第一章　数制和编码

1.1　数　　制

数制是人们对数量计数的一种统计规律。日常生活中最常遇到的进位计数制是十进制，在数字系统中，广泛采用的则是二进制、八进制、十六进制。

一种进位计数包含着两个基本的因素：

(1) 基数：它是计数制中所用到的数码的个数，一般地说，基数为 R 的计数制(简称 R 进制)中，包含的是 0、1、…、R_n、R_{-1}等数码，进位规律是“逢 R 进一”，即每个数位计满 R 就向高位进 1，称为 R 进位计数制。

(2) 位权：在一个进位计数制表示的数中，处在不同数位的数码，代表着不同的数值，某一个数位的数值是由这一位数码的值乘上处在这位的一个固定常数。不同数位上的固定常数称为位权值，简称位权。不同数位有不同的位权值。例如，十进制数个位的位权值是 1，十位的位权值是 10^1，百位的位权值是 10^2。

广义地说，一个 R 进制数 N，可以有两种表示方式：

① 并列表示方式，也称位置计数法：

$$(N)_R = (K_{n-1}K_{n-2}\cdots K_1K_0.K_{-1}K_{-2}\cdots\cdots K_{-m})_R \tag{1-1}$$

其中，n 为整数部分的数位；m 为小数部分的数位；R 表示基数；K_i 为不同数位的数值：

$$0 \leqslant K_i \leqslant R-1$$

② 多项式表示法，也称以权展开式：

$$(N)_R = (K_{n-1}R^{n-1} + K_{n-2}R^{n-2} + \cdots + K_1R^1 + K_0R^0 + K_{-1}R^{-1} + \cdots + K_{-m}R^{-m}) \tag{1-2}$$

或者写成和式：

$$(N)_R = \left(\sum_{i=-m}^{n-1} K_iR^i\right)_R$$

其中：R 代表进位制的基数；m、n 为正整数，n 代表整数部分的位数；m 代表小数部分的位数；K 代表 R 进位制中 R 个数字符号中的任何一个：

$$0 \leqslant K_i \leqslant R-1$$

1.1.1　二进制

1. 二进制数的表示

基数 R＝2 的数制为二进制。二进制数的数值表示符号只有“1”和“0”，进位规律是“逢二进一”，任意一个二进制数 N 的多项式表示为：

$$(N)_2 = K_{n-1}2^{n-1} + K_{n-2}2^{n-2} + \cdots + K_12^1 + K_02^0 + K_{-1}2^{-1} + \cdots + K_{-m}2^{-m}$$

$$=\left\{\sum_{i=-m}^{n-1} K_i 2^i\right\}_2 \tag{1-3}$$

其中：K_i 为 0 或 1，2 为位权。

例如：二进制数 1011.101 可以展开为：

$1011.101 = 1\times 2^3 + 0\times 2^2 + 1\times 2^1 + 1\times 2^0 + 1\times 2^{-1} + 0\times 2^{-2} + 1\times 2^{-3}$

2. 二进制数的运算

二进制数的算术运算比较简单，只要记住二个二进制整数的和与积的运算规律就可以了。

加法规律为：

$$0+0=0 \qquad 0+1=1+0=1 \qquad 1+1=10$$

乘法规律为：

$$0\times 0=0 \qquad 0\times 1=1\times 0=0 \qquad 1\times 1=1$$

由于二进制数每位只可能有两种数值 0 或者 1，在数字系统中，可用电子器件的两种不同的状态来表示一位二进制数，因此实现起来非常方便。例如，我们在数字系统中，用晶体管的导通表示“0”，而晶体管的截止表示“1”；或用低电位表示“0”、高电位表示“1”。所以二进制数的物理实现简单、易行、可靠，并且存储和传送也方便。其运算规则也很简单。但二进制书写位数太多，不便记忆。为此，通常用八进制和十六进制数作为二进制数的缩写。

1.1.2 八进制

八进制数的数位符号有八个，即 0～7，进位规律是“逢八进一”，基数 R＝8，任意的一个八进制数 N 的多项展开式为：

$$(N)_8 = K_{n-1}8^{n-1} + K_{n-2}8^{n-2} + \cdots + K_1 8^1 + K_0 8^0 + K_{-1}8^{-1} + \cdots + K_{-m}8^{-m}$$

$$=\left\{\sum_{i=-m}^{n-1} K_i 8^i\right\}_8 \tag{1-4}$$

其中：K_i 表示为 0～7 中的任意一个。

1.1.3 十六进制

一个十六进制数的表示符号有 16 个，即 0～9，以及用 A、B、C、D、E、F 分别表示 10～15。进位规律为“逢十六进一”，基数 R＝16，任意的一个十六进制数 N 的多项表示式为：

$$(N)_{16} = K_{n-1}16^{n-1} + K_{n-2}16^{n-2} + \cdots + K_1 16^1 + K_0 16^0 + K_{-1}16^{-1} + \cdots + K_{-m}16^{-m}$$

$$=\left\{\sum_{i=-m}^{n-1} K_i 16^i\right\}_{16} \tag{1-5}$$

其中，K_i 表示为 0～9 以及 A、B、C、D、E、F 中的任意一个。

1.1.4 二进制与八进制、十六进制之间的转换

1. 八进制转换为二进制

把每位八进制数用三位二进制数表示。

例如$(312.64)_8$，3、1、2、6、4 各用三位二进制数表示：

3	1	2	.	6	4
011	001	010	.	110	100

所以$(312.64)_8=(11001010.1101)_2$

2. 二进制数转换为八进制

二进制转换为八进制时，整数部分从低位向高位每三位分为一组，最高一组不够时，用 0 补足；小数部分从高位向低位每三位一组，最后不足三位的，在低位补 0，然后把每三位的二进制数用相应的八进制数表示。

例如$(10110.11)_2$

010	110	.	110
2	6	.	6

$(10110.11)_2=(26.6)_8$

3. 十六进制转换为二进制

把每位十六进制数用相应的四位二进制数表示。

例如$(21A.5)_{16}$

2	1	A	.	5
0010	0001	1010	.	0101

$(21A.5)_{16}=(1000011010.0101)_2$

4. 二进制转换为十六进制

整数部分由小数点向左，每四位一组，最高一组不足四位数前面补 0；小数部位由小数点向右，每四位一组，最后不足四位的，在低位补 0，然后，把每四位二进制数用相应的十六进制数表示。

例如：$(1100101.101)_2$

0110	0101	.	1010
6	5	.	A

$(1100101.101)_2=(65.A)_{16}$

由于八进制、十六进制比二进制书写简短、容易读写，也便于记忆，并且转换为二进制也较方便。因此，在数字系统中得到普遍应用。

1.1.5 二进制与十进制之间的转换

1. 二进制转换为十进制

把二进制数按权展开，利用十进制运算法则，求出其值，即可将二进制数转换为十进制数。

例如：

$$(101101.101)=1\times 2^5+1\times 2^3+1\times 2^2+1\times 2^0+1\times 2^{-1}+1\times 2^{-3}$$
$$=32+8+4+1+0.5+0.125$$
$$=(45.625)_{10}$$

2. 十进制转换为二进制

一个具有整数部位和小数部位的十进制数转换为二进制数时，应当分别将其整数部分和小数部分转换为二进制数，然后用小数点将两部分连接起来。现举例说明。

例如：将十进制数$(157)_{10}$转换为二进制数：

根据公式(1-2)：

$$(157)_{10}=K_{n-1}2^{n-1}+K_{n-2}2^{n-2}+\cdots+K_1 2^1+K_0 2^0$$

显然，等式右边除K项外都有2的因子。因此，用2除$(157)_{10}$，所得余数即为K_0，

$$2\ \big|\ 157$$
$$78\cdots\cdots\text{余数}\ 1=K_0$$

并得到等式：

$$(78)_{10}=K_{n-1}2^{n-2}+K_{n-2}2^{n-3}+\cdots+K_2 2^1+K_1$$

同样，再用2除$(78)_{10}$，余数则为K_1，

$$2\ \big|\ 78$$
$$39\cdots\cdots\text{余数}\ 0=K_1$$

再用这样的方法一直继续下去，直到商为0为止。

2	157	
2	78	余数为1，所以$K_0=1$
2	39	余数为0，所以$K_1=0$
2	19	余数为1，所以$K_2=1$
2	9	余数为1，所以$K_3=1$
2	4	余数为1，所以$K_4=1$
2	2	余数为0，所以$K_5=0$
2	1	余数为0，所以$K_6=0$
	0	余数为1，所以$K_7=1$

得$(157)_{10}=(10011101)_2$

十进制小数转换为二进制小数的方法是：不断用2乘要转换的十进制小数，将每次所得的整数(0或1)，依次记为K_{-1}、K_{-2}、…。若乘积的小数部分最后能为0，那么最后一

次乘积的整数部分记作 K_{-m}，则 $0.K_{-1}K_{-2}\cdots K_{-m}$ 即为十进制小数的二进制表达式。因为十进制小数，并不都是能用有限位的二进制小数精确表示，通常则是根据精度要求 m 位，作为十进制小数的二进制的近似表达式。

例如：将 $(0.913)_{10}$ 转换为二进制数，根据公式(1-2)：

$$(0.913)_{10} = K_{-1}2^{-1} + K_{-2}2^{-2} + \cdots + K_{-m}2^{-m}$$

两边乘以 2，则得：

$$(1.826)_{10} = K_{-1} + K_{-2}2^{-1} + \cdots + K_{-m}2^{-m+1}$$

所以 $K_{-1}=1$；那么，$(0.826)_{10}=K_{-2}2^{-1}+K_{-3}2^{-2}+\cdots+K_{-m}2^{-m+1}$

两边再乘以 2，则得：

$$(1.652)_{10} = K_{-2} + K_{-3}2^{-1} + K_{-4}2^{-2} + \cdots + K_{-m}2^{-m+2}$$

假如精度要求 m=4，转换过程如下：

$$\begin{array}{r} 0.913 \\ \times \quad 2 \\ \hline 1.862 \end{array}$$

整数部分为 1，所以 $K_{-1}=1$；

$$\begin{array}{r} 0.826 \\ \times \quad 2 \\ \hline 1.652 \end{array}$$

整数部分为 1，所以 $K_{-2}=1$；

$$\begin{array}{r} 0.652 \\ \times \quad 2 \\ \hline 1.304 \end{array}$$

整数部分为 1，所以 $K_{-3}=1$；

$$\begin{array}{r} 0.304 \\ \times \quad 2 \\ \hline 0.608 \end{array}$$

整数部分为 0，所以 $K_{-4}=0$；

因此，$(0.913)_{10}=(0.1110)_2$

任意进制与十进制之间的转换原理及方法，同二进制与十进制之间的转换原理及方法相类似，不再重复。

而任意两种进制之间的转换，一般说来是先由一种进位制转换为十进制，再由十进制转换为另一种进制，把十进制作为桥梁。例如实现 $(N)_b$ 转换为 $(N)_k$ 时，首先是将 $(N)_b$ 转换为 $(N)_{10}$，也就是将 $(N)_b$ 展开为 a 进位制的多项式，在十进制中计算其值，然后再利用基数乘除法，将 $(N)_{10}$ 转换为 $(N)_k$。

1.2 编　码

1.2.1 带符号的二进制数的编码

在通常的算术运算中，用“+”号表示正数，用“−”号表示负数。而在数字系统中，正、负数的表示方法是：把一个数的最高位作为符号位，并用“0”表示“+”；用“1”表示“−”。连同符号位在一起作为一个数，称之为机器数，它的原来的数值形式则称为这个机器数的真值。

例如：　　$X_1=+0.1101$；　　$X_2=-0.1101$

表示成机器数为：　　$X_1=0.1101$；　　$X_2=1.1101$

在数字系统中，表示机器数的方法很多，目前常用的有原码、反码和补码。

1. 原码(True Form)

原码表示法又称符号—数值表示法，正数的符号位用“0”表示，负数的符号位用“1”表示，数值部分保持不变。

(1) 小数原码的定义：

若二进制数 $X=\pm 0.X_{-1}X_{-2}\cdots X_{-m}$

1) $X>0$ 时

$$X=+0.X_{-1}X_{-2}\cdots X_{-m}$$

$$(X)_{原}=0.X_{-1}X_{-2}\cdots X_{-m}$$

2) $X<0$

$$X=-0.X_{-1}X_{-2}\cdots X_{-m}$$

$$(X)_{原}=1.X_{-1}X_{-2}\cdots X_{-m}$$

$$=1-(-0.X_{-1}X_{-2}\cdots X_{-m})$$

$$=1+X$$

例如：$X_1=+0.1101$　　则 $(X)_{原}=0.1101$

$X_2=-0.1101$　　则 $(X)_{原}=1-(-0.1101)=1.1101$

3) 零的原码有两种表示形式

$$(+0)_{原}=0.00\cdots0$$

$$(-0)_{原}=1.00\cdots0$$

所以小数原码表示为：

$$(X)_{原}=\begin{cases}X & \text{当 } 0\leqslant X<1\\ 1-X & \text{当 } -1<X\leqslant 0\end{cases}$$

(2) 整数原码的定义

若 $X=\pm X_{n-1}X_{n-2}\cdots X_0$

1) $X>0$ 时，则　　$X=+X_{n-1}X_{n-2}\cdots X_0$

$$(X)_{原}=0X_{n-1}X_{n-2}\cdots X_0$$

2) $X<0$ 时，则　　$X=-X_{n-1}X_{n-2}\cdots X_0$

$$
\begin{aligned}
(X)_{补} &= 1X_{n-1}X_{n-2}\cdots X_0 \\
&= 2^n + X_{n-1}X_{n-2}\cdots X_0 \\
&= 2^n - (-X_{n-1}X_{n-2}\cdots X_0) \\
&= 2^n - X
\end{aligned}
$$

例如：X＝－1101

$$
\begin{aligned}
(X)_{补} &= 11101 \\
&= 10000 + 1101 \\
&= 10000 - (-1101) \\
&= 2^n - X
\end{aligned}
$$

因此，整数原码定义为：

$$
(X)_{补} = \begin{cases} X & 当\ 0 \leqslant X < 2^n \\ 2^n - X & 当\ -2^n < X \leqslant 0 \end{cases}
$$

原码表示法简单易懂，但在数字系统中，要进行两个异号原码的加法运算时，需先判两数的大小，然后才能从大数中减去小数。最后，还要判结果的符号位，这就增长了运算时间。

2. 反码(One's complement)

反码的符号位表示法与原码相同，即符号“0”表示正数，符号“1”表示负数。与原码不相同的是反码数值部分的形成和它的符号位有关。正数反码的数值和原码的数值相同，而负数反码的数值是原码的数值按位求反。

(1) 整数的反码

若 $X = \pm X_{n-1}X_{n-2}\cdots X_0$

则定义为：

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

例如：$X_1 = +1101$，则$(X_1)_{反} = 01101 = 1101$

$X_2 = -1101$，则$(X_2)_{反} = (2^5 - 1) + X$

$$
\begin{aligned}
&= (100000 - 000001) + (-1101) \\
&= 11111 - 1101 \\
&= 10010
\end{aligned}
$$

(2) 小数反码的定义

若 $X = \pm 0.X_{-1}X_{-2}\cdots X_{-m}$

则定义为：

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

例如；$X_1 = +0.1101$，则$[X_1]_{反} = 0.1101$

$X_2 = -0.1101$，则$[X_2]_{反} = 2 - 2^{-4} + X$

$$
= 10.0000 - 0.0001 - 0.1101
$$

$$=1.0010$$

(3) 零的反码有两种形式：

$$[+0]_{反}=0.00\cdots0$$

$$[-0]_{反}=1.11\cdots1$$

作反码加、减法时，要将运算结果的符号位产生的进位(0或1)加到和的最低位，才能得到最后结果。

例如：将 X_1、X_2 作反码加：$X_1=+1101$，　$X_2=-0101$

$$[X_1]_{反}=01101，\quad [X_2]_{反}=11010$$

$$[X_1]_{反}+[X_2]_{反}=01101+11010$$

$$=\text{“1”}00111$$

将符号位产生的进位“1”，加到最低位，即为

$$[X_1+X_2]_{反}=00111+1=01000$$

在反码表示中，±0的表示不是唯一的，因此，使用反码不很方便。

3. 补码(Two's complement)

补码的符号表示和原码相同。“0”表示正数；“1”表示负数。正数的补码和原码、反码相同，就是二进制数值本身。负数的补码是这样得到的：将数值部分按位求反，再在最低位加1。

(1) 整数的补码

若 $X=\pm X_{n-1}X_{n-2}\cdots X_0$

则定义为：

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

例如：$X_1=+1101$，　则 $[X_1]_{补}=1101$

$X_2=-1101$，　则 $[X_2]_{补}=2^5+X$

$$=100000-1101$$

$$=10011$$

(2) 小数的补码

若 $X=\pm0.X_{-1}X_{-2}\cdots X_{-m}$

则定义为：

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

(3) 零的补码只有一种形式：

$$[0]_{补}=0.000\cdots0$$

引入补码以后，可将数字系统的减法运算用加法实现。在求得和的结果中，要将运算结果产生的进位丢掉，才得到正确结果。

例如：在补码系统中，实现 X_1-X_2 运算：

$$X_1=1101，\quad X_2=0101$$

求出：$[X_1]_{补}=1101$，　$[-X_2]_{补}=11011$

做$[X_1]_补+[-X_2]_补=1101+11011$

$=$"1"01000

丢掉最高位的"1"，即得$(X_1-X_2)_补$的正确结果。

显然，两数相减时，用补码求和运算比用原码求和要简单。但补码的缺点是负数用补码表示不直观。

表 1-1 给出了几个典型数的真值、原码、反码、补码的表示。

表 1-1

X	$[X]_原$	$[X]_反$	$[X]_补$	X	$[X]_原$	$[X]_反$	$[X]_补$
+1001	1001	1001	1001	−0.0000	1.0000	1.1111	0.0000
+0001	0001	0001	0001	−0.0010	1.0010	1.1101	1.1110
+0.1101	0.1101	0.1101	0.1101	−0011	1.0011	1.1100	1.1101
+0.0000	0.0000	0.0000	0.0000	−1010	1.1010	1.0101	1.0110

1.2.2 带小数点的数的编码

一个数既有小数部分又有整数部分，在数字系统中是如何表示的？一般有两种方式：定点表示和浮点表示。

任何数制的数 N，均可以表示为：

$$N = R^E \times M \tag{1-6}$$

其中：R 为进制的基数；E(Exponent)为阶码，取值为整数；M(Mantissa)为数 N 的尾数，取值为整数或小数。

例如：$(N_1)_{10}=(516000)_{10}=10^3\times516$

$(N_2)_{10}=(0.25)_{10}=10^{-2}\times25$

数 N_1 的阶码 $E_1=3$、尾数 $M_1=516$；数 N_2 的阶码 $E=-2$，尾数 $M=25$。

对于二进制数，

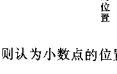
$$N = 2^E \times M \tag{1-7}$$

1. 定点表示法：

所谓的定点表示法就是在一个数中小数点的位置在数中是固定不变的。这个固定的位置是事先约定好的，不必用符号表示。在定点表示中，阶码 E 为零。

当 E=0，尾数 M 为纯整数时，则认为小数点在尾数 M 最低位右边，为整数定点。

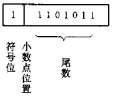
例如：N=+1011011，则表示为



当 E=0，尾数 M 为纯小数时，则认为小数点的位置在 M 最高位的左边，定点数只能

表示小数，为小数定点。

例如：N＝－0.1101011，则表示为

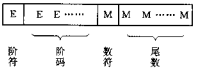


定点数表示法，数 N 范围是限定的，在小数定点时，当用 8 位二进制数表示一个数时，1 位符号位，7 位表示数值，若只考虑绝对值，最大数取值为$(0.1111111)_2=(127\div128)_{10}$

最小数取值为$(0.0000001)_2=(1\div128)_{10}$

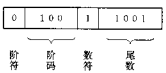
2. 浮点表示法

在数中小数点的位置不是固定不变的，而是可以变化的，这种表示法称为浮点表示法。阶码 E 和尾数 M 各自可分别采用原码、反码和补码的形式。若尾数 M 采用反码表示，阶码 E 采用补码表示，表示格式如下：



例如：$N=2^4\times(-9)$，用二进制数表示为　　$N=2^{100}\times(-1001)$

浮点表示的格式如下：



尾数 M 表示了数 N 的全部有效数字；而阶码 E 指明了小数点的位置。小数点移动的原则是小数点向左移一位，相当于尾数的数码向右移一位，而阶码加 1。

浮点数的运算规则要比定点数复杂。如阶码相同的两浮点数求和，只要将尾数作相加运算，所得为和数的尾数，而阶码仍为原来的阶码。如阶码不等的两数求和，则先将阶对齐，然后才能对尾数求和。例如：

$$a=2^{11}\times0.1001$$

$$b=2^{01}\times0.1100$$

首先对阶，让小阶向大阶看齐，阶小的尾数小数点要左移一位阶码加 1，直到阶码相同为止。对 a、b 来说，必须使 b 的尾数左移两位，阶码加 2，得：

$$b=2^{11}\times0.0011$$

然后相加：

$$\begin{array}{l} 2^{11}\times 0.1001 \\ 2^{11}\times 0.0011 \\ \hline 2^{11}\times 0.1100 \end{array}\quad(+$$

两浮点数相乘，只要阶码相加，尾数相乘；两浮点数相除，只要阶码相减，尾数相除。

1.2.3 十进制数的二进制编码

这节叙述用二进制数码按照不同规律编码来表示十进制数。这样的二进制数既具有二进制的形式，又具有十进制数的特点，便于传递、处理。

一位十进制数有0～9十个不同数码，需要用四位二进制数才能表示。四位二进制数可组合$2^4=16$种不同状态。从16种状态中取十种状态来表示0～9的编码方式很多。一般分为有权码和无权码。所谓有权码是指四位二进制数中的每一位都对应有固定的权，四位权之和为所表示的十进制数。无权码是指四位二进制数中的每一位无固定的权，而要遵循另外的规则。最常用的十进制数的二进制编码有以下几种：

1. 8421码

8421码为有权码。是十进制代码中最常见的代码，也称二-十进制码，或称BCD码(Binary-Code-Decimal)，四位二进制码从高位至低位每位的权分别为2^3、2^2、2^1、2^0，即为8、4、2、1。

8421码的编码表如表1-2所示。

显然，8421码表与普通二进制数0～9表示的形式完全相同。但二进制码1010～1111在8421码中是没有意义的。

8421码和十进制数之间的转换是一种直接按位转换。

例如：$(12)_{10}=(00010010)_{BCD}$

$(0100100101110101)_{BCD}=(4975)_{10}$

2. 5421码

5421是一种有权码，其权自高位至低位，每位分别为5、4、2、1。所以，十进制数X用5421码表示为：

$$X=a_3\cdot 5+a_2\cdot 4+a_1\cdot 2+a_0\cdot 1$$

5421码的编码表如表1-2所示。

3. 2421码

2421码为另一种有权码，也是四位代码，每位权从高位到低位为2、4、2、1，若一个2421编码的二进制数码为$a_3a_2a_1a_0$时，它表示的十进制数值为：

$$X=2a_3+4a_2+2a_1+1a_0$$

表1-2给出了它的编码表。

4. 余三码(Excess 3 code)

余三码是一种无权码。十进制数用余三码表示，要比8421码在二进制数值上多3，故称余三码，它可由8421码加0011得到。

从表 1-2 中看到，余三码表中，0 和 9；1 和 8；2 和 7；3 和 6；4 和 5 互为反码。因此，余三码作十进制数的算术运算是比较方便的。

表 1-2　四种十进制数的编码

十进制数	8421	5421	2421	余三码
0	0000	0000	0000	0011
1	0001	0001	0001	0100
2	0010	0010	0010	0101
3	0011	0011	0011	0110
4	0100	0100	0100	0111
5	0101	1000	0101	1000
6	0110	1001	0110	1001
7	0111	1010	0111	1010
8	1000	1011	1110	1011
9	1001	1100	1111	1100

1.2.4　格雷码(Gray Code)

格雷码是一种无权码，它有多种形式，表 1-3 给出四种格雷码。格雷码共同的特点是：任何两个相邻的十进制数的格雷码仅有一位不同。这种代码可以减少代码变换中产生的错误，所以它是一种高可靠性编码。从表 1-3 中的格雷码 1、2 可以看出，除十进制数的相邻两个数的格雷码只有一位不同外，格雷码 1 还有另一特点，即相对该组编码的中线(中线位于十进制数 4、5 之间)而言，其最高位的代码一一相反，而其余各位的代码则相同，就是说各个代码之间对中线一一“反射”，通常把这种具有反射性的格雷码称为反射码。

表 1-3　十进制数的四种格雷码表

十进制数	8421 BCD 码	格雷码 1	格雷码 2	典型格雷码	修改格雷码
0	0000	0000	0000	0000	0010
1	0001	0001	0001	0001	0110
2	0010	0011	0011	0011	0111
3	0011	0010	0010	0010	0101
4	0100	0110	0110	0110	0100
5	0101	1110	0111	0111	1100
6	0110	1010	0101	0101	1101
7	0111	1011	0100	0100	1111
8	1000	1001	1100	1100	1110
9	1001	1000	1000	1101	1010

在表 1-3 中的典型格雷码，不仅可以对十进制进行编码而且能对任意二进制数进行编码。对任意二进制(8421 码)进行编码规则是：设二进制数 $B=B_{n-1}B_{n-2}\cdots B_{i+1}B_i\cdots B_1B_0$，对应的格雷码 $G=G_{n-1}G_{n-2}\cdots G_{i+1}G_i\cdots G_1G_0$，则有：

$$G_i = B_{i+1} \oplus B_i$$

即格雷码的第 i 位(G_i)是二进制码第 i 位(B_i)和第 i+1 位(B_{i+1})的模 2 和。模 2 和的运算符是⊕，运算法则是：

$$0 \oplus 0 = 0; \quad 1 \oplus 0 = 0 \oplus 1 = 1; \quad 1 \oplus 1 = 0$$

所谓"模 2 和"就是不计进位的二进制加法。

修改格雷码，又称余三循环码，它具有循环和反射格雷码的特点。从表 1-3 可以看到，与十进制数 0 对应的修改格雷码为 0010，而典型格雷码的 0010 恰好对应的十进制数为 3。因此它比典型格雷码从 0000 开始计数的状态多了。

格雷码不直观，因为可靠性高，广泛用于输出、输入等场合。

1.2.5 字符编码

在数字系统中，还需要把符号、文字、图象等用二进制数表示，这样的二进制数称为字符代码。目前在国际上用得最多的字符有：十进制数 0～9；大写和小写英文字母各 26 个；一些通用的运算符号(+、-、×、÷等)及标点符号，共有 128 种。可用七位二进制数对它们进行编码。

1. 七位 ASCII 编码(Americon Standards Committee of Information Interchange 美国信息交换标准委员会)。编码方式：英文字母由 A—Z 及由 a—z 顺序编码；十进制数采用高三位相同，为 011，低四位按二进制数顺序编码。七位 ASCII 的码表如表 1-4 所示。

表 1-4 美国信息交换代码 ASCII CODE

$b_4b_3b_2b_1$ \ $b_7b_6b_5$	000	001	010	011	100	101	110	111
0 0 0 0	NUL	DLE	SP	0	@	P	`	p
0 0 0 1	SOM	DC	!	1	A	Q	a	q
0 0 1 0	STX	DC	"	2	B	R	b	r
0 0 1 1	ETX	DC	#	3	C	S	c	s
0 1 0 0	EOT	DC	$	4	D	T	d	t
0 1 0 1	ENQ	NAK	%	5	E	U	e	u
0 1 1 0	ACK	SYN	&	6	F	V	f	v
0 1 1 1	BEL	ETB	'	7	G	W	g	w
1 0 0 0	BS	CAN	(	8	H	X	h	x
1 0 0 1	HT	EM	)	9	I	Y	i	y
1 0 1 0	LF	SUB	*	:	J	Z	j	z
1 0 1 1	VT	ESC	+	;	K	[	k	
1 1 0 0	FF	FS	,	<	L	\	l	!
1 1 0 1	CR	GS	-	=	M	]	m	
1 1 1 0	SO	RS	.	>	N		n	~
1 1 1 1	SI	US	/	?	O	←	o	DEL

2. 八位 ASCII 编码：用最高二位(b_7b_6)把字符分为四类：

00：表示控制字符

01：表示数字及通用符号

10：表示大写英文字母

11：表示小写英文字母

第五位(b_5)与最高位(b_7)相同；编码的其余五位(b_4～b_0)与七位 ASCII 编码相同。

3. EBCDIC 是扩展二-十进制交换码，八位码都用来表示字符。编码方式类似七位 ASCII 码的编码方式。低四位与 BCD 8421 码相同。

第二章　逻辑代数及逻辑函数的化简

2.1　逻辑代数的基本原理

逻辑代数是英国的数学家乔治·布尔(George Boole)在19世纪中叶提出的。在其著作"逻辑的数学分析"及"思维规律"中，首先阐述了逻辑代数的基本性质和概念。因此，常常把逻辑代数称为布尔代数。目前，逻辑代数是数字系统分析和设计的数学工具。本章主要讨论逻辑代数的基本运算、基本公式以及逻辑函数的化简方法。

2.1.1　逻辑代数的基本运算

逻辑代数和普通代数的共同之处是有变量和变量的运算。逻辑代数用字母表示变量。作为逻辑变量，取值只有"0"或"1"两个。这两个值不是数量上的概念，而是表示两种不同的状态。在逻辑电路中，常用"0"或"1"表示电位的低或高，脉冲的无或有。在人们的逻辑思维中，常用"0"或"1"表示命题的假或真。逻辑代数的基本运算比较简单，只有三种——"与"运算、"或"运算、"非"运算。

1. "与"运算

当决定一事件的所有条件都具备之后，这事件才会而且一定会发生，称这种关系为"与"逻辑关系，或称为逻辑乘关系。

用两个串联的开关A、B控制一盏灯，如图2-1所示。灯亮的条件是开关A"与"开关B"同时"处在"合上"位置。假定：灯亮为"1"，不亮为"0"；开关在"合上位置"为"1"，在"断开位置"为"0"。那么，把灯的状态和两个开关所处位置之间的关系列表，如图2-1左方表所示。把这种表称为真值表(或称为功能表)。

"与"运算真值表

A	B	F
0	0	0
1	0	0
0	1	0
1	1	1

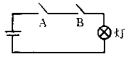


图2-1　"与"逻辑关系举例

常用真值表来表示逻辑命题的真假关系。把所有的条件(变量)的全部组合以表格形式列出来，这里为A、B，再把在每一种组合下对应的事件(函数)的值求出，这里为F，这张表格就是真值表。因为每个条件有两种状态"0"、"1"，因此，n个条件就有2^n个组合。图2-1左方为A"与"B的真值表。A"与"B用表达式表示为：

$$F=A\cdot B \quad 或者 \quad F=A\wedge B$$

一般简写为：$F=AB$，把此式称为变量A、B相"与"的逻辑表达式。

2. “或”运算

当决定一事件的各个条件中，只要具备一个条件，事件就会发生，这样的关系称为“或”逻辑关系，或称逻辑加。

用并联的两个开关 A、B 控制一盏灯，如图 2-2 所示，只要开关 A“或”开关 B 在“合上”位置，灯就亮。按照前面假定来赋值“0”、“1”，列出真值表，如图 2-2 左方所示。“或”运算的逻辑表达式为：

$$F=A+B \quad \text{或者} \quad F=A \vee B$$

一般简写为：$F=A+B$

“或”运算真值表

A	B	F
0	0	0
1	0	1
0	1	1
1	1	1

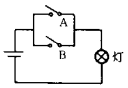


图 2-2 “或”逻辑关系举例

3. “非”运算

“非”运算，就是否定、或者称为求反。如开关 A 和灯的状态有如图 2-3 右方所示的关系，写出灯的状态和开关 A 的位置关系的真值表如图 2-3 左方所示。“非”运算的逻辑表达式为：

$$F=\overline{A}$$

“非”运算真值表

A	F
0	1
1	0

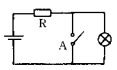


图 2-3 “非”逻辑关系举例

2.1.2 逻辑代数的基本公式、规则、附加公式

1. 基本公式

逻辑代数有些常用的基本公式，这些公式均可用真值表证明。熟悉掌握这些公式，对逻辑函数的化简是非常有用的。逻辑代数的基本公式有：

交换律
$$\left.\begin{aligned} A \cdot B &= B \cdot A \\ A+B &= B+A \end{aligned}\right\} \tag{2-1}$$

结合律
$$\left.\begin{aligned} A \cdot (BC) &= (AB) \cdot C \\ A+(B+C) &= (A+B)+C \end{aligned}\right\} \tag{2-2}$$

分配律
$$\left.\begin{aligned} A(B+C) &= AB+AC \\ A+BC &= (A+B)(A+C) \end{aligned}\right\} \tag{2-3}$$

吸收律

$$\left.\begin{aligned} &A+\overline{A}B=A+B \\ &A(\overline{A}+B)=AB \end{aligned}\right\} \quad (2\text{-}4)$$

$$\left.\begin{aligned} &A+AB=A \\ &A(A+B)=A \end{aligned}\right\} \quad (2\text{-}5)$$

反演律(德·摩根律)

$$\left.\begin{aligned} &\overline{AB}=\overline{A}+\overline{B} \\ &\overline{A+B}=\overline{A}\cdot\overline{B} \end{aligned}\right\} \quad (2\text{-}6)$$

包含律

$$\left.\begin{aligned} &AB+\overline{A}C+BC=AB+\overline{A}C \\ &(A+B)(\overline{A}+C)(B+C)=(A+B)(\overline{A}+C) \end{aligned}\right\} \quad (2\text{-}7)$$

对合律

$$\overline{\overline{A}}=A \quad (2\text{-}8)$$

重叠律

$$\left.\begin{aligned} &A+A=A \\ &A\cdot A=A \end{aligned}\right\} \quad (2\text{-}9)$$

互补律

$$\left.\begin{aligned} &A\cdot\overline{A}=0 \\ &A+\overline{A}=1 \end{aligned}\right\} \quad (2\text{-}10)$$

0 律

$$\left.\begin{aligned} &0\cdot A=0 \\ &0+A=A \end{aligned}\right\} \quad (2\text{-}11)$$

1 律

$$\left.\begin{aligned} &1\cdot A=A \\ &1+A=1 \end{aligned}\right\} \quad (2\text{-}12)$$

以上公式也是逻辑代数的基本定律。利用它们，还可以推导出常用的逻辑代数的其他公式。例如：

(1) $AB+A\overline{B}=A$

证：

$AB+A\overline{B}$

$=A(B+\overline{B})$ [根据(2-3)式]

$=A\cdot 1$ [根据(2-10)式]

$=A$ [根据(2-12)式]

所以 $AB+A\overline{B}=A$ (2-13)

(2) $(A+B)(A+\overline{B})=A$

证：

$(A+B)(A+\overline{B})$

$=A+B\overline{B}+AB+A\overline{B}$ [根据(2-3)式]

$=A+0+A$ [根据(2-10)、(2-13)式]

$=A$ [根据(2-9)、(2-11)式]

所以 $(A+B)(A+\overline{B})=A$ (2-13)

(3) $AB+\overline{A}C+BCD=AB+\overline{A}C$

用式(2-7)将上式左边改写为：

$(AB+\overline{A}C+BC)+BCD$

$=AB+\overline{A}C+BC(1+D)$ [根据(2-3)式]

$=AB+\overline{A}C+BC\cdot 1$ [根据(2-12)式]

$=AB+\overline{A}C$ [根据(2-7)式]