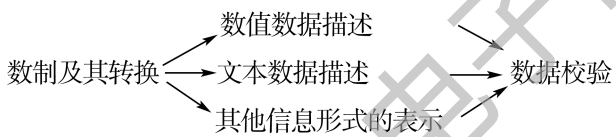
# 第 2 章 数码系统

## [单元概述]

现实生活中，人们对信息的描述已经有了广泛而深入的接触，比如各种费用的表达与计算、时间的表达与计量、图像的画图记录和生活信息的文字记录等。然而，在计算机世界中，由于计算机只能存储二进制信息，所以必须把生活中的信息用二进制信息来描述。换句话来说，要把生活中常用的数字系统变成二进制信息来描述，还要把其他的图形、文字、声音等信息也用二进制信息进行描述。

**本章线索：**




## [教学重点与难点]

**重点：**

(1)进位计数制及其转换。

(2)真值和机器数。

(3)有符号数的表示。

(4)浮点数的表示。

(5)BCD 码。

(6)文本数据的描述。

(7)校验码。

**难点：**

有符号数的表示(原码、反码、补码)。

**学习建议：**

学习本章前应复习相关课程内容，了解数制转换相关信息，并通过 Debug 等实验进一步加强对本章内容的理解。

# 2.1　数制及其转换

数制就是计数的方式和方法，分为进位计数制和非进位计数制。进位计数制如二进制、十进制、十六进制等，非进位计数制如刻痕计数和结绳计数等。非进位计数制本书不做详细的讨论，进制计数制是今后研究的重点，本章中的数制都是指进位计数制。

## 2.1.1　进位计数制中的重要概念

对于 R 进制数有两个重要的概念，基数和位权。

(1)基数(Radix)。R 进制中只允许出现 0，1，2，…，R－1 共 R 个数码，数码既可以是数也可以是码(如字母)，则 R 就称为 R 进制的基数。

(2)位权(Weight)。R 进制中，不同位置的 1 所表示的值不同，位号为 i 的 1 表示 $R^i$，$R^i$称为位权。位号 i 的确定是以小数点为基准，向左分别为 0，1，2……，向右分别为－1，－2，－3，－4，……

(3)进位规则。进位规则不是数制本身体现的，而是在进行计算过程中体现的。进位规则是指何种情况下向高位进位。R 进制中，对位加法满 R 就应向高位进位，即逢 R 进一。

## 2.1.2　几种常见的数制

(1)十进制。十进制数的基数 R＝10，共有 0，1，2，3，4，5，6，7，8，9 十个数码，各位的权值为 10 的幂，进位规则是逢十进一。任一十进制数的多项式表示法为：

$$(N)_D = d_{n-1}10^{n-1} + d_{n-2}10^{n-2} + \cdots + d_1 10^1 + d_0 10^0 + d_{-1}10^{-1} + d_{-2}10^{-2} + \cdots + d_{-m}10^{-m} = \sum_{i=-m}^{n-1} di\ 10^i$$

(2)二进制。二进制数的基数 R＝2，共有 0 和 1 两个数码，各位的权值为 2 的幂，进位规则是逢二进一。二进制是计算机领域的数制，任一二进制数的多项式表示法为：

$$(N)_B = d_{n-1}2^{n-1} + d_{n-2}2^{n-2} + \cdots + d_1 2^1 + d_0 2^0 + d_{-1}2^{-1} + d_{-2}2^{-2} + \cdots + d_{-m}2^{-m} = \sum_{i=-m}^{n-1} di\,2^i$$

(3)八进制。八进制数的基数 R＝8，共有 0，1，2，3，4，5，6，7 八个数码，各位的权值为 8 的幂，进位规则是逢八进一。八进制数主要在书写程序时使用，任一八进制数的多项式表示法为：

$$(N)_O = d_{n-1}8^{n-1} + d_{n-2}8^{n-2} + \cdots + d_1 8^1 + d_0 8^0 + d_{-1}8^{-1} + d_{-2}8^{-2} + \cdots + d_{-m}8^{-m} = \sum_{i=-m}^{n-1} di\,8^i$$

(4)十六进制。十六进制数的基数 R＝16，共有 0，1，2，3，4，5，6，7，8，9，A，B，C，D，E，F 十六个数码，各位的权值为 16 的幂，进位规则是逢十六进一。十六进制数的可以解决二进制数书写过长的问题，这样可以方便表示二进制，任一十六进制数的多项式表示法为：

$$(N)_H = d_{n-1}16^{n-1} + d_{n-2}16^{n-2} + \cdots + d_1 16^1 + d_0 16^0 + d_{-1}16^{-1} + d_{-2}16^{-2} + \cdots + d_{-m}16^{-m} = \sum_{i=-m}^{n-1} di\ 16^i$$

在计算机系统中，二进制主要用于计算机的内部数据处理，八进制和十六进制主要用于书

写程序，十进制主要用于运算结果的输出。常用进制之间的关系如表 2.1 所示。

表 2.1 常用进制对照表

| 十进制(D) | 二进制(B) | 八进制(O) | 十六进制(H) |
|---|---|---|---|
| 0 | 0000 | 0 | 0 |
| 1 | 0001 | 1 | 1 |
| 2 | 0010 | 2 | 2 |
| 3 | 0011 | 3 | 3 |
| 4 | 0100 | 4 | 4 |
| 5 | 0101 | 5 | 5 |
| 6 | 0110 | 6 | 6 |
| 7 | 0111 | 7 | 7 |
| 8 | 1000 | 10 | 8 |
| 9 | 1001 | 11 | 9 |
| 10 | 1010 | 12 | A |
| 11 | 1011 | 13 | B |
| 12 | 1100 | 14 | C |
| 13 | 1101 | 15 | D |
| 14 | 1110 | 16 | E |
| 15 | 1111 | 17 | F |

## 2.1.3 数制间的相互转换

(1)R 进制转换为十进制。

方法：按位权展开相加，可表示为：

$(a_{n-1}a_{n-2}\cdots a_1a_0.a_{-1}a_{-2}\cdots a_{-m})_R$

$=a_{n-1}\times R^{n-1}+a_{n-2}\times R^{n-2}+\cdots+a_i\times R^i+\cdots+a_1\times R^1+a_0\times R^0+a_{-1}\times R^{-1}+a_{-2}\times R^{-2}+\cdots+a_{-m}\times R^{-m}$(其中 $0\leqslant a_i\leqslant R-1$)

**例 2.1**：将二进制数 101.11 转换为十进制数。

**解**：$(101.11)_2=1\times2^2+0\times2^1+1\times2^0+1\times2^{-1}+1\times2^{-2}=(4+0+1+0.5+0.25)_{10}=(5.75)_{10}$

**例 2.2**：将十六进制数 1F.0B 转换为十进制数。

**解**：$(1F.0B)_{16}=1\times16^1+F\times16^0+0\times16^{-1}+B\times16^{-2}=(31.046875)_{10}$

**注意**：R 进制中的按权展开式是唯一的。

(2)十进制转换为 R 进制。

方法：

① 整数小数分开转换。

② 整数除以 R 取余，商为 0 为止，反向书写。

③ 小数乘以 R 取整，小数部分为 0 或取到足够精度为止，正向书写。

在将十进制转换为二进制时，考虑到二进制表示的特殊性，可以采用一些特殊的方法进行计算。

**例 2.3**：将$(3/32)_{10}$转换成二进制。

**解**：方法一：$(3/32)_{10}=1/32+2/32=1/32+1/16=(0.00011)_2$

方法二：$(3/32)_{10}=3\times2^{-5}=(11)_2\times(0.00001)_2=(0.00011)_2$

(3)二进制转换为十六进制。

方法：

① 分组转换，即以小数点为基准，向两侧每四位二进制划分为一组，不足的在离小数点远的一端补上 0，凑够 4 位。

② 将每一组二进制转换成对应的十六进制数码。

**例 2.4**：将$(1010.01)_2$转换成十六进制。

$(1010.01)_2=(1010\ .\ 0100)=(A.4)_{16}$

(4)十六进制转换为二进制。

方法：

将每一个十六进制数码转换为对应的二进制数。

### 2.1.4　二数制中的一些常用表达

常见的数值表示及用法如下。

(1)$(100\cdots0)_2=2^N$：表示 1 后面有 N 个 0 的二进制整数。

(2)$(111\cdots1)_2=2^{N+1}-1$：表示共有 N+1 个 1 的二进制整数。

(3)$(0.0\cdots01)_2=2^{-N}$：表示小数点后共有 N−1 个 0 的二进制纯小数。

(4)$(0.1\cdots11)_2=1-2^{-N}$：表示小数点后共有 N 个 1 的二进制纯小数。

(5)$(111\cdots1)_2-(X_1X_2\cdots X_N)_2=(\overline{X}_1\overline{X}_2\cdots\overline{X}_N)_2$：表示共有 N 个 1 的二进制整数减去一个 N 位的二进制整数，结果是将减数按位取反。

(6)$(0.111\cdots1)_2-(0.X_1X_2\cdots X_N)_2=(0.\overline{X}_1\overline{X}_2\cdots\overline{X}_N)_2$：表示共有 N 个 1 的二进制纯小数减去一个小数部分有 N 位的二进制纯小数，结果是将减数的小数部分按位取反。

(7)二进制数左移 K 位，相当于这个数乘以 $2^K$。

(8)二进制数右移 K 位，相当于这个数除以 $2^K$。

## 2.2　数值数据描述

### 2.2.1　无符号数和有符号数

(1)无符号数(unsigned)。无符号数的数据与数值相等，数据本身的 N+1 个二进制比特位全部用来表示其数值，没有符号位。也就是说，无符号数都为绝对数，所以不需要符号位来表示其正负信息。无符号数的全部比特位都用来表示数值信息，表示范围为$[0,2^{N+1}-1]$。

在计算机领域，无符号数通常用于表示地址，作为计数器等用途。

(2)有符号数(signed)。有符号数的数据分为符号和数值两个部分，N+1 位有符号数的二进制比特位最左一位用于表示符号，其余 N 位用于表示数值。

有符号数中涉及两个概念：真值和机器数。真值是带符号的数，即平时生活中数的表达方式。机器数是有符号数的符号数值化后在计算机存储中的数，是一个 0，1 序列。

Ⅰ. 符号的数值化："+"用"0"表示，"−"用"1"表示。

Ⅱ. 数值的编码：多种方案，如原码，反码，补码，移码等。

Ⅲ. 小数点的表示：隐藏，不表示出来，只要事先约定好即可。

例如，真值+0.5，如何把它存储在计算机中？

① 把 0.5 变成二进制的$(0.5)_{10}=(0.1)_2=(.1)_2$(小数点前面的 0 对于纯小数是没有任何意义的)。

② 解决第Ⅰ个问题,加上符号“+”,再把“+”变成“0”,即$(+.1)_2=(0.1)_2$。

③ 待会儿专门介绍编码,这里先假设编码没有问题,那第Ⅱ个问题已经解决了。

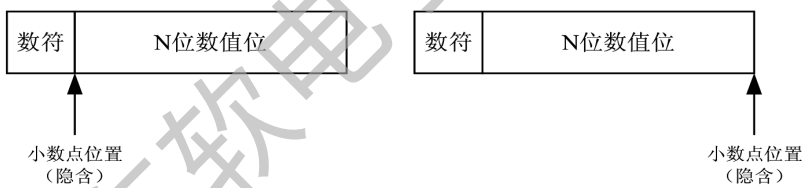
④ 第Ⅲ个问题,小数点可以不表示,(0.1)就变成了最终的编码 01,其中第一位为符号位,表示数为正,第二位表示数的大小$(0.5)_{10}$。

这样,真值就转换为机器数了。

机器数
- 无符号数
- 有符号数
  - 定点数
    - 定点小数
    - 定点整数
  - 浮点数

## 2.2.2　定点数和浮点数

(1)定点数的表示。小数点固定在某一位置的数为定点数,有两种格式,如图 2.1 所示。当小数点位于数符和第一数值位之间时,计算机内的数为纯小数;当小数点位于数值位最后时,计算机内的数为纯整数。采用定点数的计算机叫做定点机,数值部分的位数 N 决定了定点机中数的表示范围。在定点机中,由于小数点的位置固定不变,故当计算机处理的数不是纯小数或纯整数时,必须乘上一个比例因子,否则会产生“溢出”。




(a)定点小数格式　　(b)定点整数格式

图 2.1　定点数的表示形式

(2)浮点数的表示。实际上计算机中处理的数不一定是纯小数或纯整数,因此都不能直接用定点小数或定点整数表示,所以要用浮点数表示。浮点数即小数点的位置可以浮动的数,如$425.34=4253.4\times10^{-1}=4.2534\times10^{+2}$,还可以写成$0.42534\times10^{+3}$形式。显然,这里小数点的位置是变化的,但因为分别乘上了不同的 10 的方幂,故其值不变。通常,浮点数被表示成$X=M_x\times R^{Ex}$,式中$M_x$为尾数(可正可负),$E_x$为阶码(可正可负),R 是基数(或基值)。在计算机中,基数可取 2,4,8 或 16 等。

以基数 R=2 为例,二进制数 N 可写成下列不同形式(阶码采用 3 位二进制数表示):

$N=10.0101=0.100101\times2^{+010}=1.00101\times2^{+001}=1001.01\times2^{-010}=0.00100101\times2^{+100}$。

为了提高数据精度以及便于浮点数的比较,在计算机中规定浮点数的尾数用纯小数形式,故上例中$0.100101\times2^{+010}$和$0.00100101\times2^{+100}$形式是可以采用的。此外,将尾数最高位为 1 的浮点数称作规格化数,即$0.100101\times2^{+010}$为浮点数的规格化形式。浮点数表示成规格化形式后,其精度最高。

(3)浮点数的表示形式。浮点数在计算机中的形式如图 2.2 所示,采用这种数据格式的计算机叫做浮点机。

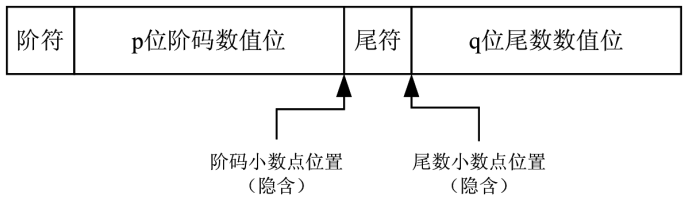



**图 2.2　浮点数的表示形式**

由图 2.2 可见浮点数由阶码和尾数两部分组成。阶码是整数,阶符和阶码的位数 p+1 合起来反映浮点数的表示范围及小数点的实际位置;尾数是小数,其位数反映了浮点数的精度;尾数的符号代表浮点数的正负。

(4)浮点数的规格化。为了提高浮点数的精度,其尾数必须为规格化数。如果不是规格化数,就要通过修改阶码并同时左移或右移尾数的办法,使其变成规格化数。将非规格化数转换成规格化数的过程叫做规格化。对于基数不同的浮点数,因其规格化数的形式不同,规格化过程也不同。

当基数为 2 时,尾数采用带符号的二进制数表示时,尾数最高数值位为 1 的数为规格化数。规格化时,尾数左移一位,阶码减 1,这种规格化叫做向左规格化,简称左规;尾数右移一位,阶码加 1,这种规格化叫做向右规格化,简称右规。

浮点机中基数一旦确定就不再变了,而且基数是隐含的,故不同基数的浮点数其表示形式完全相同。但基数不同,数的表示范围和精度等都受影响。一般来说,基数 R 越大,可表示的浮点数范围越宽,而且所表示的数其个数越多。但 R 越大,浮点数的精度反而下降。如 R=16 的浮点数,因其规格化数的尾数最高三位可能出现零,故与其尾数位数相同的 R=2 的浮点数相比,后者可能比前者多三位精度。

(5)数值范围和数据精度。由于计算机硬件限制只能用有限的二进制比特位来表示数据(假定为 N+1 位),因此对于数值数据而言,计算机的数值数据不可能把所有的数(无论整数、小数还是一般意义的实数)都表示出来,从而存在它能够表示的数值范围和数据精度。这里需要注意的是只要提到二进制都会有这样一个问题,这个二进制数有多少位,这个位数将决定数值的范围和精度。一般情况下,在出现的数据运算中出现的任一个数都要保持这种位数,不能任意减少,更不能增加。关于数值范围和数据精度有以下几个概念。

- 数值范围:一种类型的数据所能表示的最大值和最小值。
- 数据精度:一种类型的数据所能表示的有效数据位的位数。
- 溢出:超出数据表示的范围称为溢出。
- 正溢:定点数中大于所能表示的最大正数。
- 负溢:定点数中小于所能表示的最小负数。

## 2.2.3　定点数的编码方案

假定用 N+1 个二进制比特来表示定点数,最左位为符号位,右 N 位为数值位,定点数的表示法分为原码表示法、反码表示法和补码表示法。

(1)原码表示法。原码表示法把数值的绝对值转换成二进制以后在最高位加上其符号的数

值表示，原数值的符号位“＋”和“－”分别用“0”和“1”表示。

**例 2.5**：二进制数的位数为 N＋1＝8 位时，求＋5 和－5 的原码。

当 N＋1＝8 时，N＝7，即数值用 7 位表示。所以：

$(5)_D=(101)_B=(0000\quad 101)_B$

前面的 4 个 0 是为了保证数据位为 7 位加上去的，再把“＋”号变成“0”放在首位。

$[+5]_{原}=0\quad 0000101$

同理可得：$[-5]_{原}=1\quad 0000101$。

**例 2.6**：二进制数的位数为 N＋1＝8 位时，求＋0.625 和－0.625 的原码。

当 N＋1＝8 时，N＝7，即数值用 7 位表示。所以：

$(0.625)_D=(0.101)_B=(0.101\quad 0000)_B$

在上面的等式里十进制和二进制中小数点前的“0”都是为了符合日常习惯而写上去的，只是占位作用，后面的 4 个 0 是为了保证数据位为 7 位加上去的，再把“＋”号变成“0”替换原来小数点前的占位“0”。

$[+0.625]_{原}=0.\quad 1010000$

同理可得：$[-0.625]_{原}=1.\quad 1010000$。

对于定点整数，原码的定义可表示为：

$$[X]_{原}=\begin{cases}X & (X\geqslant 0)\\ 2^N-X & (X\leqslant 0)\end{cases}$$

对于定点小数原码的定义可表示为：

$$[X]_{原}=\begin{cases}X & (X\geqslant 0)\\ 1-X & (X\leqslant 0)\end{cases}$$

其中需要注意的是 0 的编码有“＋0”和“－0”的区别：$[+0]_{原}=00000000$，$[-0]_{原}=10000000$。也就是说，0 的原码不是唯一的！

原码表示 N＋1 位二进制有符号数，定点整数的表示范围为$[-(2^N-1),+(2^N-1)]$，定点小数的表示范围为$[-(1-2^{-N}),+(1-2^{-N})]$。

(2)反码表示法。反码表示法对数的处理过程与数值的符号有关，如果为正数则其编码与原码相同；如果为负数则把原码的符号位不变，其他位(数值位)取反，即 0 变为 1，1 变为 0。

**例 2.7**：二进制数的位数为 N＋1＝8 位时，求＋5 和－5 的反码。

由例 2.5 可知：$[+5]_{原}=00000101$，$[-5]_{原}=10000101$。

因为＋5 是一个正数，其反码与原码相同，有$[+5]_{反}=00000101$。

因为－5 是一个负数，其反码把原码数值位取反即可，有$[-5]_{反}=11111010$。

**例 2.8**：二进制数的位数为 N＋1＝8 位时，求＋0.625 和－0.625 的反码。

由例 2.6 可知：$[+0.625]_{原}=0.1010000$，$[-0.625]_{原}=1.1010000$。

因为＋0.625 是一个正数，其反码与原码相同，有$[+0.625]_{反}=0.1010000$。

因为－0.625 是一个负数，其反码把原码数值位取反即可，有$[-0.625]_{反}=1.0101111$。

对于定点整数的反码表示可定义为：

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

对于定点小数的反码表示可定义为：

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

其中 0 的编码不一致的问题仍然存在，其编码为：$[+0]_{反}=00000000$，$[-0]_{反}=11111111$。0 的反码不是唯一的！

反码表示 N+1 位二进制有符号数，定点整数的表示范围为$[-(2^N-1),+(2^N-1)]$，定点小数的表示范围为$[-(1-2^{-N}),+(1-2^{-N})]$。

(3)补码表示法。补码表示法对数的处理过程与数值的符号有关，如果为正数则其编码与原码相同，如果为负数则在反码的最末位加上 1，计算所得到的编码即为其补码。

**例 2.9**：二进制数的位数为 N+1=8 位时，求+5 和−5 的补码。

由例 2.5 可知：$[+5]_{原}=00000101$，$[-5]_{原}=10000101$。

因为+5 是一个正数，其补码与原码相同，有$[+5]_{补}=00000101$。

负数的补码是在反码的最末位加上 1，由例 2.7 可知$[-5]_{反}=11111010$，而 11111010+1=11111011，可以得到$[-5]_{补}=11111011$。

**例 2.10**：二进制数的位数为 N+1=8 位时，求+0.625 和−0.625 的补码。

由例 2.6 可知：$[+0.625]_{原}=0.1010000$，$[-0.625]_{原}=1.1010000$。

因为+0.625 是一个正数，其补码与原码相同，有$[+0.625]_{补}=0.1010000$。

负数的补码是在反码的最末位加上 1，由例 2.8 可知$[-0.625]_{反}=1.0101111$，而 1.0101111+0.0000001=1.0110000，可以得到$[-0.625]_{补}=1.0110000$。

对于定点整数补码的表示可定义为：

$$[X]_{补}=\begin{cases}X & (X\geqslant 0)\\ 2^{N+1}+X & (X\leqslant 0)\end{cases}$$

对于定点小数补码的表示可定义为：

$$[X]_{补}=\begin{cases}X & (X\geqslant 0)\\ 2+X & (X\leqslant 0)\end{cases}$$

对 0 编码时，$[+0]_{补}=00000000$，$[-0]_{补}=00000000$。由此可见，0 的补码是唯一的！

补码表示的 N+1 位二进制有符号数，定点整数的表示范围为$[-2^N,+(2^N-1)]$，定点小数的表示范围为$[-1,+(1-2^{-N})]$。

① 变形补码。变形补码的实质是双符号位补码，求法很简单，只需将补码的符号位多写一位即可。

**例 2.11**：二进制数的位数为 N+1=8 位时，求+5 和−5 的变形补码。

$[+5]_{原}=\ \ 0\ \ 0000101$

$[+5]_{反}=\ \ 0\ \ 0000101$

$[+5]_{补}=\ \ 0\ \ 0000101$

$[+5]_{变补}=00\ \ 0000101$

同理可得：

$[-5]_{原}=\ \ 1\ \ 0000101$

$[-5]_{反}=\ \ 1\ \ 1111010$

$[-5]_{补}=\ \ 1\ \ 1111011$

$[-5]_{变补}=11\ \ 1111011$

另外：$[+0.101]_{变补}=00\ \ .1010000$，$[-0.101]_{变补}=11\ \ .0110000$。

变形补码主要用于方便判断运算是否溢出！

② 移码。将一个数的补码的符号位取反所得的编码即为其移码。移码只用于表示定点整数。

**例 2.12**：二进制数的位数为 N+1=8 位时，求+5 和−5 的移码。

$[+5]_{补}=0\quad 0000101$

$[+5]_{移}=1\quad 0000101$

同理可得：

$[-5]_{原}=1\quad 0000101$

$[-5]_{反}=1\quad 1111010$

$[-5]_{补}=1\quad 1111011$，

$[-5]_{移}=0\quad 1111011$。

移码的表示方法可定义为：$[X]_{移}=2^N+X$。

移码中 0 的编码是：$[+0]_{移}=10000000$，$[-0]_{移}=10000000$。0 的移码表示是唯一的！

移码表示的 N+1 位二进制有符号数的表示范围为$[0,+(2^{N+1}-1)]$。

移码表示的整数可直接比较大小，通常用于表示浮点数的阶码，便于浮点数运算。

## 2.2.4 浮点数的编码方案

一般情况下，R 进制中表示一个实数 X 为：$X=M_x\times R^{E_x}$，其中 $M_x$ 称为尾数，通常为带符号的小数，$E_x$ 称为阶码，通常为带符号的整数。所以，表示一个实数 X，只需要将尾数和阶码两部分表示出即可(R 可以隐含)，由于尾数和阶码均为带符号的数，因此一个完整的实数 X 在计算机内的表示涉及到四部分：阶符(阶码的符号)、阶码值、尾符(尾数的符号)、尾数值。阶符和尾符通常只占一位，而阶码值和尾数值可采用不同的定点编码方案。尾数通常决定了浮点数的精度，而阶码通常决定了浮点数的表示范围。

(1)几个重要概念。

- 浮点数上溢：阶码比所能表示的最大正数还要大。
- 浮点数下溢：阶码比所能表示的最小负数还要小。
- 浮点数溢出处理：上溢产生中断，停止处理；下溢强行置机器 0，继续运算。
- 机器 0：尾数为 0，阶码不管为任何值，或者阶码小于它所能表示的最小数，无论尾数为任何值。
- 规格化数：在 R 进制中，要求尾数 $M_x$ 满足 $1/R\leqslant|M_x|<1$，即 R 进制中尾数的第一位为有效位(非零)。

(2)IEEE754 浮点数标准。计算机的浮点数通常采用 IEEE754 标准表示，如表 2.2 所示。

**表 2.2　　浮点数 IEEE754 标准**

| IEEE754 标准 | 符号位 | 阶码 | 尾数 | 偏移量 | 是否采用隐藏位技术 |
|---|---|---|---|---|---|
| 单精度 32 位(短实数) | 1 | 8 | 23 | 127 | 采用 |
| 双精度 64 位(长实数) | 1 | 11 | 52 | 1023 | 采用 |
| 临时实数 80 位 | 1 | 15 | 64 | 16383 | 不采用 |

浮点数真值$=(-1)^{尾符}\times(1+尾数)\times2^{(阶码-偏移量)}$

**例 2.13**：已知一单精度浮点数的机器码为：42C88000H，求其真值。

**解**：$42C88000H=(0100,0010,1100,1000,1000,0000,0000,0000)_2$

$=(0,10000101,10010001000000000000000)_2$

∴尾符为 0，尾数为 0.10010001000000000000000，阶码为 10000101

∴真值＝$(-1)^0\times(1+0.10010001000000000000000)\times2^{(10000101-1111111)}=1.10010001\times2^6=(100.25)_{10}$。

### 2.2.5　十进制数的编码

(1)BCD 码。用二进制代码来为十进制数编码，即是用若干位二进制的编码来表示十进制的每一个数码 0～9。一般用四位二进制来表示一位十进制，这样，就可以在只识别二进制的计算机中利用十进制来表示数值。

如要表示十进制的 678，只需表示出 6、7、8 三个数码即可，怎么表示呢？计算机只识别二进制，所以要用二进制来表示这些十进制数码，这就是 BCD 码的基本思想。至于如何用二进制编码来表示十进制数码，方案很多。

(2)BCD 码的编码方案。BCD 码实质都是利用四位二进制来表示一位十进制数码，方案有多种，按其编码是否根据特定的权值构成，分为有权码和无权码。有权码有 8421 码、2421 码和 5421 码等编码方式，通常用得最多的是 8421 码，所以一般 BCD 码如不特殊说明就指 8421 码。

如十进制的 128，用 8421 码的 1 表示为 0001，2 表示为 0010，8 表示为 1000，得$(128)_D=(0001\ \ 0010\ \ 1000)_{BCD}$。

无权码有余 3 码和格雷码等编码方式。余 3 码的编码方式就是在 8421 码的编码上加上 3，做数值计算得到的编码。格雷码的主要特点为任何两组相邻码字中只有一位二进制不同，它有多种编码方案。一种方案为 8421 码加一个前导 0，然后相邻两位异或得到。

几种常见的 BCD 码编码方案的关系如表 2.3 所示。

**表 2.3　　常见 BCD 码编码表**

| 十进制数码 | 8421 码 | 余 3 码 | 格雷码 |
|---|---|---|---|
| 0 | 0000 | 0011 | 0000 |
| 1 | 0001 | 0100 | 0001 |
| 2 | 0010 | 0101 | 0011 |
| 3 | 0011 | 0110 | 0010 |
| 4 | 0100 | 0111 | 0110 |
| 5 | 0101 | 1000 | 0111 |
| 6 | 0110 | 1001 | 0101 |
| 7 | 0111 | 1010 | 0100 |
| 8 | 1000 | 1011 | 1100 |
| 9 | 1001 | 1100 | 1101 |

## 2.3　文本数据描述

文本数据的表示主体是各种符号，主要是 26 个英文字符(A～Z，a～z)，10 个阿拉伯数码(0～9)，标点符号(，．；：‘“（）［］)，以及一些其他专用符号(＋　－　＊　＝　@　＃　$　％　∧)，还有就是汉字符号。

## 2.3.1 ASCII 码

目前使用最广泛的英文字符集及其编码是 ASCII 字符集和 ASCII 码(ASCII 是 American Standard Code for Information Interchange 的缩写),它同时也被国际标准化组织(International Organization for Standardization, ISO)批准为国际标准。

基本的 ASCII 字符集共有 128 个字符,其中有 96 个可打印字符,包括常用的字母、数字、标点符号等,另外还有 32 个控制字符。标准 ASCII 码使用 7 位二进制数对字符进行编码,对应的 ISO 标准为 ISO646 标准。表 2.4 中列出了基本 ASCII 字符集及其编码。

表 2.4 ASCII 码表

| ASCII 值 | 控制字符 | ASCII 值 | 控制字符 | ASCII 值 | 控制字符 | ASCII 值 | 控制字符 |
|---|---|---|---|---|---|---|---|
| 0 | NUT | 32 | (space) | 64 | @ | 96 | 、 |
| 1 | SOH | 33 | ! | 65 | A | 97 | a |
| 2 | STX | 34 | ” | 66 | B | 98 | b |
| 3 | ETX | 35 | # | 67 | C | 99 | c |
| 4 | EOT | 36 | $ | 68 | D | 100 | d |
| 5 | ENQ | 37 | % | 69 | E | 101 | e |
| 6 | ACK | 38 | & | 70 | F | 102 | f |
| 7 | BEL | 39 | , | 71 | G | 103 | g |
| 8 | BS | 40 | ( | 72 | H | 104 | h |
| 9 | HT | 41 | ) | 73 | I | 105 | i |
| 10 | LF | 42 | * | 74 | J | 106 | j |
| 11 | VT | 43 | + | 75 | K | 107 | k |
| 12 | FF | 44 | , | 76 | L | 108 | l |
| 13 | CR | 45 | — | 77 | M | 109 | m |
| 14 | SO | 46 | . | 78 | N | 110 | n |
| 15 | SI | 47 | / | 79 | O | 111 | o |
| 16 | DLE | 48 | 0 | 80 | P | 112 | p |
| 17 | DCI | 49 | 1 | 81 | Q | 113 | q |
| 18 | DC2 | 50 | 2 | 82 | R | 114 | r |
| 19 | DC3 | 51 | 3 | 83 | X | 115 | s |
| 20 | DC4 | 52 | 4 | 84 | T | 116 | t |
| 21 | NAK | 53 | 5 | 85 | U | 117 | u |
| 22 | SYN | 54 | 6 | 86 | V | 118 | v |
| 23 | TB | 55 | 7 | 87 | W | 119 | w |
| 24 | CAN | 56 | 8 | 88 | X | 120 | x |
| 25 | EM | 57 | 9 | 89 | Y | 121 | y |
| 26 | SUB | 58 | : | 90 | Z | 122 | z |
| 27 | ESC | 59 | ; | 91 | [ | 123 | { |
| 28 | FS | 60 | < | 92 | / | 124 | \| |
| 29 | GS | 61 | = | 93 | ] | 125 | } |
| 30 | RS | 62 | > | 94 | ∧ | 126 | ~ |
| 31 | US | 63 | ? | 95 | — | 127 | DEL |

字母和数字的 ASCII 码的记忆是非常简单的，只要记住了一个字母或数字的 ASCII 码（例如记住 A 的 ASCII 码为 65，0 的 ASCII 码为 48），知道相应的大小写字母之间差 32，就可以推算出其余字母、数字的 ASCII 码。

虽然标准 ASCII 码是 7 位编码，但由于计算机基本处理单位为字节（1Byte＝8bit），所以一般仍以一个字节来存放一个 ASCII 字符。每一个字节中多余出来的一位（最高位）在计算机内部通常保持为 0（在数据传输时可用作奇偶校验位）。

由于标准 ASCII 字符集字符数目有限，在实际应用中往往无法满足要求。为此，国际标准化组织又制定了 ISO2022 标准，它规定了在保持与 ISO646 兼容的前提下将 ASCII 字符集扩充为 8 位代码的统一方法。ISO 陆续制定了一批适用于不同地区的扩充 ASCII 字符集，每种扩充 ASCII 字符集分别可以扩充 128 个字符，这些扩充字符的编码均为高位为 1 的 8 位代码（即十进制数 128～255），称为扩展 ASCII 码。表 2.5 展示的是最流行的一套扩展 ASCII 字符集和编码。

**表 2.5　　扩展 ASCII 字符集和编码**

| | | |
|---|---|---|
| NUL 空 | VT 垂直制表 | SYN 空转同步 |
| SOH 标题开始 | FF 走纸控制 | ETB 信息组传送结束 |
| STX 正文开始 | CR 回车 | CAN 作废 |
| ETX 正文结束 | SO 移位输出 | EM 纸尽 |
| EOY 传输结束 | SI 移位输入 | SUB 换置 |
| ENQ 询问字符 | DLE 空格 | ESC 换码 |
| ACK 承认 | DC1 设备控制 1 | FS 文字分隔符 |
| BEL 报警 | DC2 设备控制 2 | GS 组分隔符 |
| BS 退一格 | DC3 设备控制 3 | RS 记录分隔符 |
| HT 横向列表 | DC4 设备控制 4 | US 单元分隔符 |
| LF 换行 | NAK 否定 | DEL 删除 |



## 2.3.2　汉字的编码

汉字的编码表示涉及到汉字的输入编码、汉字的存储编码以及汉字的输出编码。汉字的输入编码指的是如何通过键盘上已有的字符组合来代表欲输入的汉字，常见的有字形码（如五笔输入编码）和拼音码（如全拼、双拼等）以及音形结合编码。汉字的存储编码是指汉字在计算机内部以什么编码形式存储。汉字在计算机内部的存储编码表示，即汉字的内码。汉字的输出编码是指如何在显示器和打印机等输出设备上显示汉字的字形，也叫汉字的点阵码。

（1）输入码（外码）。输入码也叫外码，是用来将汉字输入到计算机中的一组键盘符号。英文字母只有 26 个，可以把所有的字符都放到键盘上，而使用这种办法把所有的汉字都放到键盘上，是不可能的。所以汉字系统需要有自己的输入码体系，使汉字与键盘能建立对应关系。目前常用的输入码有拼音码、智能全拼输入法和五笔字型输入法。

（2）交换码。计算机内部处理的信息，都是用二进制代码表示的，汉字也不例外。而二进制代码使用起来是不方便的，于是需要采用信息交换码。我国标准总局 1981 年制定了中华人民

共和国国家标准 GB2312－80《信息交换用汉字编码字符集－基本集》，即国标码。国标码字符集中收集了常用汉字和图形符号 7445 个，其中图形符号 682 个，汉字 6763 个，按照汉字的使用频度分为两级，第一级为常用汉字 3755 个，第二级为次常用汉字 3008 个。为了避开 ASCII 字符中的不可打印字符 0100001－1111110（十六进制为 21－7E），国标码表示汉字的范围为 2121－7E7E（十六进制）。

区位码是国标码的另一种表现形式，把国标 GB2312－80 中的汉字、图形符号组成一个 94×94 的方阵，分为 94 个“区”，每区包含 94 个“位”，其中“区”的序号由 01 至 94，“位”的序号也是从 01 至 94。94 个区中位置总数＝94×94＝8836 个，其中 7445 个汉字和图形字符中的每一个占一个位置后，还剩下 1391 个空位，这 1391 个位置空下来保留备用。所以给定“区”值和“位”值，用四位数字就可以确定一个汉字或图形符号，其中前两位是“区”号，后两位是“位”号。区位码编码的最大优点是没有重码，但由于编码缺少规律，很难记忆。94 个区可以分为五组：

01－15 区：是各种图形符号、制表符和一些主要国家的语言字母，其中 01－09 区为标准符号区，共有 682 个常用符号。

10－15 区：为自定义符号区，可留作用户自己定义。

16－55 区：是一级汉字区，共有 3755 个常用汉字，以拼音为序排列。

56－87 区：是二级汉字区，共有 3008 个次常用汉字，以部首为序排列。

88－94 区：自定义汉字区，可留作用户自己定义。

(3)机内码。根据国标码的规定，每一个汉字都有了确定的二进制代码，但是这个代码在计算机内部处理时会与 ASCII 码发生冲突，为解决这个问题，把国标码的每一个字节的首位置 1。由于 ASCII 码只用 7 位编码，最高位置为 0。所以，这个首位上的“1”就可以作为识别汉字代码的标志，计算机在处理到首位是“1”的码就把它理解为汉字的信息，在处理到首位是“0”的码就把它理解为是 ASCII 码。经过这样处理后的国标码就是机内码。

(4)汉字的字形码。字形码是汉字的输出码，输出汉字时都采用图形方式，无论汉字的笔画多少，每个汉字都可以写在同样大小的方块中。为了能准确地表达汉字的字形，每一个汉字都有相应的字形码，目前大多数汉字系统中都是以点阵的方式来存储和输出汉字的字形。所谓点阵就是将字符（包括汉字图形）看成一个矩形框内一些横竖排列的点的集合，有笔画的位置用黑点表示，没笔画的位置用白点表示。在计算机中用一组二进制数表示点阵，用 0 表示白点，用 1 表示黑点。一般的汉字系统中汉字字形点阵有 16×16，24×24，48×48 几种，点阵越大对每个汉字的修饰作用就越强，打印质量也就越高。通常用 16×16 点阵来显示汉字，每一行上的 16 个点需用两个字节表示，一个 16×16 点阵的汉字字形码需要 2×16＝32 个字节表示，这 32 个字节中的信息是汉字的数字化信息，即汉字字模。

(5)区位码、国标码和汉字的内码。汉字区位码的区号和位号各占一个字节，区号和位号都用十进制表示。汉字的国标码用 4 位十六进制数描述。汉字在计算机内用两个字节存储，即汉字的内码是 16 位二进制，但是为了书写方便还是用十六进制。

汉字的机内码、国际码和区位码之间的关系是：

$(\text{汉字的十进制区号})_D=(\text{国标码的高两位})_H-(20)_H$

$(\text{汉字的十进制位号})_D=(\text{国标码的低两位})_H-(20)_H$

$(\text{汉字的内码})_H=(\text{国标码})_H+(8080)_H$

**例 2.10**：“啊”字的国标码为 3021H（可以参考 GB2312－80 文档）。

它的区位码为 3021H－2020H＝1001H，即它在第 16 区，第 01 位。

它在计算机内部的存储的内码为：3021H＋8080H＝B0A1H。

# 2.4　其他信息形式的表示

计算机内任何形式的信息最终都将转换为二进制数据进行存储，对于较复杂的信息形式都采用各自领域的编码形式，也可以说成定义了不同的数据结构，使顺序的二进制数据不同部分表述不同意义。本节简要介绍图像、音频和视频的一些相关内容。

## 2.4.1　图像的格式简介

图像的存储主要是把图像信息分为两部分，一部分为图像的外部信息，如图像的名称、大小、高度和宽度等等，另一部分为图像的颜色信息，也就是图像的每个点的颜色，大部分图像数据所用空间都花费在此，所以大多数图像格式都采用一定压缩算法来节约空间。

(1)BMP 格式。BMP 是英文 Bitmap(位图)的简写，它是 Windows 操作系统中的标准图像文件格式，能够被多种 Windows 应用程序所支持。随着 Windows 操作系统的流行与丰富的 Windows 应用程序的开发，BMP 位图格式理所当然地被广泛应用。这种格式包含的图像信息较丰富，几乎不进行压缩，但由此导致了它与生俱生来的缺点——占用磁盘空间过大，所以，目前 BMP 在单机上比较流行。

(2)GIF 格式。GIF 是英文 Graphics Interchange Format(图形交换格式)的缩写。顾名思义，这种格式是用来交换图片的。GIF 格式的特点是压缩比高，磁盘空间占用较少，所以这种图像格式迅速得到了广泛的应用。最初的 GIF 只是简单地用来存储单幅静止图像，后来随着技术发展，可以同时存储若干幅静止图像进而形成连续的动画，使之成为当时支持 2D 动画为数不多的格式之一。此外，考虑到网络传输中的实际情况，GIF 图像格式还增加了渐显方式，也就是说，在图像传输过程中，用户可以先看到图像的大致轮廓，然后随着传输过程的继续而逐步看清图像中的细节部分，从而适应了用户从朦胧到清楚的观赏心理。目前 Internet 上大量采用的彩色动画文件多为这种格式的文件。但 GIF 有个小小的缺点，即不能存储超过 256 色的图像。尽管如此，这种格式仍在网络上大行其道，这和 GIF 图像文件短小、下载速度快、可用许多具有同样大小的图像文件组成动画等优势是分不开的。

(3)JPEG 格式。JPEG 也是常见的一种图像格式，JPEG 文件的扩展名为.jpg 或.jpeg，其压缩技术十分先进，它用有损压缩方式去除冗余的图像和彩色数据，获得极高压缩率的同时，能展现十分丰富生动的图像。换句话说，就是可以用最少的磁盘空间得到较好的图像质量。同时，JPEG 还是一种很灵活的格式，具有调节图像质量的功能，允许用不同的压缩比例对这种文件压缩，比如，最高可以把 1.37MB 的 BMP 位图文件压缩至 20.3KB。当然，完全可以在图像质量和文件尺寸之间找到平衡点。由于 JPEG 优异的品质和杰出的表现，它的应用也非常广泛，特别是在网络和光盘读物上，肯定都能找到它的影子。目前各类浏览器均支持 JPEG 这种图像格式，因为 JPEG 格式的文件尺寸较小，下载速度快，使得 Web 页有可能以较短的下载时间提供大量美观的图像，JPEG 同时也就顺理成章地成为网络上最受欢迎的图像格式。

## 2.4.2　有关音频编码

自然界中的声音种类繁多，波形极其复杂，通常采用的是脉冲代码调制编码，即 PCM 编码。PCM 脉冲编码调制是 Pulse Code Modulation 的缩写。PCM 通过抽样、量化、编码三个步骤将连续变化的模拟信号转换为数字编码。

(1)采样率和采样大小。声音其实是一种能量波,因此也有频率和振幅的特征,频率对应于时间轴线,振幅对应于电平轴线。波是无限光滑的,弦线可以看成由无数点组成,由于存储空间是相对有限的,数字编码过程中,必须对弦线的点进行采样。采样的过程就是抽取某点的频率值,很显然,在一秒钟内抽取的点越多,取得的频率信息更丰富。要满足人耳的听觉要求,则需要至少每秒进行 40k 次采样,用 40kHz 表达,这个 40kHz 就是采样率。光有频率信息是不够的,还必须获得该频率的能量值并量化,用于表示信号强度,量化电平数为 2 的整数次幂。采样率和采样大小的值越大,记录的波形更接近原始信号。

(2)有损和无损。根据采样率和采样大小可以得知,相对自然界的信号,音频编码最多只能做到无限接近,至少目前的技术只能这样了。相对于自然界的信号,任何数字音频编码方案都是有损的,因为无法完全还原。在计算机应用中,能够达到最高保真水平的就是 PCM 编码,被广泛用于素材保存及音乐欣赏,CD,DVD 以及常见的 WAV 文件中均有应用。因此,习惯上称 PCM 为无损编码。尽管 PCM 代表了数字音频中最佳的保真水准,并不意味着 PCM 就能够确保信号绝对保真,PCM 也只能做到最大程度的无限接近。一般把 MP3 列入有损音频编码范畴,是相对 PCM 编码的。强调编码的相对有损和无损,是为了说明要做到真正的无损是困难的,就像用数字去表达圆周率,不管精度多高,也只是无限接近,而不是真正等于圆周率的值。

(3)使用音频压缩技术的原因。计算一个 PCM 音频流的码率是一件很轻松的事情,采样率值×采样大小值×声道数,单位为 bps。一个采样率为 44.1KHz,采样大小为 16bit,双声道的 PCM 编码的 WAV 文件,它的数据速率则为 44.1K×16×2=1411.2Kbps。通常所说的 128K 的 MP3,对应的 WAV 的参数,就是 1411.2Kbps,这个参数也被称为数据带宽,它和 ADSL 中的带宽是一个概念。将码率除以 8,就可以得到这个 WAV 的数据速率,即 176.4KB/s。这表示存储一秒钟采样率为44.1KHz,采样大小为 16bit,双声道的 PCM 编码的音频信号,需要 176.4KB 的空间,1 分钟则约为 10.34M,这对大部分用户是不可接受的。要降低磁盘占用,只有 2 种方法,降低采样指标或者压缩。降低指标是不可取的,因此专家们研发了各种压缩方案。由于用途和针对的目标市场不一样,各种音频压缩编码所达到的音质和压缩比都不一样,但是有一点是可以肯定的,他们都被压缩过。

(4)流特征。随着网络的发展,人们对在线收听音乐提出了要求,因此也要求音频文件能够一边读一边播放,而不需要把这个文件全部读出后才可以播放,这样就可以做到不用完全下载就可以实现收听了。也可以做到一边解码一边播放,正是由于这种特征,可以实现在线的直播。

(5)PCM 编码。PCM 编码最大的优点就是音质好,最大的缺点就是体积大。常见的 Audio CD 就采用了 PCM 编码,一张光盘的容量只能容纳 72 分钟的音乐信息。

(6)WAVE。这是一种古老的音频文件格式,由微软开发。WAVE 是一种文件格式,符合(RIFF Resource Interchange File Format)规范。所有的 WAVE 都有一个文件头,这个文件头是音频流的编码参数。WAVE 对音频流的编码没有硬性规定。在 Windows 平台下,基于 PCM 编码的 WAVE 是被支持得最好的音频格式,所有音频软件都能完美支持,由于本身可以达到较高音质的要求,因此,WAVE 也是音乐编辑创作的首选格式,适合保存音乐素材。

### 2.4.3 视频格式中采用的技术

(1)无声时代的 FLC。FLC 是 Autodesk 开发的一种视频格式,仅仅支持 256 色,但支持色彩抖动技术,因此在很多情况下与真彩视频区别不是很大,不支持音频信号,现在看来这种格式已经毫无用处,但在没有真彩显卡没有声卡的 DOS 时代确实是最好的也是唯一的选择。最重要的是,Autodesk 全系列的动画制作软件都提供了对这种格式的支持,包括著名的 3D

StudioX，因此这种格式代表了一个时代的视频编码水平。直到今日，仍旧有不少视频编辑软件可以读取和生成这种格式。

(2)载歌载舞的 AVI。AVI——Audio Video Interleave，即音频视频交叉存取格式。1992 年初，Microsoft 公司推出了 AVI 技术及其应用软件 VFW(Video for Windows)。在 AVI 文件中，运动图像和伴音数据是以交织的方式存储，并独立于硬件设备。这种按交替方式组织音频和视像数据的方式使读取视频数据流时能更有效地从存储媒介得到连续的信息。构成一个 AVI 文件的主要参数包括视像参数、伴音参数和压缩参数等。AVI 文件用的是 AVI RIFF 形式，AVI RIFF 形式由字串“AVI”标识。所有的 AVI 文件都包括两个必须的 LIST 块。这些块定义了流和数据流的格式。AVI 文件可能还包括一个索引块。

只要遵循这个标准，任何视频编码方案都可以使用在 AVI 文件中。这意味着 AVI 有着非常好的扩充性。这个规范由于是由微软制定的，微软全系列的软件包括编程工具 VB，VC 都提供了最直接的支持，因此更加奠定了 AVI 在 PC 上的视频霸主地位。由于 AVI 本身的开放性，获得了众多编码技术研发商的支持，不同的编码使得 AVI 不断被完善，现在几乎所有运行在 PC 上的通用视频编辑系统，都是以支持 AVI 为主的。AVI 的出现宣告了 PC 上无声时代的结束，不断完善的 AVI 格式代表了多媒体在 PC 上的兴起。

(3)容量与质量兼顾的 MPEG 系列编码。和 AVI 相反，MPEG 不是简单的一种文件格式，而是编码方案。

MPEG－1 制定于 1991 年底，处理的是标准图像交换格式(Standard Interchange Format，SIF)或者称为源输入格式(Source Input Format，SIF)的多媒体流。是针对 1.5Mbps 以下数据传输率的数字存储媒质运动图像及其伴音编码的国际标准，伴音标准后来衍生为今天的 MP3 编码方案。MPEG－1 规范了 PAL 制(352×288，25 帧/秒)和 NTSC 制(为 352×240，30 帧/秒)模式下的流量标准，提供了相当于家用录像系统(VHS)的影音质量，此时视频数据传输率被压缩至 1.15Mbps，其视频压缩率为 26∶1。

MPEG－2 是 1994 年发布的国际标准草案(DIS)，在视频编码算法上基本和 MPEG－1 相同，只是有了一些小小的改良，例如增加隔行扫描电视的编码。

MPEG－3 最初为 HDTV 制定，由于 MPEG－2 的快速发展，MPEG－3 还未彻底完成便宣告淘汰。

MPEG－4 于 1998 年被公布，和 MPEG－2 不同，MPEG－4 追求的不是高品质而是高压缩率以及适用于网络的交互能力。MPEG－4 提供了非常惊人的压缩率。

MJPEG，这并不是专门为 PC 准备的，而是为专业级甚至广播级的视频采集与在设备端回放而准备的，所以 MJPEG 包含了为传统模拟电视优化的隔行扫描电视的算法。

(4)属于网络的流媒体。RealNetworks RealVideo，采用的是 RealNetworks 公司自己开发的 Real G2 Codec，它具有很多先进的设计，例如 SVT(Scalable Video Technology)、双向编码(Two—Encoding，类似于 VBR)。RealMedia 音频部分采用的是 RealAudio，可以接纳很多音频编码方案，可实现声音在单声道、立体声音乐不同速率下的压缩。

Windows Media，视频编码采用的是非常先进的 MPEG－4 视频压缩技术，被称作 Microsoft MPEG－4 Video Codec，音频编码采用的是微软自行开发的一种编码方案，目前没有公布技术资料，在低流量下提供了令人满意的音质和画质。

事实上，常见的 MPG 文件也具有流媒体的最大特征——边读边放。

# 2.5 数据校验

计算机对数据进行传送、存储和操作的过程中，都有可能由于硬件故障、软件错误或信息干扰等原因而导致数据出错。为了有效地防止、减少或避免错码现象，就必须采取相应的技术手段来解决这个问题。校验码就是解决这一问题的主要手段。

数据进行传送、存储和操作的过程中可能出现的问题包括代码是否出错、错在何处以及如何纠正。

校验码分为两类：一类为检错码，即检查代码是否出错，但不能确定何处出错，不能修改；另一类为纠错码，即不仅能检查出错误，还能定位错误并纠正。常用的校验码有奇偶校验码、海明码、循环冗余码 CRC 等。

## 2.5.1 校验码的工作原理

在合法的数据编码之间加入一些不允许出现的(非法的)编码，使合法的数据编码出现某些错误时就变成非法编码，这样就可以通过检测编码的合法性来达到发现错误的目的。合理地安排非法编码的数量和编码规则，就可以提高发现错误的能力。

在校验码中码距是非常重要的一个参数，码距就是一种编码方案中任意两个合法编码之间不相同的位数的最小值。例如，4 位二进制有 16 种不同的编码，任意两个合法编码之间最少有一个二进制位不同，则它的码距为 1。码距与错误的校验有直接关系，码距为 1 的编码不能发现错误。仍以 4 位二进制数编码为例，如果它的十六个编码全为合法编码，那么任何一个编码出现错误后都会变成另外一个合法的编码，因此就查不出错误。例如希望传送的代码为 0000，经传送后产生了错误，被传送为 0001，而 0001 也是一个合法代码，这样就不能发现错误，因此码距大于 1 的编码才能发现错误或校正错误。

## 2.5.2 奇偶校验码

奇偶校验码是奇校验码和偶校验码的统称，是一种最基本的检错码。奇偶校验码是由 n－1 位信息元和 1 位校验元组成，可以表示成为(n，n－1)。如果是奇校验码，在附加上一个校验元以后，码长为 n 的码字中“1”的个数为奇数个；如果是偶校验码，在附加上一个校验元以后，码长为 n 的码字中“1”的个数为偶数个。

如果一个偶校验码的码字用 $A=[a_{n-1}a_{n-2}\cdots a_1a_0]$ 表示，则

$$S=a_{n-1}\oplus a_{n-2}\oplus\cdots\oplus a_1\oplus a_0 \qquad (2-1)$$

式中 S 为校验元，式(2－1)通常被称为校验方程。利用式(2－1)，由信息元即可求出校验元。另外，如果发生单个(或奇数个)错误，就会破坏这个关系式，因此通过该式能检测码字中是否发生了单个或奇数个错误。

**例 2.14**：被校验的代码为 1000001，则奇偶校验码分别为：

奇校验码：11000001　　(最左位为校验位)

偶校验码：01000001　　(最左位为校验位)

## 2.5.3 海明码

海明码是由 R. Hanming 在 1950 年首次提出的，它是一种可以纠正一位差错的编码。可

以借用简单奇偶校验码的生成原理来说明海明码的构造方法。

若 k＝n－1 位信息位 $a_{n-1}a_{n-2}\cdots a_1$ 加上一位偶校验位 $a_0$，构成一个 n 位的码字 $a_{n-1}a_{n-2}\cdots a_1a_0$，则在接收端校验时，可按关系式 $S=a_{n-1}\oplus a_{n-2}\oplus\cdots\oplus a_1\oplus a_0$ 来计算。若求得 S＝0，则表示无错；若 S＝1，则有错。上式可称为监督关系式，S 称为校正因子。在奇偶校验情况下，只有一个监督关系式和一个校正因子，其取值只有 0 或 1 两种情况，分别代表无错和有错两种结果，还不能指出错所在的位置。不难设想，若增加冗余位，也即相应地增加了监督关系式和校正因子，就能区分更多的情况。

如果有两个校正因子 $S_1$ 和 $S_0$，则 $S_1S_0$ 取值就有 00，01，10 或 11 四种可能的组合，也就能区分四种不同的情况。若其中一种取值用于表示无错（如 00），则另外三种（如 01，10 及 11）便可以用来指出不同情况的差错，从而可以进一步区分出是哪一位错。

设信息位为 k 位，增加 r 位冗余位，构成一个 n＝k＋r 位的码字。若希望用 r 个监督关系式产生的 r 个校正因子来区分无错和在码字中的 n 个不同位置的一位错，则要求满足关系式 $2^r\geqslant n+1$ 或 $2^r\geqslant k+r+1$

以 k＝4 为例来说明，要满足上述不等式，必须 r≥3。假设取 r＝3，则 n＝k＋r＝7，即在 4 位信息位 $a_6a_5a_4a_3$ 后面加上 3 位冗余位 $a_2a_1a_0$，构成 7 位码字 $a_6a_5a_4a_3a_2a_1a_0$，其中 $a_2$，$a_1$ 和 $a_0$ 分别由 4 位信息位中某几位半加得到，在校验时，$a_2$，$a_1$ 和 $a_0$ 就分别和这些位半加构成三个不同的监督关系式。在无错时，这三个关系式的值 $S_2$，$S_1$ 和 $S_0$ 全为“0”。若 $a_2$ 错，则 $S_2$＝1，而 $S_1$＝$S_0$＝0；若 $a_1$ 错，则 $S_1$＝1，而 $S_2$＝$S_0$＝0；若 $a_0$ 错，则 $S_0$＝1，而 $S_2$＝$S_1$＝0。$S_2$，$S_1$ 和 $S_0$ 这三个校正因子的其它 4 种编码值可用来区分 $a_3$，$a_4$，$a_5$，$a_6$ 中的一位错，其对应关系如表 2.6 所示。当然，也可以规定成另外的对应关系，这并不影响讨论的一般性。

**表 2.6　　$S_2$ $S_1$ $S_0$ 值与错码位置的对应关系**

| $S_2$ $S_1$ $S_0$ | 000 | 001 | 010 | 100 | 011 | 101 | 110 | 111 |
|---|---|---|---|---|---|---|---|---|
| 错码位置 | 无错 | $a_0$ | $a_1$ | $a_2$ | $a_3$ | $a_4$ | $a_5$ | $a_6$ |

由表 2.6 可以看出，$a_2$，$a_4$，$a_5$ 或 $a_6$ 的一位错都应使 $S_2$＝1，由此可以得到监督关系式 $S_2=a_2\oplus a_4\oplus a_5\oplus a_6$，同理可得 $S_1=a_1\oplus a_3\oplus a_5\oplus a_6$，$S_0=a_0\oplus a_3\oplus a_4\oplus a_6$。

在发送端编码时，信息位 $a_6$，$a_5$，$a_4$ 和 $a_3$ 的值取决于输入信号，它们在具体的应用中有确定的值。冗余位 $a_2$、$a_1$ 和 $a_0$ 的值应根据信息位的取值按监督关系式来确定，使上述三式中的 $S_2$，$S_1$ 和 $S_0$ 取值为零，即：

$a_2\oplus a_4\oplus a_5\oplus a_6=0$

$a_1\oplus a_3\oplus a_5\oplus a_6=0$

$a_0\oplus a_3\oplus a_4\oplus a_6=0$

由此可求得：$a_2=a_4\oplus a_5\oplus a_6$

$a_1=a_3\oplus a_5\oplus a_6$

$a_0=a_3\oplus a_4\oplus a_6$

已知信息位后，按上述三式即可算出各冗余位。对于本例来说，各种信息位算出的冗余位如表 2.7 所示。

表 2.7 由信息位算出的海明码冗余位

| 信息位 $a_6a_5a_4a_3$ | 冗余位 $a_2a_1a_0$ | 信息位 $a_6a_5a_4a_3$ | 冗余位 $a_2a_1a_0$ |
|---|---|---|---|
| 0000 | 000 | 1000 | 111 |
| 0001 | 011 | 1001 | 100 |
| 0010 | 101 | 1010 | 010 |
| 0011 | 110 | 1011 | 001 |
| 0100 | 110 | 1100 | 001 |
| 0101 | 101 | 1101 | 010 |
| 0110 | 011 | 1110 | 100 |
| 0111 | 000 | 1111 | 111 |

在接收端收到每个码字后，按监督关系式算出 $S_2$，$S_1$ 和 $S_0$，若它们全为“0”，则认为无错；若不全为“0”，在一位错的情况下，可通过查表 2.6 来判定是哪一位错，从而纠正之。例如码字 0010101 传输中发生一位错，在接收端收到的为 0011101，代入监督关系式可算得 $S_2=0$，$S_1=1$ 和 $S_0=1$，由表 2.6 可查得 $S_2S_1S_0=011$ 对应于 $a_3$ 错，因而可将 0011101 纠正为 00101010。

上述海明码的编码效率为 4/7。若 K=7，按 $2^r \geqslant k+r+1$ 可算得 r 至少为 4，此时编码，效率为 7/11。可见，信息位位数越多时，编码效率就越高。

## 2.5.4 循环冗余校验码(CRC)

循环冗余校验码的英文名称为 Cyclical Redundancy Check，简称 CRC。

循环冗余校验码是数据通信领域中最常用的一种差错校验码，其特征是信息字段和校验字段的长度可以任意选定。

循环冗余校验码的基本思想是利用线性编码理论，在发送端根据要传送的 k 位二进制码序列，以一定的规则产生一个校验用的监督码(既 CRC 码)r 位，并附在信息后边，构成一个新的二进制码序列数共(k+r)位，最后发送出去。在接收端，则根据信息码和 CRC 码之间所遵循的规则进行检验，以确定传送中是否出错。

CRC 的本质是模 2 除法的余数，采用的除数不同，CRC 的类型也就不一样。通常，CRC 的除数用生成多项式来表示。最常用的 CRC 码的生成多项式有 CRC16 和 CRC32。以 CRC16 为例，16 位的 CRC 码的产生规则是先将要发送的二进制序列数左移 16 位(既乘以 $2^{16}$)后，再除以一个多项式，最后所得到的余数既是 CRC 码。

生成 CRC 码的基本原理规定，任意一个由二进制位串组成的代码都可以和一个系数仅为‘0’和‘1’取值的多项式一一对应。例如：代码 1010111 对应的多项式为 $x^6+x^4+x^2+x+1$，而多项式为 $x^5+x^3+x^2+x+1$ 对应的代码 101111。

从 CRC 码集选择的原则可知，若设码字长度为 N，信息字段为 K 位，校验字段为 R 位(N=K+R)，则对于 CRC 码集中的任一码字，存在且仅存在一个 R 次多项式 g(x)，使得：

$$g(x)=x^R \cdot m(x)+r(x)$$

其中：m(x)为 K 次信息多项式，r(x)为 R−1 次校验多项式，g(x)称为生成多项式：

$$g(x)=g_0+g_1x+g_2x^2+\ldots+g_{(R-1)}x^{(R-1)}+g_Rx^R$$

发送方通过指定的 g(x)产生 CRC 码字，接收方则通过该 g(x)来验证收到的 CRC 码字。

例如：信息字段代码为：1011001；对应 $m(x)=x^6+x^4+x^3+1$ 假设生成多项式为：g(x)=

$x^4+x^3+1$；则对应 g(x)的代码为：11001。则 $x^4 m(x)=x^{10}+x^8+x^7+x^4$ 对应的代码记为：10110010000；采用多项式除法，得余数 1010（即校验字段为 1010）。

在发送方，发出的传输字段为：1 0 1 1 0 0 1　　1 0 1 0

信息字段　　校验字段

在接收方：使用相同的生成码进行校验：接收到的字段/生成码（二进制除法）如果能够除尽，则正确。

下面以发送数据是字符“12”为例看一下的生成码、解码的过程。发送的数据为“12”字符对应 ASCII 码的 16 进制为 3132H，对应的二进制为 0011 0001 0011 0010B。

设生成多项式为 1 0001 0000 0010 0001B=11021H。最高位 $2^{16}$，把数据 0011 0001 0011 0010 乘上 $2^{16}$ 得到 0011 0001 0011 0010 0000 0000 0000 0000 0000。用此数除上生成多项式 1 0001 0000 0010 0001 得余数即 CRC 校验码 0010 0000 1011 0101B=20B5H。最终发送端发送数据码=数据+CRC 校验码=313220B5H。

当接受到 313220B5H 这一数据后，用此数据除上生成多项式 1 0001 0000 0010 0001B=11021H。如果能够除尽，则表明数据正确，否则表明数据错误。在验证数据正确后把接受数据去掉 CRC 校验码 20B5H，得到数据 3132H，即数据是字符“12”。

循环冗余码在数据存储和数据通讯领域，无处不在。著名的通讯协议 X.25 的 FCS（帧检错序列）采用的是 CRC，而 CCITT，ARJ，LHA 等压缩工具软件采用的是 CRC32，磁盘驱动器的读写采用了 CRC16，通用的图像存储格式 GIF，TIFF 等也都用 CRC 作为检错手段。

# 2.6　相关知识介绍

## 2.6.1　编码

编码指用预先规定的方法将文字、数字或其他对象编成数码，或将信息、数据转换成规定的电脉冲信号。编码在电子计算机、电视、遥控和通讯等方面广泛使用。编码是信息从一种形式或格式转换为另一种形式的过程。解码，是编码的逆过程。

在计算机硬件中，编码是指用代码来表示各组数据资料，使其成为可利用计算机进行处理和分析的信息。代码是用来表示事物的记号，它可以用数字、字母、特殊的符号或它们之间的组合来表示。n 位二进制数可以组合成 2n 个不同的信息，给每个信息规定一个具体码组，这种过程也叫编码。

## 2.6.2　Unicode

Unicode 是一种在计算机上使用的字符编码。Unicode 是为了解决传统的字符编码方案的局限而产生的，例如 ISO－8859 所定义的字符虽然在不同的国家中广泛地使用，可是在不同国家间却经常出现不兼容的情况。很多传统的编码方式都有一个共同的问题，即容许电脑处理双语环境（通常使用拉丁字母以及其本地语言），但却无法同时支持多语言环境（指可同时处理多种语言混合的情况）。

在文字处理方面，Unicode 为每一个字符而非字形定义唯一的代码。换句话说，Unicode 以一种抽象的方式（即数字）来处理字符，并将视觉上的演绎工作（例如字体大小、外观形状、字体形态、文体等）留给其他软件来处理。

几乎所有电脑系统都支持基本拉丁字母，并各自支持不同的其他编码方式。Unicode 为了和它们相互兼容，其首 256 字符保留给 ISO 8859－1 所定义的字符，使既有的西欧语系文字的转换不需特别考量；并且把大量相同的字符重复编到不同的字符码中去，使得旧有纷杂的编码方式得以和 Unicode 编码间互相直接转换，而不会丢失任何信息。Unicode 是国际组织制定的可以容纳世界上所有文字和符号的字符编码方案。Unicode 用数字 0－0x10FFFF 来映射这些字符，最多可以容纳 1114112 个字符，或者说有 1114112 个码位。码位就是可以分配给字符的数字。UTF－8、UTF－16、UTF－32 都是将数字转换到程序数据的编码方案。在 Unicode 中：汉字“字”对应的数字是 23383（十进制），十六进制表示为 5B57。在 Unicode 中，我们有很多方式将数字 23383 表示成程序中的数据，包括：UTF－8、UTF－16、UTF－32。UTF 是“UCS Transformation Format”的缩写，可以翻译成 Unicode 字符集转换格式，即怎样将 Unicode 定义的数字转换成程序数据。例如，“汉字”对应的数字是 0x6c49 和 0x5b57。表 2.8 给出了 Unicode 中文汉字的编码。

**表 2.8　　Unicode 中文汉字的编码**

| U＋ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 4e00 | 一 | 丁 | 丂 | 七 | 丄 | 丅 | 丆 | 万 | 丈 | 三 | 上 | 下 | 丌 | 不 | 与 | 丏 |
| 4e10 | 丐 | 丑 | 丒 | 专 | 且 | 丕 | 世 | 丗 | 丘 | 丙 | 业 | 丛 | 东 | 丝 | 丞 | 丢 |
| 4e20 | 两 | 丣 | 两 | 严 | 并 | 丧 | 丨 | 丩 | 个 | 丫 | 丬 | 中 | 丮 | 丯 | 丰 | 丱 |
| 4e30 | 串 | 丳 | 临 | 丵 | 丶 | 丷 | 丸 | 丹 | 为 | 主 | 丼 | 丽 | 举 | 丿 | 乀 | 乁 |
| 4e40 | 乂 | 乃 | 乄 | 久 | 乆 | 乇 | 么 | 义 | 乊 | 之 | 乌 | 乍 | 乎 | 乏 | 乐 | 乑 |
| 4e50 | 乒 | 乓 | 乔 | 乕 | 乖 | 乘 | 乙 | 乚 | 乛 | 乜 | 九 | 乞 | 也 | 习 | 乡 | 书 |
| 4e60 | 乥 | 乧 | 乨 | 乩 | 乪 | 乫 | 乬 | 乭 | 乮 | 乯 | 买 | 乱 | 乲 | 乳 | 乴 | 乵 |
| 4e70 | 乶 | 乷 | 乸 | 乹 | 乺 | 乻 | 乼 | 乽 | 乾 | 乿 | 亀 | 亁 | 亂 | 亃 | 亄 | 亅 |
| 4e80 | 了 | 亇 | 予 | 争 | 亊 | 事 | 二 | 亍 | 于 | 亏 | 亐 | 云 | 互 | 亓 | 五 | 井 |
| 4e90 | 亖 | 亗 | 亘 | 亙 | 亚 | 些 | 亜 | 亝 | 亞 | 亟 | 亡 | 亢 | 交 | 亥 | 亦 | 产 |
| 4ea0 | 亨 | 亩 | 亪 | 享 | 京 | 亭 | 亮 | 亯 | 亰 | 亱 | 亲 | 亳 | 亴 | 亵 | 亶 | 亷 |
| 4eb0 | 亸 | 亹 | 人 | 亻 | 亼 | 亽 | 亾 | 亿 | 什 | 仁 | 仂 | 仃 | 仄 | 仅 | 仆 | 仇 |
| 4ec0 | 仉 | 今 | 介 | 仌 | 仍 | 从 | 仏 | 仐 | 仑 | 仒 | 仓 | 仔 | 仕 | 他 | 仗 | 付 |
| 4ed0 | 仙 | 仚 | 仛 | 仜 | 仝 | 仞 | 仟 | 仠 | 仡 | 仢 | 代 | 令 | 以 | 仦 | 仧 | 仨 |
| 4ee0 | 仩 | 仪 | 仫 | 们 | 仭 | 仮 | 仯 | 仰 | 仱 | 仲 | 仳 | 仴 | 仵 | 件 | 价 | 仸 |
| 4ef0 | 仹 | 仺 | 任 | 任 | 份 | 仼 | 仿 | 伀 | 企 | 伂 | 伃 | 伄 | 伅 | 伆 | 伇 | 伈 |
| 4f00 | 伉 | 伊 | 伋 | 伌 | 伍 | 伎 | 伏 | 伐 | 休 | 伒 | 伓 | 伔 | 伕 | 伖 | 众 | 优 |
| 4f10 | 伙 | 会 | 伛 | 伜 | 伝 | 伞 | 伟 | 传 | 伡 | 伢 | 伣 | 伤 | 伥 | 伦 | 伧 | 伨 |
| 4f20 | 伩 | 伪 | 伫 | 伬 | 伭 | 伮 | 伯 | 估 | 伱 | 伲 | 伳 | 伴 | 伵 | 伶 | 伷 | 伸 |
| 4f30 | 伹 | 伺 | 伻 | 似 | 伽 | 伾 | 伿 | 佀 | 佁 | 佂 | 佃 | 佄 | 佅 | 但 | 佇 | 佈 |

（续表）

| U+ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 4f40 | 侣 | 侁 | 征 | 佃 | 佣 | 侏 | 但 | 伫 | 布 | 佐 | 彼 | 侶 | 侃 | 位 | 低 | 住 |
| 4f50 | 佐 | 佑 | 侠 | 体 | 占 | 何 | 佖 | 佗 | 佘 | 余 | 佚 | 佛 | 作 | 佝 | 佞 | 佟 |
| 4f60 | 你 | 侖 | 佢 | 佣 | 佤 | 佥 | 佑 | 伟 | 佻 | 佩 | 佪 | 佫 | 佬 | 佯 | 佮 | 佯 |
| 4f70 | 佰 | 佥 | 佫 | 佳 | 佴 | 并 | 佶 | 佷 | 佸 | 佹 | 佺 | 佻 | 佼 | 欣 | 佾 | 使 |
| 4f80 | 侀 | 侁 | 侂 | 侃 | 侄 | 侅 | 来 | 侇 | 侈 | 侍 | 侥 | 例 | 侩 | 侍 | 侏 | 侏 |
| 4f90 | 侖 | 侑 | 侫 | 侔 | 侔 | 侀 | 仑 | 侗 | 侂 | 侃 | 徇 | 供 | 俏 | 依 | 伽 | 俯 |
| 4fa0 | 侠 | 侀 | 俜 | 侣 | 俦 | 侥 | 侦 | 侧 | 侨 | 侩 | 侪 | 侵 | 侬 | 侭 | 侮 | 侯 |
| 4fb0 | 偌 | 偍 | 侵 | 侳 | 俞 | 侵 | 侣 | 局 | 佢 | 侹 | 俦 | 悦 | 俦 | 偶 | 俦 | 便 |
| 4fc0 | 俀 | 俣 | 系 | 促 | 俄 | 俅 | 徐 | 侄 | 借 | 悟 | 俊 | 偲 | 浦 | 狠 | 俎 | 俏 |
| 4fd0 | 俐 | 俑 | 恍 | 侄 | 倪 | 侏 | 侶 | 俗 | 俘 | 俙 | 俚 | 俛 | 俜 | 保 | 俞 | 俟 |
| 4fe0 | 侠 | 信 | 修 | 俣 | 俤 | 俥 | 俦 | 徒 | 俨 | 俩 | 俪 | 俅 | 俶 | 俭 | 修 | 俯 |
| 4ff0 | 俙 | 俱 | 俙 | 俳 | 俊 | 表 | 俶 | 俪 | 俸 | 俹 | 俺 | 备 | 倩 | 倣 | 俾 | 倱 |

## 2.6.3　BMP 图片格式介绍

BMP：全称为 Bitmap 是 Windows 操作系统中的标准图像文件格式，可以分成两类：设备相关位图（DDB）和设备无关位图（DIB），使用非常广。它采用位映射存储格式，除了图像深度可选以外，不采用其他任何压缩，因此，BMP 文件所占用的空间很大。BMP 文件的图像深度可选 lbit、4bit、8bit 及 24bit。BMP 文件存储数据时，图像的扫描方式是按从左到右、从下到上的顺序。由于 BMP 文件格式是 Windows 环境中交换与图有关的数据的一种标准，因此在 Windows 环境中运行的图形图像软件都支持 BMP 图像格式。典型的 BMP 图像文件由四部分组成：

（1）位图头文件数据结构，它包含 BMP 图像文件的类型、显示内容等信息；

（2）位图信息数据结构，它包含有 BMP 图像的宽、高、压缩方法，以及定义颜色等信息；

（3）调色板，这个部分是可选的，有些位图需要调色板，有些位图，比如真彩色图（24 位的 BMP）就不需要调色板；

（4）位图数据，这部分的内容根据 BMP 位图使用的位数不同而不同，在 24 位图中直接使用 RGB 颜色，而其他的小于 24 位的使用调色板中颜色索引值。

位图一共有两种类型，即：设备相关位图（DDB）和设备无关位图（DIB）。DDB 位图在早期的 Windows 系统（Windows 3.0 以前）中是很普遍的，事实上它也是唯一的。然而，随着显示器制造技术的进步，以及显示设备的多样化，DDB 位图的一些固有的问题开始浮现出来了。比如，它不能够存储（或者说获取）创建这张图片的原始设备的分辨率，这样，应用程序就不能快速的判断客户机的显示设备是否适合显示这张图片。为了解决这一难题，微软创建了 DIB 位图格式。

**1. 设备无关位图（Device－Independent Bitmap）**

DIB 位图包含下列的颜色和尺寸信息：

(1)原始设备(即创建图片的设备)的颜色格式。

(2)原始设备的分辨率。

(3)原始设备的调色板。

(4)一个位数组,由红、绿、蓝(RGB)三个值代表一个像素。

(5)一个数组压缩标志,用于表明数据的压缩方案(如果需要的话)。

以上这些信息保存在 BITMAPINFO 结构中,该结构由 BITMAPINFOHEADER 结构和两个或更多个 RGBQUAD 结构所组成。BITMAPINFOHEADER 结构所包含的成员表明了图像的尺寸、原始设备的颜色格式、以及数据压缩方案等信息。RGBQUAD 结构标识了像素所用到的颜色数据。

DIB 位图也有两种形式,即:底到上型 DIB(bottom－up),和顶到下型 DIB(top－down)。底到上型 DIB 的原点(origin)在图像的左下角,而顶到下型 DIB 的原点在图像的左上角。如果 DIB 的高度值(由 BITMAPINFOHEADER 结构中的 biHeight 成员标识)是一个正值,那么就表明这个 DIB 是一个底到上型 DIB,如果高度值是一个负值,那么它就是一个顶到下型 DIB。注意:顶到下型的 DIB 位图是不能被压缩的。

位图的颜色格式是通过颜色面板值(planes)和颜色位值(bitcount)计算得来的,颜色面板值永远是 1,而颜色位值则可以是 1、4、8、16、24、32 其中的一个。如果它是 1,则表示位图是一张单色位图(译者注:通常是黑白位图,只有黑和白两种颜色,当然它也可以是任意两种指定的颜色),如果它是 4,则表示这是一张 VGA 位图,如果它是 8、16、24、或是 32,则表示该位图是其他设备所产生的位图。

调色板是被保存在一个 RGBQUAD 结构的数组中,该结构指出了每一种颜色的红、绿、蓝的分量值。位数组中的每一个索引都对应于一个调色板项(即一个 RGBQUAD 结构)。

Win32 API 支持位数据的压缩(只对 8 位和 4 位的底到上型 DIB 位图)。压缩方法是采用运行长度编码方案(RLE),RLE 使用两个字节来描述一个句法,第一个字节表示重复像素的个数,第二个字节表示重复像素的索引值。有关压缩位图的详细信息请参见对 BITMAPINFO-HEADER 结构的解释。

**2.设备相关位图 (Device－Dependent Bitmaps)**

设备相关位图(DDB)之所以现在还被系统支持,只是为了兼容旧的 Windows 3.0 软件,如果程序员现在要开发一个与位图有关的程序,则应该尽量使用或生成 DIB 格式的位图。

DDB 位图是被一个单个结构 BITMAP 所描述,这个结构的成员标明了该位图的宽度、高度、设备的颜色格式等信息。

DDB 位图也有两种类型,即:可废弃的(discardable)DDB 和不可废弃的(nondiscardable)DDB。可废弃的 DDB 位图就是一种当系统内存缺乏,并且该位图也没有被选入设备描述表(DC)的时候,系统就会把该 DDB 位图从内存中清除(即废弃)。不可废弃的 DDB 则是无论系统内存多少都不会被系统清除的 DDB。API 函数 CreateDiscardableBitmap()函数可用于创建可废弃位图。而函数 CreateBitmap()、CreateCompatibleBitmap()、和 CreateBitmapIndirect()可用于创建不可废弃的位图。其对应的数据结构为:

(1)BMP 文件组成

BMP 文件由文件头、位图信息头、颜色信息和图形数据四部分组成。

(2)BMP 文件头(14 字节)

BMP 文件头数据结构含有 BMP 文件的类型、文件大小和位图起始位置等信息。

其结构定义如下:

```
typedef struct tagBITMAPFILEHEADER
{
    WORD bfType; //位图文件的类型,必须为 BM(1－2 字节)
    DWORD bfSize; //位图文件的大小,以字节为单位(3－6 字节)
    WORD bfReserved1; //位图文件保留字,必须为 0(7－8 字节)
    WORD bfReserved2; //位图文件保留字,必须为 0(9－10 字节)
    DWORD bfOffBits; //位图数据的起始位置,以相对于位图(11－14 字节)
    //文件头的偏移量表示,以字节为单位
} BITMAPFILEHEADER;
```

位图信息头(40 字节)

BMP 位图信息头数据用于说明位图的尺寸等信息。

```
typedef struct tagBITMAPINFOHEADER{
    DWORD biSize; //本结构所占用字节数(15－18 字节)
    LONG biWidth; //位图的宽度,以像素为单位(19－22 字节)
    LONG biHeight; //位图的高度,以像素为单位(23－26 字节)
    WORD biPlanes; //目标设备的级别,必须为 1(27－28 字节)
    WORD biBitCount;//每个像素所需的位数,必须是 1(双色),(29－30 字节)
    // 4(16 色),8(256 色)16(高彩色)或 24(真彩色)之一
    DWORD biCompression; //位图压缩类型,必须是 0(不压缩),(31－34 字节)
    // 1(BI_RLE8 压缩类型)或 2(BI_RLE4 压缩类型)之一
    DWORD biSizeImage; //位图的大小(其中包含了为了补齐行数是 4 的倍数而添加的
空字节),以字节为单位(35－38 字节)
    LONG biXPelsPerMeter; //位图水平分辨率,每米像素数(39－42 字节)
    LONG biYPelsPerMeter; //位图垂直分辨率,每米像素数(43－46 字节)
    DWORD biClrUsed;//位图实际使用的颜色表中的颜色数(47－50 字节)
    DWORD biClrImportant;//位图显示过程中重要的颜色数(51－54 字节)
} BITMAPINFOHEADER;
```

颜色表

颜色表用于说明位图中的颜色,它有若干个表项,每一个表项是一个 RGBQUAD 类型的结构,定义一种颜色。RGBQUAD 结构的定义如下:

```
typedef struct tagRGBQUAD {
    BYTE rgbBlue;//蓝色的亮度(值范围为 0－255)
    BYTE rgbGreen; //绿色的亮度(值范围为 0－255)
    BYTE rgbRed; //红色的亮度(值范围为 0－255)
    BYTE rgbReserved;//保留,必须为 0
} RGBQUAD;
```

颜色表中 RGBQUAD 结构数据的个数有 biBitCount 来确定:

当 biBitCount＝1,4,8 时,分别有 2,16,256 个表项;

当 biBitCount＝24 时,没有颜色表项。

位图信息头和颜色表组成位图信息,BITMAPINFO 结构定义如下:

```
typedef struct tagBITMAPINFO {
```

BITMAPINFOHEADER bmiHeader; //位图信息头
RGBQUAD bmiColors[1]; //颜色表
} BITMAPINFO;

位图数据

位图数据记录了位图的每一个像素值,记录顺序是在扫描行内是从左到右,扫描行之间是从下到上。位图的一个像素值所占的字节数:

当 biBitCount=1 时,8 个像素占 1 个字节;

当 biBitCount=4 时,2 个像素占 1 个字节;

当 biBitCount=8 时,1 个像素占 1 个字节;

当 biBitCount=24 时,1 个像素占 3 个字节,按顺序分别为 B,G,R;

Windows 规定一个扫描行所占的字节数必须是 4 的倍数(即以 long 为单位),不足的以 0 填充,

biSizeImage = ((((bi.biWidth * bi.biBitCount) + 31) & ~31) / 8) * bi.biHeight;

如某 BMP 文件开头具体数据举例为:

424D 46900000 0000 0000 4600 0000 2800 0000 8000 0000 9000 0000 0100 * 1000 0300 0000 00900000 A00F 0000 A00F0000 0000 00000000 0000 * 00F8 E007 1F00 0000 * 02F1 84F1 04F1 84F1 84F1 06F2 84F1 06F2 04F2 86F2 06F2 86F2 86F2 .... ....

## 2.6.4 浮点数格式 IEEE 754

IEEE 二进制浮点数算术标准(IEEE 754)是 20 世纪 80 年代以来最广泛使用的浮点数运算标准,为许多 CPU 与浮点运算器所采用。这个标准定义了表示浮点数的格式(包括负零-0)与反常值(denormal number),一些特殊数值(无穷(Inf)与非数值(NaN)),以及这些数值的"浮点数运算符";它也指明了四种数值舍入规则和五种例外状况(包括例外发生的时机与处理方式)。

IEEE 754 规定了四种表示浮点数值的方式:单精确度(32 位)、双精确度(64 位)、延伸单精确度(43 比特以上,很少使用)与延伸双精确度(79 比特以上,通常以 80 比特实做)。只有 32 位模式有强制要求,其他都是选择性的。大部分编程语言都有提供 IEEE 浮点数格式与算术,但有些将其列为非必需的。例如,IEEE 754 问世之前就有的 C 语言,现在有包括 IEEE 算术,但不算作强制要求(C 语言的 float 通常是指 IEEE 单精确度,而 double 是指双精确度)。

二进制浮点数是以符号数值表示法的格式存储如图 2.3 所示包含三个部分:

(1)符号位(sign bit):最高有效位被指定为符号位;

(2)指数部份(exponent):即次高有效的 e 个比特位,存储指数;

(3)尾数(significand):最后剩下的 f 个低有效位的比特位,存储尾数的小数部份。


图 2.3 浮点数的格式存储

### 1.32 位单精度浮点数

IEEE 754 标准所定义的单精度浮点数的长度为 32 位,按位域可划分为:符号位、阶码位与

尾数位。符号位是 1 位，阶码位是 8 位，尾数位是 23 位。指数部分即使用所谓的偏正值形式表示，偏正值为实际的指数大小与一个固定值(32 位的情况是 127)的和。采用这种方式表示的目的是简化比较。因为，指数的值可能为正也可能为负，如果采用补码表示的话，全体符号位 S 和 Exp 自身的符号位将导致不能简单的进行大小比较。正因为如此，指数部分通常采用一个无符号的正数值存储。单精度的指数部分是－126～＋127 加上偏移值 127，指数值的大小从 1～254(0 和 255 是特殊值)。浮点小数计算时，指数值减去偏正值将是实际的指数大小。表 2.9 展示了单精度浮点数的各种极值情况。

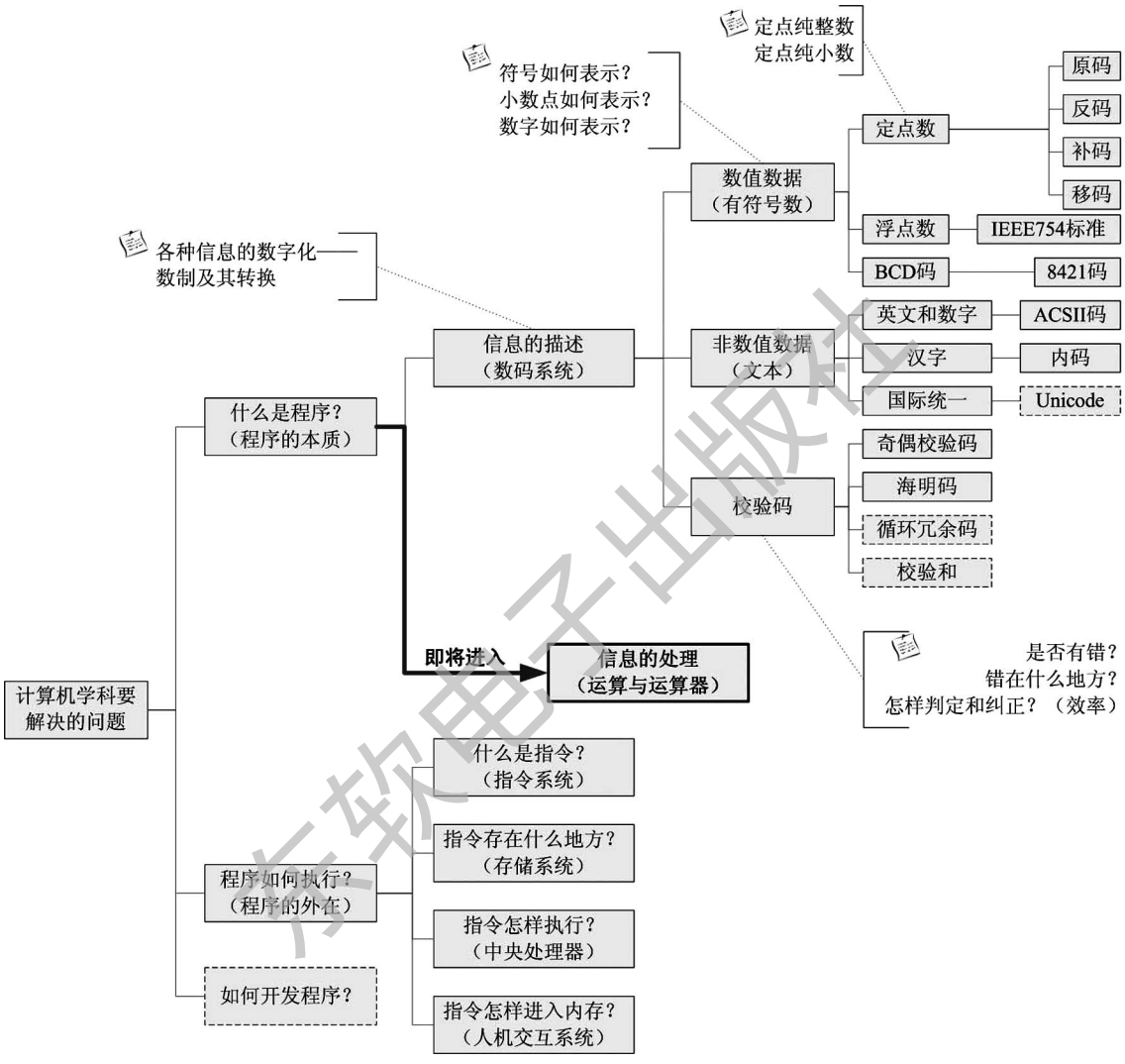
**表 2.9　　单精度浮点数各种极值**

| 类别 | 正负号 | 实际指数 | 有偏移指数 | 指数域 | 尾数域 | 数值 |
|---|---|---|---|---|---|---|
| 零 | 0 | －127 | 0 | 0000 0000 | 000 0000 0000 0000 0000 0000 | 0.0 |
| 负零 | 1 | －127 | 0 | 0000 0000 | 000 0000 0000 0000 0000 0000 | －0.0 |
| 1 | 0 | 0 | 127 | 0111 1111 | 000 0000 0000 0000 0000 0000 | 1.0 |
| －1 | 1 | 0 | 127 | 0111 1111 | 000 0000 0000 0000 0000 0000 | －1.0 |
| 最小的非规约数 | * | －126 | 0 | 0000 0000 | 000 0000 0000 0000 0000 0001 | $\pm 2^{-23} \times 2^{-126} = \pm 2^{-149} \approx \pm 1.4 \times 10^{-45}$ |
| 中间大小的非规约数 | * | －126 | 0 | 0000 0000 | 100 0000 0000 0000 0000 0000 | $\pm 2^{-1} \times 2^{-126} = \pm 2^{-127} \approx \pm 5.88 \times 10^{-39}$ |
| 最大的非规约数 | * | －126 | 0 | 0000 0000 | 111 1111 1111 1111 1111 1111 | $\pm (1-2^{-23}) \times 2^{-126} \approx \pm 1.18 \times 10^{-38}$ |
| 最小的规约数 | * | －126 | 1 | 0000 0001 | 000 0000 0000 0000 0000 0000 | $\pm 2^{-126} \approx \pm 1.18 \times 10^{-38}$ |
| 最大的规约数 | * | 127 | 254 | 1111 1110 | 111 1111 1111 1111 1111 1111 | $\pm (2-2^{-23}) \times 2^{127} \approx \pm 3.4 \times 10^{38}$ |
| 正无穷 | 0 | 128 | 255 | 1111 1111 | 000 0000 0000 0000 0000 0000 | $+\infty$ |
| 负无穷 | 1 | 128 | 255 | 1111 1111 | 000 0000 0000 0000 0000 0000 | $-\infty$ |
| NaN | * | 128 | 255 | 1111 1111 | non zero | 非数值 |
| * 符号位可以为 0 或 1 | | | | | | |

### 2.64 位双精度浮点数

相对于单精度浮点数格式，双精度的阶码变为 11 位，尾数变为 52 位。指数部分即使用所谓的偏正值形式表示，偏正值为实际的指数大小与一个固定值(64 位的情况是 1023)的和。采用这种方式表示的目的是简化比较。因为，指数的值可能为正也可能为负，如果采用补码表示

的话，全体符号位 S 和 Exp 自身的符号位将导致不能简单的进行大小比较。正因为如此，指数部分通常采用一个无符号的正数值存储。双精度的指数部分是－1022～＋1023 加上 1023，指数值的大小从 1～2046(0(2 进位全为 0)和 2047(2 进位全为 1)是特殊值)。浮点小数计算时，指数值减去偏正值将是实际的指数大小。




课程逻辑图

# 实践环节设计

## 项目 1：汉字的内码计算与查看(UP(2/4))

### 一、能力要求

(1)计算机基础知识：掌握关于汉字在计算机内存储的基础知识。(重要)

(2)验证假设与结论：验证汉字区位码、国标码和内码之间的关系。(中等)

(3)查询印刷资料和电子文献:查询 GB2312－80 文档。(重要)

(4)书面交流:规范地撰写项目报告。(重要)

**二、项目构思**

通过简单的工具软件来查看汉字在计算机内的存储格式;

学习通过 GB2312－80 文档来计算汉字的区位码、国标码和内码,以加深对汉字内码、国标码、区位码等概念的认识和理解;

学习 DEBUG 的使用。

**三、项目设计**

通过 GB2312－80 文档和汉字内码查看器来计算汉字“中”“国”“人”以及自己姓名的区位码、国标码和内码,并通过简单的工具软件来查看汉字在计算机内的存储格式,并将两者进行比较,观察结果是否一致。

**四、项目实施**

(1)在 GB2312－80 国家标准文档中查找汉字的区位号,以“中”字为例。

① 双击打开 GB2312－80 编码表。

② 在左上角的“编辑”菜单中选“查找”,在查找内容中键入“中”字,如图 2.4 所示。


图 2.4　GB2312－80 编码表

③ 单击“查找下一个(F)”按钮,进行查找,结果如图 2.5 所示(注意要查找的结果必须出现在排列的汉字矩阵中,不能出现在矩阵外,可多次按“查找下一个(F)”按钮,直到查到为止),得到“中”字位于第 54 区,第 48 位(位号为左边的 4 加上上面的 8,即 48)。

- 国标第 54区

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | 帧 | 症 | 郑 | 证 | 芝 | 枝 | 支 | 吱 | 蜘 |
| 1 | 知 | 胺 | 脂 | 汁 | 之 | 织 | 职 | 直 | 植 | 殖 |
| 2 | 执 | 值 | 侄 | 址 | 指 | 止 | 趾 | 只 | 旨 | 纸 |
| 3 | 志 | 挚 | 掷 | 至 | 致 | 置 | 帜 | 峙 | 制 | 智 |
| 4 | 秩 | 稚 | 质 | 炙 | 痔 | 滞 | 治 | 窒 | 中 | 盅 |
| 5 | 忠 | 钟 | 衷 | 终 | 种 | 肿 | 重 | 仲 | 众 | 舟 |
| 6 | 周 | 州 | 洲 | 诌 | 粥 | 轴 | 肘 | 帚 | 咒 | 皱 |
| 7 | 宙 | 昼 | 骤 | 珠 | 株 | 蛛 | 朱 | 猪 | 诸 | 诛 |
| 8 | 逐 | 竹 | 烛 | 煮 | 拄 | 瞩 | 嘱 | 主 | 著 | 柱 |
| 9 | 助 | 蛀 | 贮 | 铸 | 筑 | | | | | |

图 2.5　查找结果

(2)将“中”字的区号和位号分别转换为十六进制,得到它的区位码。

区号:54=36H　　位号:48=30H

则“中”字的区位码为:3630H。

(3)将区位码加上 2020H 即得到它的国标码。

则“中”字的国标码为:5650H。

(4)将国标码加上 8080H 即得到它的内码(在计算机中的存储形式)。

则“中”字的内码为:D6D0H。

**五、项目运行**

按表 2.10 格式在项目报告中记录运行结果(其中“张三”用自己的姓名替代)。

**表 2.10　　项目运行结果**

| 汉字 | 区号 | 位号 | 区位码 | 国标码 | 内码 |
|---|---|---|---|---|---|
| 中 | | | | | |
| 国 | | | | | |
| 人 | | | | | |
| 张 | | | | | |
| 三 | | | | | |

**六、目标检验**

学生自查:对照所学理论知识,计算汉字的各种表示方式,并与运行结果对比。

教师检查:按照标准答案以及学生撰写项目报告情况给出项目成绩。

项目考试:本项目是项目考试内容之一。

## 项目 2:文本文件的存储格式(UP(2/4))

**一、能力要求**

(1)计算机基础知识:掌握关于字母、数字等文本在计算机内存储的基础知识。(重要)

(2)分析问题:分析各种类型文本的表示方式。(中等)

(3)验证假设与结论:验证字母、数字等文本使用 ASCII 码表示以及文本文件大小的假设。(不重要)

(4)具有综合和通用化能力:综合运用所学到的关于字母、数字以及汉字存储的相关知识。(不重要)

(5)估计与定性分析:估算出文本文件的存储大小。(不重要)

(6)查询印刷资料和电子文献:查询 ACSII 编码表。(重要)

(7)书面的交流:规范地撰写项目报告。(重要)

**二、项目构思**

通过 DEBUG 软件来查看文本文件在计算机内的存储格式,以加深对 ASCII 码、汉字内码等概念的认识和理解,同时学习 DEBUG 的使用。

**三、项目设计**

在 C 盘下新建 TXT 目录,在 TXT 目录中用记事本生成一个含有 ASCII 码和汉字的文本文件 test.txt,包含若干换行,再用 DEBUG 软件查看该文件的内容和长度。

**四、项目实施**

(1)用 Windows 操作系统下的记事本程序在 TXT 目录中生成 test.txt 文件。

文件内容为：

abc

ABC

012

中国人

张三

**注意**：有四个换行，最后一行没有换行符。

(2)计算文件的长度。换行符在计算机内部被表示成回车符(0DH)和换行符(0AH)，英文字母和数字均以 ASCII 码的形式存储，每个字符占一个字节，而汉字则以内码形式存储，每个汉字占两个字节，所以可以计算该文件的大小为：9×1(英文和数字)＋5×2(3 个汉字)＋4×2(4 个换行)＝27 字节。

(3)在 Windows 下查看文件 test.txt 的长度，观察是否与计算的文件大小一致，方法为指向文件后按鼠标右键，选“属性”，如图 2.6 所示。

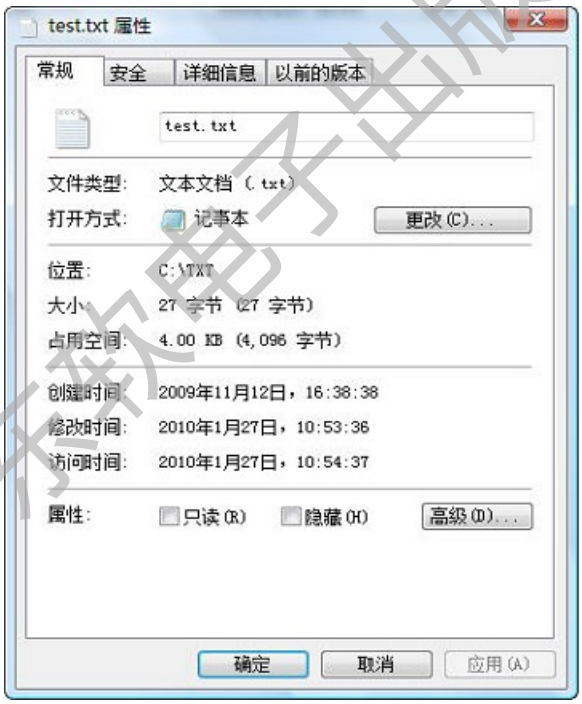



**图 2.6　test.txt 文件的属性**

(4)使用 DEBUG 查看文件的内容和长度。

① 首先进入 DOS 方式(点 Windows 左下角的“开始”按钮，再选择“运行”，键入“cmd”，回车)。

② 键入命令“cd c:\txt”，回车，进入 test.txt 文件所在的目录 TXT。

③ 键入“DEBUG test.txt”，回车，出现“－”(短横线，为 DEBUG 软件的提示符)。

④ 键入“r”,回车,查看文件的长度。

**注意**:DEBUG 软件装入文件后,文件的长度存储在 BX 和 CX 中,单位是字节,其中 BX 是高位,CX 是低位,本例中 BX=0000,CX=001B,由于 DEBUG 中全部是十六进制,因此文件 test.txt 的长度为:0000001BH,即十进制的 27 字节。这与前面计算和查看的文件长度一致。

⑤ 键入“d”,回车,显示文件的内容。

**五、项目运行**

按表 2.11 格式在项目报告中记录运行结果(其中,“张三”用自己的姓名替代):

表 2.11 项目运行结果

| 文本内容 | 内存显示 |
|---|---|
| abc | |
| ABC | |
| 012 | |
| 中国人 | |
| 张三 | |

**六、目标检验**

学生自查:对照所学理论知识,计算文本的各种表示方式,并与运行结果对比。

教师检查:按照标准答案以及学生撰写项目报告情况给出项目成绩。

项目考试:本项目是项目考试内容之一。

## 项目 3:整数在计算机内的存储形式(UP(2/4))

**一、能力要求**

(1)计算机基础知识:掌握关于整数原、反、补码的基础知识。(重要)

(2)分析问题:分析内存显示内容的含义。(中等)

(3)验证假设与结论:验证整数使用补码存储的假设。(不重要)

(4)具有综合和通用化能力:能够对不同精度的数据进行识别和综合。(不重要)

(5)估计与定性分析:估计不同的数值在计算机内的表示方式。(中等)

(6)书面的交流:规范地撰写项目报告。(重要)

**二、项目构思**

通过简单的汇编程序和 DEBUG 来查看整数在计算机内的存储形式,加深对补码的理解和认识,学习 DEBUG 的使用。

**三、项目设计**

将含有已经编写好的汇编程序 int.asm 以及进行汇编和链接用的程序 masm.exe 和 link.exe的文件夹 INT 复制到 C 盘的根目录下。在已经编写好的汇编程序 int.asm 中修改相应的数据分别为±15、±63、±127 以及±自己学号后两位,再对该程序进行汇编和链接,最后用 DEBUG 软件查看生成的可执行文件 int.exe 中的数据存储。

**四、项目实施**

(1)在 int.asm 中修改相应的数据。

用记事本等纯文本工具打开 int.asm,修改相应的数据(下面程序中的粗斜体部分),从而

查看某整数在计算机内的存储形式。注意：程序的其它部分不要改动，改动后注意保存。例如本例中将查看两个整数＋15 和－15 分别用 8 位，16 位，32 位，64 位存储时的存储形式。

int.asm 源文件的内容：

```
data segment

     db +15        ;   8 位整数   db: define byte
     db -15
     dw +15        ;  16 位整数   dw: define word
     dw -15
     dd +15        ;  32 位整数   dd: define double word
     dd -15
     dq +15        ;  64 位整数   dq: define quardword
     dq -15
data ends

code segment
  assume ds:data,cs:code
 main proc far
 start:
     mov ax,data    ;让 DS 指向数据段
     mov ds,ax

     mov ax,4c00h
     int 21h
 main endp
code ends
   end start
```

(2)汇编和链接。

① 进入 DOS 方式(点 Windows 左下角的“开始”按钮，再选择“运行”，键入“cmd”，回车)。

② 进入 int.asm 程序所在的目录：键入“cd c:\int”，回车。

③ 用“masm　int.asm”命令进行汇编，将源程序汇编成目标程序，连续回车，直到出现 DOS 提示符。

④ 用“link　int.obj”命令进行链接，将目标文件链接成 exe 文件，连续回车，直到出现 DOS 提示符。

(3)利用 DEBUG 查看 int.exe 的数据存储。

① 键入命令“DEBUG int.exe”，回车，出现“－”(短横线，为 DEBUG 软件的提示符)。

② 键入命令“u”，回车，观察输出结果。

③ 找到第一行“MOV　AX,13CE”处的数“13CE”(注意，不同的计算机显示的数可能不同，假设为 X)，然后键入“dX:0”，回车，本例中键入“d13CE:0”。

④ 查看所显示的整数的存储形式。(注意存储时高位是存储在高地址单元中的)

⑤ 将所查到的整数的存储形式和按定点整数补码算出的形式比较，观察结果是否一致。

**五、项目运行**

按表 2.12 格式在项目报告中记录运行结果：

表 2.12 项目运行结果

| 数值 | 8 位表示 | 16 位表示 | 32 位表示 | 64 位表示 |
|---|---|---|---|---|
| ＋15 | | | | |
| －15 | | | | |
| ＋63 | | | | |
| －63 | | | | |
| ＋127 | | | | |
| －127 | | | | |
| ＋自己学号后两位 | | | | |
| －自己学号后两位 | | | | |

### 六、目标检验

学生自查：对照所学理论知识，计算整数的各种表示方式，并与运行结果对比。

教师检查：按照标准答案以及学生撰写项目报告情况给出项目成绩。

项目考试：本项目是项目考试内容之一。

## 项目 4*：浮点数在计算机内的存储形式(UP(2/4))

### 一、能力要求

(1)计算机基础知识：掌握关于浮点数使用尾数与阶码表示的基础知识。(重要)

(2)分析问题：分析内存显示内容的含义。(中等)

(3)验证假设与结论：验证浮点数使用尾数与阶码表示的假设。(不重要)

(4)具有综合和通用化能力：能够对不同精度的数据进行识别和综合。(中等)

(5)估计与定性分析：估计浮点数在计算机中的存储形式。(不重要)

(6)查询印刷资料和电子文献：查询 IEEE754 相关文档。(不重要)

(7)书面的交流：规范地撰写项目报告。(重要)

### 二、项目构思

通过简单的汇编程序和 DEBUG 来查看浮点数在计算机内的存储格式，加深对浮点数的认识和理解，学习 DEBUG 的使用。

### 三、项目设计

将含有已经编写好的汇编程序 float.asm 以及进行汇编和链接用的程序 masm.exe 和 link.exe的文件夹 FLOAT 复制到 C 盘的根目录下。在已经编写好的汇编程序 float.asm 中修改相应的数据分别为＋100.25，－74.75，＋64.5，－255.125，再对该程序进行汇编和链接，最后用 DEBUG 软件查看生成的可执行文件 float.exe 中的数据存储。

### 四、项目实施

(1)在 float.asm 中修改相应的数据。用记事本等纯文本工具打开 float.asm 文件，修改相应的数据(下面程序中的粗斜体部分)，从而查看某实数在计算机内的存储形式。注意，程序的其他部分不要改动，改动后注意保存。例如本例中将查看两个浮点数的存储形式：＋100.25 和－74.75。

float.asm 源文件的内容：

```
data segment
```

```
dd + 100.25    ;定义一个 float 型的实数,dd:define double word,在此修改。
dd - 74.75   ;利用 dd 定义的数占双字长,即 32 位。
dq + 100.25    ;定义一个 double 型的实数,dq:define quardword。
dq - 74.75    ;利用 dq 定义的数占四字长,即 64 位。

data ends

code segment
  assume ds:data,cs:code
 main proc far
 start:
     mov ax,data    ;让 DS 指向数据段
     mov ds,ax

     mov ax,4c00h
     int 21h
 main endp
code ends
   end start
```

(2)汇编和链接。

① 进入 DOS 方式(点 Windows 左下角的“开始”按钮,再选择“运行”,键入“cmd”,回车)。

② 进入 float.asm 程序所在的目录:键入“cd c:\float”,回车。

③ 用“masm float.asm”命令进行汇编,将源程序汇编成目标程序,连续回车,直到出现 DOS 提示符。

④ 用“link float.obj”命令进行链接,将目标文件链接成 exe 文件,连续回车,直到出现 DOS 提示符。

(3)利用 DEBUG 查看 float.exe 的数据存储。

① 键入命令“DEBUG float.exe”,回车,出现“-”(短横线,为 DEBUG 软件的提示符)。

② 键入命令“u”,回车,观察输出显示。

③ 找到第一行“MOV　AX,13CE”处的数“13CE”(注意,不同的计算机上显示的可能不同,假设为 X),然后键入“dX:0”,回车,本例中键入“d13CE:0”。

④ 查看所显示的浮点数的存储形式,前两个浮点数每个占 4 字节,高位存储在高地址,如显示为:00 80 C8 42,实际的数为:42C88000H,它即为+100.25 在计算机内的单精度存储形式,而紧接着的 4 字节,即显示为:00 80 95 C2,实际的数为:C2958000H,它即为-74.75 在计算机内的存储形式。接下来的两个浮点数每个占 8 字节,分别显示为:00 00 00 00 00 10 59 40 和 00 00 00 00 00 B0 52 C0,实际的两个双精度的实数为:40 59 10 00 00 00 00 00H 和 C0 52 B0 00 00 00 00 00H,分别为+100.25 和-74.75。

⑤ 将所查到的数的存储形式和按 IEEE754 标准算出的形式比较,观察结果是否一致。

**五、项目运行**

按表 2.13 格式在项目报告中记录运行结果。

表 2.13 项目运行结果

| 数值 | 单精度内存表示 | 单精度 IEEE754 表示 | 双精度内存表示 | 双精度 IEEE754 表示 |
|---|---|---|---|---|
| +100.25 | | | | |
| −74.75 | | | | |
| +64.5 | | | | |
| −255.125 | | | | |

**六、目标检验**

学生自查：对照所学理论知识，计算整数的各种表示方式，并与运行结果对比。

教师检查：按照标准答案以及学生撰写项目报告情况给出项目成绩。

# 教学效果测评

本单元将通过作业、期中考试、项目报告、项目考试以及期末考试进行效果测评，其中，作业如下。

**一、单项选择题**

**1. 以下叙述错误的是_____。**

A. 八进制数据逢八进一

B. 任何进制的数据都有基数和各位的“位权”

C. 表示信息的数字符号称为代码

D. 二进制数据的加减运算规则与逻辑加运算规则相同

**2. 十进制分数 27/64 的十六进制数表示为_____。**

A. 0.011011H　　B. 0.33H　　C. 0.63H　　D. 0.6CH

**3. 八进制数中的 1 位对应于二进制数的_____。**

A. 2 位　　B. 3 位　　C. 4 位　　D. 5 位

**4. 假定下列字符码中有奇偶校验位，但没有数据错误，采用偶校验的字符码是_____。**

A. 11001011　　B. 11010110　　C. 11000001　　D. 11001001

**5. 下列叙述正确的是_____。**

A. 原码是表示无符号数的编码方法

B. 对一个数据的原码的各位取反而且在末位再加 1 就可以得到这个数据的补码

C. 定点数表示的是整数

D. 二进制数据表示在计算机中容易实现

**6. 在余三码表示的二—十进制数中，代码 0011 代表十进制数的_____。**

A. 2　　B. 3　　C. 0　　D. 4

**7. 十进制小数转换成十六进制数可采用_____。**

A. 除基(10)取余法　　B. 除基(16)取余法

C. 乘基(10)取整法　　D. 乘基(16)取整法

8. 二进制数 1001101B 的十进制数表示为_____。

A. 4DH　　B. 95D　　C. 77D　　D. 9AD

9. 目前的计算机，从原理上讲_____。

A. 指令以二进制形式存放，数据以十进制形式存放

B. 指令以十进制形式存放，数据以二进制形式存放

C. 指令和数据都以二进制形式存放

D. 指令和数据都以十进制形式存放

10. 根据国标规定，每个汉字在计算机内占用_____存储。

A. 一个字节　　B. 二个字节　　C. 三个字节　　D. 四个字节

11. 设 X＝－0.1011，则$[X]_{补}$为_____。

A. 1.1011　　B. 1.0100　　C. 1.0101　　D. 1.1001

12. 十进制数 2000 化成十六进制数是_____。

A. $(7CD)_{16}$　　B. $(7D0)_{16}$　　C. $(7E0)_{16}$　　D. $(7F0)_{16}$

13. 下列数中最大的数是_____。

A. $(10011001)_2$　　B. $(227)_8$　　C. $(98)_{16}$　　D. $(152)_{10}$

14. _____表示法主要用于表示浮点数中的阶码。

A. 原码　　B. 补码　　C. 反码　　D. 移码

15. 在小型或微型计算机里，普遍采用的字符编码是_____。

A. BCD 码　　B. 16 进制　　C. 格雷码　　D. ASCⅡ码

16. 一个 8 位的二进制整数，若采用补码表示，且由 3 个“1”和 5 个“0”组成，则最小值为_____。

A. －127　　B. －32　　C. －125　　D. －3

17. 下列数中最小的数是_____。

A. $(100101)_2$　　B. $(50)_8$　　C. $(100010)_{BCD}$　　D. $(625)_{16}$

18. 下列数中最大的数为_____。

A. $(10010101)_2$　　B. $(227)_8$　　C. $(76)_8$　　D. $(143)_{10}$

19. 用 16 位字长（其中 1 位符号位）表示定点补码整数时，所能表示的数的绝对值范围是_____。

A. $[\ 0,2^{16}-1\ ]$　　B. $[\ 0,2^{15}-1\ ]$　　C. $[\ 0,2^{14}-1\ ]$　　D. $[0,2^{15}]$

20. 用 32 位字长（其中 1 位符号位）表示定点原码小数时，所能表示的数的绝对值范围是_____。

A. $[0,1-2^{-32}]$　　B. $[0,1-2^{-31}]$　　C. $[0,1-2^{-30}]$　　D. $[0,1]$

21. 已知 X 为整数，且$[X]_{补}$＝10011011，则 X 的十进制数值是_____。

A. ＋155　　B. －101　　C. －155　　D. ＋101

22. 某机字长 32 位，其中 1 位符号位，31 位表示数值。若用定点小数表示，则最大正小数为_____。

A. $+(1-2^{-32})$　　B. $+(1-2^{-31})$　　C. $2^{-32}$　　D. $2^{-31}$

23. 若浮点数用补码表示，则判断运算结果是否为规格化数的方法是_____。

A. 阶符与数符相同为规格化数

B. 阶符与数符相异为规格化数

C. 数符与尾数小数点后第一位数字相异为规格化数

D. 数符与尾数小数点后第一位数字相同为规格化数

**24. 定点 16 位字长的字，采用补码形式表示时，一个字所能表示的整数范围是_____。**

A. $-2^{15}\sim+(2^{15}-1)$　　B. $-(2^{15}-1)\sim+(2^{15}-1)$

C. $-(2^{15}+1)\sim+2^{15}$　　D. $-2^{15}\sim+2^{15}$

**25. 定点 8 位字长的字，采用补码表示时，一个字所能表示的整数范围是_____。**

A. $-128\sim+127$　　B. $-127\sim+127$

C. $-129\sim+128$　　D. $-128\sim+128$

**26. 在机器数_____中，零的表示形式是唯一的。**

A. 原码　　B. 补码　　C. 补码和移码　　D. 反码

**27. 文本文档 test. txt 的内容如下（前三行末尾有回车换行符），则该文档的大小为_____。**

abc
ABC
012
中国人

A. 12 字节　　B. 15 字节　　C. 18 字节　　D. 21 字节

**28. 假定下列字符码中有奇偶校验位，但没有数据错误，采用奇校验的字符码有_____。（四个数为①10011010　②11010000　③11010111　④10111100）**

A. ①③　　B. ①　　C. ②④　　D. ④

**29. 若浮点数的机器表示中，尾数用补码表示，则判断该浮点数是否为规格化的方法是尾数的最高数值位_____。**

A. 为 0　　B. 为 1　　C. 与尾符相反　　D. 与尾符相同

**30. 一种 BCD 码，其中 0～9 的编码为 0000，0001，0010，0011，0100，0101，0110，0111，1110，1111，则各位的权值为_____。**

A. 8421　　B. 5421　　C. 4421　　D. 2421

**31. 如果存在三进制（逢 3 进 1），那么十进制中的数 7 在三进制中表示为_____。**

A. 21　　B. 7　　C. 30　　D. 03

**32. 16 进制数据 16. 5 化为八进制数据是_____。**

A. 26. 24　　B. 46. 5　　C. 75. 9　　D. 32. 88

**33. 在 GB2312－80 编码表中，“中”字位于第 54 区，第 48 位，则“中”字的内码为_____。**

A. B9FAH　　B. D6D0H　　C. C8CBH　　D. CBC8H

**34. 奇偶校验码可以检查出_____位错误。**

A. 1　　B. 2　　C. 6　　D. 8

**35. 二进制数 0101，左移两位所得到的十进制值为_____。**

A. 2　　B. 10　　C. 20　　D. 30

**36. 一个 8 位二进制数采用原码表示，加上 80H，得到的结果为_____。**

A. 数值增加 128　　B. 原数的相反数　　C. 原数的绝对值　　D. 没有意义

**37. 下列符号“v”“D”“3”“人”在计算机内各占用_____个字节。**

A. 1，2，3，4　　B. 1，1，3，2　　C. 1，1，1，2　　D. 1，1，2，2

**38. 在计算机内部用于汉字存储和运算等处理的信息代码为_____。**

A. 汉字机内码　　B. 汉字字形码　　C. 汉字输入码　　D. 汉字交换码

**39. 用 BCD 码表示一个 5 位十进制数，得到的 0,1 序列有_____位。**

A. 15　　B. 20　　C. 19　　D. 25

**40. _____的补码是将二进制位按位取反后在最低位上加 1。**

A. 正数　　B. 负数　　C. 浮点数　　D. 规格数

**41. 关于奇偶校验码说法正确的是_____。**

A. 只能检查出 1 位出错　　B. 能检查出偶数位数据出错

C. 能检查出奇数位数据出错　　D. 能自动纠正 1 位数据出错

## 二、填空题

1. 8 位二进制含 1 位符号位的定点补码小数的数值范围为_____。

2. 二进制数 100110 采用偶校验后的校验码是_____。

3. 表示一个带符号数的方法有原码表示法、补码表示法和_____表示法。

4. 设 A 为 8 位二进制寄存器，进行 A · 00001111→A 运算后，A 中_____，其余位不变。

5. 设 A 和 B 两个寄存器的内容进行或运算，若运算的结果是_____，那么 A、B 寄存器的内容必定为零。

6. 1 位十进制数，用 BCD 码表示需_____位二进制码，用 ASCII 码表示需_____位二进制码。

7. 一个定点数由_____和_____两部分组成，根据小数点位置不同，定点数有_____和_____两种表示方法。

8. 移码表示法主要用于表示浮点数的_____，有利于比较两个_____数的大小和进行_____操作。

9. 若$[x_1]_{补}=11001100$，$[x_2]_{原}=1.0110$，则数 $x_1$ 和 $x_2$ 的十进制数真值分别是_____和_____。

10. 字符信息是_____数据，国际上采用的字符系统是 7 位的_____码。

11. 已知数字 8 的 ASCII 码、8421 码和余三码，请写明它们是何种代码。

1000 _____，1011 _____，0111000 _____。

12. 将下列数由大到小排序：$(00101001)_{BCD}$，$(52)_8$，$(101001)_2$，$(233)_{16}$。_____

13. 已知 0 和 9 的 ASCII 码分别为 0110000 和 0111001，则 3 的 ASCII 码为_____，6 的 ASCII 码为_____。

14. 计算机中带符号数据表示常采用的格式有_____和_____两种。

15. 已知 x 的补码分别为 0.10100 和 1.10111，则其真值分别为_____和_____。

16. 一个汉字的机内码为 B4F3H，那么这个汉字的区位码为_____（用 16 进制表示）。

17. 0 的编码是唯一的码有_____和_____。

18. 已知一个数的移码是 11110111，那么它的原码是_____。

19. 采用 BCD 码表示的数据在运算时遵循_____进制的运算原则。

## 三、简答题

1. 设机器字长 32 位，定点表示，数值位 31 位，数符 1 位，问：

(1)定点原码整数表示时，最大正数是多少？最小负数是多少？

(2)定点原码小数表示时,最大正数是多少? 最小负数是多少?

2.求十进制数－113 的原码表示、反码表示、补码表示和移码表示形式(用 8 位二进制表示,并设最高位为符号位,数值为 7 位)。

3.将十进制数 $354\frac{5}{8}$ 转换成二进制数、八进制数、十六进制数和 BCD 数。

4.设机器字长为 16 位,定点表示时,数值位 15 位,数符 1 位。

(1) 定点原码整数表示时,最大正数为多少? 最小负数为多少?

(2) 定点原码小数表示时,最大正数为多少? 最小负数为多少?

5.设二进制浮点数的阶码是 3 位,阶符 1 位,尾数 6 位,尾符 1 位,写出它的最大正数、最大负数和最小负数(0 除外),并写出各自相应的十进制数的数值(用规格化数)。

6.将下列十进制数按阶码用移码、尾数用补码的格式写成浮点数的表示形式,其中阶码 3 位,阶符 1 位,尾数 6 位,尾符 1 位(用规格化数)。

(1)7.75　　(2)－7/128　　(3)726.55　　(4)－9.98

7.将 X＝－19/64 表示成定点数(8 位)及浮点规格化数(阶码 3 位,阶符 1 位,尾数 7 位,尾符 1 位),对于定点数请用原码、补码、反码的形式表示;对于浮点数请用原码、补码、反码以及阶码用移码,尾数用补码的形式表示。

8.设有两个正的浮点数 $N1=2^{e1}\cdot n1$ 和 $N2=2^{e2}\cdot n2$,

(1)若 e1＞e2,是否就有 N1＞N2?

(2)若尾数 n1,n2 都是规格化数,上述结论成立吗?

9.设某机字长为 8 位,X＝－0.00111B,要求用补码求下列机器数:

(1)$[X/2]_{补}$　　(2)$[X/4]_{补}$　　(3)$[2X]_{补}$

10.试比较下列各数对中两个数的大小:

(1)$(2001)_{10}$和$(2001)_{8}$

(2)$(4095)_{10}$和$(7776)_{8}$

(3)$(0.115)_{10}$和$(0.115)_{16}$

(4)$(0.625)_{10}$和$(0.505)_{8}$

11.校验码可以分为哪两类,各有什么特点?

12.已知 0 和 9 的 ASCII 码分别为 0110000 和 0111001,请分别写出 3,5,7 的 ASCII 码、8421 码及余三码。

13.奇校验和偶校验有哪些异同点?

14.在下表中空白处填入适当答案(采用 8 位二进制定点整数,最左 1 位为符 0 号位)。

| 真值 x(十进制) | 真值 x(二进制) | $[x]_{原}$ | $[x]_{反}$ | $[x]_{补}$ | $[x]_{移}$ |
|---|---|---|---|---|---|
| －127 | | | | | |
| －1 | | | | | |
| ＋0 | | | | | |
| 1 | | | | | |
| ＋127 | | | | | |

15.定点数表示中,小数点的位置如何约定?