第 1 章　数制与编码

众所周知，在计算机系统中使用的是二进制数，而人类在日常生活中使用的则是十进制数。因此，当我们要将数据输入到计算机时，就需要进行数制转换（Base Conversion），将十进制转换为二进制；而当计算机将数据输出（显示、打印）时，则需要将二进制转换为十进制，因此我们必须熟悉二进制数和十进制数之间的转换。为记忆、阅读方便，人们还经常使用八进制和十六进制，因此我们也要熟悉它们与二进制及十进制之间的关系。

除了数字外，数字系统还要存储、处理一些字符，如字母、运算符、各种符号、汉字等，这就需要用二进制码去表示这些信息，以适应数字系统中的二进制。

1.1　数　　制

任意进制的数都是由若干位数字组成的，每位上的数字所表示的意义是不相同的，也就是它们的**权**不同。例如，十进制数的 345.67 所表示的意义为：

$$(345.67)_{10} = 3 \times 10^2 + 4 \times 10^1 + 5 \times 10^0 + 6 \times 10^{-1} + 7 \times 10^{-2}$$

也就是说，在 345.67 中，3 的权为 10^2，4 的权为 10^1，…

一般情况下，R 进制的数 N 可表示为如下形式：

$$(N)_{\mathrm{R}} = K_{n-1}K_{n-2}\cdots K_0.K_{-1}K_{-2}\cdots K_{-m} \tag{1.1}$$

式中，R 为基数，K_i 为在 0，1，…，R–1 范围中取值的数字。这个数的按权展开式为

$$\begin{aligned}(N)_{\mathrm{R}} &= K_{n-1}R^{n-1} + K_{n-2}R^{n-2} + \cdots + K_0R^0 + K_{-1}R^{-1} + K_{-2}R^{-2} + \cdots + K_{-m}R^{-m} \\ &= \sum_{i=-m}^{n-1} K_i R^i\end{aligned} \tag{1.2}$$

由式（1.2）可见，第 i 位上的数字 K_i 所表示的数的大小为 $K_i \times R^i$，这个 R^i 就是 K_i 的权。整个数的大小是所有数字的加权之和。

对于十进制有 $N_{\mathrm{D}} = \sum_{i=-m}^{n-1} d_i \cdot 10^i$，$d_i$ 的取值范围为 0, 1,…, 9；

对于二进制有 $N_{\mathrm{B}} = \sum_{i=-m}^{n-1} b_i \cdot 2^i$，$b_i$ 的取值范围为 0,1；

对于八进制有 $N_{\mathrm{Q}} = \sum_{i=-m}^{n-1} q_i \cdot 8^i$，$q_i$ 的取值范围为 0,1,…,7；

对于十六进制有 $N_{\mathrm{H}} = \sum_{i=-m}^{n-1} h_i \cdot 16^i$，$h_i$ 的取值范围为 0, 1,…, 9, A, B, C, D, E, F。其中，A,B,C,D,E,F 分别对应于十进制的 10, 11, 12, 13, 14, 15。

1.2 数 制 转 换

1.2.1 二、八、十六进制到十进制的转换

根据数制（Number System）的定义，二、八、十六进制到十进制的转换用加权相加公式 $\sum_{i=-m}^{n-1} K_i R^i$ 直接相加即可。

【例 1.1】 $(101.001)_2 = 1\cdot2^2+0\cdot2^1+1\cdot2^0+0\cdot2^{-1}+0\cdot2^{-2}+1\cdot2^{-3} = (5.125)_{10}$

【例 1.2】 $(32.56)_8 = 3\cdot8^1+2\cdot8^0+5\cdot8^{-1}+6\cdot8^{-2} = (26.71875)_{10}$

【例 1.3】 $(ED.A)_{16} = 14\cdot16^1+13\cdot16^0+10\cdot16^{-1} = (237.625)_{10}$

1.2.2 二、八、十六进制之间的转换

表 1.1 所示为对应十进制数 0～15 的二、八、十六进制数。

表 1.1 十、二、八、十六进制数对照表

十进制数	二进制数	八进制数	十六进制数
0	0000	00	0
1	0001	01	1
2	0010	02	2
3	0011	03	3
4	0100	04	4
5	0101	05	5
6	0110	06	6
7	0111	07	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

观察表 1.1 中二进制数和八进制数之间的关系可知，每三位二进制数对应一位八进制数，由此可得二进制到八进制和八进制到二进制的转换方法。

【例 1.4】 $(10010111.1101)_2 = (\underline{010}\ \underline{010}\ \underline{111}.\underline{110}\ \underline{100})_2 = (227.64)_8$

【例 1.5】 $(227.64)_8 = (\underline{010}\ \underline{010}\ \underline{111}.\underline{110}\ \underline{100})_2 = (10010111.1101)_2$

例 1.4 说明了二进制到八进制的转换方法：小数点左边从右向左每三位一组，最左边一组不够三位时左边补 0；小数点右边从左向右每三位一组，最右边一组不够三位时右边补 0。然后按顺序分别写出每三位二进制数所对应的八进制数即可。例 1.5 说明了八进制到二进制的转换方法：将八进制数转换成二进制数时，只要将每位八进制数按顺序分别写成它们所对应的二进制数即可。注意，整数部分除最高位外，每个三位二进制数中最左边的 0 不能省略；小数部分除最低位外，每个三位二进制数中最右边的 0 也不能省略。

观察表 1.1 中二进制数和十六进制数之间的关系可以看出，每 4 位二进制数对应一位十六进制数。由此可得二进制到十六进制和十六进制到二进制的转换方法。

【例 1.6】 $(110110111.011)_2=(\underline{0001}\ \underline{1011}\ \underline{0111}.\underline{0110})_2=(1B7.6)_{16}$

【例 1.7】 $(1C7.6)_{16}=(\underline{0001}\ \underline{1100}\ \underline{0111}.\underline{0110})_2=(111000111.011)_2$

例 1.6 说明了二进制到十六进制的转换方法：小数点左边从右向左每 4 位一组，最左边一组不够 4 位时左边补 0；小数点右边从左向右每 4 位一组，最右边一组不够 4 位时右边补 0。然后按顺序分别写出每 4 位二进制数所对应的十六进制数即可。例 1.7 说明了十六进制到二进制的转换方法：将十六进制数转换成二进制数时，只要将每位十六进制数按顺序分别写成它们所对应的二进制数即可。注意，中间位的所有的 0 均不能省略。

八进制数到十六进制数之间的转换可以通过转换为二进制数作为中间过程来方便地完成。例 1.8 和例 1.9 说明了这个转换过程。

【例 1.8】 $(1C7.6)_{16}=(\underline{0001}\ \underline{1100}\ \underline{0111}.\underline{0110})_2=(\underline{111}\ \underline{000}\ \underline{111}.\underline{011})_2=(707.3)_8$

【例 1.9】 $(707.3)_8=(\underline{111}\ \underline{000}\ \underline{111}.\underline{011})_2=(\underline{0001}\ \underline{1100}\ \underline{0111}.\underline{0110})_2=(1C7.6)_{16}$

1.2.3　十进制到二、八、十六进制的转换

将十进制数转换到二、八、十六进制数时，要将整数部分和小数部分分别进行转换，整数部分用连除法，而小数部分用连乘法。

1. 十进制数到二进制数的转换

整数部分转换用连除法，即用 2 去除所要转换的数，所得余数即为 b_0；再用 2 去除上一步所得的商，所得余数为 b_1，…，一直除到商为 0 时为止。

【例 1.10】 $(59)_{10}=(?)_2$

解： $0 \leftarrow 1 \leftarrow 3 \leftarrow 7 \leftarrow 14 \leftarrow 29 \leftarrow 59$　连除以 2，商写在箭头左侧，直到商为 0；

1　1　1　0　1　1　余数写在商下边，连起来就是所求二进制

b_5　b_4　b_3　b_2　b_1　b_0　数，b_0 在最右边。

所以$(59)_{10}=(111011)_2$。

小数部分转换用连乘法，将小数部分乘以 2，所得积的整数部分即为 b_{-1}；积的小数部分再乘以 2，所得积的整数部分为 b_{-2}，…，一直乘到所要求的精度为止。

【例 1.11】 $(0.8125)_{10}=(?)_2$

解： $0.8125\rightarrow0.625\rightarrow0.25\rightarrow0.5\rightarrow0$　乘以 2，积的小数部分写在箭头右侧；

1　1　0　1　积的整数部分写在小数部分下侧，连

b_{-1}　b_{-2}　b_{-3}　b_{-4}　起来就是所求二进制数，b_{-1} 在最左边。

所以$(0.8125)_{10}=(0.1101)_2$。

例 1.11 中的小数部分最后可以乘到 0，此时的转换是精确转换，即十进制数与转换后得到的二进制数相等。

【例 1.12】 $(0.62)_{10}=(?)_2$，要求小数点后精确到 5 位。

解： $0.62\rightarrow0.24\rightarrow0.48\rightarrow0.96\rightarrow0.92\rightarrow0.84$ 乘以 2，积的小数部分写在箭头右侧；

1　0　0　1　1　积的整数部分写在小数部分下侧，连

b_{-1}　b_{-2}　b_{-3}　b_{-4}　b_{-5}　起来就是所求二进制数，b_{-1} 在最左边。

所以$(0.62)_{10}\approx(0.10011)_2$。

例 1.12 中的小数部分永远乘不到 0，此时不能做精确转换，只能取近似值。转换误差为

$|0.62_{10}-0.10011_2| = 0.62-(2^{-1}+2^{-4}+2^{-5}) = 0.62-0.59375 = 0.02625$。如果要求精度更高，可增加转换后小数的位数。

【例 1.13】 $(59.62)_{10} = (?)_2$，要求转换结果精确到小数点后 5 位。

解：将整数、小数部分分别转换，如例 1.10 和例 1.12 所示，有

$$(59.62)_{10} \approx (111011.10011)_2$$

2. 十进制数到八进制数和十六进制数的转换

方法一：与十进制数转换到二进制数类似，将十进制数转换到八进制数（十六进制数）时，整数部分连除以 8（16），余数为 q_i（h_i）；小数部分连乘以 8（16），整数部分为 q_{-i}（h_{-i}）。

【例 1.14】 $(59)_{10} = (?)_8$

解： $0 \leftarrow 7 \leftarrow 59$　　除以 8，商写在箭头左侧，除到商为 0 时为止；

7　3　　余数写在商下边，连起来就是所求八进制数，q_0 在最右边。

q_1　q_0

所以 $(59)_{10} = (73)_8$。

【例 1.15】 $(59)_{10} = (?)_{16}$

解： $0 \leftarrow 3 \leftarrow 59$　　除以 16，商写在箭头左侧，除到商为 0 时为止；

3　11　　余数写在商下边，余数所对应的十六进制数连起

3　B　　来就是所求十六进制数，h_0 在最右边。

h_1　h_0

所以$(59)_{10} = (3B)_8$。

【例 1.16】 $(0.8125)_{10} = (?)_8$

解： $0.8125 \rightarrow 0.5 \rightarrow 0.$　　乘以 8，积的小数部分写在箭头右侧；

6　4　　积的整数部分写在小数部分下侧，连起来

q_{-1}　q_{-2}　　就是所求八进制数，q_{-1} 在最左边。

所以$(0.8125)_{10} = (0.64)_8$。

例 1.16 中的小数部分最后可以乘到 0，此时的转换是精确转换，即十进制数与转换后得到的八进制数相等。

【例 1.17】 $(0.62)_{10} = (?)_8$，要求小数点后精确到 5 位。

解： $0.62 \rightarrow 0.96 \rightarrow 0.68 \rightarrow 0.44 \rightarrow 0.52 \rightarrow 0.16$ 乘以 8，积的小数部分写在箭头右侧；积的

4　7　5　3　4　整数部分写在小数部分下侧，连起来就是所

q_{-1}　q_{-2}　q_{-3}　q_{-4}　q_{-5}　求八进制数，q_{-1} 在最左边。

所以$(0.62)_{10} \approx (0.47534)_8$。

例 1.17 中的小数部分永远乘不到 0，此时只能取近似值。

用方法一将十进制小数转换为十六进制小数的例子读者可自己练习。

方法二：先将十进制数转换为二进制数，再将二进制数转换为八进制数或十六进制数。

比较方法一与方法二可见，由于乘 8（16）、除 8（16）比乘 2、除 2 在运算数时更复杂，所以用方法二可减轻心算负担；但用方法一运算的次数较少，也有优点。读者可自行确定使用那种方法。

1.3　二进制符号数的表示方法

所谓符号数，就是带正、负号的数。在数字系统中所有信息都是由二进制码表示的，数的正、负也由二进制码表示。本节介绍二进制符号数的表示方法（Signed Number Representation）。

1.3.1　原码表示法

所谓原码表示法，就是用一位二进制数表示符号：0 表示正数，1 表示负数；而数的大小则以该数的绝对值表示。符号位通常放在最高位。如某数字系统中用 8 位存储器存放数据，其中最高位为符号位，其余 7 位为数的绝对值。例如：

$$(+37)_{10}=(+0100101)_2=(00100101)_{原}$$
$$(-37)_{10}=(-0100101)_2=(10100101)_{原}$$
$$(+0)_{10}=(+0000000)_2=(00000000)_{原}$$
$$(-0)_{10}=(-0000000)_2=(10000000)_{原}$$
$$(+127)_{10}=(+1111111)_2=(01111111)_{原}$$
$$(-127)_{10}=(-1111111)_2=(11111111)_{原}$$

以上例子说明：n 位数字系统采用原码表示法时所能表示的十进制数的范围为$-(2^{n-1}-1)$～$+(2^{n-1}-1)$，其中 0 有两种表示形式：+0 和–0。

1.3.2　反码表示法

1．反码（1's Complement）

二进制数每一位上的数字不是 0 就是 1。定义 0 的反码为 1，1 的反码为 0。一个二进制数的反码定义为将二进制数的每一位分别求反而得到的二进制码。

2．符号数的反码表示法

所谓符合数的反码表示法，就是用一位二进制数表示符号：0 表示正数，1 表示负数；正数的大小用原码表示，而负数的大小则以该数的反码表示。符号位通常放在最高位。如某数字系统中用 8 位存储器存放数据，其中最高位为符号位，其余 7 位存放数的大小。例如：

$$(+37)_{10}=(+0100101)_2=(00100101)_{反}$$
$$(-37)_{10}=(-0100101)_2=(11011010)_{反}$$
$$(+0)_{10}=(+0000000)_2=(00000000)_{反}$$
$$(-0)_{10}=(-0000000)_2=(11111111)_{反}$$
$$(+127)_{10}=(+1111111)_2=(01111111)_{反}$$
$$(-127)_{10}=(-1111111)_2=(10000000)_{反}$$

以上例子说明：n 位反码表示法所能表示的十进制数的范围为$-(2^{n-1}-1)$～$+(2^{n-1}-1)$，其中 0 有两种表示法。

1.3.3　补码表示法

1．补码（2's Complement）

设数 N 为有 n 位整数、m 位小数的二进制数，则 N 的补码定义为：

$$(N)_{补,n}=2^n-N \tag{1.3}$$

由定义可知：N 的补码与 N 的大小有关，还与位数 n 有关。

【例 1.18】 $(11001)_{补,8}=2^8-11001=11100111$

【例 1.19】 $(11001.0101)_{补,8}=2^8-11001.0101=11100110.1011$

2．补码的求法

利用补码的定义式（1.3）当然可以求一个数的补码，但较为烦琐。补码有简单的求法，下面就是两种简单的求法。

方法一：将原码补足 n 位后求反加 1 即得其补码。

【例 1.20】 求二进制数 $N=10001$ 的补码，设字长 $n=8$ 位。

解：因为补码与位数有关，故先将数 N 的整数部分补齐为 8 位：$N=00010001$

对 N 求反：11101110

将求反后的数加 1 得：11101111

此即 $n=8$ 时 N 的补码。

方法二：将原码补足 n 位后，从右往左第一个 1 及其右边的 0 不变，其余各位求反即得 N 的补码。

【例 1.21】 求二进制数 $N=10010$ 的补码，设字长 $n=8$ 位。

解：因为补码与位数有关，故先将数 N 补齐为 8 位：

$$N=00010010=\underline{000100}\ \underline{10}$$

$$(N)_{补.8}=\underline{111011}\ \underline{10}$$

其他各位求反 ↑ ↑ 最右边一个 1 及其右边的 0 不变

如果所给二进制数为小数，则应将其整数部分补齐为 n 位。

3．符号数的补码表示法

所谓符号数的补码表示法，就是用一位二进制数表示符号：0 表示正数，1 表示负数；正数的大小用原码表示，而负数的大小则以其绝对值的原码的补码表示。符号位放在最高位。如某数字系统中用 8 位存储器存放数据，其中最高位为符号位，其余各位存放数的大小。例如：

$$(+37)_{10}=(+0100101)_2=(00100101)_{补.8}$$

$$(-37)_{10}=(-0100101)_2=(11011011)_{补.8}$$

$$(+0)_{10}=(+0000000)_2=(00000000)_{补.8}$$

$$(-0)_{10}=(-0000000)_2=(00000000)_{补.8}$$

$$(+127)_{10}=(+1111111)_2=(01111111)_{补.8}$$

$$(-127)_{10}=(-1111111)_2=(10000001)_{补.8}$$

$$(-128)_{10}=(-10000000)_2=(10000000)_{补.8}$$

以上例子说明：+0 和–0 的补码一样，为全 0；n 位符号数的补码表示法所能表示的十进制数的范围为$-2^{n-1}\sim+(2^{n-1}-1)$。

求负数的补码时可以不特别考虑符号位，而将符号位作为一位数来处理。如在 $n=8$ 时求–100101 的补码，可以这样求：将 100101 补齐为 8 位 00100101，其补码为 11011011，其中最高位符号位为 1，表明这是负数。

4．利用补码求符号数的加减运算

如果将加数和被加数均以其补码表示，则只用加法运算器就可完成加减运算。显然这样可以节省硬件，降低生产成本。运算时符号位与其他位一样参与运算。若符号位产生进位，则在结果中忽略该进位，不予考虑。

【例 1.22】 设 $n=8$，有两个正数 $A=10011$，$B=1101$。试用补码求 $A+B$，$A-B$，$B-A$，$-A-B$。

解：$(A)_{补.8}=00010011$，$(B)_{补.8}=00001101$，$(-A)_{补.8}=11101101$，$(-B)_{补.8}=11110011$

$A+B=100000$　　$A-B=110$　　$-A+B=11111010$　　$-A-B=11100000$

```
  00010011      00010011      11101101      11101101
+ 00001101    + 11110011    + 00001101    + 11110011
----------    ----------    ----------    ----------
  00100000     100000110      11111010     111100000
```

例 1.22 说明，在运算时符号位如同其他位一样参与运算；运算结果若符号位有进位，则该进位位在结果中不予考虑；运算结果以补码形式表示。读者可验证结果的正确性。

为什么用补码相加可以做加减运算呢？下面给予证明。

设有两个 n 位正数 N_1、N_2，则 $-N_1$、$-N_2$ 的补码分别为 2^n-N_1 和 2^n-N_2。在 n 位加法器中进行加减运算时共有如下 4 种情况：

① N_1+N_2 就是两个正数相加，结果为正数；

② $N_1-N_2=N_1+(2^n-N_2)=2^n-(N_2-N_1)$，结果取决于 N_2-N_1 的符号：如果 $N_2>N_1$，则结果为负数，$2^n-(N_2-N_1)$ 就是 $-(N_2-N_1)$ 的补码；如果 $N_2<N_1$，则结果为 $2^n+(N_1-N_2)$，由于 $N_1-N_2>0$，而 2^n 为第 $n-1$ 位的进位，位于第 n 位（n 位运算器的最高位为第 $n-1$ 位）上，在 n 位运算器之外，所以结果为 N_1-N_2，是正数；

③ N_2-N_1，结果与 N_1-N_2 类似；

④ $-N_1-N_2=(2^n-N_1)+(2^n-N_2)=2^n+[2^n-(N_1+N_2)]$，其中第 1 个 2^n 为第 $n-1$ 位的进位，位于第 n 位上，在 n 位运算器之外，舍去不管；而 $[2^n-(N_1+N_2)]$ 就是负数 $-(N_1+N_2)$ 的补码。

由此证明了用补码进行加减运算的正确性。

对于 $n=8$ 的情况，当然运算结果不能超出 8 位补码所能表示的数值范围，否则会产生所谓的**溢出**，即运算结果发生错误。

【例 1.23】 设 $n=8$，有两个正数 $A=110011$，$B=1101101$。试用补码求 $A+B$，$A-B$，$B-A$，$-A-B$。

解： $(A)_{补,8}=00110011$，$(B)_{补,8}=01101101$，$(-A)_{补,8}=11001101$，$(-B)_{补,8}=10010011$

```
A+B = 10100000      A-B = 11000110      -A+B = 00111010      -A-B = 01100000
  00110011            00110011            11001101             11001101
+ 01101101          + 10010011          + 01101101           + 10010011
----------          ----------          ----------           ----------
  10100000            11000110           100111010            101100000
```

例 1.23 中，$A+B$ 的结果为负数，而 $-A-B$ 的结果为正数，二者显然错了；而 $-A+B$ 和 $A-B$ 结果正确。

两个符号相异的数相加，结果的绝对值小于任一加数的绝对值，所以此时运算结果不会超出 n 位符号数的表示范围，即不会发生溢出。所以例 1.23 中 $A-B$ 和 $-A+B$ 运算结果肯定正确。

两个正数相加，由于两个数的符号位均为 0，所以符号位肯定不会产生进位；如果此时两个数的绝对值之和不大于 $(2^{n-1}-1)$，则第 $n-2$ 位（即最高数字位）就不会产生进位，运算结果就正确；如果此时两个数的绝对值之和大于 $(2^{n-1}-1)$，则第 $n-2$ 位就会产生进位，这个进位使第 $n-1$ 位（即符号位）为 1，结果成了负数，显然错了，例 1.23 中的 $A+B$ 就是这种情况。

两个负数相加，由于两个数的符号位均为 1，所以此时符号位（第 $n-1$ 位）肯定有进位；如果此时第 $n-2$ 位有进位，则运算结果为负数，结果正确；如果此时第 $n-2$ 位无进位，则运算结果为正数，结果显然错了，例 1.23 中的 $-A-B$ 就是这种情况。

综上所述，利用符号数的补码进行加减运算时，如果两个加数的绝对值之和大于 n 位符号数的表示范围，则 $A+B$ 和 $-A-B$ 的运算结果就会发生错误。这类错误称为**溢出**。溢出只发生在两个加数的符号位相同时。在设计加法器时必须考虑溢出问题，并在溢出时给出报警信号，以提示运算结果出错。

根据以上分析并观察例 1.22 和例 1.23 可知：当第 $n-1$ 位（符号位）和第 $n-2$ 位（最高数字位）不同时有进位（两个负数相加时）或不同时无进位（两个正数相加时）时有溢出发生。设计加法器时可根据这个原理设计溢出指示电路。

1.3.4 符号数小结

本节介绍了符号数的三种表示法，对于加减运算最重要的是补码表示法。表 1.2 所示为 $n=8$ 时对应十进制数–128～+127 之间的二进制数的三种编码结果。

表 1.2 符号数的三种表示法

十进制数	二进制数	原码	反码	补码
–128	–10000000	—	—	10000000
–127	–01111111	11111111	10000000	10000001
–126	–01111110	11111110	10000001	10000010
⋮	⋮	⋮	⋮	⋮
–1	–00000001	10000001	11111110	11111111
–0	–00000000	10000000	11111111	00000000
+0	+00000000	00000000	00000000	00000000
+1	+00000001	00000001	00000001	00000001
⋮	⋮	⋮	⋮	⋮
+125	+01111101	01111101	01111101	01111101
+126	+01111110	01111110	01111110	01111110
+127	+01111111	01111111	01111111	01111111

最后强调说明一下“补码”与“符号数的补码表示”之间的关系。求一个数的补码，与符号数无关，按 1.3.3 节中 2 所给的方法运算即可，在此只涉及求补码；而符号数的表示，则涉及两个概念：求补和符号数的表示。所以，习题 1-9 只是求补练习（所给的数只有正数）；而习题 1-10 则是考求补和符号数的表示两个概念（所给的数既有正数，也有负数）。

1.4 二-十进制编码

数字系统中使用的是二进制数，而在许多场合特别是在输入（如键盘等）/输出（如显示、打印等）时需要处理十进制数。那么十进制数在机器里面是怎样表示的呢？这就是本节所要讨论的问题，即二-十进制码（Binary Coded Decimal，BCD）。

若要表示 0～9 这 10 个十进制数字，需要 10 种组合。三位二进制码只有 8 种组合，不够用；而 4 位二进制码有 16 种组合，可用。当然，位数大于 4 的任意多位二进制数均可用，但那样会造成资源浪费，故一般情况下均用 4 位二进制数对一位十进制数进行编码。用以表示十进制数字的二进制编码称为 BCD 码。从 4 位二进制码的 16 种组合中取 10 种去表示 10 个十进制数字，共有 $C_{16}^{10}=16!/(10!\cdot(16-10)!)$ 种取法；而每种取法又有 10！种分配方法；故共有 16！/6！种可能的 BCD 编码方法可供选择，当然我们不可能全用。表 1.3 所示为几种常用的 BCD 码。

表 1.3 常用的 BCD 码

十进制数	8421BCD	5421BCD	2421BCD	余 3 码	余 3 循环码	备注
0	0000	0000	0000	0011	0010	有效编码
1	0001	0001	0001	0100	0110	
2	0010	0010	0010	0101	0111	
3	0011	0011	0011	0110	0101	
4	0100	0100	0100	0111	0100	
5	0101	1000	1011	1000	1100	
6	0110	1001	1100	1001	1101	
7	0111	1010	1101	1010	1111	
8	1000	1011	1110	1011	1110	
9	1001	1100	1111	1100	1010	

续表

十进制数	8421BCD	5421BCD	2421BCD	余 3 码	余 3 循环码	备注
—	1010	0101	0101	0000	0000	无效编码
—	1011	0110	0110	0001	0001	
—	1100	0111	0111	0010	0011	
—	1101	1101	1000	1101	1011	
—	1110	1110	1001	1110	1001	
—	1111	1111	1010	1111	1000	

表 1.3 中，8421BCD、5421BCD 和 2421BCD 三种编码为**有权码**，即它们从高到低的各位都有固定的权值（8421BCD 码从高到低各位的权值分别为 8、4、2、1，5421BCD 和 2421BCD 的各位的权类似定义），而所有 4 位码加权相加的结果就是它所表示的十进制数字。而余 3 码和余 3 循环码则是**无权码**，因为它的各位没有固定的权值。观察表 1.3 可知，若是各位的权分别为 8、4、2、1，则对应于同一个十进制数字，余 3 码比 8421BCD 码多 3，余 3 码由此得名。

表 1.3 中的前 10 组编码分别对应十进制数字 0～9，称为**有效编码**；而后 6 组没有对应任何十进制数字，没有任何意义，是无效的，故称为**无效编码**。

8421BCD 各位的权与 4 位二进制数相同，故又称为自然码。5421BCD、2421BCD、余 3 码和余 3 循环码各有特点，如 5421BCD 中的 0、1、2、3、4 的最高位变成 1 就分别得到 5、6、7、8、9；而 2421BCD 和余 3 码则具有**自反**特性，即以 4、5 间直线为轴反对称：直线以上（下）的码求反即得直线以下（上）处于它所对称位置的码；余 3 循环码中相邻的编码只有一位不同（0 和 9 的编码也只有一位不同），且最高位自反，其他位以 4、5 间直线为轴对称。这些编码在许多场合有重要应用。

用 BCD 码表示十进制数时，每一位十进制数均需 4 位二进码表示。

【例 1.24】 $(216)_{10} = (0010\ 0001\ 0110)_{8421} = (0010\ 0001\ 1001)_{5421} = (0101\ 0100\ 1001)_{余3}$

两个 BCD 码相加，结果必须还是 BCD 码，并且还必须是合法的 BCD 码。

【例 1.25】 $(0010\ 0001\ 0110)_{8421}+(0011\ 1001\ 0011)_{8421} = (0101\ \underline{1010}\ 1001)_{8421}$，其中第 2 位 BCD 码为 1010，是非法的 8421BCD，要对其进行调整：本位加 6（0110），结果为 0000，并向高位进 1，所以最后结果应为$(\underline{0110\ 0000}\ 1001)_{8421}$。

1.5　格　雷　码

在组合电路中，为避免译码噪声（毛刺）的产生，常使用格雷（Gray）码。格雷码的特点：相邻两个编码中只有一位不同，其他各位均相同。在控制系统中人们常常使用格雷码。表 1.4 所示为对应 4 位二进制码的 4 位格雷码。由表 1.4 可见，第一个格雷码和最后一个格雷码也只有一位不同。余 3 循环码就是取的 4 位格雷码中的 10 组编码。

表 1.4　4 位格雷码

序号	二进制码	格雷码	序号	二进制码	格雷码
0	0000	0000	8	1000	1100
1	0001	0001	9	1001	1101
2	0010	0011	10	1010	1111
3	0011	0010	11	1011	1110
4	0100	0110	12	1100	1010
5	0101	0111	13	1101	1011
6	0110	0101	14	1110	1001
7	0111	0100	15	1111	1000

由表 1.4 还可以看出，格雷码的最高位以 7、8 间的中线为轴自反；而低位则以此线为轴对称。据此可由 n 位格雷码方便地写出 n+1 位格雷码。一位格雷码只有两个：0、1；由此可写出两位、由两位可写出三位、由三位可写出四位、…、由 n–1 位可写出 n 位格雷码，例如：

一位格雷码：	两位格雷码：	三位格雷码：	四位格雷码：
0	00	000	……
1	01	001	
	11	011	
	10	010	
		110	
		111	
		101	
		100	

二进制码与格雷码的相互转换可由表 1.4 得到，也可以用解析式通过运算得到。

定义：逻辑变量 A、B 的取值范围为 0、1，则它们的**异或**运算定义为

$$F = A \oplus B = \begin{cases} 1 & \text{当} A \neq B \text{时} \\ 0 & \text{当} A = B \text{时} \end{cases}$$

有了异或运算，就可以由已知二进制码求出它所对应的格雷码；反之，也可以由已知格雷码求出它所对应的二进制码。设给定二进制码为 $B_{n-1}\cdots B_2B_1B_0$，则它所对应的格雷码 $G_{n-1}\cdots G_2G_1G_0$ 可由式(1.4）求出：

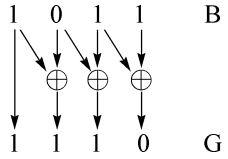
$$G_{n-1} = B_{n-1},\quad G_i = B_{i+1} \oplus B_i \quad i = n-2, n-3, \cdots, 2, 1, 0 \tag{1.4}$$

若给定格雷码 $G_{n-1}\cdots G_2G_1G_0$，则它所对应的二进制码 $B_{n-1}\cdots B_2B_1B_0$ 可由式（1.5）求出：

$$B_{n-1} = G_{n-1},\quad B_i = B_{i+1} \oplus G_i \quad i = n-2, n-3, \cdots, 2, 1, 0 \tag{1.5}$$

【例 1.26】 试写出对应二进制码 1011 的格雷码。

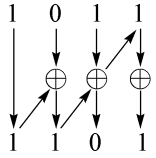
解：由式（1.4）得



所以所求格雷码为 1110。

【例 1.27】 试写出对应格雷码 1011 的二进制码。

解：由式（1.5）得



注意：先求高位再求低位。

所以所求格雷码为 1101。

读者可以对照表 1.4 验证。

1.6　ASCII 字符集

在数字系统或计算机系统中除了需要表示十进制数外，还经常要表示一些人机交流用的其他信息，如大小写字母、+、–、×、÷、=、&、%等字符，DEL、ESC、CR 等控制符。目前广泛使用的是 ASCII（American Standard Codes for Information Interchange）字符集，又称为 ASCII 码，如表 1.5 所示。

表 1.5　ASCII 字符集

			列号（$b_6b_5b_4$）							
		B	000	001	010	011	100	101	110	111
	B	H	0	1	2	3	4	5	6	7
行号($b_3b_2b_1b_0$)	0000	0	NUL	DLE	SP	0	@	P	‘	p
	0001	1	SOH	DC1	!	1	A	Q	a	q
	0010	2	STX	DC2	“	2	B	R	b	r
	0011	3	ETX	DC3	#	3	C	S	c	s
	0100	4	EOT	DC4	$	4	D	T	d	t
	0101	5	ENQ	NAK	%	5	E	U	e	u
	0110	6	ACK	SYN	&	6	F	V	f	v
	0111	7	BEL	ETB	’	7	G	W	g	w
	1000	8	BS	CAN	(	8	H	X	h	x
	1001	9	HT	EM	)	9	I	Y	i	y
	1010	A	LF	SUB	*	:	J	Z	j	z
	1011	B	VT	ESC	+	;	K	[	k	{
	1100	C	FF	FS	,	<	L	\	l	|
	1101	D	CR	GS	-	=	M	]	m	}
	1110	E	SO	RS	.	>	N	^	n	～
	1111	F	SI	US	/	?	O	-	o	DEL

由表 1.5 可知，ASCII 字符集共有 128 个编码，其中控制符 33 个，字符 95 个。00H～1FH、7FH 为控制符编码，其余（20H～7EH）为字符编码。每个 ASCII 码均由 7 位二进制数（b_6～b_0）组成，又可将它们看成为两位十六进制码，如 20H 为空格 SP 的 ASCII 编码；30H～39H 分别是数字 0～9 的 ASCII 编码；41H～5AH 分别是大写字母 A～Z 的 ASCII 编码等。

1.7　检错码和纠错码

由于各种原因，数字信号（数据）在传输过程中常常会发生错误，即所谓的误码。在不同的应用场合对错误的处理方法也不一样，有的场合只要检测出有无错码即可，这时就需要检错（Error Detecting）码；而在另外的场合则不仅需要检测出错码，而且还要将错码纠正过来，这时就需要纠错（Error Correcting）码。本节简单介绍检错码和纠错码。

1.7.1　检错码

最简单的检错码是奇偶校验码（Parity）。

为检验传输过程是否出错，除要发送的信息码之外，再多发送一位校验位，信息码与校验位共同组成的码就是**奇偶校验码**。校验位的确定方法：如果采用奇校验（Odd Parity），则信息码与校验位所

构成的奇偶校验码中“1”的个数为奇数；如果采用偶校验（Even Parity），则信息码与校验位所构成的奇偶校验码中“1”的个数为偶数。根据收发协议，接收端接收到发送端发送来的信息后，先判断“1”的个数的奇偶性，若“1”的个数为奇（偶）数，则认为接收正确，否则认为接收错误。校验位一般放在最高位。判断“1”的个数的奇偶性，可由逻辑运算“异或”完成。

【例 1.28】 设要发送的 8 位信息码为 01000001，则采用奇、偶校验时所发送的校验位分别为 1、0。所发送的奇偶校验码分别为：101000001、001000001。

可见，使系统具有检错功能所花费的代价是降低信息码的传送速率。设传送一位码元需要一个单位的时间 T，则传送 n 位信息码需要时间 nT；传送奇偶校验码需要时间$(n+1)T$，比前者多用了 T。在传输奇偶检验码时，信息码的传输效率是$\eta = nT/(n+1)T = n/n+1$。当 $n=8$ 时，$\eta = 8/9\times100\% \approx 88.89\%$。

数据在传输过程中，若所传输的码发生了偶数位错误，则其奇偶性不会改变，接收端不能将其检测出来。因此，奇偶校验码只能用来检测奇数个错误。这种检错方法一般用于误码率较低的场合。

1.7.2 纠错码

一种比较简单的纠错码是所谓的二维奇偶纠错码，它利用行、列的奇偶性进行纠错。图 1.1 所示二维奇偶纠错码的示意图，它的行列均使用奇（偶）校验。数据传输以数据块为单位进行。接收完一个数据块后，对该数据块行、列的奇偶性进行检查，有错时将错误纠正过来。例如，若第 X 行、第 Y 列相交的码元发生错误，则接收端可检测到第 X 行、第 Y 列出错，将第 X 行、第 Y 列相交的码元求反即可纠正错误。

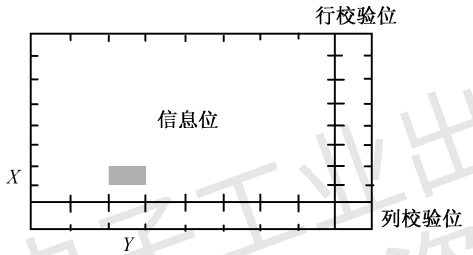


图 1.1 二维奇偶纠错码

本节介绍了简单的检错和纠错编码。实际系统中的出错情况、编码方式往往复杂得多。编码是通信与电子系统的一个重要组成部分，许多人员都在从事这方面的研究工作。这已不属于本课程的范畴，有兴趣的读者可阅读这方面的书籍。

小 结

为解决人机交流问题，引入了二、八、十、十六进制数及它们之间的相互转换；为解决只用加法器就能实现加、减运算，从而达到节省硬件的目的，引入了补码、反码的定义及求法；为解决在机器里表示正、负数的问题，引入了符号数的计算机表示方法；介绍并举例说明了利用补码进行二进制数的加减运算的正确性；为用机器表示十进制数，介绍了十进制数的二进制编码，即 BCD 码；为解决控制系统中译码噪声的问题，引入了“相邻码组逻辑相邻”的格雷码；还介绍了美国标准编码系统 ASCII 字符集；最后介绍了在数据传输中极为重要的检错、纠错的概念，给出了奇偶校验码的定义及其适合的应用场合。

习 题

1-1 例 1.12 中转换前后两个数的绝对值哪个大？为什么？

1-2 将下列二进制数转换为八进制数、十六进制数和十进制数。

11001101.101，10010011.1111

1-3 将下列十进制数转换为二进制数、八进制数和十六进制数。

121.56，73.85

1-4　将下列十六进制数转换为二进制数、八进制数和十进制数。

89.0F，E5.CD

1-5　试求例 1.17 的转换误差，比较例 1.12 的转换误差，哪个大？为什么？

1-6　用 16 位二进数表示符号数。试分别写出原码、反码和补码可表示的数值范围。

1-7　设 $n=8$，试求下列数对应的二进制数的反码：

24，43，65，79

1-8　设 $n=8$，试求下列二进制数的反码：

101101，–101101，10100，–10100

1-9　设 $n=8$，试求下列数对应的二进制数的补码：

23，123，79，97

1-10　设 $n=8$，试求下列二进制数的补码：

101101，–101101，10100，–10100，101.001，–101.001

1-11　为什么将 N 求反加 1 即为 N 的补码？

1-12　试证明利用补码进行加减运算的正确性。

1-13　设 $A=65$，$B=56$，$n=8$。试用补码求下列运算，并验证其结果是否正确：

$A+B$，$A-B$，$-A+B$，$-A-B$

1-14　设 $A=65$，$B=75$，$n=8$。试用补码求下列运算，并验证其结果是否正确：

$A+B$，$A-B$，$-A+B$，$-A-B$

如果结果有错，为什么？

1-15　如何判断补码运算有无溢出？

1-16　试分别写出下列十进制数的 8421BCD、5421BCD、2421BCD 和余 3 码。

325，108，61.325

1-17　试完成下列 BCD 码运算：

$$(0011\ 1001\ 0001)_{8421}+(0101\ 1000\ 0010)_{8421}=\ ?$$

1-18　试写出对应下列二进制数的格雷码：

1010，1101

1-19　试写出对应下列格雷码的二进制数：

1010，1101

1-20　试写出“Hello everyone”的 ASCII 编码，分别使用二进制和十六进制。

1-21　设要用奇偶校验码传送 ASCII 字符串“BIT”，试分别写出其奇校验码和偶校验码。在这种情况下传输效率降低了多少？

1-22　设发送端发送的奇偶校验码为 101100110，而在接收端收到的码元序列为：①111100110；②101010110。问本题中采用的是奇校验还是偶校验？接收结果①、②中哪个是对的？哪个是错的？为什么？

1-23　用二维奇偶纠错码去纠错，有无可能纠正所有的错误？若不能，什么情况下不能？试列出不能纠错的情况并说明原因。