第 2 章　计算机中的数据表示

计算机只认识二进制编码所表示的数据，也只能对二进制表示的数据进行加工和处理。为了使计算机更方便地对数据进行处理，就首先必须弄清楚各种各样的数据在计算机中是如何表示的，进而明了计算机硬件是如何对这些数据进行各种运算和加工的。本章将作为本书的基础知识，描述计算机中数据——包括数值数据和非数值数据的表示方法。

2.1 数据编码

2.1.1 数值数据的编码

2.1.1.1 概述

1. 进位计数制及其转换

常见的进位计数制有十进制、二进制、八进制和十六进制。

十进制数中共有 0～9 十个数码，其计数特点及进位原则为“逢十进一”。十进制的基数为 10，位权为 10^I(I 是整数)。十进制数后面常用字母 D 标记或者缺省。

计算机中常用的计数制还有二进制、八进制、十六进制。

二进制数中只有 0 和 1 两个数码，其计数特点及进位原则为“逢二进一”。二进制的基数为 2，位权为 2^I(I 是整数)。二进制数的后面常用字母 B 标记。

八进制数中共有 0～7 八个数码，其计数特点及进位原则为“逢八进一”。八进制的基数为 8，位权为 8^I(I 是整数)。八进制数的后面常用字母 O 标记。

十六进制数中共有 0～9、A、B、C、D、E、F 十六个数码，其计数特点及进位原则为“逢十六进一”。十六进制的基数为 16，位权为 16^I(I 是整数)。十六进制数的后面常用字母 H 标记。

任何一种进位计数制表示的数都可以写成按权展开的多项式之和，即任意一个 r 进制数 N 可表示为：

$$N_r = \sum_{i=m-1}^{-k} D_i \times r^i \qquad (2-1)$$

其中的 D_i 为该数制采用的基本数符，r^i 是权，r 是基数。

数值数据是表示数量多少和数值大小的数据，即在数轴上能找到其对应的点。

各种数值数据在计算机中表示的形式称为机器数。机器数对应的实际数值称为数的真值。

小数点位置固定的数称为定点数，又有无符号数和有符号数之分。

2. 无符号数及有符号数

⑴ 无符号数

所谓无符号数即没有符号的数，数中的每一位均用来表示数值。所以，8 位二进制无符号数所表示的数值范围是 0～255。而 16 位无符号数的表示范围为 0～65535。

⑵ 有符号数

对有符号数而言，由于机器是无法直接识别“+、－”（即正、“负）符号，但

由于“正”、“负”恰好是两种截然不同的状态，若用“0”表示“正”，用“1”表示“负”，则符号也被数字化了。并且规定将符号放在有效数字的前面，这样就组成了有符号数。

3. 定点数与浮点数

⑴ 定点数

在数据的机器表示中，若约定小数点的位置固定不变，则称为定点数。有两种形式的定点数：定点整数（纯整数，小数点定在最低有效数值位之后）和定点小数（纯小数，小数点在最高有效数值位之前）。具体表示形式如图 2.1 所示。

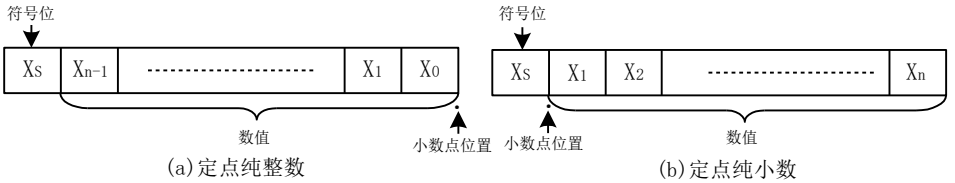

图2.1 有符号定点数的表示形式

⑵ 浮点数

基数为 2 的数 F 的浮点表示为：

$$F = M \times 2^{E} \qquad (2-2)$$

其中 M 称为尾数，E 称为阶码。

尾数为带符号的纯小数，阶码通常为带符号的纯整数。计算机中浮点数的一般表示格式如如图 2.2 所示。

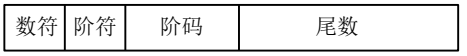

图2.2 浮点数的一般表示形式

有关有符号的定点数及浮点数的详细讨论将放在本章的后面。

2.1.1.2 原码

1. 定义

原码是机器数中最简单的一种表示形式，其符号位为 0 表示正数。符号位为 1 表示负数，数值位即真值的绝对值。

⑴整数原码的定义

根据图 2.1(a)，若整数用二进制 n 位表示，则整数原码的定义为：

$$[X]_{原} = \begin{cases} X & 2^{n-1} > X \geqslant 0 \\ 2^{n-1} - X & 0 \geqslant X > -2^{n-1} \end{cases} \qquad (2-3)$$

式中 X 为真值，n－1 为整数数值位的位数。

原码的构成可直接用定义实现；也可用符号位后面跟数的绝对值的方法实现。

例 1：当 X=+35 时，若用 8 位二进制编码的原码表示，则

$[X]_{原}$ =00100011

若 X=－35，同样用 8 位编码表示，则

$[X]_{原}$ =10100011

从上面的定义可以看到，符号位总是放在最高位。同时，原码表示又称作带符

号的绝对值表示，即在符号的后面跟着的就是该数据的绝对值。

⑵ 小数原码的定义

根据图 2.2(b)，若小数用二进制 n 位表示，则小数原码的定义为：

$$[X]_{原}=\begin{cases} X & 1>X\geqslant 0 \\ 1-X & 0\geqslant X>-1 \end{cases} \qquad (2-4)$$

根据式(2－4)，纯小数的原码表示可以表示为：

对于正数：$[X]_{原}=0.X_1X_2\cdots X_{n-1}$

对于负数：$[X]_{原}=1.X_1X_2\cdots X_{n-1}$

值得注意的是，在计算机中小数点是隐含的，也是不用表示的。上面表示式中的小数点纯属强调小数点的位置。

例 2：若纯小数 X＝0.46875，用包括符号位为 8 位的定点原码表示，则可表示为：

$[X]_{原}$ =0.0111100

若纯小数 X＝－0.46875，用包括符号位的定点原码表示，则可表示为：

$[X]_{原}$ =1.0111100

数值原码表示法简单直观，但加减运算却很麻烦。同时，对于数值 0，用原码表示则不是唯一的，有两种表示形式，以 8 位原码表示的 0 为：

$[+0]_{原}$=0.0000000 或$[+0]_{原}$=00000000

$[-0]_{原}$=1.0000000 或$[-0]_{原}$=10000000

可见$[+0]_{原}$不等于$[-0]_{原}$，这就是原码中的“零”有两种表示形式。

利用上述定义，原码 n 位(包括一位符号位)整数及纯小数所能表示的数值范围分别为：$-(2^{n-1}-1)\sim+(2^{n-1}-1)$和$-(1-2^{-(n-1)})\sim+(1-2^{-(n-1)})$。

2.1.1.3 补码

1. 补数的概念

在日常生活中，常会遇到“补数”的概念。如时钟指示 6 点，欲使它指示 3 点，即可按顺时针方向将分针转 9 圈，也可按逆时针方向将分针转 3 圈，结果是一致的。假设顺时针方向转为正，逆时针方向转为负，则有

$$\begin{array}{r} 6 \\ -3 \\ \hline 3 \end{array} \qquad \begin{array}{r} 6 \\ +9 \\ \hline 15 \end{array}$$

由于时钟的时针转一圈能指示 12 个小时，这“12”在时钟里是不被显示而自动丢失的，即 15-12=3，故 15 点和 3 点均显示 3 点。这样-3 和+9 对时钟而言其作用是一致的。在数学上称 12 为模，写作 mod 12，而称+9 是-3 以 12 为模的补数。

将补数的概念用到计算机中，便出现了补码这种机器数。

2. 补码的定义

⑴ 整数补码的定义

同样根据图 2.1(a)，若整数用二进制 n 位表示，则整数补码的定义为：

$$[X]_{补}=\begin{cases} X & 2^{n-1}>X\geqslant 0 \\ 2^{n}+X & 0>X\geqslant -2^{n-1} \end{cases} \quad (\mathrm{mod}\,2^{n}) \qquad (2-5)$$

由式(2－5)可以看到，对正数来说，补码与原码的定义完全一样。同样的例子，假定 X=+35 时，若用 8 位二进制编码的补码表示，则

$[X]_{补}$ =00100011

但是，对负数而言，两者是不同的。现仍以 X=－35 为例，可以利用式(2－5)向定义来求得该数的 8 位补码表示。但这种方法相对比较麻烦。简单的方法有如下几种：

①将 X=＋35 的原码表示，包括符号位在内各位取反，再在最低位上加 1。

②将 X=－35 的原码表示，不包括符号位各位取反，再在最低位上加 1。

③将 X=＋35 的原码表示，从最低位逐位向高位找起，找到第一个 1 不变，以后各位 1 变 0、0 变 1 直至符号位。

⑵ 小数补码的定义

小数用二进制 n 位表示，则小数补码的定义为

$$[X]_{补}=\begin{cases} X & 1>X\geqslant 0 \\ 2+X & 0>X\geqslant -1 \end{cases} \quad (\text{mod } 2) \qquad (2-6)$$

根据式(2－6)，纯小数的补码同样可以表示为：

对于正数：$[X]_{补}=0.X_1X_2\cdots X_{n-1}$

对于负数：$[X]_{补}=1.X_1X_2\cdots X_{n-1}$ $10.00\text{---}0-0.X_1X_2\text{---}X_{n-1}$ (mod2)

同样，小数点是隐含的。

例 3：若纯小数 X＝0.46875，用包括符号位的 8 位定点补码表示，则可表示为：

$[X]_{补}$ =0.0111100

可以看到，对于正数，补码纯小数表示与原码是一样的。对于负数纯小数，构成其补码表示形式所采用的方法与整数一样。也就是说，上面所提到的四种方法均可使用。因此，当 X＝－0.46875 时，用包括符号位的 8 位定点补码表示，则可表示为：

$[X]_{补}$ =1.1000100

通过上述说明，n 位补码表示的整数数值范围为：$-2^{n-1}\sim+(2^{n-1}-1)$。n 位补码表示的小数数值范围为：$-1\sim(1-2^{-n+1})$。

3. 补码的特点

⑴ 0 的表示是唯一的

$[+0]_{补}$ =0.0000000

$[-0]_{补}$ =2+（-0.0000000）=10.0000000-0.0000000=0.0000000

显然$[+0]_{补}$ =$[-0]_{补}$=0.0000000，即补码中的“零”只有一种表示形式。

⑵ 变形码

当模数为 4 时，形成了双符号位的补码，如 X=－0.1001，对（mod 2^2）而言。

$[X]_{补}=2^2+X$=100.0000000-0.1001000=11.0111000

这种双符号位的补码又叫做变形补码，它在阶码运算和溢出判断中，有其特殊作用。

⑶ 求补运算

在许多处理器中都设置求补指令，即对操作数求其补码。具体运算就是将操作数，包括符号位在内，各位取反再在最低位上加上 1。我们注意到一个正数求补就变做负数，例如，对＋68 求补，其结果必为－68。就如同上面对＋35 求补就得到－35 的结果一样。同样，对－68 求补，其结果必为＋68。

求补运算可用上面所描述的四种方法的任一种来实现，当然以简单快速为好，下章中将有具体的实现方案。

⑷ 简化加减法

利用补码实现两数相加是很方便的，补码加法的运算法则为：

$$[X+Y]_{补}=[X]_{补}+[Y]_{补} \qquad (2-7)$$

由式（2－7）可以看到，两数和的补码就等于两数补码之和。

对于减法，正如上面所描述的，对补码求补就相当于在其前面加了一个负号。也就是说 $[[X]_{补}]_{求补}=[-X]_{补}$；$[[-X]_{补}]_{求补}=[X]_{补}$。有了这样的特性，就使得减法运算完全可以用加法来实现，即

$$\begin{aligned}[X-Y]_{补}&=[X]_{补}+[-Y]_{补}\\&=[X]_{补}+[[Y]_{补}]_{求补}\end{aligned} \qquad (2-8)$$

可见，利用补码，减法运算可用加法来实现。这也是所有常见的处理器中上设置加法器不设置减法器的原因。从而简化了处理器的结构，现举例说明。

例 4：欲求 68－35＝？

可以将上式写作：68＋（－35）＝Z，则

$$[Z]_{补}=[68]_{补}+[(-35)]_{补}=01000100+11011101=00100001$$

所获得的结果正是 33。

⑸ 算术或逻辑左移

对于用补码表示的数据，只要没有超出所规定的数值范围，每算术或逻辑左移一次，即各位顺序向左移一位，最高位移出，最低位补进一个 0。相当于该数据乘 2。但必须注意前题条件。

⑹ 算术右移

算术右移规定保持最高位（即符号位）不变，并将包括最高位的数据顺序右移一位，最低位移出。

补码表示的数据，每算术右移一次相当于除 2。

2.1.1.4 反码

反码通常用来作为由原码求补码或者由补码求原码的中间过渡。

1. 反码的定义

⑴ 整数反码的定义

$$[X]_{反}=\begin{cases}X & 2^{n-1}>X\geqslant 0\\ (2^{n}-1)+X & 0\geqslant X>-2^{n-1}\end{cases} \quad (\mathrm{mod}\ (2^{n}-1)) \qquad (2-9)$$

整数反码的定义为：

由式(2－9)可以看到，正整数的反码表示与原码及补码表示是相同的。对于负数可直接利用(2－9)式来获得。也可以将设负数绝对值相同的正数原码包据符号位各位取反获得。还可以用该负数的原码表示，保持符号位不变，其余各位取反来获得。也许这就是反码的由来。

例 5：若 X＝35，其 8 位反码表示为：

$$[X]_{反}=00100011$$

若 X＝－35，其反码表示为：

$[X]_{反}$ =11011100

⑵ 小数反码的定义

小数反码的定义为：

$$[X]_{反}=\begin{cases} X & 1>X\geqslant 0 \\ (2-2^{-n+1})+X & 0\geqslant X>-1 \end{cases} \quad (\mathrm{mod}\ (2-2^{-n+1})) \qquad (2-10)$$

2. 特点

⑴ 0 的表示

在数值的反码表示中，0 同样有两种表示形式，用 8 位表示如下：

$[+0]_{反}$=0.0000000＝00000000

$[-0]_{反}$=1.1111111＝11111111

⑵ 反码与补码的关系

从反码及补码的定义可以看到：

$[X]_{反}=2-2^{-n+1}+X$

$[X]_{补}=2+X$

可见，

$$[X]_{补}=[X]_{反}+2^{-n+1} \qquad (2-11)$$

式(2－11)进一步验证了前面的结论：只要在某数值的反码的最低位上加 1 即可获得该数值的补码。

⑶ 数值范围

n 位反码表示的整数数值范围为：$-(2^{n-1}-1)\sim+(2^{n-1}-1)$

n 位反码表示的小数数值范围为：$-(1-2^{-(n-1)})\sim+(1-2^{-(n-1)})$

移码

1. 移码的由来

有符号数在计算机中除了用原码、补码和反码表示外，还用另一种机器数——移码表示，由于它的一些突出的优点，目前已被广泛采用。

当真值用补码表示时，，由于符号位和数值部分一起编码，与习惯上的表示法不同，因此人们很难从补码的形式上直接判断其真值的大小，例如：

十进制数 X=＋31,对应的二进制数为+11111，若用 8 位表示，则$[x]_{补}$=00011111;

十进制数 X=－31,对应的二进制数为-11111，若用 8 位表示，则$[x]_{补}$=11100001;

上述补码表示中，从代码形式看，符号位也是一位二进制数。按这 8 位二进制代码比较其大小的话，会得出 11100001＞00011111， 其实恰恰相反。

如果我们对每个真值加上一个 2^n(n 为整数的位数，此处可为 7，即 8－1)，情况就发生了变化，如：

X=00011111 加上 2^7 可得 10011111；

X=11100001 加上 2^7 可得 01100001；

比较它们的结果可见，10011111＞01100001。这样一来，从 8 位代码本身就可看出真值的实际大小。

在上面的例子中，实际上是在原来补码表示的编码上，加上一个偏移量。例子中，由于编码长度为 8 位，使用的偏移量为 2^7 。这或许是移码的由来。

2. 移码的定义

由于移码多用于浮点数中表示阶码，均为整数，只介绍定点整数的移码表示。当用包括符号位为 n 位字长时，整数移码的定义为：

$[X]_{移}=2^{n-1}+X \quad (2^{n-1}>X\geqslant -2^{n-1}) \quad (\mathrm{mod}\ 2^n)$ (2－12)

要获得整数的移码表示，可以利用定义来实现。也可以先求出该数的补码表示，而后将符号位取反。仍以前面的例子，当 X＝35，其 8 位字长移码表示为 ：

先求出 $[X]_{补}=00100011$，再将其符号位取反，即$[X]_{移}=10100011$。

也可以直接用定义，此时的 $2^{n-1}=10000000$，X＝00100011，则$[X]_{移}=2^{n-1}+X=10100011$。

若 X＝－35，先求出 $[X]_{补}=11011101$，则$[X]_{移}=01011101$。

用定义求，则$[X]_{移}=2^{n-1}+X=10000000+11011101=01011101$。

3. 特点

⑴ 移码就是在真值上加一个常数 2^{n-1} 。在数轴上移码所表示的范围恰好对应与真值在数轴上的范围向轴的正方向移动 2^{n-1} 个单元，如图 2.3 所示。

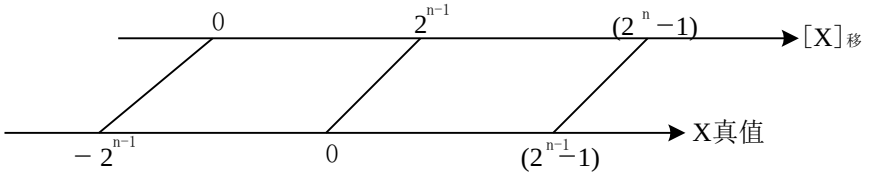

图2.3 移码在数轴上的表示

⑵ 移码与补码的关系

移码与补码的关系可用图 2.4 表示。

由图 2.4 可以看到，补码与移码间的关系十分密切，只要将补码的符号位取反，则补码就转换成了相应的移码；同样，只要将移码的符号位取反，则移码就转换成了相应的补码。

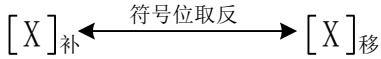

图2.4 补码与移码的关系

进而可以想到，只要字长相同，补码与移码所能表示的数值范围是相同的。

⑶ 在这里再强调说明，移码码值的大小就反映了数值的大小。因此，正数移码的码值一定大于负数移码的码值。也就是说，大码值所表示的数值一定大于小码值所表示的数值。

2.1.1.6 不同编码码比较

原码表示很直观。若采用原码作乘除运算，可取其绝对值(原码的数值部分)直接运算，并按同号相乘除取正、异号相乘除取负的原则，单独处理符号位，比较方便。但原码加减运算时，其运算比较复杂。例如，当两个操作数符号不同且要作加法运算时，先要判断两数绝对值大小，然后将绝对值大的数减去绝对值小的数，结果的符号以绝对值大的数为准。运算步骤既复杂又费时，而且本来是加法运算却要用减法器实现。而机器数采用补码时，就能找到一个与负数等价的正数来代替该负数，就可把减法操作用加法代替，这样在计算机中就可以只设加法器，但是根据补码的定义，在形成补码的过程中又出现了减法，因此，引入了反码，作为由原码求补码或者由补码求原码的中间过渡，这样由真值通过原码求补码就可避免减法运算。

因此，原、反、补码三种机器数的特点可归纳如下：

① 当真值为正时，原码、补码和反码的表示形式均相同，即符号位用“0”表示，数值部分与真值相同。

② 当真值为负时，原码、补码和反码的表示形式不同，但其符号位都用“1”表示，而数值部分有如下关系，即补码是原码的“求反加 1”，反码是原码的“每位取反”。

表 2.1 列出了 8 位字长中所有二进制代码组合与无符号数及定点整数原码、补码和反码所代表真值的对应关系。

表 2.1　8 位不同编码对应的真值范围

二进制 代码	无符号数 对应的真值	原码对应 的真值	补码对应 的真值	反码对应 的真值
00000000	0	+0	+0	+0
00000001	1	+1	+1	+1
00000010	2	+2	+2	+2
⋮	⋮	⋮	⋮	⋮
01111110	126	+126	+126	+126
01111111	127	+127	+127	+127
10000000	128	-0	-128	-127
10000001	129	-1	-127	-126
10000010	130	-2	-126	-125
⋮	⋮	⋮	⋮	⋮
11111101	253	-125	-3	-2
11111110	254	-126	-2	-1
11111111	255	-127	-1	-0

2.1.2　数据的浮点表示

在前一小节中，已经详细地讨论了有关定点数的编码问题，本小节将说明浮点数的表示方法。有关定点数和浮点数的运算将留待下一章讨论。

2.1.2.1　浮点数的表示方法

1. 概述

实际上，计算机中处理的数不一定是纯小数或纯整数，而且有些数据的数值范围相差很大（如电子的质量 9×10^{-28} 克，太阳的质量 2×10^{33} 克），它们都不能直接用定点小数或定点整数表示，但均可用浮点数表示。浮点数即小数点的位置可以浮动的数，如

$352.47=3.5247\times10^{2}=3524.7\times10^{-1}=0.35247\times10^{3}$

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。浮点数被表示成一般形式为

$$F=M\times R^{E} \qquad (2-13)$$

式中 M 为尾数（可正可负），E 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。

当基数 R=2 时，式(2－13)就变成了式(2－2)。此时，数 F 可写成下列不同形式：

$F=11.0101=0.110101\times2^{10}=1.10101\times2^{1}=1101.01\times2^{-10}=0.00110101\times2^{100}=\cdots$

2. 浮点数的表示

浮点数在机器中的形式有两种：①如前图 2.2 所示的形式。②如下图 2.6 所示的形式。至于采用哪种形式，是由计算机的设计人员所决定。能够对浮点数进行处理

的处理器称为浮点处理器。

可以看到，浮点数的两种表示形式本质上是一样的。不同的仅仅是规定数值的符号位规定的位置不同。

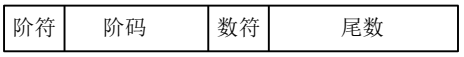

图2.6 浮点数的另一种表示形式

2.1.2.2 浮点数的所表示的数值范围

浮点数可分为非规格化浮点数和规格化浮点数，它们有一些不同，下面分别讨论。

1. 非规格化浮点数

由浮点数的表示中可以看到，浮点数由阶码 E 和尾数 M 两部分组成。需要强调的是：①阶码是整数，阶符和阶码的位数 k 合起来决定浮点数的表示数值范围也就是决定了所表示的数值的大小。阶符决定阶码的正负。②尾数是小数，其位数 n 主要用于决定浮点数的精度；③尾数的符号表示浮点数的正负。

浮点数的表示范围：

以通式 $F=M\times R^E$ 为例，设浮点数的基值 R=2；阶码的数值位取 k 位，阶符 1 位且采用补码表示；尾数的数值位取 n 位，尾符 1 位，同样采用补码表示。当浮点数为非规格化数时，可先分别求出阶码和尾数的表示范围。

阶码的最小值为：-2^k ，阶码的最大值为：(2^k-1)。

尾数的最小负值为：－1，尾数的最大负值为：-2^{-n} ；

尾数的最小正值为：$+2^{-n}$ ，尾数的最大正值为：$+(1-2^{-n})$ 。

根据上面的分析，此非规格化浮点数在数轴上的表示范围如图 2.7 所示。

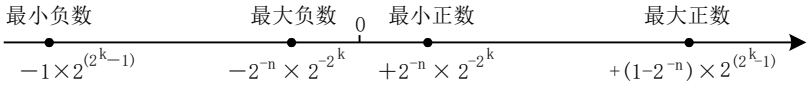

图2.7 非规格化浮点数的数值范围

2. 规格化浮点数

在计算机中，为了充分利用尾数的二进制编码表示更多的有效数字，为了使浮点数有统一的表示形式，也为了使浮点保持更高的精度。通常，浮点数采用规格化形式来表示。

对浮点数的规格化，就是将尾数的绝对值限定在一个规定的数值范围内。当基值为 2 时，规格化浮点数尾数的绝对值应在 1/2～1 之间。使尾数的绝对值在此范围内是可以通过改变小数点的位置（相应地改变阶码）便可以做到。

当尾数 M 用补码表示，当 M≥0 时，规格化尾数的形式必须为：

$$M=0.1XXXX\cdots X \qquad (2-14)$$

式(2－14)中，X 为任意二进制值，0 或 1 皆可。

当 M＜0 时，规格化尾数的形式必须为：

$$M=1.0XXXX\cdots X \qquad (2-15)$$

同样，式(2－15)中，X 为任意二进制值，0 或 1 皆可。

根据上面对规格化浮点数的定义，可以得到规格化尾数的数值范为。

尾数的最小负值为：－1，尾数的最大负值为：（$1/2+2^{-n}$）；

尾数的最小正值为：＋1/2 ，尾数的最大正值为：$+(1-2^{-n})$ 。

对于规格化浮点数来说，其阶码所表示的数值范围与非规格化浮点数是一样的。

因此，可以确定规格化浮点数所能表示的数值范围如图 2.8 所示。

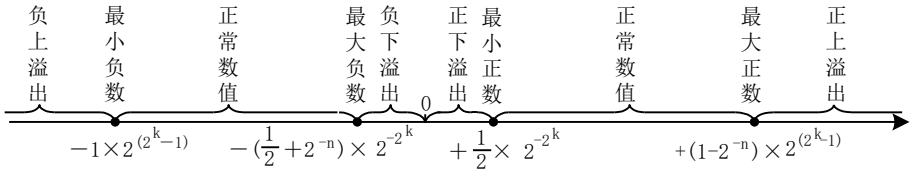

图2.8 规格化浮点数的数值范围

比较图 2.8 和 2.7，可以发现非规格化浮点数和规格化浮点数所能表示的数值范围主要不同是绝对值最小的有效数值。

由图 2.8 可见，规格化浮点数的数值范围：

最大正数为： $+(1-2^{-n})\times2^{(2^k-1)}$

最小正数为： $+\frac{1}{2}\times2^{-2^k}$

最大负数为： $-(\frac{1}{2}+2^{-n})\times2^{-2^k}$

最小负数为： $-1\times2^{(2^k-1)}$

当浮点数阶码大于最大阶码时，称为“上溢”，此时机器停止运算，进行中断溢出处理；当浮点数阶码小于最小阶码时，称为“下溢”，此时“溢出”的数绝对值很小，通常将尾数各位强置为零，按机器零处理，此时机器可以继续运行。图 2.8 中表示了浮点数所能表示的数值范围及溢出的情况。

一旦浮点数的位数确定后，合理分配阶码和尾数的位数，直接影响浮点数的表示范围和精度。

从上面的讨论可以看到，利用数值的浮点数表示，可将有限字长的二进制编码表示更大的数值范围。例如，16 位二进制编码：

无符号数的范围为：0～65535。

补码定点整数的范围为：－32768～＋32767。

对于浮点数，可有多种方案。假定阶符 1 位、阶码 6 位，数符 1 位、尾数 8 位。若阶码用移码表示，尾数用补码表示。则该浮点数所能表示的数值范围是：

-2^{63} ～＋（$1-2^{-8}$）$\times2^{63}$

可见，同样字长的浮点数所能表示的数值范围要大得多。

3. 规格化

浮点数在进行运算前和运算后，必须对其尾数规格化，使其成为规格化数。当尾数不是规格化数时，就要通过修改阶码并同时左右移尾数的办法，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，规格化数依据式(2－14)和(2－15)。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）。

其他基数的规格化过程不做说明。

4. 定点数和浮点数的比较：

定点数和浮点数可从如下几个方面进行比较：

①当浮点机和定点机中的数其位数相同时，浮点数的表示范围比定点数大得多。

②当浮点数为规格化数时，其精度远比定点数高。

③浮点数运算要分阶码部分和尾数部分，而且运算结果都要求规格化，故浮点运算步骤比定点运算步骤多，运算速度比定点低，运算线路比定点复杂。

④在溢出的判断方法上，浮点数是对规格化数的阶码进行判断，而定点数是对数值本身进行判断。如定点小数其绝对值必须小于 1，否则即“溢出”，此时要求机器停止运算，进行处理。为防止溢出，运算前必须选择好合适的比例因子，但这项工作做起来比较麻烦，给编程带来不便。而浮点数的表示范围远比定点数大，仅当“上溢”时机器才停止运算，故一般不必考虑比例因子的选择。

综之，浮点数在数的表示范围、数的精度、溢出处理和程序编程方面（不取比例因子）均优于定点数。但在运算规则、运算速度及硬件成本方面又不如定点数。因此，究竟选用定点数还是浮点数，应根据具体应用综合考虑。一般来说，通用的大型计算机大多采用浮点数，或同时采用定、浮点数；小型、微型及某些专用机、控制机则大多采用定点数。当需要作浮点运算时，可通过软件实现，也可外加浮点扩展硬件（如协处理器）来实现。现举例说明。

例 6： 将十进制数 X=+13/128 写成二进制定点数和浮点数（尾数部分取 7 位，阶码部分取 7 位，阶符和数符各取 1 位，阶码采用移码，尾数用补码表示），分别写出该数的定点数和浮点数的表示形式。

解：令 x=+13/128

其二进制形式：　　X=0.0001101

定点数表示：　　X=0.0001101

浮点数规格化表示：$X=0.1101000\times 2^{-11}$

定点表示：　$[X]_{原}=[X]_{补}=[X]_{反}=0.00011011$

浮点表示形式如图 2.9 所示：

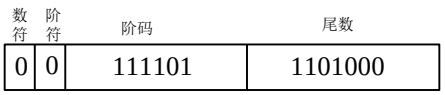

图2.9 例中浮点数的表示形式

例 7： 设浮点数字长为 16 位，其中阶码为 6 位（含一位阶符），尾数为 10 位（含一位数符），写出 X= –(53/512)对应的浮点规格化数阶码用移码，尾数用补码的形式。解：

$X= -(53/512)= -0.000110101= 2^{-11}\times(-0.110101000)$

尾数的规格化补码形式为：1.001011000

阶码的移码表示为：011101

该数用另一种浮点表示形式如图 2.10 所示。

图2.10 例2浮点数的表示形式

值得注意的是，当一个浮点数尾数为 0 时，不论其阶码为何值；或阶码等于或小于它所能表示的最小数时，不管其尾数为何值，机器都把浮点数当作零看待，并称之为“机器零”。如果浮点数的阶码用移码表示，尾数用补码表示，则当阶码为它所能表示的最小数-2^k(式中 k 为阶码的位数)且尾数为 0 时，其阶码（移码）全为 0，尾数（补码）也全为 0，这样的机器零为 0000…0000 全零表示，有利于简化机器中

判“0”电路。

2.1.2.3　IEEE－754 标准

1. 工业标准 754 概述

1985年，IEEE发表了一份关于单精度和双精度浮点数的浮点表示标准，这个标准官方称为IEEE-754（1985）。以后又不断加以发展，SUN公司于2005年推出《数值计算指南》(中译本名)，对该标准进行了更加详细和深入地讨论，给出了多种格式及程序。因此，《数值计算指南》更加全面也非常实用。IEEE－754标准已获得了广泛的认可，并已用于当前所有各类处理器和浮点协处理器中。

IEEE－754 规定了单精度和双精度两种基本的浮点格式，以及双精度扩展等多种浮点格式。常用的 IEEE－754 格式参数如表 2.2 所示。

表 2.2　常用的 IEEE－754 格式

参　数	单精度浮点数	双精度浮点数	双精度扩展浮点数
浮点数长度（bit）	32	64	80
尾数长度 p（bit）	23	52	64
符号位 s	1	1	1
指数 E 的长度（bit）	8	11	15
最大指数 Emax	+127	+1023	+16383
最小指数 Emin	-126	-1022	-16382
指数偏移量	+127	+1023	+16383
可表示的实数范围	10^{-38}～10^{+38}	10^{-308}～10^{+308}	10^{-4932}～10^{+4932}

需要说明的是，在 IEEE－754 标准的具体规定中，还有多种形式。在本小节中只对该标准做最简单地介绍，读者欲了解更详细的细节，可查阅相关文献资料。

IEEE－754标准的表示形式如下：

$$（-1）^{S}\ 2^{E}（b_0 \blacklozenge b_1b_2b_3\cdots b_{p-1}）\qquad (2-16)$$

其中：$（-1）^{S}$该浮点数的数符，当s为0时表示为正数，s为1时为负数；

E为指数，用移码表示；

$（b_0 \diamond b_1b_2b_3\cdots b_{p-1}）$为尾数，共P位，用原码表示。

在IEEE－754标准中，特别要说明的就是尾数在规格化时的处理。也就是说在规格化的过程中必须使b_0为1，而且小数应当在◆位置上，是隐含的。规格化时将b_0去掉，也是隐含的。这相当于使尾数增加了一位，在使用时应注意到这种情况。

2. 单精度格式

⑴ 格式

在这里仅介绍最基本的IEEE－754标准的单精度格式，其格式如图2.11所示。

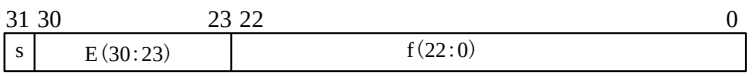

图2.11 IEEE-754单精度浮点数的格式

由图2.11可以看到，754所定义的浮点数由三个字段构成：s为数符为1位，f为尾数为23位，阶码为8位(含1位阶符)。

值得强调的是，IEEE－754中阶码采用移码，正如表2.2所示，对单精度浮点数来说，移码的偏移量不是前面所提到的2^7(+128)而是(2^7-1)即+127。同时，规定尾数用原码表示，规格化时b_0必须为1而且应隐去。有关IEEE－754标准的单精度格式的

详细规定见表2.3。

表 2.3　IEEE－754 单精度格式位模式表示的值

单精度格式位模式	IEEE 浮点数的值
0＜e＜255	$(-1)^s\times2^{e-127}\times1.f$　（正规数）
e＝0；f≠0（f 中至少有一位不为零）	$(-1)^s\times2^{-126}\times0.f$　（次正规数）
e＝0；f＝0（f 的所有位均为零）	$(-1)^s\times0.0$　（有符号的零）
s＝0；e＝255；f＝0（f 的所有位均为零）	＋INF　（正无穷大）
s＝1；e＝255；f＝0（f 的所有位均为零）	－INF　（负无穷大）
s＝u；e＝255；f≠0（f 中至少有一位不为零）	NaN　（非数值）

由表 2.3 可以看到，正规数尾数有效数字的前导位（小数点左侧的位 b_0）为 1，与 23 位尾数一起提供了 24 位的精度。 u 为无关。

次正规数有效数字的前导位为 0。在 IEEE－754 标准中，单精度格式次正规数也称为单精度格式非规格化数。

⑵ 说明

根据上述描述，可以得到 IEEE－754 标准的单精度浮点数的结论：

①由于规定 E＝e－127，并且 0＜e＜255。故阶码 E 的正常值应为其真值-126～+127 或 e 为偏移后的值，规定 e 在+1～+254 为正规数。

②规格化数为：$N=(-1)^s\times2^{e-127}\times(1.f)$　　(2－17)

③所能表示的正数范围：$N=+2^{+127}\times(1+1-2^{-23})\sim+2^{-126}\times(1+0)$

所能表示的负数范围：$N=-2^{+127}\times(1+1-2^{-23})\sim-2^{-126}\times(1+0)$

④当 e=0 或 e=255 时，在 IEEE－754 标准中表示特殊的数。

⑶ 举例

为了说朋IEEE7－54浮点数的应用，现举例如下。

例8：现欲利用IEEE754标准将数176.0625表示为单精度浮点数。

首先将该十进制来转换成二进制数：

$$（176.0625）_{10}＝（10110000.0001）_2$$

对上面二进制数规格化：$10110000.0001＝1◆01100000001\times2^7$

这就保证了使b_0为1，而且小数点在◆位置上。将b_0去掉并扩展为单精度浮点数所规定的23位尾数：01100000001000000000000。

现在，再来求取阶码。现指数为7即真值，而单精度浮点数规定指数的偏移量为127（请注意不是前面移码描述中所提到的128），即在指数7上加127，也就是e＝7＋127＝134。即在00000111上加01111111，从而得到则指数的移码表示为10000110。

最后，将$（176.0625）_{10}$表示为IEEE－754标准的单精度浮点数：

0　10000110　01100000001000000000000

⑷ IEEE 754标准双精度浮点数的说明

在这里顺便将 IEEE 754 标准双精度浮点数做最简单说明，不再涉及更多的细节。

① 阶码 E 的正常值应为其真值-1022～+1023 编移+1023，即 e 为+1～+2043。

②规格化数为：$N=(-1)^s\times 2^{e-1023}\times(1.f)$ (2－18)

③所能表示的正数范围：$N=+2^{+1023}\times(1+1-2^{-52})\sim+2^{-1022}\times(1+0)$

所能表示的页数范围：$N=-2^{+1023}\times(1+1-2^{-52})\sim-2^{-1022}\times(1+0)$

④当 e=0 或 e=2047 时，在 IEEE 754 标准中表示特殊的数。

2.1.3 BCD 码

前面已经提到，计算机只能识别处理二进制数据，上面所描述的定点数、浮点数均是用二进制编码来表示。但在人们的日常生活中更多更习惯地是使用十进制数。因此，也就希望将十进制数引入计算机中。

在计算机中，采用 4 位二进制编码来表示 1 位十进制数。4 位编码有 16 中形式，从中取出 10 种来表示十进制数的 0～9 十个数字，因而有多种编码方案。下面简单说明常见的几种。

2.1.3.1 有权码

1. 定义

在这种实现方案中，对应的 1 位十进制数的 4 位二进制编码，每一位都有确定的权值。例如，在 8421 码的方案中，表示 1 位十进制数的 4 位二进制编码，从最高位到最低位的权值依次为 8、4、2、1。因此，可用 0000、0001、0010、…、1001 这十种编码分别表示十进制数 0、1、2、…、9。这就是用二进制编码来表示十进制数的一种编码，简称 BCD 码。而 4 位二进制编码 16 种编码中只用 10 种定义十进制数，剩下的 6 种编码对 BCD 而言是非法的，是不允许出现的。

有确定权值的 BCD 编码有多种方案，现列在如下表 2.4 中。

表 2.4 有权码的 BCD 编码

十进制数	8421 码	2421 码	5211 码	4311 码
0	0000	0000	0000	0000
1	0001	0001	0001	0001
2	0010	0010	0011	0011
3	0011	0011	0101	0100
4	0100	0100	0111	1000
5	0101	1011	1000	0111
6	0110	1100	1001	1011
7	0111	1101	1100	1100
8	1000	1110	1110	1110
9	1001	1111	1111	1111

在上表 2.4 中所列出的有权 BCD 码中，用得最多的是 8421 码。

2. 运算

由于计算机中是以二进制进行运算，当在进行 BCD 数运算时，运算的结果有可能出现未定义的非法数据，这必然导致结果的错误。要保证结果正确，则需要对运算的结果进行校正。

例 9：求两 8421BCD 数 49+24=？

列出式子如下：

```
  0100  1001
+ 0010  0100
-------------
```

0110　1101

从上面两位 BCD 加法可以看到，在计算机中，处理器确实对用两个 2 位的 8421BCD 码相加，但相加的结果是错误的，必须进行校正。此处只需加 06H 即 00000110 就可获得正确的 BCD 结果。有关校正的细节此处不再说明，只是希望记得 BCD 数的算术运算后需要进行校正，方能保证结果总是正确的。

2.1.3.2 无权码

在对应的 1 位十进制数的 4 位二进制编码中，二进制各位没有确定的权值，这种 BCD 码称为无权码。常见的无权码如表 2.5 所示。

表 2.5 无权码的 BCD 编码

十进制数	余 3 码	格雷码(1)	格雷码(2)
0	0011	0000	0000
1	0100	0001	0100
2	0101	0011	0110
3	0110	0010	0010
4	0111	0110	1010
5	1000	1110	1011
6	1001	1010	0011
7	1010	1000	0001
8	1011	1100	1001
9	1100	0100	1000

表中的余 3 码实际上在有权的 8421 码的基础上，在每位编码上加 3（0011）构成的。同样，利用余 3 码进行算术运算时，也需要进行校正，此处不再说明。

从表 2.5 中可以发现格雷码的编码规则，任何两相邻十进制数的格雷码只有 1 位二进制位不相同，其余 3 位是相同的。当十进制数从某一个数变为其相邻的数时，其相应的格雷码只需改变 1 位。可以想像，格雷码有多种方案，表 2.5 中仅列出常用的两种方案。

2.2 非数值数据的编码

2.2.1 ASCⅡ码

现代计算机不仅处理数值领域的问题，而且大量处理非数值领域的问题．这样一来，必然要引入文字、字母以及某些专用符号，以便表示文字语言、逻辑语言等信息．例如，人机交互信息时使用英文字母、标点符号、十进制数以及诸如$、%、+等符号。然而，计算机只能处理二进制数据，上述信息应用到计算机中时，都必须编写成二进制格式的代码，也就是字符信息用数据表示，称为符号数据。

目前国际上普遍采用的一种字符系统是七位的 ASCII 码(美国国家信息交换标准码)，它包括 10 个十进制数码，大小写 26 个英文字母和一定数量的专用符号、控制命令等总共约 128 个元素。因此，用二进制编码表示只需要 7 位。若加上一个奇（偶）校验位，共 8 位，刚好可用一个字节表示。

ASCII 码尽管是美国信息交换的一种标准，但该编码已被国际标准化组织 ISO 采纳，已经成为一种国际通用的信息交换用标准代码。

ASCII 码采用 7 个二进制位对字符进行编码，可表示 128 个符号：低 4 位组 $d_3d_2d_1d_0$ 用作行编码，高 3 位组 $d_6d_5d_4$ 用作列编码，其格式如图 2.12 所示。

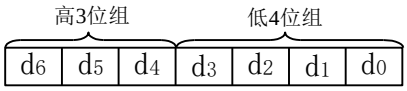

图2.12 ASCII码的构成格式

表 2.6 列出了 ASCII 码，是美国国家信息交换标准委员会制定的 7 位二进制码，共有 128 种元素。

表 2.6 ASCII 编码表

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

注：表 2.6 中的控制字符：

LF	换行	NAK	否定回答		
NUL	空行	VT	纵向制表	SYN	同步空转
SOH	标题开始	FF	改换格式	ETB	信息组传送结束
STX	文件开始	CR	回车	CAN	作废
ETX	文件结束	SO	移出	EM	记录媒体结束
EOT	传送结束	SI	移入	SUB	代替
ENQ	询问	DEL	删除	ESC	脱离
ACK	回答	DC1	制备控制 1	FS	字段分隔
BEL	报警	DC2	制备控制 2	GS	字组分隔

ASCII 码规定 8 个二进制位的最高一位为 0，余下的 7 位可以给出 128 个编码，表示 128 个不同的字符。其中 95 个编码，对应着计算机终端能敲入并且可以显示的 95 个字符，打印设备也能打印这 95 个字符，如大小写各 26 个英文字母，0～9 这 10 个数字符，通用的运算符和标点符号+、一、*、＼、>、一、<、，、 •、：、；、?、!、(、)、[、]等等。另外的 33 个字符，其编码值为 0～31 和 127[即（0000000）到（0011111）和（1111111）]，则不对应任何一个可以显示或打印的实际字符，它们被用作控制码，控制计算机某些外围设备的工作特性和某些计算机软件的运行情况。

ASCII 编码和 128 个字符的对应关系如表 2.6 所示. 表中编码符号的排列次序为 d7d6d5d4d3d2d1，其中 d8，恒为 0，表中未给出，d7d6d5 为高位部分，d4d3d2d1 为低位部分。可以看出，十进制的 8421 码可以去掉 d7d6d5 (=011)而得到。

字符串是指连续的一串字符，通常方式下，它们占用主存中连续的多个字节，每个字节存一个字符。当主存字由 2 个或 4 个字节组成时，在同一个主存字中，既可按从低位字节向高位字节的顺序存放字符串内容，也可按从高位字节向低位字节的次序顺序存放字符串内容。这两种存放方式都是常用方式。

最后要提及的是，有的厂家制定了 8 位的字符编码方案。例如，IBM 公司使用 EBCDIC 码(扩充二进制编码的十进制交换码)。它采用 8 位码, 有 256 个编码状态, 是 ASCII 码的两倍。但只选用其中一部分。0-9 十个数字符的高 4 位编码为 1111, 低 4 位仍为 0000～1001。大、小写英文字母的编码同样满足正常的排序要求, 而且有简单的对应关系，有关细节不再说明。

2.2.2 汉字编码

汉字的计算机处理技术远比拼音文字复杂，英文是一种拼音文字，只需要配备 26 个字母键，并规定 26 个字母的编码(比如通用的 ASCII 码)就能方便地输入英文信息了。

汉字是象形文字，字数很多，如果每一个汉字规定一个编码，就要有成千上万种编码。另外汉字字形结构复杂。仅是汉字的部首笔划就有数百种，汉字的同音字、同形字、同义字也很多。汉字的上述特点使得汉字的编码方案种类繁多，根据其用途可将这些编码分为三类：汉字输入编码方案、国际码及汉字内码和汉字字模码。

2.2.2.1 汉字输入编码

汉字输入编码是研究最多的，方案有数百种之多。根据其特点，这些方案可以归结为下列几种类型：

1. 汉字拼音编码

这是一类以汉语拼音为基础的汉字输入编码。在汉语拼音键盘或经过处理的西文键盘上，根据汉字读音直接键入拼音。只要会汉语拼音，就能使用，当遇到同音异字时，屏幕显示若干同音汉字，再输入序号，选定一个汉字，送到计算机。这是非专业汉字输入人员常用的一种简便的汉字输入方法。

2. 汉字字形编码

所有的汉字都由横、竖、撇、点、折、弯有限的几种笔划构成，并且又都可分为‘左右’、‘上下’、‘包围’、‘单体’有限的几种构架，每种笔划都赋予一个编码并规定选取字形构架的顺序，不同的汉字因为组成的笔划和字形构架顺序不同，就能获得一组不同的编码，来表达一个特定的汉字，广泛使用的“五笔字形”就属这一种。

首尾码是另一种根据字形进行的输入方式，它将汉字的左上部笔画约定为字首码，右下部笔画约定为字尾码，由一个首码一个尾码描述一个汉字，当有重码时再送人附加信息。

一般字形编码表示法的字形基本部件与输入键的对应关系都比较复杂，这种类型的输入方法多为专业汉字输入人员所用。

3. 汉字直接数字编码

直接利用一串数字表示一个汉字，电报码就属这一种。另外国标码、区位码、机内码也是每个汉字对应一组惟一的数据，因此也可把它们归于这一类，这一类的共同特点是只需数字键盘即可输入，缺点是其编码难以记忆。

4. 整字编码

设置汉字整字大键盘，每个汉字占一个键，类似中文打字机，操作人员选取汉字，机器根据所选汉字在盘面上的位置将其对应编码送人计算机。

5. 手写输入

一块插在 USB 接口上的汉字手写板，再配上厂家所提供的专用软件，使用者便可以很方便地输入汉字。本书及作者的其他几本书全都是利用手写输入完成的。有多个厂家提供这种形式的汉字手写输入。显然，识别规整的印刷体汉字要容易得多

6. 语音输入

每台计算机上都配有用于语音输入的麦克风，在此基础上配上专门的语音识别软件，就可以实现汉字的语音输入。届时，使用者上要对着麦克风将汉字文件念一遍，便可方便地输入。而且，技术发展到今天，无论是识别率还是响应时间，早已达到实用的水平。

上面所介绍的几种汉字输入方式有的是手工输入，手写要求写字规范，而其他方法需要一定的时间才能熟练掌握。理想的输入方式是语音输入。

2.2.2.2 国标码和汉字内码

1. 国标码

为了使汉字信息交换有一个通用的标准，1981 年我国制定推行的 GB2312—80 国家标准《信息交换用汉字编码字符集(基本集)》，简称国标码。

在该国家标准中，挑选常用的汉字 3755 个，次常用汉字 3008 个，共 6763 个汉字，以及俄文字母、日语假名、拉丁字母、希腊字母、汉语拼音以及一般符号、数字等共 682 个非汉字符号。这些常用汉字和非汉字符加在一起共 7445 个字符，国标码规定了它们的标准编码。

2. 汉字内码

汉字内部码（简称汉字内码）是汉字在设备或信息处理系统内部最基本的表达形式。汉字内码也是汉字在计算机内用于存储、检索、交换的信息代码，汉字内码的设计要求与西文信息处理有较好的兼容性，同时与国标码有简单的转换关系。下面的例子可以看到，如果给国标码的每字节最高位加 1，作为汉字标识符，就成为一种汉字内码。只要从最高位标识符就能区分这两种代码。标识符是 0，即是 ASCII 码，标识符是 1，则是汉字内码。

汉字内码大多数采用两字节长的代码，也有三字节长、四字节长的汉字内码。

3. 汉字区位码

GB2312-80 国标字符集构成一个二维平面，分成 94 行和 94 列，并将行号称做区号，将列号称为位号。因此，在此字符集中的每一个汉字或符号对应唯一的一个区号和位号。而且，区位号最大是 94，故区号和位号均用 7 位二进制编码来表示（区号编码在前，位号编码在后），这 14 位二进制编码就称为汉字的区位码。区位码常用于汉字输入。

4. 编码间的关系

区位码并不是国标码，但由区位码可以构成国标码（又称国标交换码）。在汉

字区位码上分别各加上 20H，则区位码就变成国标码。例如，汉字“大”字的区位号分别是 20 和 83，其区位码可表示为 1453H（14 位二进制编码）。在区号及位号上分别加 20H，则汉字“大”字的国标码 3473H。但清注意，34H 和 73H 均为 7 位二进制编码，即 0110100 1110011。

在计算机内部，为了存储和处理上的方便，上述每个 7 位二进制编码均用一个字节来表示。上述汉字“大”字的国标码在计算机中表示为：00110100 01110011。这样做又带来了新的问题：汉字国标码的每一个字节都有可能与 ASCII 码重到一起，以至于会产生混淆而无法识别。于是，就规定国标码的每个 7 位二进制编码的前面必须再加一位 1，构成一个字节，这就解决了上述矛盾。因此，汉字“大”字的编码就变成了 10110100 11110011。这时一个汉字字符就用 2 个字节（16 位）来表示，且每个字节的最高位必须为 1。汉字的这种编码就叫做汉字的机内码，简称汉字内码。

由上所述，使我们弄清了上面三种编码之间的关系：

区位码(14 位区位各 7 位)，区位分别加 20H 变成国标码(仍为 14 位)，国标码的每个 7 位编码的前面再加一位 1，变成 8 位，从而构成两个字节的内码。

5. 其他标准

为了统一地表示世界各国的文字，1993 年国际标准化组织公布“通用多八位编码字符集”的国际标准 ISO/IEC 10646，简称 UCS。UCS 包含了中、日、韩等国的文字。这一标准为包括汉字在内的各种正在使用的文字规定了统一的编码方案。我国相应的国家标准为 GB13000。

UCS－4 规定每一字符用 4 个字节表示，可构成多达 13 亿的字符空间。但这也给字符的存储、传输及处理带来很大的麻烦。于是，便从 UCS 中选出一个子集，用 2 个字节表示一个字符，这就是 UCS－2，又称 Unicod。

Unicod 用两个字节共定义了 49194 个符号及数字，其中包括中、日、韩、越、新加坡及中国台湾所使用的 27484 个汉字。

2.2.2.3 汉字输出

当汉字需要在屏幕上显示或需要在打印机上打印时，需要将机内码转换成汉字字形码。它是表示汉字字形的字模数据，通常用点阵、矢量函数等方式表示。

1. 点阵字模

用点阵表示字形时，汉字字形码指的就是这个汉字字形点阵的代码，字形码也称字模码。汉字字模就是用 0、1 表示汉字的字形，将汉字放入 n 行×n 列的正方形内，该正方形共有 n^2 个小方格，每个小方格用一位二进制表示，凡是笔划经过的方格值为 1，未经过的值为 0。简易型汉字为 16×16 点阵，高精度型汉字为 24×24 点阵，32×32 点阵，48×48 点阵。上述每个汉字字模分别需要 32、72、128、288 个字节存放。显然，点数愈多，输出的汉字愈美观。将每一个汉字的点阵信息集中存储在一起，就构成了汉字字形库。

2. 矢量字模

汉字的矢量表示法是将汉字看做是由笔画组成的图形，提取每个笔画的坐标值，这些坐标值就可以决定每一笔画的位置，将每一个汉字的所有坐标值信息组合起来就是该汉字字形的矢量信息。同样，将每一个汉字的矢量信息集中在一起也可从构成汉字字库。

对于同样数量的汉字来说，点阵信息构成的汉字字形库比矢量汉字字形库需要更大的存储空间。对于一个 48×48 点阵的汉字，就需要 288 个字节的存储空间。而矢量汉字所需存储空间要小一些。但是，从打即显示的汉字来看，点阵方式所显示的汉字更逼真一些，除非对矢量字模进行更复杂的处理。

当需要汉字输出时，利用汉字字形检索程序根据汉字内码从字形库中找到相应的字形码，由输出设备输出。

存放在磁盘上的字模库称为软字库。存放在 ROM 中的字模库称为硬字库，也称为“汉卡”。

2.3 检错与纠错编码

元件故障、噪声干扰等各种因素常常导致计算机在传输、存储或处理信息的过程中出现错误。例如，将一位 1 从部件 A 传送到部件 B，可能由于传送信道中的噪声干扰而受到破坏，以至于在接收部件 B 收到的是 0 而不是 1。为了防止这种错误，可将信号采用专门的逻辑线路进行编码以检测错误，甚至校正错误。本节将介绍常用的几种检错及纠错方法。

2.3.1 奇偶校验码

最简单且应用广泛的检错码是采用一位校验位的奇偶校验。

2.3.1.1 水平奇偶校验

水平奇偶校验就是对每一个数据的编码添加校验位，使信息位与校验位处于同一行上。

1. 水奇校验

设数据 $X=(x_0x_1\cdots x_{n-1})$是一个 n 位字，若在其高位前增加 1 位奇校验位 c，则包括奇校验位的数据就变成了 $X'=(cx_0x_1\cdots x_{n-1})$。则奇校验定义为：

$$c \oplus x_0 \oplus x_1 \oplus \cdots \oplus x_{n-1} = 1 \qquad (2-19)$$

式中 $\oplus$ 代表按位加。之所以称为奇校验，是因为必须保证数据(包括奇校验位在内)的 n+1 位中，1 的个数为为奇数。在将数据加上奇校验时可做如下运算：

$$\overline{c} = x_0 \oplus x_1 \oplus \cdots \oplus x_{n-1} \qquad (2-20)$$

就是说，奇校验位应为数据 $X=(x_0x_1\cdots x_{n-1})$各位模 2 加的结果取反。

例如，X=01010100，采用奇校验时，奇校验位 c 必须为 0，则加了奇校验的数据 X′=001010100。

当数据 X=01010101，采用奇校验时，奇校验位 c 必须为 1，则加了奇校验的数据 X′=101010101。

当数据加上奇校验后，便可以存储或传送到通信的对方。在从存储器读出或通信对方收到该数据后，可利用（2－19）式进行计算，若（2－19）式成立，则认为在存储或传输过程中，未发生 1 位出错。

2. 水平偶校验

偶校验的概念与奇校验是一样的，定义就是加上偶校验后，必须保证数据(包括偶校验位在内)的 n+1 位中，1 的个数为为偶数。也就是必须保证：

$$c \oplus x_0 \oplus x_1 \oplus \cdots \oplus x_{n-1} = 0 \qquad (2-21)$$

在将数据 $X=(x_0x_1\cdots x_{n-1})$加上偶校验时，可利用下式求出偶校验位 c：

$$c = x_0 \oplus x_1 \oplus \cdots \oplus x_{n-1} \qquad (2-22)$$

也就是说，偶校验位就寻等于数据各位的模 2 加。

例如，X=01010100，采用偶校验时，偶校验位 c 必须为 1，则加了奇校验的数据 X′=101010100。

当数据 X=01010101，采用偶校验时，偶校验位 c 必须为 0，则加了奇校验的数据 X′=001010101。

同样，当数据加上偶校验后，便可以存储或传送到通信的对方。在从存储器读出或通信对方收到该数据后，可利用（2－21）式进行计算，若（2－21）式成立，则认为在存储或传输过程中，未发生 1 位出错。

从上面的描述中可以想到，奇偶校验可以检测出 1 位错误(或奇数位错误，但多于 1 位错的几率很小)。但无法检测出 2 位或偶数位错，也无法识别错误信息的位置。但是，由于奇偶校验原理简单，实现起来非常容易。因此，这种方法得到广泛地应用。

2.3.1.2 垂直奇偶校验码

这种校验码把数据分成若干组，一组数据占一行，排列整齐，针对每一列采用奇校验或是偶校验，再加一行校验码，。

例 10：对于 32 位数据 10100101 00110110 11001100 10101011，其垂直奇校验和垂直偶校验如下表 2.7 所示。

表 2.7 加垂直校验的数据

	垂直奇校验码	垂直偶校验码
数据	1 0 1 0 0 1 0 1	1 0 1 0 0 1 0 1
	0 0 1 1 0 1 1 0	0 0 1 1 0 1 1 0
	1 1 0 0 1 1 0 0	1 1 0 0 1 1 0 0
	1 0 1 0 1 0 1 1	1 0 1 0 1 0 1 1
校验位	0 0 0 0 1 0 1 1	1 1 1 1 0 1 0 0

2.3.1.3 水平垂直校验码

在垂直校验码的基础上，对每个数据再增加一位水平校验位，便构成水平垂直校验码。

例 11：对于 32 位数据 10100101 00110110 11001100 10101011，其水平垂直奇校验和偶校验如下表 2.8 所示。

表 2.8 加二维校验的数据

	水平垂直奇校验码								水平垂直偶校验码								
	水平校验位	数 据								水平校验位	数 据							
	1	1	0	1	0	0	1	0	1	0	1	0	1	0	0	1	0	1
	1	0	0	1	1	0	1	1	0	0	0	0	1	1	0	1	1	0
	1	1	1	0	0	1	1	0	0	0	1	1	0	0	1	1	0	0
	0	1	0	1	0	1	0	1	1	1	1	0	1	0	1	0	1	1
垂直校验位	0	0	0	0	0	1	0	1	1	1	1	1	1	1	0	1	0	0

2.3.2 海明码

在计算机运行过程中，由于种种原因致使数据在存储过程中可能出现差错。为了能及时发现错误并及时纠正错误，通常将原数据配成海明编码。

2.3.2.1 海明码的组成

海明码具有一位纠错能力。由编码纠错理论得知，任何一种编码是否具有检错能力和纠错能力，都与编码的最小距离有关。所谓编码最小距离是指在一种编码系统中，任意两组合法代码之间的最少二进制位数的差异。根据纠错理论得：

$$L-1=D+C \quad 且\ D \geqslant C \qquad （2－23）$$

即编码最小距离 L 越大，则其检测错误的位数 D 也越大，纠正错误的位数 C 也越大，且纠错能力恒小于或等于检错能力。如当编码最小距离 L=3 时，这种编码可视为最多能检错二位，或能检错一位、纠错一位。可见，倘若能在信息编码中增加几位检测位，增大 L，显然便能提高检错和纠错能力。海明码就是根据这一理论提出的具有一位纠错能力的编码。

设欲检测的二进制代码为 n 位，为使其具有纠错能力，须增添 k 位检测位，组成 n+k 位的代码。为了能准确对错误定位以及指出代码没错，新增添的检测位数 k 应满足：

$$2^k \geqslant n+k+1 \qquad （2－24）$$

由此关系可求得不同代码长度 n 所需检测位的位数 k, 如表 2.7 所示。

表 2.9 代码长度与检测位位数的关系

n	K(最小)
1	2
2~4	3
5~11	4
12~26	5
27~57	6
58~120	7

K 的位数确定后，便可由它们承担检测任务，设定它们被传送代码中的位置及它们的取值。

在说明海明码在工程上如何编码、如何译码的具体应用之前，首先解释海明码的产生及译码过程。

下面表 2.8 表示海明码及其校验方程。

表 2.10　海明码及其校验方程

bit 内容	D_{15} D_{14} D_{13} D_{12} D_{11} H_4 D_{10} D_9 D_8 D_7 D_6 D_5 D_4 H_3 D_3 D_2 D_1 H_2 D_0 H_1 H_0
bit 位置	21 20 19 18 15 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
校验方程	$P_0=D_{15}\oplus D_{13}\oplus D_{11}\oplus D_{10}\oplus D_8\oplus D_6\oplus D_4\oplus D_3\oplus D_1\oplus D_0\oplus H_0$
	$P_1=D_{13}\oplus D_{12}\oplus D_{10}\oplus D_9\oplus D_6\oplus D_5\oplus D_3\oplus D_2\oplus D_0\oplus H_1$
	$P_2=D_{15}\oplus D_{14}\oplus D_{10}\oplus D_9\oplus D_8\oplus D_7\oplus D_3\oplus D_2\oplus D_1\oplus H_2$
	$P_3=D_{10}\oplus D_9\oplus D_8\oplus D_7\oplus D_6\oplus D_5\oplus D_4\oplus H_3$
	$P_4=D_{15}\oplus D_{14}\oplus D_{13}\oplus D_{12}\oplus D_{11}\oplus H_4$

在表 2.8 中, bit 内容是指 16bit 数据和相应的 5 位海明码构成的编码中，各位所处的位置。下一行 bit 位置就指出各位的位置编号。请注意到每一位的位置编号，这在纠错时会用到。

P_0 到 P_4 是对应 16 位信息位的校验方程。而表 2.8 中 H_0 到 H_4 为海明硬码的 bit0 到 bit4。对于一个字节（8 位）信息而言只要利用 4 位海明码 H_0 ～ H_3 就够了。校验方程只需表中的 P_0 到 P_3 ，其计算公式只需表 2.8 中虚线右边部分即可。

在应用中，需要产生海明码，海明码是在已知信息（8 位或 16 位）的基础上，假定所有的校验位 H_i 均为 0 的条件下，利用表 2.8 中的校验方程计算出 P_0 、 P_1 、…。当把信息写入内存单元时，同时将计算出来的 P_i 看成相应的 H_i 一并写入，这就是海明校验码。

当读出校验时，存入内存单元的信息位和海明码校验位同时读出。这时，数据各位和海明校验方各位均为校验方程所利用，通过它们计算校验方程的值。若各校验方程的计算结果均为 0，则说明数据正确。若计算结果不为 0，则它们的编码值就表示出错 bit 的的位置编号 。根据出错的位置编号，便可知是哪一位出了错，也就很容易将出错位纠正过来。

现在以 8 位数据为例，说明海明码的产生及纠错过程。

例 12：假如 8 位数据如下：

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	1	0	1	1	1	0

为计算 P_i，首先假定 H_0 ～ H_3 均为 0，利用表 2.8 所给出的校验方程的右边部分计算出 P_0 到 P_3 如下：

$P_0=D_6\oplus D_4\oplus D_3\oplus D_1\oplus D_0\oplus H_0 = 0\oplus 0\oplus 1\oplus 1\oplus 0\oplus 0=0$

$P_1=D_6\oplus D_5\oplus D_3\oplus D_2\oplus D_0\oplus H_1=0\oplus 1\oplus 1\oplus 1\oplus 0\oplus 0=1$

$P_2=D_7\oplus D_3\oplus D_2\oplus D_1\oplus H_2 =1\oplus 1\oplus 1\oplus 1\oplus 0=0$

$$P_3=D_7 \oplus D_6 \oplus D_5 \oplus D_4 \oplus H_3 = 1 \oplus 0 \oplus 1 \oplus 0 \oplus 0 = 0$$

这样，就求出了 P_0 到 P_3 的值。用 H_0、H_1、H_2、H_3 代替刚求出的 P 值，则海明码就得到了，它们分别为：

$$H_0=P_0=0$$
$$H_1=P_1=1$$
$$H_2=P_2=0$$
$$H_3=P_3=0$$

在存储上面的 8 位数据时，同时将上述 H_0、H_1、H_2、H_3 一并存入内存单元，此时每一内存单元存储 12 位。在存储数据和海明码时，可以按照表 2.8 的格式将它们混合为 12 位编码：$D_7D_6D_5D_4H_3D_3D_2D_1H_2D_0H_1H_0$ 进行存储，也可以不混在一起分别进行存储，即：$D_7D_6D_5D_4D_3D_2D_1D_0H_3H_2H_1H_0$ 进行存储。

当数据由此存储单元读出时，8 位数据和 4 位海明校验码一并读出。这时所读出的 H_0 ～ H_3 就是原先写入的 0100。在读出的数据和海明码均已知的情况下，同样使用表 2.8 给出的校验方程计算 P_i 的值。若读出的 12 位均正确，则所计算的校验方程的值均为 0。

假如，读出的数据 D_2 出错，由原先的 1 变为 0，则校验方程的计算值 P_0 、P_1 、P_2 、P_3 分别为 0、1、1、0 。此时，P_3 ～P_0 的编码十进制值为 6，它表示在 bit 位置 6 上的那一位数出错。请注意表 2.8 中的 bit 位置一栏，在位置 6 上对应的是数据位 D_2 ，这就是说 D_2 出错。可见，读出后，计算出的 P_3 ～P_0 的编码值就是出错 bit 的位置。

知道了在数据存取过程中哪一位出错，就能够把错误纠正过来。利用硬件电路实现纠错是比较简单的。正如前面所提到的，P_3 ～P_0 的编码十进制值就是出错 bit 的序号。可以利用常见的 4—16 译码器芯片，将译码器输出与相应的数据位相连接，从而可达到纠正 8 位数据中任何一位错的目的。纠错电路原理框图如图 2.13 所示。

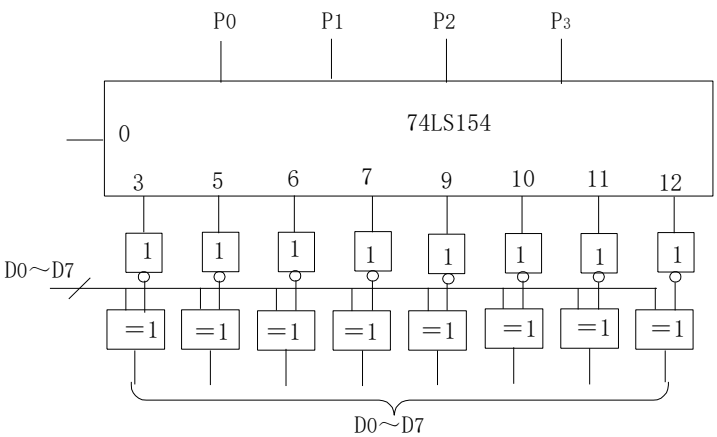

图2.13　纠错电路原理图

在图 2.13 中，译码器选用 54LS154，在它的四个译码输入端上加上 P_3 ～P_0，它可以有 16 个译码输出。选择对应 8 位数据的输出端，与相应的数据位相异或，从而达到纠错的目的。由图 2.13 可以看到，当四个输入全为 0 时，即数据正确时，译码器输出端 0 的输出有效（低电平）。即该端输出为低电平时，表示数据没错；而当该端

输出不为低电平时，表示数据有错。

海明码不仅用于存储器的校验，在通信传输、磁盘记录中同样可用。

2.3.3 循环冗余校验码（CRC 码）

2.3.3.1 概述

循环冗余校验码（CRC 码）可以发现并纠正信息在存储或传送过程中出现的错误。因此 CRC 校验码在磁介质存储器和计算机之间通信方面得到广泛应用。

CRC 码是基于模 2 运算而建立编码规律的校验码。模 2 运算的特点是不考虑进位和借位的运算，其规律如下：

模 2 加法　按位加，不考虑进位：0+0=0，0+1=1，1+0=1，1+1=0．例如 1101+1011=0110。

模 2 减法　按位减，不考虑借位：0-0=0，0-1=1，1-0=1，1-1=0．例如 1101-1011=0110。

可见模 2 减法与模 2 加法运算结果相同，因此可用模 2 加法代替模 2 减法。

模 2 乘法　按模 2 加求部分积之和，不考虑进位。例如 1010×1101=1110010。

模 2 除法　按模 2 减求部分余数，不借位，每求一位商应使部分余数减少一位。进商的规则是：余数首位为 1 商取 1，余数首位为 0 商取 0。当余数位数小于除数位数时即为最后余数。上述这些法则，在后面求 CRC 校验码时将会用到。

例如 10000÷101=101+01／101，商为 101，余数为 01。

2.3.3.2 CRC 码的编码方法

循环冗余校验码是这样构成的：

将 n 位的信源二进制代码 M（x）左移 k 位后，被二进制长度为 k＋1 位的生成多项式 G（x）相除所得的 k 位余数就是校验位。将此 k 位校验位接在原信源代码 M（x）之后，就构成了长度为 n＋k 位的循环冗余校验码 CRC。其格式如下图 2.14 所示。

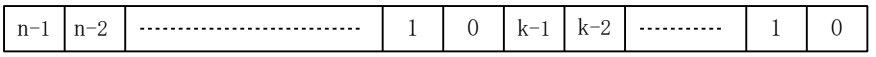

图2.14 CRC构成格式

1. 左移构成一个多项式

利用生成多项式，可为 n 个数据位产生 k 个校验位，构成编码长度为 k+n 的编码。

设待编的信息码组用多项式 M（x）表示：

$$M(x) = D_{n-1}x^{n-1} + D_{n-2}x^{n-2} + \cdots + D_1 x^1 + D_0 x^0 \qquad (2-25)$$

式中 D_i 为 1 或 0。

将信息码组左移 k 位，移空处补 0，则得 M（x）$\cdot x^k$，即成 n+k 位多项式：

$$D_{n-1+k}\, D_{n-2+k} \cdots\cdots D_{2+k}\, D_{1+k}\, D_{0+k}$$

空出的 k 位将来用来拼接 k 位校验码。

2. 求余数

CRC 码就是用多项式 M（x）• x^k 除以生成多项式 G（x）（即产生校验码的多项式），所得余数作为校验位。

为了得到 k 位余数（校验位），G（x）必须是 k+1 位。

在进行除法的过程中，利用概述中所提到的法则。因此，在这里要特别强调这些法则是非常重要的。利用模 2 除法即可得到余数 R 和商。

将余数拼接在左移了 k 位后的信息位后面，就构成了这个有效信息的 CRC 码。

因此，所得 CRC 码是一个可被生成多项式 G（X）除尽的数码。如果 CRC 码在传输过程中不出错，其余数必为 0；如果传输过程中出错，则余数不为 0，再由该余数指出哪一位出错，即可纠正之。

3. 举例

例 13：已知有效信息为 1100，试用生成多项式 G（X）=1011 将其构成 CRC 码。

解：

有效信息 M（x）=1100= x^3+x^2　　(n=4)

由 G（x）=1011= x^3+x+1。生成多项式为 4 位，即 k+1= 4 ，所以 k=3。

将有效信息左移 k 位后得：1100000。再被 G（X）模 2 除，过程如下：

```
          1 1 1 0
      ┌──────────
1 0 1 1 ) 1 1 0 0 0 0 0
          1 0 1 1
          ─────────
            1 1 1 0
            1 0 1 1
            ───────
              1 0 1 0
              1 0 1 1
              ───────
                0 0 1 0
                0 0 0 0
                ───────
                  0 1 0
```

将余数 010 放在前面信息位左移 3 次所空出的位置上，即将余数与信息拼接在一起就构成了 CRC 码如下：

M（x）• x^3+R(x)=1100000+010=1100010 为 CRC 码。

总的 CRC 编码为 7 位，有效信息位为 4 位，故上述 1100010　码又称（7，4）码。这里的（7，4）码，即为码制，还可以有（7，3）码制和（7，6）码制等。

2.3.3.3 CRC 码的译码和纠错

将收到（或从存储介质中读出）的循环校验码用约定的生成多项式 G（X）去除，如果无错，则余数应为 0，如果某一位出错，则余数不为 0，不同的出错位其余数也不同，表 2.9 列出了对应 G（X）=1011 的出错模式。

表 2.11 对应 G（X）=1011 的（7，4）循环的出错模式

序号	N_1	N_2	N_3	N_4	N_5	N_6	N_7	余数	出错位
正确	1	1	0	0	0	1	0	000	无
错	1	1	0	0	0	1	1	001	7

误	1	1	0	0	0	0	0	010	6
	1	1	0	0	1	1	0	100	5
	1	1	0	1	0	1	0	011	4
	1	1	1	0	0	1	0	110	3
	1	0	0	0	0	1	0	111	2
	0	1	0	0	0	1	0	101	1

可以证明，更换不同的待测码字，余数和出错位的对应关系不变，只与码制和生成多项式有关。表 2.9 给出的关系只对应 G（x）=1011 的（7，4）码，对于其他码制或选择用其他生成多项式，出错模式将发生变化。

如果循环码有一位出借，用 G（x）作模 2 除将得到一个不为 0 的余数。如果对余数补 0 继续除下去，将发现各次所得余数将按表 2.9 顺序循环。例如第 7 位出错，其余数为 001，补 0 后再除，第二次余数为 010，以后依次为 100，011……反复循环，这就是“循环码”的名称由来。这个特点正好用来纠错，即当出现不为零的余数后，一方面对余数补 0 继续作模 2 除，另一方面将被检测的校验码字循环左移。由表 2.9 可见，当出现余数为 101 时，出错位也移到了 N_1 位置。可通过异或门将它纠正后在下一次移位时送回 N_7。这样当移满一个循环（对 7，4 码共移七次）后，就得到一个纠正后的码字。

值得指出的是，并不是任何一个（k+1）位多项式都可以作为生成多项式。从检错和纠错的要求出发，生成多项式应满足以下要求：

① 任何一位发生错误，都应该使余数不为零;

② 不同位发生错误应使余数不同;

③ 对余数继续作模 2 除,应使余数循环。

好在前人已为我们提供了多种检纠错性能很好的生成多项式供选用。例如，国际电联推荐了多种不同长度的生成多项式，其中长度为 17 位的生成多项式为：

$G(x)=X^{16}+X^{12}+X^5+1$

当需要使用生成多项式时可找有关资料选用。

习　　题

2.1　实现下列各数的转换。

(1) $(97.8125)_{10}$ ＝ (　　)$_2$ ＝ (　　)$_8$ ＝ (　　)$_{16}$

(2) $(110101.011)_2$ ＝ (　　)$_{10}$ ＝ (　　)$_8$ ＝ (　　)$_{16}$＝ (　　)$_{8421BCD}$

(3) $(0011\ 0110\ 1001.\ 0101)_{8421BCD}$ ＝ (　　)$_{10}$ ＝ (　　)$_2$ ＝ (　　)$_{16}$

(4) $(2A7C.5E)_{16}$ ＝ (　　)$_{10}$ ＝ (　　)$_2$

2.2　已知$[x]_{原}$，求$[x]_{补}$和$[x]_{反}$。

① $[x]_{原}$＝0.1010110　　② $[x]_{原}$＝1.0010110

③ $[x]_{原}$＝01010110　　④ $[x]_{原}$＝11010010

2.3　已知$[x]_{补}$，求 x。

① $[x]_{补}=1.1101101$　　② $[x]_{补}=0.1010110$

③ $[x]_{补}=10000000$　　④ $[x]_{补}=11010010$

2.4　假设机器字长为 8 位，求下列补码所对应 X 的十进制真值。

(1)　$[2X]_{补}=90H$

(2)　$[\frac{1}{2}X]_{补}=C2H$

(3)　$[-X]_{补}=FEH$

2.5　设 x 为定点小数，$[x]_{补}=1.x_6x_5x_4x_3x_2x_1x_0$，最高位为符号位。

(1)　若要 $x<-\frac{1}{2}$，$x_6x_5x_4x_3x_2x_1x_0$ 应满足什么条件？

(2)　若要 $-\frac{1}{2}\leqslant x<-\frac{1}{4}$，$x_6x_5x_4x_3x_2x_1x_0$ 应满足什么条件？

2.6　假设机器字长为 8 位，已知$[X]_{补}=3AH$，$[Y]_{补}=C5H$，求：

$[2X]_{补}$，$[2Y]_{补}$，$[\frac{1}{2}X]_{补}$，$[\frac{1}{4}Y]_{补}$，$[-X]_{补}$，$[-Y]_{补}$

$[X]_{原}$，$[Y]_{原}$，$[X]_{反}$，$[Y]_{反}$，$[X]_{移}$，$[Y]_{移}$

2.7　若机器字长为 32 位，问整数补码和小数补码可表示数的个数为多少？可表示的真值范围各是多少？

2.8　分别写出下列各十进制数的原码、反码和补码，用 8 位二进制数表示（最高位为符号位）。如果是小数，小数点在最高位（符号位）之后；如果是整数，小数点在最低位之后，并写出其对应的移码。

(1) 用整数表示的－1　(2) 用小数表示的－1　(3) 用整数表示的＋0

(4) 用小数表示的－0　(5) 45/64　(6) －1/128

(7) ＋128　(8) －128　(9) ＋127

(10) －127　(11) 89　(12) －32

2.9　若约定小数点在 8 位二进制数的最右端（定点整数），试分别写出下列各种情况下 W、X、Y、Z 的真值。

(1)　$[W]_{补}=[X]_{原}=[Y]_{反}=[Z]_{移}=00H$

(2)　$[W]_{补}=[X]_{原}=[Y]_{反}=[Z]_{移}=80H$

(3)　$[W]_{补}=[X]_{原}=[Y]_{反}=[Z]_{移}=FFH$

2.10　多项选择题。可供选择的答案：A. 原码　B. 反码　C. 补码　D. 移码

(1)　在数值数据的编码表示中，0 有惟一表示的编码有（　　）；

(2)　符号位用 0 表示正、用 1 表示负的编码有（　　）；

(3)　满足若真值大，则码值大的编码是（　　）；

(4)　存在负数的真值越大，则码值越小现象的编码是（　　）；

(5)　负数的码值大于正数的码值的编码有（　　）；

2.11　假设机器字长为 8 位，

(1)　码值为 80H：若表示真值 0，则为（　）码；若表示真值－128，则为（　）码；若表示真值－127，则为（　）码；若表示真值－0，则为（　）码。

(2)　码值为 FFH：若表示真值 127，则为（　）码；若表示真值－127，则为（　）码；若表示真值－1，则为（　）码；若表示真值－0，则为（　）码。

2.12　X 在什么范围内，有$[X]_{补}>[X]_{原}$？在什么范围内，有$[X]_{补}=[X]_{原}$？当 $X<$

0 时，试求出满足$[X]_{补}=[X]_{原}$的真值 X。

2.13 一个用 8 位二进制表示的整数补码，如何判断其正负？如何判断其有无十进制的百位？如何判断其奇偶？如何判断其能否被 8 整除？

2.14 若机器字长为 n 位二进制，可用来表示多少个不同的数？就下列三种情况，分别写出所能表示的最大值和最小值：

(1) 无符号数；

(2) 用原码表示的整数；

(3) 用补码表示的小数。

2.15 试写出下列各种情况下用 16 位二进制所能表示的数的范围（用十进制表示）以及对应的二进制代码。

(1) 无符号的整数；

(2) 补码表示的有符号整数；

(3) 补码表示的有符号小数；

(4) 移码表示的有符号整数；

(5) 原码表示的有符号小数。

2.16 已知字母“A”的 ASCII 编码为 1000001，字母“a”的 ASCII 编码为 1100001，数字“0”的 ASCII 编码为 0110000。求字符“D”、“K”、“f”、“h”、“5”、“7”的 7 位 ASCII 编码，并在最高位加入奇偶校验位（偶校验），形成带奇偶校验位的 8 位 ASCII 编码。

2.17 下列代码若看作 ASCII 码、整数补码、BCD 码时分别代表什么？

(1) 78H (2) 39H

2.18 汉字的编码分为哪几类？各有什么用途？

2.19 汉字字模码采用 32×32 点阵，一个汉字占多少字节？如果该点阵字库要包含 GB2312 收录的所有 7445 个图形字符（包括简化汉字及符号、字母、日文假名等），试计算该字库需要多大的存储容量？

2.20 假设计算机使用的是 24 位字，试在如下情况下，利用 24 位来表示 365。

(1) 如果计算机使用原码表示定点整数，如何表示十进制数 365？

(2) 如果计算机使用 8 位 ASCII 编码和偶校验，如何表示字符串“365”？

(3) 如果计算机使用压缩的 BCD 编码，如何表示数字＋365？

2.21 设浮点数字长 16 位，基值为 2（以 2 为底）。其中阶码 6 位（含一位阶符），用移码表示；尾数 10 位（含一位数符），用补码表示。

(1) 求能表示的规格化浮点数的范围，填写下表，并与 16 位定点补码整数和定点补码小数的表示范围进行比较。

表 2.12 习题 2.21 附表

	阶码（十六进制）	尾数（十六进制）	真值（十进制）
最大正数			
最小正数			
最大负数			
最小负数			

(2) 判断下列十进制数能否表示成此格式的规格化浮点数，若可以，请写出对应的码值。

① 3.14； ② －1917； ③ 105/512； ④ -10^{-6}； ⑤ 10^{10}

2.22 以 IEEE 754 短浮点数格式（32 位）表示下列十进制数。

① ＋5.3125　　② －365.59375　　③ ＋21

④ －35/8　　⑤ 324　　⑥ 56789.25

2.23　约定生成多项式为 $G(x)=x^3+x+1$，试计算下述信息字的 CRC 编码字，并在接收端进行除法校验。

① 1010110　　② 01011001

2.24　假设正在使用的一种纠错码可以校正长度为 8 的存储字的全部单位错误。计算结果表明，需要 4 位校验位，编码字的全部长度为 12 位。编码字的产生方式采用本章介绍的海明算法。现在接收器收到如下代码字：010111010110，假定采用偶校验，请问收到的这个字是否为合法的编码字？如果不是，根据纠错编码，指出错误发生在哪一位？