绝对值最大的数**是

$$1.1111\ 1111\ 1111\ 1111\ 1111\ 111\text{B}\times 2^{127}\approx 3.402\ 823\ 5\times 10^{+38}$$

绝对值最小的数为

$$1.\ 0000\ 0000\ 0000\ 0000\ 0000\ 000\text{B}\times 2^{-126}\approx 1.175\ 494\ 4\times 10^{-38}$$

【例 3-8】 如果一个 32 位的单精度位模式(就是 32 位单精度表示形式)是

1 1000 1100 0011 0000 0000 0000 0000 000

那么它表示的数是多少？

解：最高位是符号位，为 1，说明是负数。

后面 8 位 1000 1100B＝140，减偏移量 127，实际指数是 140－127＝13。尾数部分是 0011 0000 0000 0000 0000 000，那么，它表示的数是

$$-1.0011\times 2^{13}=-1.1875\times 1024\times 8=-9728$$

【例 3-9】 一个数 1.1101 0001B$\times 2^{-3}$的 32 位单精度位模式是什么？

解：这是一个规范化的数，正数，最高位为 0。指数为－3，移码，－3＋127＝124＝0111 1100B，这是指数部分。尾数部分就是小数点后的数，不够的补 0，即

0 0111 1100 1101 0001 0000 0000 0000 000

一个 IEEE 754 位模式，设符号位为 S，阶码位为 E，尾数位为 f。当 E 全为 0，f 全为 0，s 也为 0 时，表示 0.0；当 E 全 1，f 全 0 时，s 为 0 时，表示正无穷；当 E 全 1，f 全 0，s 为 1 时，表示负无穷；当 E 全 1，f 不为 0 时，表示这不是一个数，常表示为 NaN(Not a Number)，例如负数开平方时。

若 E 全为 0，而 f 不为 0，那么单精度模式表示的数是

$$0.\text{f}\times 2^{-126}$$

这样形式的数称为**非规格化的数**。单精度能表示的绝对值**最小的非规格化数**是

$$0.0000\ 0000\ 0000\ 0000\ 0000\ 001\times 2^{-126}=1.0\times 2^{-149}\approx 1.401\ 298\ 5\times 10^{-45}$$

绝对值**最大的非规格化数**为

$$0.1111\ 1111\ 1111\ 1111\ 1111\ 111\times 2^{-126}\approx 1.175\ 494\ 2\times 10^{-38}$$

注意，这两个数都比规格化形式表示的最小数还要小，这就是非规格化数的意义，用来表示绝对值更小的数。

【例 3-10】 若单精度的位模式是 1 0000 0000 0101 1010 0000 0000 0000 000 它表示的数是多少？

解：符号位为 1，说明是负数，指数全 0，尾数不为 0，说明是非规格化数。它表示的数是：$-0.0101\ 1010\text{B}\times 2^{-126}$。

【例 3-11】 若一个数是 $1.01101B\times2^{-130}$，能不能表示为单精度模式？如果能，写出位模式。

解： $1.01101B\times2^{-130}$ 这样书写是规范化的格式，但指数部分小于－126，超出了单精度指数－126～127 的范围，所以按规格化的单精度格式是无法保存的。若写成 $0.000101101\times2^{-126}$，这是一个非规格化的形式，指数和尾数都在单精度能表示的范围内，位模式为

0 0000 0000 0001 0110 1000 0000 0000 000

双精度格式与单精度格式的位模式类似，指数的偏移量是 $2^{k-1}-1=2^{11-1}-1=1023$（其中 k 是指数位数），阶码位的取值 0～2047，0 和 2047 保留，规格化数可用 1～2046，减去偏移量，实际指数范围是－1022～1023。

浮点数解决了大数和小数的表示问题，但要注意，只是部分或者说在一定范围内解决。使用浮点数有两点要清楚：一是浮点数的表示范围仍然是有限的，数量和精度都是有限的；第二是实数的表示常常是不精确的，比如 0.1 的问题，转换为二进制是无限循环小数，但不管是单精度还是双精度的尾数部分都是有限的，不可能无穷，这就使得 0.1 在计算机中实际上不是精确的 0.1。大家运行一下下列 Python 程序，分析结果：

```
a=0.0
for i in range(10):          #这个循环执行几次？
    a=a+0.1;                 #0.1 被加了几次？
print(a==1.0)                #这是逻辑值，结果是 True 还是 Flase 呢？
print(a)                     #a 的值是多少呢？是 1 吗？
```

6. 十进制数的 BCD 码表示

大家注意到，信息的表示可以有各种各样的方法。数的表示其实不一定用数字，更不一定用二进制。例如，比赛的第 1 名、第 2 名、第 3 名，常被称为冠军、亚军和季军。在计算机中，一个十进制数除了可以转换成二进制表示外，还可以将十进制数的每一位数码符号都用二进制编码来表示。这种十进制数的表示方法称为“二进制编码的十进制数”，简称 **BCD 码**（binary coded decimal）。所谓**编码**，可以理解为“用代号表示”。

十进制使用 10 个数字符号 0～9。要给这 10 个符号各自一个代号，需要 4 个二进制位。但 4 个二进制位可以给出的编码个数是 16 个，超出了需求。所以可以按一定的规则，取 10 个来与 0～9 对应就形成了不同的 BCD 码。表 3-2 是不同编码方案使用的 4 位编码。

表 3-2 常用的 BCD 码

十进制数字	8421 码	5421 码	2421 码	余 3 码	BCD Gray 码
0	0000	0000	0000	0011	0000
1	0001	0001	0001	0100	0001

续表

十进制数字	8421 码	5421 码	2421 码	余 3 码	BCD Gray 码
2	0010	0010	0010	0101	0011
3	0011	0011	0011	0110	0010
4	0100	0100	0100	0111	0110
5	0101	1000	1011	1000	0111
6	0110	1001	1100	1001	0101
7	0111	1010	1101	1010	0100
8	1000	1011	1110	1011	1100
9	1001	1100	1111	1100	1000

(1) 8421 码

4 位二进制编码从高位到低位的位权依次设为 8、4、2、1，按权展开后与十进制数码对应，就是 8421 码，这样的编码方法称为**有权 BCD 码**。例如，8421 码表示的 1001，按 8421 权展开是 1×8＋0×4＋0×2＋1×1＝9，所以 1001 就是数字 9 的编码。数字 6 分解为权的和是 4＋2，它的 BCD 码 0110。一个十进制数的 8421 BCD 编码就是将十进制数的每一位直接写成对应的 8421 编码。例如，十进制的 1896 的 8421 BCD 码表示就是 0001 1000 1001 0110。反过来，若一个十进制数的 8421 BCD 编码为 0111 0100 0101，每 4 位写成一个十进制数字，它代表的是 745。

(2) 5421 和 2421 码

5421 码和 2421 码也是有权 BCD 码，它们从高位到低位的权值分别为 5、4、2、1 和 2、4、2、1。比如 2421 码，1101 按权展开是 2＋4＋0＋1＝7，它是 7 的编码。8 的 2421 码是 1110，前 3 位的位权分别为 2、4、2，加起来是 8。十进制 1896 的 2421BCD 码是 0001 1110 1111 1100。

注意，由于按权 2421，1010 展开是 4，0100 展开也是 4，就是说十进制数字 4 可以有两种编码，同样 2 也有两种编码，所以，这种 BCD 码的编码方案是不唯一的，使用时要先做约定。表 3-2 中的 2421 编码用 0100 表示 4。

(3) 余 3 码

余 3 码是 8421 BCD 码的每个码组加 3(0011B)形成的，例如，7 的 8421 码是 0111，加 0011，0111＋0011＝1010，是 7 的余三码编码。余三码常用于 BCD 码的运算电路中。

(4) Gray 码

Gray 码(格雷码)因 1953 年公开的弗兰克·格雷(Frank Gray，1887－1969，美国发明家)专利 pulse code communication(脉冲编码通信)而得名，其最基本的特性是任何相邻的两组编码中，仅有一位不同，因此又称为单位距离码。Gray 码的单位距离特性有很重要的意义。例如，两个相邻的十进制数 7 和 8，其对应的二进制数为 0111 和 1000。在用二进制数进行加 1 计数时，如果从 7 变为 8，二进制数的 4 位都要改变，但在实际物理实现时，4 位改变不可能完全同时发生，若从低位到高位逐位变化(串行计数器就是这样

变化的），则中间会出现 0111→0110→0100→1000 多种状态，即会出现短暂的错误编码 0110 和 0100。如果使用 Gray 码来完成同样的任务，因为相邻的两个 Gray 码只有一位变化，就可以避免出现这种错误。

格雷码和余 3 码都是无权码，就是每个数位上没有固定的权值。

3.1.3 非数值信息的表示

计算机发明的初期主要是用来进行科学运算的，但随着计算机应用的不断扩展和丰富，现在计算机很大一部分功能是用来处理非数值信息的，如文字、图像、视频和声音信息等。与数值信息类似，非数值信息在计算机中也是用二进制编码来表示的，但具体的表示方法随信息类型的不同和应用的不同有很大区别。下面就来讨论一下非数值信息的编码与表示。

1. 文字信息的表示

【课堂提问 3-4】 英文使用了 26 个字母（不分大小写），试着给出一种二进制编码方案（包括使用多少位（码长）以及如何编码）来表示这些字母。如果区分大小写，编码方案应如何改动？

每一种语言的文字都是由字母、数字、标点符号及一些特殊符号所组成，它们通称为字符（character）。**字符**是各种文字和符号的总称，包括各国家的文字符号、标点符号、图形符号和数字等，与某种用途相关的字符集合称为**字符集**。字符集种类很多，每个字符集包含的字符个数不同，常见的字符集有：ASCII 字符集、GB2312 字符集、BIG5 字符集、GB18030 字符集和 Unicode 字符集等。要让计算机能够识别、存储和处理各种文字，首先要对相应语言所使用的字符集中的字符进行编码，就是每种符号设定一个数字或二进制形式的代号。编码的码长与字符集的大小有关，如 8 位编码可以表示大小为 256（2^8）个字符的字符集，16 位编码可以表示 65536（2^{16}）个字符大小的字符集。

【课堂提问 3-5】 请举出身边事物的编码。

字符集中每个字符都使用一个唯一的编码来表示（字符的二进制表示），所有的编码就构成了该字符集的编码表，简称**码表**。世界上不同文字系统的字符集有很多，相应的编码也有很多。

1）英文字符编码

计算机技术发源于美国，因此最早的信息编码也来源于美国。计算机中使用最广泛的西文字符集及其编码是 ASCII 码（American Standard Code for Information Interchange，美国信息交换标准编码），它已被国际标准化组织（International Organization for Standardization，ISO）批准为国际标准，称为 ISO 646 标准。

（1）标准 ASCII 编码

标准 ASCII 码采用 7 位编码（如果用 8 位来存储，则最高位总为 0），编码范围是 00H～7FH，总共可以表示 128 个符号。ASCII 字符集包括英文字母、阿拉伯数字、标点符号以及一些计算机系统中使用的控制编码等。标准 ASCII 码编码表如表 3-3 所示，其中行号代表编码的高位，列号代表编码的低位，是十六进制形式，如字母 A，行是 4，列是

1，则字符 A 的编码是 41H(十进制 65)。

表 3-3　标准 ASCII 代码表(十六进制)

低位/高位	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2	SP	!	"	#	$	%	&	'	(	)	*	+	,	−	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

标准 ASCII 码中的 00H～20H 和 7FH 为 33 个控制编码(有时也称为控制字符)，用来表示计算机中的控制动作。这些控制编码与早期计算机系统使用电传打字机作为操作终端有很大的关系。例如，CR(编码为 0DH)表示回车，当电传打字机接收到此字符编码时，就会使打印头回到起始位置，而不是将它打印出来。其他一些编码则用来表示数字、字母和标点符号等**可打印字符**(也叫**可显示字符**)。数字字符 0～9 的 ASCII 码是连续的，为 30H～39H，十进制是 48～57；大写英文字母 A～Z 和小写英文字母 a～z 的 ASCII 码也是连续的，分别为 41H～54H(十进制是 65～90)和 61H～74H(十进制是 97～122)。因此在知道一个数字或字母的编码后，即可推算出其他数字和字母的编码。

标准 ASCII 码共有 94 个可打印的字符，称为**图形字符**。这些字符在计算机键盘上都有相应的键，当操作者按下某一个键时，该键对应的字符的 ASCII 编码就被输入到计算机的存储器中供进一步处理。

只支持 ASCII 码的应用系统会忽略字节流中每个字节的最高位，只认为低 7 位是有效位。例如，因特网中的邮件传输协议 SMTP 就只支持标准 ASCII 编码，所以 SMTP 为了传输采用 8 位编码的信息(如包含汉字字符的邮件)，就必须使用 BASE64 或者其他编码方式对传输的内容重新编码。

(2) 扩充 ASCII 编码和 ISO8859

7 位标准 ASCII 编码用来处理英文没有什么问题，但是若还要表示其他国家的文字，128 个编码显然不够用。于是 IBM 公司在设计微型计算机时将 ASCII 编码进行了扩充，使用 8 位二进制来编码其他一些西欧语系的文字符号，具体方法就是使用一个字节的全部 8 位，最高位为 0 的 128 个编码与标准 ASCII 编码完全相同，最高位为 1 的 128 个编码则用来编码扩充的文字符号，扩充出来的 128 个编码称为**扩展 ASCII 编码**，对应的字符称为**扩展 ASCII 字符**。

这种扩充 ASCII 编码的方法也被 ISO 所采纳，这就是 **ISO8859** 编码标准(也称为 ISO Latin)。ISO8859 标准使用了 8 位编码，收录的字符除标准 ASCII 字符外，还包括西

欧语言、希腊语、阿拉伯语、希伯来语等语系所对应的文字符号。而根据语系的不同，ISO8859 的编码标准又分为几个部分，如 ISO8859-1(Latin-1)所包含的符号可以用于西班牙语、丹麦语、德语、意大利语等，而 ISO8859-15(Latin-15)则可以用于法语、芬兰语等。每种编码的 0～127 与 ASCII 编码完全相同，只是 128～255(相应二进制编码的最高位为 1)才根据语系的不同而不同。这意味着不同国家的扩展 ASCII 字符往往是不兼容的，也就是说表示的是不同的文字符号。

2) 中文字符的表示

【课堂提问 3-6】 中文属于象形文字，它所使用的字符集要远远大于属于拼音文字的西文(如英文、希腊文、西班牙文等)。基本的汉字字符集大约有 7000 个汉字。试着给出一种二进制编码方案(包括使用多少位(码长)以及如何编码)来表示这些汉字。如果希望编码方案能够兼容 ASCII 编码，应该如何处理呢？

(1) 国家汉字编码标准 GB2312-80

汉字编码的基本思路是将汉字以及汉语中用的大小写字母、数字、标点符号、日语符号、希腊字母、俄文字母、拼音符号、注音字母、图形符号等编为 94 行 94 列的表格，每行叫做一个区，每列叫做一个位。这样，每个符号对应一个区号和一个位号，例如“啊”字在 16 区，第 1 列，那么 1601 就是“啊”字的数字编码，这个数字叫**区位码**。区位码是 4 位的十进制数，它的前两位是区号，也叫**区码**；后两位是位号，也叫**位码**。区位码的十六进制形式是将区码和位码分别转换为两位的十六进制数。如 1601，16 转换为十六进制是 10H，01 转换为十六进制是 01H，那么“啊”字的区位码的十六进制是 1001H，在计算机中存储，占两个字节。

区位码表中，01～09 区为符号、数字区；16～87 区为汉字区；10～15 区、88～94 区是有待进一步标准化的空白区。汉字区中 16～55 区收录常用汉字 3755 个，称为第一级汉字，按汉语拼音字母/笔形顺序排列；56～87 区收录次常用汉字 3008 个，称为第二级汉字，按部首/笔画顺序排列。共计 6763 个汉字。

汉字的显示和打印也需要使用控制符。ASCII 中的控制符使用的编码是 00～1FH 共 32 个编码。为了和 ASCII 中的控制符兼容，汉字的编码决定不再使用 00～1FH 的数字，区码和位码都不用。于是，将区码和位码都后移 32(20H)，即各加 20H，这样得到的每个汉字符号的编码称为**国标码**。如，“啊”字的区位码为 1001H，1001H＋2020H＝3021H 就是“啊”字的国标码。

然而，当 ASCII 编码的文件和国标码编码的文件在一个系统中时，遇到 3021H，那么它是 ASCII 中的“0”和“!”呢？还是汉字“啊”呢？这就分不清了。于是，为了和 ASCII 兼容，汉字编码决定不再使用 00～7FH 共 128 个编码位。再将国标码的区码和位码部分各自再后移 128(80H)个位置，即各加 80H，这样得到的编码称为**机内码**。例如，“啊”字的国标码为 3021H，3021H＋8080H＝B0A1H 就是“啊”的机内码。

事实上，目前一般计算机中保存的汉字，保存的都是它们的机内码。机内码占两个字节，每个字节的最高位都是 1，所以，当遇到一个字节最高位不是 1(值小于 128)时，那么它是一个 ASCII 符号，最高位是 1(大于 128)时，那么它和后面一个字节表示一个汉字符号。

上述编码方法形成国家汉字编码标准——“信息交换用汉字编码字符集(基本集)”，代号是 GB2312-80，简称国标码，于 1980 年发布。

(2) GBK

GBK(“国标”、“扩展”汉语拼音的第一个字母)编码，是在 GB2312-80 标准基础上的扩展，叫做《汉字内码扩展规范》，使用双字节编码方案，其编码范围为 8140H～FEFEH(剔除 xx7F)，共 23940 个码位，收录 21003 个汉字，完全兼容 GB2312-80 标准，支持国际标准 ISO/IEC10646-1 和国家标准 GB13000-1 中的全部中日韩汉字，并包含了 BIG5 编码中的所有汉字，集简体、繁体字与一体。GBK 编码方案于 1995 年 12 月正式发布，中文版的 Windows 95，Windows 98，Windows NT 以及 Windows 2000，Windows XP，Windows 7 等都支持 GBK 编码方案。

(3) GB18030

GB18030 的全称是《信息交换用汉字编码字符集基本集的补充》，是我国计算机系统必须遵循的基础性国家标准之一。GB18030 有两个版本：GB18030-2000 和 GB18030-2005。GB18030-2000 是 GBK 的取代版本，2000 年发布，它的主要特点是在 GBK 基础上增加了 CJK 统一汉字扩充 A 的汉字，汉字字符数达到 27533 个。GB18030-2005 于 2005 年发布，主要特点是在 GB18030-2000 基础上增加了 CJK 统一汉字扩充 B 的汉字，包括多种我国少数民族文字(如藏、蒙古、傣、彝、朝鲜、维吾尔文等)，汉字字符数达到 70244 个。

GB18030 标准采用单字节、双字节和四字节三种方式对字符编码。单字节使用 0X00～0X7F 码位(ASCII)。双字节部分，首字节码位是 0X81～0XFE，尾字节码位分别是 0X40～0X7E 和 0X80～0XFE。四字节部分范围为 0X81308130～0XFE39FE39，其中第一、三字节编码码位均为 0X81～0XFE，第二、四字节编码码位均为 0X30～0X39。这里以**0X 开头的数表示是十六进制数**(大小写相同)。GB18030 的码位结构包含的字符数及与其他标准的关系见表 3-4，其中的码位的编码范围均是十六进制。

表 3-4　GB18030-2005 标准码位结构及与其他标准的关系

编码方案			第 1 字节	第 2 字节	第 3 字节	第 4 字节	码位数	字符数	字符类型
GB18030-2005，70244 汉字	GB18030-2000，27533 汉字		00～7F					128	ASCII
		GBK，21003 汉字	81～A0	40～FE			6080	6080	GBK 汉字
			AA～FE	40～A0			8160	8160	GBK 汉字
			B0～F7	A1～FE			6768	6763	GB2312 汉字
			81～82	30～39	81～FE	30～39	25200	6530	CJK 统一汉字扩充 A
			95～98	30～39	81～FE	30～39	50400	42711	CJK 统一汉字扩充 B

从表中看出，后面的标准与前面的标准是兼容的，这样，以前编辑的文字，在新的标准下仍能使用。兼容性是系统设计时应考虑的重要指标。

（4）BIG5

BIG5 是我国台湾地区使用的汉字编码字符集，它包含了台湾地区使用的 13060 个繁体汉字和 420 个图形符号。BIG5 也使用 16 位编码方案，以两个字节来存放一个汉字编码。第一个字节称为“高位字节”，第二个字节称为“低位字节”。“高位字节”的编码范围为 0x81～0xFE，“低位字节”的编码范围是 0x40～0x7E 及 0xA1～0xFE。

（5）CJK

CJK（Chinese Japanese Korean）编码是 ISO 10646 通用字符集（universal character set，UCS）在汉字编码上的具体实现。与 CJK 编码相对应的国家标准号为 GB13000-90。CJK 采用的是双字节形式的基本多文种平面。在 65536 个码位的空间中，定义了几乎所有国家或地区的语言文字和符号。其中从 0x4E00～0x9FA5 的连续区域包含了 20902 个来自中国（包括台湾地区）、日本、韩国的汉字。CJK 是 GB2312-80、BIG5 等字符集的超集。

3）ANSI 信息编码

ANSI（American National Standards Institute，美国国家标准协会）是专为计算机工业建立标准的机构，在世界上具有相当重要的影响力。**ANSI 编码**是 ANSI 制定的一种采用双字节编码的信息编码标准。ANSI 编码包含很多代码页，每个代码页与字符集有关，例如代码页 936 对应的就是 GB2312-80，代码页 950 对应的就是 BIG5，代码页 65001 对应的就是 UTF-8 等。

不同 ANSI 代码页之间互不兼容，使用某个 ANSI 代码页编码的信息无法在采用其他 ANSI 代码页的系统中正常显示，这就是为什么日文版/繁体中文版软件在简体中文系统中会出现乱码的原因。显然，属于两种语言的文字无法存储在同一个使用 ANSI 编码的文本中。

4）ISO 10646 和 Unicode 编码

ISO 10646 是国际标准化组织 ISO（International Organization for Standardization）和国际电工委员会 IEC（International Electrotechnical Commission）制订的实现全球所有文字统一编码标准——《信息技术通用编码字符集》的代号，简称 UCS（universal coded character set），1993 年发布第 1 个版本。Unicode 是多语言软件制造商组成的统一码联盟制订的统一编码标准，于 1994 年发布。它们的目的是制订一个全世界统一的编码标准，以便全球文字能兼容使用。1991 年左右，两个项目的参与者都认识到，世界不需要两个不兼容的“统一字符集”。于是，它们开始合并双方的工作成果，并为创立一个单一编码表而协同工作。两个项目仍都存在，并独立地公布各自的标准，但都同意保持两者标准的码表兼容，所以，Unicode 和 UCS 或 ISO 10646 是一致的。

（1）编码方式

最初 Unicode 字符分 17 组编排，码位从 0x0000～0x10FFFF，最高一个字节就是组号，每组称为一个平面（Plane），每个平面有 256 行和 256 列，共 65536 个码位，总码位有 1114112 个，然而目前只用了少数平面。

通用字符集（UCS）UCS-2 用两个字节编码，UCS-4 用 4 个字节编码。

UCS-4 根据最高位为 0 的最高字节分成 $2^7=128$ 个组（group）。每个组再根据次高字节分为 256 个平面（plane）。每个平面根据第 3 个字节分为 256 行（row），每行有

256 个码位(cell)。理论上能容纳的字符数是 128×256×256×256＝2 147 483 648 个。group 0 的平面 0 被称作基本多语言面 BMP(Basic Multilingual Plane)。如果 UCS-4 的前两个字节为全零,那么去掉这两个零字节就得到了 UCS-2。Unicode 计划使用的 17 个平面,在 group 0。在 Unicode 5.0.0 版本中,已定义的码位只有 238605 个,分布在平面 0、平面 1、平面 2、平面 14、平面 15、平面 16。平面 0 上定义了 27973 个汉字,平面 2 上定义了 43253 个汉字,共 71226 个。Unicode 所定义的 6 个平面中,第 0 平面(BMP)最为重要,目前实现的也是这个平面。常用的中文定义在 0x4E00-0x9FA5 中。

(2) 实现方式

Unicode 的实现方式不同于编码方式。一个字符的 Unicode 编码是确定的,但它在计算机中的实现方式可能不同,这其中的原因还是源于兼容、存储效率和传输效率问题。

Unicode 在诞生之日起就面临着一个严峻的问题:如何与已在世界范围内广泛使用的 ASCII 字符集兼容。ASCII 字符是单个字节的,比如 A 的 ASCII 是 65。而 Unicode 是双字节的,比如 A 的 Unicode 是 0065,这就造成了一个非常大的问题:以前处理 ASCII 码的方法不能被用来处理 Unicode 了。另一个更加严重的问题是,C 语言使用'\0'作为字符串结尾,而 Unicode 里恰恰有很多字符都包括了一个字节的 0,这样一来,C 语言的字符串函数将无法正常处理 Unicode,除非把世界上所有用 C 写的程序以及它们所用的函数库全部换掉。此外,一个仅包含标准 ASCII 字符的 Unicode 文件,如果每个字符都使用 Unicode 编码,则其第一字节的 8 位始终为 0,这就造成了很大的浪费,降低了信息的存储、传输和处理效率。

于是,就有了如何在计算机中实现 Unicode 的问题。解决这个问题的方案类似于霍夫曼(Huffman)编码的变长编码思想。这就是 **UTF(UCS Transformation Format)**。它是将 Unicode 编码规则和在计算机的实现对应起来的一个规则。现在流行的 UTF 有三种:UTF-8、UTF-16 和 UTF-32。

① UTF-8

UTF-8 以字节为基本单位对 Unicode 进行编码。从 Unicode 到 UTF-8 的编码方式见表 3-5。

表 3-5　UTF-8 的编码方式

Unicode 编码(十六进制)	UTF-8 字节流(二进制)
00000000-0000007F	0xxxxxxx
00000080-000007FF	110xxxxx 10xxxxxx
00000800-0000FFFF	1110xxxx 10xxxxxx 10xxxxxx
00010000-001FFFFF	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx
00200000-03FFFFFF	111110xx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx
04000000-7FFFFFFF	1111110x 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx

UTF-8 的特点是对不同范围的字符使用不同长度的编码,也就是**变长编码**,最大长度是 6 个字节。对于 0x00-0x7F 的字符,UTF-8 编码与 ASCII 编码完全相同。最后一行