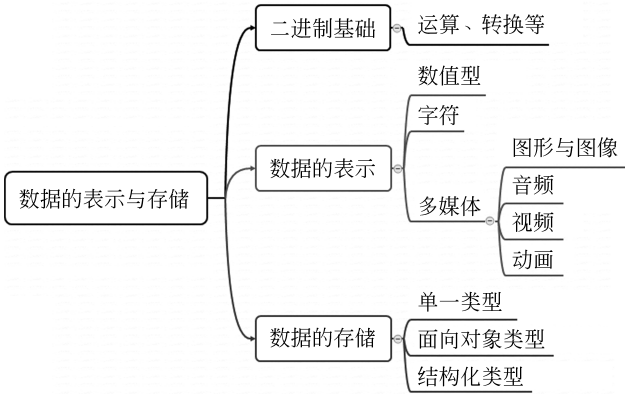
第3章

数据的表示与存储

在数据处理中最常用到的基本概念就是数据和信息，它们既有区别又有联系。数据是用来描述客观事物的可识别的符号，这些符号既可以是数字，也可以是文字、声音、图形和图像等多种表现形式。信息是指经过加工处理可以产生影响的数据表现形式，这种数据形式对于接收者是有意义的。由此可见，数据是信息的载体，信息是数据处理的结果。数据处理的目的就是从原始数据中得到有用的信息。因此，数据处理也称信息处理。

显然，用计算机进行的数据处理，这些数据不仅仅是数值型数据，还包括非数值型数据。采用什么方法可以较好地解决数值型数据和非数值型数据的表示、处理、传输及存储就是本章要介绍的内容。



3.1 数据在计算机中的表示

3.1.1 数制

数制也称记数制，是用一组固定的符号和统一的规则来表示数值的方法。人们生活中买东西时使用的就是十进制，即逢十进一。但也有不使用十进制的时候，例如，时间有12小时制和24小时制；每星期有7天，第8天属于下周一等等。在不同的场合人们很自然地会使用不同的数制进行运算，以符合当时的习惯。

实际上，在讨论数制时都会涉及两个基本要素：基数和位权。

1）基数

基数是指在某种进位记数制中，每个数位上能够使用数字的个数。例如，二进制的基数为 2，每个数位上能够使用的数字符号为 0、1；十进制的基数为 10，每个数位上能够使用的数字符号为 0～9。

表 3.1 为常见进制的表示方法。

表 3.1 常见进制的表示方法

项 目	二进制	八进制	十进制	十六进制
数字符号	0，1	0～7	0～9	0～9，A，B，C，D，E，F
规则	逢二进一	逢八进一	逢十进一	逢十六进一
基数	2	8	10	16
位权	2^i	8^i	10^i	16^i
表示方法	B	O	D	H

不同进制数的两种表示方法。

(1) 字母后缀。

例如，二进制数 1010 表示为 1010B，八进制数 23.45 表示为 23.45O，十进制数 123.45 表示为 123.45D，十六进制数 A1.23 表示为 A1.23H。一般来说，对于十进制数的后缀可以省略。

(2) 括号外面加下标。

例如，上述数值可表示为$(1010)_2$、$(23.45)_8$、$(123.45)_{10}$、$(A1.23)_{16}$。

再举个例子，我有两个苹果，用十进制表示就是$(2)_{10}$，而用二进制表示就是$(10)_2$。为什么数字不一样呢？因为使用了不同的记数规则，二进制“逢二进一”。但不管用什么样的计数规则，我的苹果还是两个，没多也没少，所以$(2)_{10}=(10)_2$就很好理解了。

注意：以下内容中约定，十进制数按平时习惯均不做特别标注。

2）位权

位权是指一个数字在某个固定位置上所代表的值，处在不同位置上的数字代表的值不同。例如，十进制数 123，1 的位权是 100，2 的位权是 10，3 的位权是 1。位权与基数的关系：各进制中位权的值是基数的对应位次幂。位幂次的排列方式以小数点为界，整数自右向左，最低位为基数的 0 次幂；小数自左向右，最高位为基数的－1 次幂。任何一种数制表示的数都可以写成如下按位权展开的多项式之和。

例 3.1 将 123.45、1010.11B、23.45O、A1.23H 按位权展开。

$$123.45=1\times10^2+2\times10^1+3\times10^0+4\times10^{-1}+5\times10^{-2}$$

$$(1010.11)_2=1\times2^3+0\times2^2+1\times2^1+0\times2^0+1\times2^{-1}+1\times2^{-2}$$

$$(23.45)_8=2\times8^1+3\times8^0+4\times8^{-1}+5\times8^{-2}$$

$$(A1.23)_{16}=10\times16^1+1\times16^0+2\times16^{-1}+3\times16^{-2}$$

3.1.2　二进制

众所周知，在计算机中处理、存放的数据都是以 0 和 1 的二进制编码形式存放的。采用二进制编码基于以下 3 个原因。

(1) 物理上容易实现，可靠性强。电子元器件大都具有两种稳定的状态，如电压的高与低，电路的通与不通等，这两种状态正好可以使用二进制数的 0 和 1 表示。

(2) 运算简单，通用性强。二进制的运算比十进制的运算简单，如二进制的乘法运算只有 3 种：1×0=0×1=0，0×0=0，1×1=1。如果是十进制运算，则有 55 种情况。

(3) 计算机中二进制数的 0、1 数字与逻辑值“假”和“真”正好吻合，便于表示和进行逻辑运算。

对于数值型数据，都转换为二进制的形式。对于非数值型数据，如字符、文字、图形、声音、视频等，也要使用二进制进行编码表示为二进制数的形式，计算机才能对它们进行处理。此外，由于人们日常生活、工作当中使用的都是十进制数，而计算机内部采用的是二进制数，因此，计算机在输出处理结果时还应把二进制数转换成十进制数的形式。

3.1.3　二进制运算

二进制数的算术运算与十进制数的算术运算一样，有加、减、乘、除四则运算，但运算更简单。

1. 二进制数的算术运算

(1) 二进制数加法、减法运算。

二进制数的加法运算规则：0+0=0，0+1=1，1+0=1，1+1=10(向高位进位)；两个二进制数相加时，每位最多有 3 个数：被加数、加数和来自低位的进位数。

二进制数的减法运算规则：0−0=0，0−1=1(向高位借位)，1−0=1，1−1=0；两个二进制数相减时，每位最多有 3 个数：被减数、减数和向高位的借位数。

(2) 二进制数乘法、除法运算。

二进制数的乘法运算规则：0×0=0×1=1×0=0，1×1=1

二进制数的除法运算规则：0/1=0，0/0 和 1/0(无意义)，1/1=1

2. 二进制的逻辑运算

在逻辑代数里，表示“真”与“假”、“是”与“否”、“有”与“无”这种具有逻辑属性的变量称为逻辑变量。使用二进制数 1 表示“真”，0 表示“假”，就可以将二进制数与逻辑取值对应起来。逻辑变量的取值只有两种：真和假，即 1 和 0。

逻辑运算有 3 种最基本的运算：逻辑“或”、逻辑“与”和逻辑“非”。此外，还有“异或”运算等。计算机的逻辑运算是按位进行，没有进位或借位关系。

(1) 逻辑“或”运算。

逻辑“或”通常用符号＋或∨表示。逻辑“或”运算规则如下：

$$0 \vee 0=0,0 \vee 1=1,1 \vee 0=1,1 \vee 1=1$$

只要两个逻辑变量中有一个为1,逻辑“或”的结果就为1;只有两个逻辑变量同时为0时,结果才为0。

例 3.2 $X=(10100100)_2$,$Y=(10111010)_2$,求 $X \vee Y$ 的结果。

$$\begin{array}{rr} & 10100100 \\ \vee & 10111010 \\ \hline & 10111110 \end{array}$$

即:$X \vee Y=(10111110)_2$

(2) 逻辑“与”运算。

逻辑“与”通常用符号·或∧表示。逻辑“与”运算规则如下:

$$0 \wedge 0=0,\quad 0 \wedge 1=0,\quad 1 \wedge 0=0,\quad 1 \wedge 1=1$$

只要两个逻辑变量中有一个为0,逻辑“与”的结果就为0;只有两个逻辑变量同时为1时,结果才为1。

例 3.3 $X=(10100100)_2$,$Y=(10111010)_2$,求 $X \wedge Y$ 的结果。

$$\begin{array}{rr} & 10100100 \\ \wedge & 10111010 \\ \hline & 10100000 \end{array}$$

即:$X \wedge Y=(10100000)_2$

(3) 逻辑“非”运算。

逻辑“非”运算又称“求反”运算,通常在逻辑数据上面加一横线表示。对某数进行逻辑“非”运算,就是对它的各位求反,即0变为1,1变为0。

逻辑“非”运算规则:

$$\overline{0}=1,\quad \overline{1}=0$$

例如,$X=(10100100)_2$,则 $\overline{X}=(01011011)_2$。

在计算机中,逻辑数据的值用于判断某个条件是否成立,成立为1(真),不成立为0(假)。当要对多个条件进行判断时,就需要用逻辑变量和逻辑运算符把它们连接起来进行运算,结果为逻辑值。

在逻辑运算中,将逻辑变量的各种可能组合与对应的运算结果列成表格,这个表格称为真值表,它是全面描述逻辑运算关系的工具之一。一般在真值表中用1或T表示“真”(True),用0或F表示“假”(False)。

3.1.4 进制的转换

由于计算机在进行数据处理时使用的是二进制,因此,其处理结果自然也是二进制数。如果计算机把这个处理结果以二进制数的形式输出(显示、打印)给用户,用户很可能看不懂,必须要把它(二进制结果)转换成平时采用的十进制数。

二进制只有0和1两个数字可以使用,而十进制有0~9十个数字可以使用,在相同位数的情况下,二进制数表示的数值范围比十进制数表示的数值范围小。例如,两位二进制数表示的数值有$(00)_2$、$(01)_2$、$(10)_2$和$(11)_2$共4个,而十进制数表示的数值有00、01、02、……、98、99共100个。由此可以看出,用二进制表示比较大一点的数,例如1000,需

要的二进制位数比用十进制表示时需要的位数多很多，书写起来很不方便。为解决这个问题，可以采用八进制和十六进制来替代二进制，因为二进制、八进制、十六进制之间很容易相互转换。实际上在许多计算机应用软件中表示存储地址、颜色时，通常都是使用十六进制来描述的。

V3.1 二进制与十进制转换

1. 二进制数、八进制数、十六进制数转换成十进制数

二进制数、八进制数、十六进制数转换成十进制数采用上述介绍的按位权展开方法，把各项相加即可。

例 3.4　将$(101.01)_2$、$(24.4)_8$、$(35.C)_{16}$转换成十进制数。

$$(101.01)_2 = 1\times2^2 + 0\times2^1 + 1\times2^0 + 0\times2^{-1} + 1\times2^{-2}$$
$$= 4 + 0 + 1 + 0 + 0.25 = 5.25$$
$$(24.4)_8 = 2\times8^1 + 4\times8^0 + 4\times8^{-1} = 16 + 4 + 0.5 = 20.5$$
$$(35.C)_{16} = 3\times16^1 + 5\times16^0 + 12\times16^{-1} = 48 + 5 + 0.75 = 53.75$$

2. 十进制数转换成二进制数、八进制数、十六进制数

(1) 整数部分：除以基数取余数法。

整数部分的转换采用“除以基数取余数法”，即用基数多次除被转换的十进制数，直至商为 0，每次相除所得余数，按照第一次除所得为最低位，最后一次除所得为最高位把余数排列起来，便可得到转换结果。

例 3.5　将十进制数 13 转换成二进制数。

```
2 | 13
2 |  6    ……1   低位
2 |  3    ……0
2 |  1    ……1
     0    ……1   高位
```

即：$13=(1101)_2$

例 3.6　将十进制数 156 转换成八进制数和十六进制数。

```
8 | 156
8 |  19   ……4   低位
8 |  2    ……3
     0    ……2   高位
```

```
16 | 156
16 |  9    ……  C   低位
      0    ……  9   高位
```

即：$156=(234)_8=(9C)_{16}$

注意：对于不同进制整数之间的互相转换，可以使用 Windows 操作系统提供的“计算器”很方便地解决。具体方法如下。

① 单击 Windows 屏幕“开始”按钮，在弹出的程序列表中选择“计算器”命令，启动计

算器。

② 单击“计算器”窗口左上角的“打开导航”→“程序员”命令。

③ 单击“计算器”窗口靠左上的基数列表中的基数(需要转换的数的基数),其中HEX表示十六进制,DEC表示十进制,OCT表示八进制,BIN表示二进制。

④ 输入需要转换的整数,基数列表旁边会显示不同进制数的转换结果。

(2) 小数部分:乘以基数取整数法。

用十进制小数乘以基数不断取整数,直至小数部分为0(对于不能为0的小数,只需达到所求精度即可)。所得的整数从小数点自左向右排列,首次取得的整数在最左。

例 3.7 将十进制数0.625转换成二进制数和八进制数。

```
    0.625                          0.625
  ×     2                        ×     8
  -------                        -------
  [1].250  ……1  高位              [5].000  ……5  高位
  ×     2                        ×     8
  -------                        -------
  [0].500  ……0                    [0].160  ……0
  ×     2                        ×     8
  -------                        -------
  [1].000  ……1  低位              [1].280  ……1  低位
```

即:$0.625=(0.101)_2\approx(0.501)_8$

由上述例题可以看出,当包含有小数的数值要转换成不同进制数时,很多情况下是无法实现精确转换的。

3. 二进制数、八进制数、十六进制数之间的转换方法

由于$2^3=8$,$2^4=16$,即1位八进制数所表示的范围与3位二进制数所表示的范围相同,1位十六进制数所表示的范围与4位二进制数所表示的范围相同,如表3.2所示。利用这个特点进行转换,方法就比较简单。

表 3.2 二进制、八进制、十进制和十六进制的数据对应关系表

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	0000	0	0	8	1000	10	8
1	0001	1	1	9	1001	11	9
2	0010	2	2	10	1010	12	A
3	0011	3	3	11	1011	13	B
4	0100	4	4	12	1100	14	C
5	0101	5	5	13	1101	15	D
6	0110	6	6	14	1110	16	E
7	0111	7	7	15	1111	17	F

二进制数转换成八进制数的方法:以小数点为界向左右两边进行分组,每3位为一

组，不足 3 位就用 0 补足，每组用一个八进制数表示即可，简称“三合一”。同样，二进制数转换成十六进制数的方法：以小数点为界向左右两边进行分组，每 4 位为一组，不足 4 位就用 0 补足，每组用一个十六进制数表示即可，简称“四合一”。

例 3.8　将二进制数$(10110101.10101)_2$转换成八进制数和十六进制数。

$(\underline{010}\ \underline{110}\ \underline{101}.\underline{101}\ \underline{010})_2=(265.52)_8$（整数高位和小数低位分别补 0）

2　6　5　5　2

$(\underline{1011}\ \underline{0101}.\underline{1010}\ \underline{1000})_2=(B5.A8)_{16}$（小数低位补 0）

B　5　A　8

同样，将八进制数转换成二进制数时，只需把每位八进制数拆分为 3 位二进制数，简称“一拆三”；将十六进制数转换成二进制数时，只需把每位十六进制数拆分为 4 位二进制数，简称“一拆四”。

例 3.9　将八进制数$(123.25)_8$和十六进制数$(38.D2)_{16}$转换为二进制数。

$(123.25)_8=(\underline{001}\ \underline{010}\ \underline{011}.\underline{010}\ \underline{101})_2=(1010011.010101)_2$

$(38.D2)_{16}=(\underline{0011}\ \underline{1000}.\underline{1101}\ \underline{0010})_2=(111000.1101001)_2$

整数最高位的 0 和小数最低位的 0 可以去掉。

当需要将八进制数和十六进制数相互转换时，可以先将其转换为二进制数，然后再将这个二进制数转换成对应的八（十六）进制数即可。

例 3.10　将八进制数$(123.25)_8$转换为十六进制数。

$(123.25)_8=(\underline{001}\ \underline{010}\ \underline{011}.\underline{010}\ \underline{101})_2=(\underline{0000}\ \underline{0101}\ \underline{0011}.\underline{0101}\ \underline{0100})_2$

0　5　3　5　4

即：$(123.25)_8=(53.54)_{16}$

3.1.5　数值在计算机中的表示与存储

前面介绍了数制和二进制的基本运算规则，但在实际应用当中，参与算术运算的数据都是带有正、负符号的，多数都还带有小数，符号怎么表示？这些数据在计算机中如何存储？这些问题都有待解决，下面逐一介绍。

1. 机器数和真值

在计算机中，对于数学上的正（+）、负（−）号也只能使用二进制中的 0 和 1 两个数字表示。规定用 0 表示正，1 表示负，因为计算机存储信息时是以字节为单位，符号就存放在该字节的最高位，称为**符号位**，也称**数符**。

例 3.11　一个 8 位二进制数 −0110011，它在计算机中表示为 10110011，如图 3.1 所示。

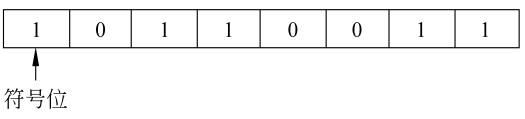


图 3.1　机器数

这种把符号数值化了的数称为“机器数”，而它代表的数值称为该机器数的“真值”。在例 3.11 中，10110011 为机器数，－0110011 为该机器数的真值。

数值的符号经过数字化后就能被计算机识别了，但由于此时符号位和数值是一起存放的，数据在进行运算时，符号位如果参与一起运算，有时会产生错误的结果，见例 3.12。

例 3.12 计算(－9)＋6＝？结果应为－3。可是在计算机中按照上述所说的符号位也一起参与运算，其结果为

$$\begin{array}{rll} & 10001001 & \cdots\cdots -9 \text{ 的机器数} \\ + & 00000110 & \cdots\cdots +6 \text{ 的机器数} \\ \hline & 10001111 & \cdots\cdots -15 \text{ 运算结果} \end{array}$$

显然这个结果是错误的。若要把符号位单独分开进行判断，反而把本来简单的运算变得复杂了。这就说明上述的方法只是解决了正数和负数的表示问题，没有解决数值运算的问题。通过分析可以看出，这都是由负数的符号位产生的问题，那么还有其他办法吗？实际上符号数还有多种的表示方式，常见的有原码、反码和补码方法。为简单起见，下面以整数、一字节为例说明。

(1) 原码。

原码的表示规则：机器数的最高位表示符号位，正数的符号位为 0，负数的符号位为 1，其余各位是数值的绝对值，通常用$[X]_{原}$方式表示。

例 3.13 整数 X 分别取不同的值时，给出其原码的表示结果。

$$[+1]_{原}=00000001 \qquad [-1]_{原}=10000001$$
$$[+1011011]_{原}=01011011 \qquad [-1011011]_{原}=11011011$$
$$[+127]_{原}=01111111 \qquad [-127]_{原}=11111111$$

从上例可见，8 位原码表示的最大值是正的 127(即 2^7-1)，最小值是负的 127。

这个方法就是前面采用的方法，简单方便，机器数与真值转换容易，但存在以下问题：①数值 0 的表示有两种形式，$[+0]_{原}=00000000$，$[-0]_{原}=10000000$，给计算机判断带来不便；②做算术运算时，符号位需要单独处理，增加了运算规则的复杂性，即增加了机器的设计成本。

(2) 反码。

反码的表示规则：正数的反码就是其原码，负数的反码是符号位为 1，其余各位按位取反，通常用$[X]_{反}$方式表示。

例 3.14 设整数 X 分别取不同的值时，以下就是其反码的表示结果。

$$[+1]_{反}=00000001 \qquad [-1]_{反}=11111110$$
$$[+91]_{反}=01011011 \qquad [-91]_{反}=10100100$$
$$[+127]_{反}=01111111 \qquad [-127]_{反}=10000000$$

从例 3.14 可见，8 位反码表示方法与原码表示方法、数值表示范围都一样。同时，0 也是有两种不同的表示形式，$[+0]_{反}=00000000$，$[-0]_{反}=11111111$，存在二义性。

虽然反码与原码从外观看有很大区别，但运算同样都不方便，一般只用于求补码的过程。

(3) 补码。

补码的表示规则：正数的补码就是其原码，负数的补码是其反码在最低位加 1，通常用$[X]_{补}$方式表示。

例 3.15　设整数 X 分别取不同的值时，以下就是其补码的表示结果。

$[+1]_{补}=00000001$　　$[-1]_{补}=11111111$

$[+91]_{补}=01011011$　　$[-91]_{补}=10100101$

$[+127]_{补}=01111111$　　$[-127]_{补}=10000001$

补码表示方法中，0 的表示方式是唯一的：$[+0]_{补}=00000000$，$[-0]_{补}=00000000$。

从上例可见，编码 10000000 没有使用，可以用来扩充补码所能表示的数值范围，即用它表示负 128，那补码的数值范围就是－128～＋127。当然$[10000000]_{补}$中的最高位就比较特殊，既可看作符号位负数，也可作为数值位，其数值为－128，与原码、反码不同。下面看看补码的运算。

例 3.16　(－9)＋15 的运算。

```
    11110111      …… －9 的补码
+   00001111      ……＋15 的补码
-------------
  [1]00000110     ……运算结果
```

运算结果的高位 1 丢弃，运算结果为$[00000110]_{补}$，符号位为 0，说明这是个正数，其对应的真值是它数值位的数值 0000110，转换为十进制数是 6，计算结果正确。

再看看补码的运算能否解决例 3.12 出现的问题。

例 3.17　(－9)＋6 的运算。

```
    11110111      ……－9 的补码
+   00000110      ……＋6 的补码
-------------
    11111101      ……运算结果
```

运算结果为$[11111101]_{补}$，符号位为 1，说明结果为负数。对负数的补码求其真值的方法，即对该补码的数值位再做一次求补操作即可。

$[[11111101]_{补}]_{补}=10000011$，即－0000011，转换为十进制数就是－3，运算结果正确。

计算机在执行数值运算时，运行的结果只能在补码表示数值的范围内，超出的数位(进位)只能丢弃。

例 3.18　(－9)＋(－6)的运算。

```
    11110111      ……－9 的补码
+   11111010      ……－6 的补码
-------------
  [1]11110001     ……运算结果
```

运算结果的高位 1 丢弃，剩余的 8 位$[11110001]_{补}$，按照例 3.16 的方法可求出运算结果为十进制数－15。

例 3.19　90＋60 的运算。

```
    01011010      ……90 的补码
+   00111100      ……60 的补码
-------------
    10010110      ……运算结果
```

运算结果$[10010110]_{补}$的最高位为1，说明结果为负数。为什么两个正数相加的结果会是个负数呢？原因是这两个数相加的结果已经超出了使用一字节补码表示数值的范围(−128～+127)，这种现象称为“溢出”。

从上述例子可见，利用补码方法可以很方便地实现正负数的加法运算，规则简单。只要参与运算的数据是在数值表示范围内，符号位同数值位一样参与运算而不需要单独考虑。如果运算的结果为正数，结果的真值就是其本身；否则，就对结果再次求补即可得到其真值。且允许产生最高位的进位(丢弃)。

此外，计算(−9)+6，这种一个负数加上一个正数的运算，实际上是可以将它改写成6−9的形式，即是将加法运算转换成为减法运算，这可以简化电路设计。实际上计算机就是使用补码进行加法、减法运算。

图 3.2　时钟

对于利用补码实现减法转换为加法，不妨看看日常生活中熟悉的例子。如图3.2所示，假定时间采用12小时制，由于时钟出现故障，现在的时间应该是上午1点，但时钟显示的时间是3点。为了重新调整时钟，该怎么做？有两种方法。第一种方法是把时针向逆时针方向拨2小时；第二种方法是把时针向顺时针方向拨10小时，都可以达到目的。

这里，先设定时针往顺时针方向为正，逆时针方向为负。按第一种方法就很好理解，就是3−2=1；第二种方法就是3+10=13，因为采用的是12小时制即十二进制，逢十二进一，高位丢弃，余数为1，就是1点。同样，有其他错误时间也一样可以使用这两种方法进行校准。

由于时钟超过12的表示，相当于舍弃了进位，可以看到：3−2等同于3+10，3−1等同于3+11，9−4等同于9+8，…。在十二进制中(不考虑进位，即减去模)，上述式子中的−2、−1、−4可以用+10、+11、+8代替，即减法变成加法。

可以从上述例子得出一个结论：若两个数a、b之和等于R，则称a和b互为“补数”(补码)，R称为模。上面例子中，10和2、11和1、8和4就是以12为模的互为补码。可以看出，求a的补码b时，直接用$R-a$即可，有了b可以方便地把减法运算转换为加法运算，例如上面的式子3−2，先求出2的补码，即12−2=10，所以3−2可以转换为3+10=13，丢弃模12，13−12=1，得出结果为1。

模R的取值是指定数值范围中包含的整数个数。上面时钟例子，时钟共有12小时，故R的取值为12。计算机中的数据都是采用定长方式存储，8位二进制能表示的整数有$2^8=256$，$R=256$，推广到n位二进制，即$R=2^n$。以一字节为例，看看利用模R求补码的方法，能否在二进制和十进制中实现减法运算转换为加法运算。

例 3.20　105−83的运算。

方法一，采用十进制方法，8位二进制能表示的整数有$2^8=256$，所以$R=256$。按照公式$R=a+b$，已知$a=83$，求得$b=R-83=256-83=173$。

所以105−83可转换为105+173=278，丢弃模256，278−256=22，得出的结果正确。

方法二，采用二进制补码方法。

$$\begin{array}{rll} & 01101001 & \cdots\cdots 105\text{ 的补码} \\ + & 10101101 & \cdots\cdots -83\text{ 的补码} \\ \hline & \boxed{1}\,00010110 & \cdots\cdots\text{运算结果} \end{array}$$

丢弃高位进位 1，运算结果$[00010110]_{补}$的符号位是 0，说明结果是正数，其真值就是$(10110)_2$，转换为十进制就是 22，结果正确。

由此可见，在指定范围内，用补码方法实现减法运算转换成加法运算不但适用于二进制，在其他进制中也同样适用。

2. 实数在计算机中的表示

解决了数的符号表示和运算问题，接下来就要解决实数存储问题，之前讨论的数值都没有涉及小数，实际应用中经常使用的是实数，要采用什么样的形式存放？计算机中小数点是不占数位的，因此，就要规定小数点的位置来表示数值，分为定点整数、定点小数和浮点数 3 种形式。

(1) 定点整数。

定点整数规定小数点的位置在最低位的右边，这种方法表示的数为纯整数，如图 3.3 所示。

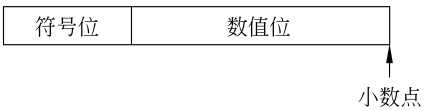


图 3.3　定点整数表示

(2) 定点小数。

定点小数规定小数点的位置在符号位的右边，这种方法表示的数为纯小数，如图 3.4 所示。

(3) 浮点数。

在计算机中定点数通常只用于表示纯整数或纯小数，其数值表示范围很有限，在实际应用中远远不够，尤其是在科学实验中需要表示特别大或者特别小的数值时，这时就要采用类似科学记数法的形式，称为浮点数表示，如图 3.5 所示。

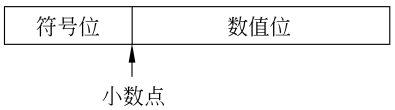


图 3.4　定点小数表示

图 3.5　浮点数表示

浮点数由阶码和尾数组成：阶码用定点数表示，阶码所占的位数确定了数值的范围；尾数用定点小数表示，尾数所占的位数确定了数值精度，即小数点后有效位数。因此，实数就是用浮点数来表示和存储的。

为了唯一地表示浮点数在计算机中的存储，对尾数采用了规格化的处理，规定尾数的最高位为 1，即所有规格化数必须转换成$\pm 0.1\times\times\times\cdots\times\times\times\times 2^{\pm p}$形式，其中 p 是指数(即阶码)，最终也要转换为二进制数，对于不符合要求的可以通过阶码调整。

IEEE 在 1985 年制定了 IEEE 754 标准，统一浮点数的存储格式。因此，在程序设计语言中比较常见的有以下两种类型的浮点数。

① 单精度浮点数(Float 或 Single)占 4 字节，其中阶符占 1 位，阶码占 7 位，尾符占

1 位，尾数占 23 位。

② 双精度浮点数(Double)占 8 字节，其中阶符占 1 位，阶码占 10 位，尾符占 1 位，尾数占 52 位。

可见双精度浮点数比单精度浮点数占用的存储空间更大，表示的数值范围更广，且精度更高。

例 3.21 36.5 采用单精度浮点数在计算机中的存储形式如图 3.6 所示。

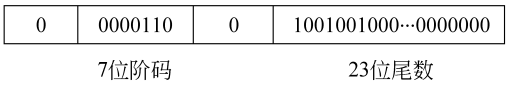


图 3.6 单精度浮点数存储格式

规格化表示：$36.5=100100.1B=0.1001001\times 2^{6}B=0.1001001\times 2^{110B}B$

从十进制数转换成二进制数的过程，要使十进制实数完全转换为二进制实数，十进制实数的小数部分最后一位必须是 5(当然这是必要条件，并非充分条件)，所以，一个十进制小数能用浮点数精确地表示，其小数部分的要求也是一样的。

3.2 字符的编码

字符包括西文字符(英文字母、数字、各种常用符号)、中文字符和一些特殊控制字符。计算机中的数据都是以二进制的形式存储和处理的，因此，字符也必须按特定的规则用二进制进行编码，即数字化后才能输入计算机进行处理。字符编码的方法：先确定需要编码的字符，把它们按一定的规律排好顺序，然后就可以开始从头到尾进行编码。当然，编码的长度与字符的总数有关，字符越多，长度越长。

3.2.1 ASCII 码

对于英文字符的编码，最常用的是美国信息交换标准代码(American Standard Code for Information Interchange，ASCII)。它是由美国国家标准研究所(American National Standards Institute，ANSI)制定的，供不同计算机在相互通信时共同使用，后来它被国际标准化组织(International Organization for Standardization，ISO)定为国际标准，称为 ISO 646 标准。ASCII 码采用 7 位二进制编码，共有 $2^{7}=128$ 种不同的组合，可以表示 128 个字符，包括 10 个数字字符 0～9、52 个大小写的英文字母以及特殊字符和控制字符，详见附录 A。

由附录 A 可以看到，ASCII 表的基本规律如下。

(1) 十进制码值 0～32 和 127，即表中的 NUL～SP 和 DEL 共 34 个字符为非图形字符(也称控制字符，不可见)，其余 94 个字符为图形字符，可显示出来。

(2) 字符 0～9、A～Z、a～z 的码值都是按从小到大顺序排列。

(3) 数字字符的码值小于英文字符。

(4) 小写字母比大写字母的码值大 32，为大小写字母的相互转换提供了方便。

计算机内信息的存储和处理是以字节(8 个二进制位)为单位来进行操作，因此，一个

字符在计算机内实际上是用 8 位二进制数表示。正常情况下，一个英文字符的 ASCII 码的最高位 d7 为 0。在需要进行奇偶校验时，这一位可用于存放奇偶校验码，这时称这一位为奇偶校验位。

英文字符除了常用的 ASCII 码外，还有 BCD 码(Binary-Coded Decimal)，它将十进制数的每位分别用 4 位二进制数表示，又称二-十进制编码；还有另一种 EBCDIC 码(Extended Binary Coded Decimal Interchange Code)，即扩展的二-十进制交换码，这种编码主要在大型计算机中使用。

3.2.2　汉字的表示

GB 2312—1980《信息交换用汉字编码字符集 基本集》收集和定义了 6763 个汉字及 682 个拉丁字母、俄文字母、汉语拼音字母、数字和常用符号等，共 7445 个汉字和字符。其中使用频度较高的 3755 个汉字为一级汉字，按汉字拼音字母顺序排列，使用频度较低的 3008 个汉字为二级汉字，按部首排列。

V3.2 汉字的编码

汉字字符比 ASCII 码表中的字符多很多，因此其编码比英文字符编码复杂。要在计算机中处理汉字，需要解决汉字的输入、处理、输出以及汉字的存储、传输等如下问题。

(1) 键盘上无汉字，不能直接利用键盘输入，需要输入码来对应。

(2) 汉字在计算机中的存储需要机内码来表示，以便存储、处理。

(3) 汉字量大、字形变化复杂，需要用对应的字库来存储。

计算机汉字处理流程如下(见图 3.7)。

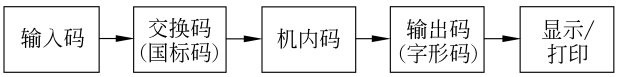


图 3.7　汉字处理流程

(1) 将汉字以输入码方式输入计算机中。

(2) 将输入码转换成计算机能够识别的汉字机内码进行处理、存储。

(3) 将机内码转换成汉字字形码输出。

1. 汉字输入码

汉字输入码是指从键盘上输入汉字时采用的编码，又称**外码**。汉字输入码有多种，目前采用较多的有：以拼音为基础的编码方案，例如“微软拼音”“搜狗拼音”等；字形编码方案，例如“微软五笔”“搜狗五笔”等。汉字输入码进入到计算机后必须转换成机内码。

2. 汉字交换码(国标码)和机内码

1980 年，我国颁布了汉字编码的国家标准 GB 2312－1980，这个字符集是我国中文信息处理技术的发展基础，也是目前国内所有汉字系统的统一标准，GB 2312－1980 称为**国标码**。GB 2312－1980 规定每个汉字用 2 字节的二进制编码，每字节的最高位为 0，其余 7 位用来表示汉字信息。为保证中、英文字符兼容，在计算机内部能区分 ASCII 字符

和汉字，将汉字国标码的 2 字节二进制编码的最高位置 1，从而得到对应的汉字机内码。汉字机内码是汉字在计算机内部被存储、处理和传输时使用的编码，简称**内码**。表 3.3 为中/英文字符的编码方案。

表 3.3　中/英文字符的编码方案

ASCII 码：	0	ASCII 码低 7 位		
国标码：	0	第一个字节低 7 位	0	第二个字节低 7 位
机内码：	1	第一个字节低 7 位	1	第二个字节低 7 位

例如，汉字“阿”的国标码 2 字节二进制编码为 00110000B 和 00100010B，对应的十六进制数为 30H 和 22H；而其机内码的 2 字节二进制编码为 10110000B、10100010B，对应的十六进制数为 B0H 和 A2H。

计算机处理字符数据时，如遇到最高位为 1 的字节，则可将该字节连同其后续最高位也为 1 的另一字节看作 1 个汉字的机内码；如遇到最高位为 0 的字节，则可将其判定为一个 ASCII 英文字符，这样就实现了汉字、英文字符的共存。

GB 18030 全称《信息技术 中文编码字符集》，是中华人民共和国国家标准所规定的变长多字节字符集。其对 GB 2312—1980 完全向后兼容，与 GBK 基本向后兼容，并支持 Unicode(GB 13000)的所有码位。GB 18030 当前版本为 GB 18030—2005，该标准规定了信息技术用的中文图形字符及其二进制编码的十六进制表示。

Unicode(**统一码、万国码**)在 1992 年被国际标准化组织确定为国际标准 ISO 10646，成为可以用于表示世界上所有文字和符号的字符编码方案。目前，所有的计算机都支持 Unicode。Unicode 用一些基本的保留字符制定了三套编码方式，分别是 UTF-8、UTF-16 和 UTF-32，UTF 是 Unicode Transformation Format 的缩写。在 UTF-8 中，字符是以 8 位二进制即一字节来编码的。用一或几字节表示一个字符，这种方式的最大好处是保留了 ASCII 字符的编码作为它的一部分。而其他字符，例如中国、日本、韩国等大部分常用字，使用 3 字节编码；UTF-16 和 UTF-32 分别是 Unicode 的前 16 位和前 32 位编码方式。

以上介绍的只是简体字的编码方式，繁体字的编码方式与简体字的编码方式不同，所以，在使用一些应用软件或者浏览网页时会出现乱码现象，如图 3.8 所示。

图 3.8　汉字乱码

如果浏览网页时出现乱码，可以右击页面，在弹出的快捷菜单中选择“编码”→Unicode(UTF-8)即可。乱码的原因很可能是选择了“中文(简体)(GBK)”造成的。如果还不成功那就是编码选择错了，要通过查看网页源文件头部，看一下网页的编码设置是哪一个，改成相应的即可。同理，如果是应用软件编辑窗口(如网页制作软件)内容乱码，通过应用软件的菜单中“编码”命令重新设置即可。

3. 汉字字形码

汉字**字形码**又称汉字字模，用于汉字在显示屏或打印机输出。汉字字形码通常有两种表示方式：点阵方式和矢量方式。

用点阵表示汉字时，汉字字形码指的是这个汉字字形点阵的代码。根据输出汉字的要求不同，点阵的多少也不同。常用的点阵有 16×16、24×24、32×32、64×64 或者更高。图 3.9 是“景”字的 24×24 点阵字形显示图。有笔画的小方格用二进制数 1 表示，没有笔画的小方格用二进制数 0 表示。显然，点阵规模越大字形越美观，但它所占的存储空间也越大。以 16×16 点阵为例，每个汉字字形就要占用 32 字节(16×16/8＝32 字节)，7400 多汉字和字符大约占用 256KB。如果觉得 16×16 点阵不够精细，可以把每个格子分成两半成为 32×32(即提高了分辨率)，每个汉字要占用的字节数就是 32×32÷8＝128 字节，比原来多了 3 倍存储空间。因此，字形点阵只能用来构成汉字库，一般存储在硬盘上，当要显示输出时才调入内存进行处理、输出，而不长驻内存当中。

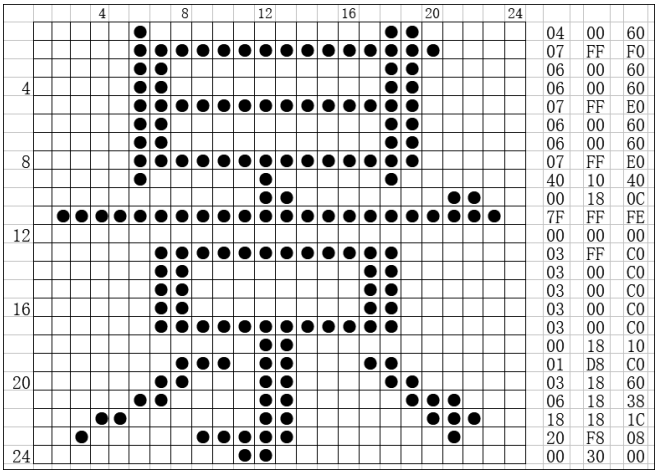


图 3.9　“景”字的 24×24 点阵

矢量方式存储的是描述汉字字形的轮廓特征，当要输出汉字时，通过计算机的计算生成所需大小和形状的汉字。矢量化字形与最终文字显示的大小、分辨率无关，因此产生高质量的汉字输出。

点阵方式和矢量方式的区别：前者编码、存储方式简单，无须转换直接输出，但字形放大后产生的效果差；矢量方式特点正好与前者相反。

3.3 多媒体数据的表示

媒体是指承载或传递信息的载体。在日常生活、工作当中，报纸、图书、杂志、广播、电影、电视均是媒体，都以它们各自的媒体形式传播信息。它们中有的以文字作为媒体，有的以声音作为媒体，有的以图像作为媒体，还有的将文字、图像、声音结合在一起作为媒体。同样的信息，在不同领域中采用的媒体形式也可以不同。

在信息领域中，**多媒体**是指文本、图形、图像、声音、影像等这些媒体和计算机程序融合在一起形成的信息媒体，运用存储与再现技术得到的计算机中的数字信息。

多媒体信息处理是指对文字、声音、图形、图像等多媒体信息在计算机运算下的综合处理。在传统媒体中，声音、图形、图像等媒体几乎都是以模拟信号的方式进行存储和传播的，而在计算机多媒体系统中将以数字的形式对这些信息进行存储、处理和传播。

3.3.1 图形与图像

在计算机科学中，**图形**(Graphics)和**图像**(Image)两个概念是有区别的。

图形(见图 3.10(a))一般指用计算机软件绘制的由直线、圆、圆弧、任意曲线等图元组成的画面，以矢量图形文件形式存储。矢量文件中存储的是一组描述各个图元的大小、位置、形状、颜色、维数等属性的指令集合，通过相应的绘图软件读取这些指令可将其转换为输出设备上显示的图形。因此，矢量图文件的最大优点是对图形中的各个图元进行缩放、移动、旋转而不失真，且占用的存储空间小。图形多用于计算机辅助设计(CAD)、三维动画创作等。

(a) 各种线条画的图形

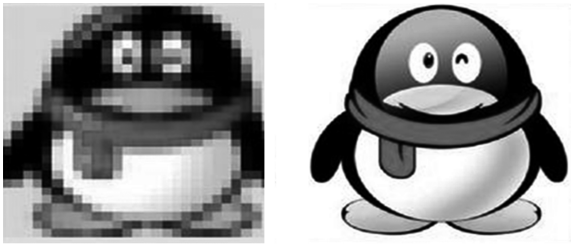
(b) 数码照相机拍的图像

图 3.10 图形与图像

图像(见图 3.10(b))则是指由输入设备(如扫描仪、数码相机等)捕捉的实际场景画面，经数字化后以位图形式存储的画面。位图文件中存储的是构成图像的每个像素点的亮度、颜色，位图文件的大小与分辨率和色彩的颜色种类有关，放大、缩小会失真，占用的空间比矢量文件大。

图像有多种分类方法，按照图像数据的表示方法，主要分为位图图像和矢量图像。位图图像也称点阵图像或栅格图像，是由像素(图片元素)的单个点组成的。这些点可以进

行不同的排列和染色以构成图样，其存储形式就像前面介绍汉字字形码一样（见图 3.9），每像素占一个小方格存储，用一个二进制位表示，像素的颜色也是用若干个二进制位表示（以字节为单位，如 1 字节可以表示 256 种颜色，常说的真彩颜色是用 3 字节表示颜色），然后把这些点位像素数据存储在计算机中。位图的缺点和点阵汉字一样，放大时会失真。数码照相机拍摄的照片就是数字化后存储在照相机的存储卡中。矢量图和矢量汉字一样，存储的不是像素数据，而是存放各种图形元素（如点、线、圆、矩形等对象数据），包括颜色、形状、轮廓、大小等数据。因此，矢量图形数据量与图像的复杂程度有关，与图像的分辨率无关，所以图像放大时不会失真。图 3.11 为放大后的位图与矢量图。



(a) 位图　　(b) 矢量图

图 3.11　放大后的位图与矢量图

由图元/线条组成的图形，如工程图、三维造型或艺术字等，特别适合用于文字设计、图案设计、版式设计、标志设计、工艺美术设计、插图等，较多用在工程作图软件，如 AutoCAD。位图可以表现的色彩比较多，表现力强、细腻、层次多、细节多，能够制作出颜色和色调变化丰富的图像，可以逼真地表现自然界的景观，特别适合制作由无规律的像素点组成的图像，如风景、人物、山水等，一般较多用 Photoshop 软件处理。但位图图像在缩放和旋转时会产生失真现象，同时文件较大，对内存和硬盘空间容量的需求较高。总之，图形、图像技术是相互关联的，把图形处理技术和图像处理技术相结合可以使视觉效果和质量更加完善。

1. 图像的数字化

现实中的图像都是模拟信号，是不能直接用计算机进行处理的，需要转换成用一系列的 0、1 数字所表示的数字图像，这个过程就是将模拟图像数字化，通常用采样、量化和编码方法来解决。

1）**采样**

采样就是将二维空间上连续的图像转换成离散点的过程，采样的实质就是用多少个像素（Pixels）点来描述这一幅图像，称为图像的分辨率，用“列数×行数”表示，分辨率越高，图像越清晰，存储容量也就越大。

2）**量化**

量化是在图像离散化后，将表示图像色彩浓淡的连续变化值离散化为整数值的过程。把量化时所确定的整数值取值个数称为量化级数，表示量化的色彩值（或亮度）所需的二进制位数称为量化字长。一般可用 8 位、16 位、24 位或以上来表示图像的颜色，24 位可

以表示 $2^{24}=16\ 777\ 216$ 种颜色，称为**真彩色**。量化字长越大，越能真实地反映原有图像的颜色，但得到的数字图像的存储容量也越大。

在多媒体计算机中，图像的色彩值称为图像的颜色深度，有 3 种表示方式。

(1) 黑白图：图像的颜色深度用一个二进制位 1 和 0 分别表示纯白、纯黑两种状态。

(2) 灰度图：图像的颜色深度用一字节表示，灰度级别就有 256($2^8=256$)级，通过调整黑白两色的程度(称为**灰度**)来显示单色图像。

(3) RGB(Red、Green、Blue，三原色)：24 位真彩色显示时，由红、绿、蓝三原色通过不同的强度混合而成。强度分为 256 级，占 24 位(3 字节)，就构成了 $2^{24}=16\ 777\ 216$ 种“真彩色”图像。

3) **编码**

将采样和量化的数据转换成为二进制数的表示形式。

图像的分辨率和像素位的颜色深度决定图像文件的大小，即图像的质量，其计算公式为

$$\text{列数}\times\text{行数}\times\text{颜色深度}\div 8=\text{图像字节数}$$

例 3.22 要表示一个分辨率为 1024×1024 的 24 位“真彩色”图像，则图像大小为

$$1024\times1024\times24\div8\text{b}=3\text{MB}$$

这仅仅是一张数字化后的图像存储容量，可见其数据量巨大，无论是存储还是传输都不方便，必须采用压缩技术解决。

2. 图形的存储

图形文件是一种以数学的方式对图形进行描述，在原制作软件和库文件的环境下，通过计算机可以任意缩放图形，又不损失其细节的文件。

矢量图形通常由直线、圆、弧线等各种矢量对象组成，图形中的每个对象都是一个独立的实体，都独立地定义了各自的色彩、形状、轮廓、尺寸以及位置等属性，通过数学方法将这些矢量对象组合而成的图形，以一组指令的形式保存。对于图像中更为复杂的曲面、光照、材质等效果，同样也是使用这种形式保存。

矢量图形的特性如下。

(1) 由于矢量图形把线段、形状及文本定义为数学方程，因此矢量图形与分辨率无关，对图形的展现更为细致、真实。它可以将图形的尺寸任意改变而不会导致失真和降低图形的质量。这是矢量图形一个非常有用的特点。

(2) 由于矢量图形与分辨率无关，因此矢量图形可以自动适应输出设备的最大分辨率。以打印机作为输出设备时，打印机把矢量图形的数学方程变成打印机的像素，无论打印的图形有多大，图形看上去都十分均匀清晰。

(3) 由于矢量图形是以数学方法描述的图形，它并不存储图形的每一点，而只存储图形内容的轮廓部分，因此矢量图形的存储空间较位图图像的存储空间要小得多。

(4) 在矢量图形中，文件大小取决于图形中所包含对象的数量和复杂程度，因此矢量图形文件大小与输出图形的大小几乎没有关系，这一点与位图图像正好相反。

(5) 在矢量图形中可以只编辑其中某个对象而不影响图形中的其他对象。矢量图形

中的对象可以互相覆盖而不会互相影响。

3. 常见图形和图像文件格式

1) BMP(.bmp)

BMP(Bitmap)是一种与设备无关的图像文件格式,是 Windows 环境中经常使用的一种位图格式。其特点是包含的图像信息较丰富,几乎不进行压缩,故文件占用空间较大。大多数图像处理软件都支持此格式。

2) JPEG(.jpg 或.jpeg)

JPEG 是由联合照片专家组(Joint Photographic Experts Group)开发的。它既是一种文件格式,又是一种压缩技术。JPEG 作为一种灵活的格式,具有调节图像质量的功能,允许用不同的压缩比例压缩图像文件。它用有损压缩方式去除冗余的图像和彩色数据,在获得极高压缩率的同时也能展现十分丰富生动的图像。JPEG 应用非常广泛,大多数图像处理软件均支持此格式。

JPEG 2000 作为 JPEG 的升级版,其压缩率比 JPEG 高约 30%,同时支持有损和无损压缩。JPEG 2000 格式有一个极其重要的特征是它能实现渐进传输,即先传输图像的轮廓,再逐步传输数据,不断提高图像质量,让图像由朦胧到清晰显示。此外,JPEG 2000 还支持“感兴趣区域”特性,可以任意指定影像上感兴趣区域的压缩质量,还可以选择指定的部分先解压缩。JPEG 2000 和 JPEG 相比优势明显,且向下兼容,因此可取代传统的 JPEG 格式。

3) GIF(.gif)

GIF(Graphic Interchange Format)是 CompuServe 公司开发的图像文件格式,采用了压缩存储技术。GIF 同时支持线图、灰度和索引图像,但最多支持 256 种色彩的图像。其特点是压缩比高、磁盘空间占用较少、下载速度快,可以存储简单的动画,被广泛用于 Internet 中。

4) PNG(.png)

PNG(Portable Network Graphics,可移植的网络图像)是流式图像文件。压缩比高,并且是无损压缩,适合在网络中传播,但是它不支持动画功能。

5) WMF(.wmf)

WMF(Windows Metafile Format)是 Windows 中常见的一种图元文件格式,它属于矢量图形,其特点是文件非常小,可以任意缩放而不影响图像质量,整个图形常由各个独立的组成部分拼接而成,但其图形往往较粗糙。Windows 中许多剪贴画图像以该格式存储,广泛应用于桌面出版印刷领域。

6) SVG(.svg)

SVG(Scalable Vector Graphics)是一种基于 XML,由 World Wide Web Consortium (W3C)开发的,开放、标准的矢量图形文件。它可以使图像在改变尺寸的情况下,图形质量不会有损失。与 JPEG 和 GIF 图像比起来,其尺寸更小,可压缩性更强,方便下载。文本在 SVG 图像中保留可编辑和可搜寻的状态,可用任何文字处理工具打开 SVG 图像,直接用代码来描绘图像,也可以通过改变部分代码使图像具有交互功能,并随时插入

HTML 中通过浏览器观看。它还可以在任何分辨率下被高质量地打印，非常适用于设计高分辨率的 Web 图形页面，是目前比较流行的图像文件格式。

3.3.2 音频

声音是通过空气振动发出的，在介质中传播时，实际上是一种波，称为**声波**，通常用模拟波的方式表示。声波的物理元素包括振幅、频率：振幅决定了音量，频率决定了音调。**音频**是声音的信息表示，通常指在 20～20 000Hz 频率范围的声音信号，是连续变化的模拟信号，而计算机只能处理数字信号，必须把它转换成数字信号计算机才能处理，这就是音频的数字化，如图 3.12 所示。

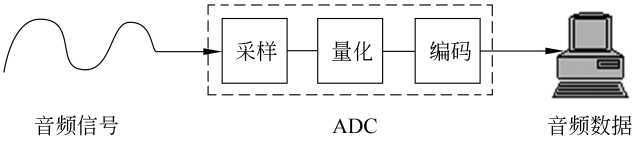


图 3.12 音频的数字化过程

1. 音频的数字化

音频的数字化过程要经过采样、量化和编码。采样和量化的过程可由模数转换器(Analog-to-Digital Converter，ADC)实现。ADC 以固定的频率采样、量化，经采样和量化的声音信号再经编码后就成为数字音频信号，以数字声波文件形式保存在计算机存储介质中。若要将数字音频输出，则通过数模转换器(Digital-to-Analog Converter，DAC)将数字信号转换成原始的模拟信号即可。

在数字化过程中，**采样频率**、**采样精度**(量化位数)和**声道数**是非常重要的指标。采样频率是指每秒要采集多少个声音样本，频率越高，声音的保真度越高，声音的质量就越好。一般使用的频率为 11.025kHz(语音效果)、22.05kHz(音乐效果)、44.1kHz(高保真效果)。采样精度是指每个声音样本需要用多少个二进制位来表示，它反映声音波形幅度的精度，一般分为 8 位采样、16 位采样。样本位数的多少影响声音的质量，位数越多，声音的质量就越好。声道数是指使用的声音通道个数，用来表明声音记录是产生一个波形还是两个波形，即通常所说的单声道、立体声。图 3.13 为采用 16 位和 24 位的音频采样。

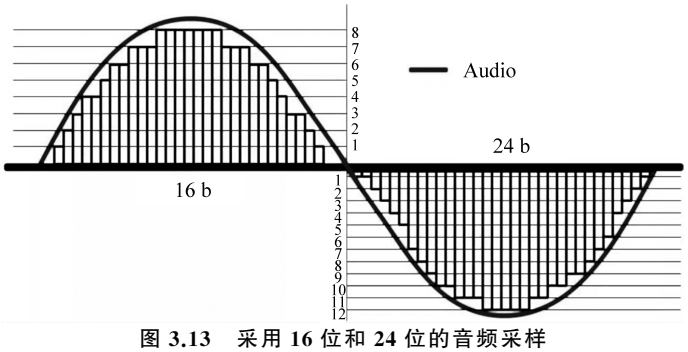


图 3.13 采用 16 位和 24 位的音频采样