数字逻辑

毛法尧　编著

华中工学院出版社

数 字 逻 辑

毛法尧 编著

华中工学院出版社

内 容 简 介

本书系统地介绍了逻辑设计的基本理论和设计方法，并主要讨论了如何利用中、小规模集成电路进行逻辑设计的问题。对于数字系统设计及大规模集成电路的应用也作了必要的介绍。

全书共七章，内容分成三部分：第一部分（第一、二、三章）介绍逻辑设计的数学知识——布尔代数、数制及函数简化，并讨论组合逻辑电路的分析和设计方法；第二部分（第四、五章）介绍同步和异步时序逻辑电路的基本规律和设计方法；第三部分（第六、七章）介绍如何用中、大规模集成电路进行逻辑设计及用寄存器传送语言进行系统设计的方法。

本书可作为高等学校电子计算机、计算机软件、无线电及自动化等专业的教材，也可供从事数字技术的工程技术人员及高等学校有关专业师生自学参考。

数 字 逻 辑

毛法尧 编著

责任编辑 唐元瑜

*

华中工学院出版社出版

（武昌喻家山）

新华书店湖北发行所发行

华中工学院出版社沔阳印刷厂印刷

*

开本：787×1092 1/16 印张：13.75 字数：334 000字

1986年3月第一版 1986年3月第一次印刷

印数：1—5,000

统一书号：15255—061 定价：2.75元

前　言

本书是根据高等学校工科电子类电子计算机专业和高等学校理工科计算机软件专业“数字逻辑”课程教学大纲的要求，并考虑到无线电、自动化等专业学习“数字逻辑”课程的需要以及编者结合自己的教学实践，为满足教学需要而编写的。

数字逻辑是电子计算机的基础理论之一。熟练地掌握逻辑设计的理论和方法，对于从事计算机研制和应用的广大科技工作者来说是十分必要的。

本书较系统地介绍了逻辑设计的基本理论和设计方法，并主要讨论了如何利用中、小规模集成电路进行逻辑设计。对于数字系统设计及大规模集成电路的应用也作了必要的介绍。编写本书的主要目的是，使读者了解和掌握逻辑设计的方法和技能，并对电子计算机硬件有一定的认识。读者学完本书后，不仅能熟悉从对系统提出要求开始，直到用集成电路实现所需逻辑功能为止的整个设计过程，而且还可利用书中介绍的有关知识设计所需要的系统和部件。

全书共七章，内容分成三部分：第一部分（第一、二、三章）首先介绍了逻辑设计的数学知识——布尔代数、数制及函数简化，在此基础上较详细地讨论了组合逻辑电路的分析和设计方法；第二部分（第四、五章）分别介绍了时序逻辑电路的两种基本类型——同步及异步时序逻辑电路的基本规律及设计方法；第三部分（第六、七章）对数字系统的设计作了讨论，介绍了如何用中、大规模集成电路进行逻辑设计及用寄存器传送语言进行系统设计的方法。

本书在内容和叙述上力求深入浅出、重点明确、便于自学。

在本书的编写过程中，曾得到华中工学院计算机系及计算机教研室的领导和同志们的热情支持和帮助。华中工学院计算机系副主任兼计算机教研室主任徐则琨副教授审阅了本书的原稿，提出了许多宝贵的意见。华中工学院出版社为本书的出版给予了大力的支持。谨借此一并表示衷心的感谢。

鉴于编者水平有限，书中如有错误和不妥之处，请读者批评指正。

编　者

1985年6月于华中工学院

目　录

第一章　数制与编码

计算技术的发展促进了科学技术和生产的飞跃发展。现在，数字计算机已广泛地应用于科学计算、数据处理和生产过程控制等领域中。数字计算机是数字系统中最常见的、最有代表性的一种设备。此外，如电话交换机、数字化仪器和电传打字机等设备也都属于数字系统。

数字系统的特点是它所处理的信息都是离散元素。这些离散元素可以是十进制数字、某种字母、各种算符及标点符号等。各种离散元素按不同方式排列可以表示大量的信息。例如，字母C、O、M、P、U、T、E和R就可以形成单词COMPUTER，数字1984表示一个确定的数等等。由于早期的计算机都用于数值计算，在这种情况下所用的离散元素是数字，数字计算机这个名词就是由此而来的。

在数字系统中，信息的离散元素是以称为讯号的物理量来表示的，电压和电流就是最常用的电讯号。通常，数字系统的讯号都只有两个离散量，因此，称为二进制的。由于只有导通和截止两种状态的电子器件十分可靠，再加上人类的逻辑思维方式也倾向于二值，因此，数字系统常采用二进制讯号。

讯号的离散量有的是自然形成的，有的则是将连续过程有意加以量化后而得到的。例如，一份学生成绩单就是天然的离散过程，上面有学生的姓名、课程名称、成绩等等。又如科学工作者在科学研究工作中常常要观察连续的过程，但仅将一些特殊的数值记录下来列成表格，这样就将连续的信息量化了，所以表格中的每一个数字都已成为离散量。

数字系统处理的是离散元素，而这些离散元素通常是以二进制数的形式出现的。其它离散元素，如十进制数字或字母符号也可用二进制代码来表示。为此，必须讨论数的代码特征及运算。

本章介绍数制和编码的概念。通过本章学习，将熟悉数字系统中数制和编码的表示方法、性质及相互间的转换，为进一步学习以后各章打下基础。

1.1　进位计数制

在日常生活中，人们通常采用十进制数来计数，它的基数为10，每位数可用下列十个数字之一来表示，即0，1，2，3，4，5，6，7，8，9。

当我们看到一个十进制数，如632.45时，我们就会立刻想到：这个数的最左位为百位（6代表600），第二位为十位(3代表30)，第三位为个位(2代表2)，小数点右面第一位为十分位(4代表4/10)，第二位为百分位(5代表5/100)。这里百、十、个、十分之一和百分之一都是10的次幂，它取决于系数所在的位置，我们称之为“权”。确切地说，632.45应该写成

$$(N)_{10}=6(10)^2+3(10)^1+2(10)^0+4(10)^{-1}+5(10)^{-2}=(632.45)_{10}$$

这样，一个十进制数可以用基数为10的次幂之和来表示。

十进制数的计数规律是“逢十进一”，也就是说，每位数累计不能超过10，计满10就应向高位进1。

一般说来，对于任意的数N，我们都能表达成以r为基数的r进制数。即

$$(N)_r = a_{n-1}(r)^{n-1} + \cdots + a_1(r)^1 + a_0(r)^0 + a_{-1}(r)^{-1} + \cdots + a_{-m}(r)^{-m}$$

这里n代表整数位数，m代表小数位数，系数a_i可以取0～r-1之间的任意一个整数。r进制数的计数规律是“逢r进一”。

习惯上总是依次写下各个系数而省略r的各个次幂。这样，一个带小数点的r进制数可用一串系数表示：

$$(N)_r = (a_{n-1}a_{n-2}\cdots a_1a_0.a_{-1}\cdots a_{-m})_r$$

由于a_0和a_{-1}之间出现小数点，故数$(N)_r$分成两部分，其中小数点的右边为数的小数部分，而左边为数的整数部分。并且，系数a_i可以是r个数字中的一个，下标i的数值给定了系数所处的位置。若基数r小于10时，通常都从十进制数中借用r个数字作为r进制的数字，当基数r大于10时，有时则以字母作补充。例如，在十六进制中，开头十个数字是从十进制中借来的，并分别将字母A、B、C、D、E及F代替数字10、11、12、13、14、15。

表1.1列出了当r为10、2、8及16时，各种进位计数制中开头的十六个自然数。

表 1.1 不同进位计数制的各种数字

十进制数 (r=10)	二进制数 (r=2)	八进制数 (r=8)	十六进制数 (r=16)
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

选择什么样的进位计数制来表示数，对数字系统的成本和性能影响很大。在数字系统中，常用二进制来表示数字和进行运算。这是因为二进制数只有0和1两个数字符号，容易表示；二进制运算规则简单，便于进行算术运算；实现二进制算术运算的逻辑电路的设计也较容易。

二进制运算中，相应的规则有：

加法规则

$0 + 0 = 0$　　$0 + 1 = 1$

$1 + 0 = 1$　　$1 + 1 = 0$（同时向较高邻位进1）

乘法规则

$0 \times 0 = 0$　　$0 \times 1 = 0$

$1 \times 0 = 0$　　$1 \times 1 = 1$

下面举几个二进制数运算的例子。

例1.1

```
   1 1 1   ←进位
   1 1 0 1
+) 1 0 1 1
-------------
 1 1 0 0 0
```

两个二进制数的加法运算和十进制数的加法运算相似，但采用“逢二进一”的法则，每位数累计到2时，本位就记为0，且向较高邻位进1。

例1.2

(*a*)

```
     1     ←借位
   1 1 1 0 1
-) 1 0 0 1 1
-------------
   1 0 1 0
```

(*b*)

```
   1 1 1 1   ←借位
   1 0 0 0 0
-)       1 1
-------------
   1 1 0 1
```

二进制减法采用“借一当二”的法则，减法运算从低位起按位进行，在遇到0减1时，就要向较高邻位借1，也就是从那一位减去1。

例1.3

```
       1 1 0 1
     ×) 1 0 1 1
-----------------
       1 1 0 1
     1 1 0 1
   0 0 0 0
 1 1 0 1
-----------------
1 0 0 0 1 1 1 1
```

二进制数的乘法运算和十进制数的乘法运算相似，所不同的是对部分积进行累加时要按“逢二进一”的原则。

例1.4

```
           1 1 0 1 …商
       ┌──────────
1 0 1 1 | 1 0 0 1 0 0 0 1
          1 0 1 1
          ---------
            1 1 1 0
            1 0 1 1
            -------
              1 1 0 1
              1 0 1 1
              -------
                  1 0 …余数
```

二进制数的除法运算同十进制数的除法运算类似，但采用二进制数的运算规则。

1.2 数制转换

在数字计算机和其它数字系统中，普遍使用二进制数，并且二进制的数字系统也只能加工二进制数或用二进制代码表示的其它进位制数。由于人们习惯于使用十进制数，所以，在信息处理中首先必须把十进制数转换成数字计算机能加工和处理的二进制数，然后再将二进

制的计算结果转换成人们习惯的十进制数。这里就存在一个不同数制的相互转换问题。

1.2.1 任意进制数转换成十进制数

二进制数转换成十进制数是很方便的，即只要把二进制数写成2的次幂的展开式，并将式中各乘积项的积算出来，然后各项相加，即可得到与该二进制数相对应的十进制数。例如

$$(11010.101)_2 = 1(2)^4 + 1(2)^3 + 0(2)^2 + 1(2)^1 + 0(2)^0 + 1(2)^{-1} + 0(2)^{-2} + 1(2)^{-3}$$
$$= 16 + 8 + 2 + 0.5 + 0.125 = (26.625)_{10}$$

在这个二进制数中，有五个有效位的数字为1，所以其等效十进制数为五个相应的乘积项之和。对于一个以r为基数的数也可以通过将各个系数分别乘以相应的次幂并相加而转换成等效的十进制数。

下面用同样的方法将一个五进制数转换成十进制数。例如

$$(314.2)_6 = 3(5)^2 + 1(5)^1 + 4(5)^0 + 2(5)^{-1}$$
$$= 75 + 5 + 4 + 0.4 = (84.4)_{10}$$

同样，也可以将十六进制数转换成十进制数。应该指出的是：在转换过程中，数字A～F要转换成对应的十进制数10～15。例如

$$(5A.E)_{16} = 5(16)^1 + A(16)^0 + E(16)^{-1} = 5(16)^1 + 10(16)^0 + 14(16)^{-1}$$
$$= 80 + 10 + 0.875 = (90.875)_{10}$$

1.2.2 十进制数转换成任意进制数

十进制数转换成二进制数时，需将待转换的数分成整数部分和小数部分，并分别加以转换。当一个十进制数写成

$(N)_{10}$ = 〈整数部分〉$_{10}$.〈小数部分〉$_{10}$转换时，首先将〈整数部分〉$_{10}$转换成〈整数部分〉$_2$，然后再将〈小数部分〉$_{10}$转换成〈小数部分〉$_2$。待整数部分和小数部分确定后，就可写成

$$(N)_2 = \langle\text{整数部分}\rangle_2.\langle\text{小数部分}\rangle_2$$

下面我们通过具体的例子来说明十进制数转换成二进制数的方法。

假如给定一个十进制数

$$(N)_{10} = d_{n-1}d_{n-2}\cdots d_0.d_{-1}d_{-2}\cdots d_{-m} = (I)_{10}.(F)_{10}$$

其中，$(I)_{10}$和$(F)_{10}$分别表示十进制数N的整数部分和小数部分。

我们希望把十进制数$(N)_{10}$转换成二进制数$(N)_2$，二进制数$(N)_2$可以写成

$$(N)_2 = b_{u-1}b_{u-2}\cdots b_0.b_{-1}b_{-2}\cdots b_{-v} = (I)_2.(F)_2$$

首先，将十进制数的整数部分$(I)_{10} = d_{n-1}d_{n-2}\cdots d_0$转换成对应的二进制数的整数部分$(I)_2 = b_{u-1}b_{u-2}\cdots b_0$。于是

$$(I)_{10} = b_{u-1}(2)^{u-1} + b_{u-2}(2)^{u-2} + \cdots + b_1(2)^1 + b_0(2)^0$$

用2除$(I)_{10}$，得到整数商Q_1和余数b_0。即

$$(I)_{10} = 2Q_1 + b_0$$

其中，$Q_1 = b_{u-1}(2)^{u-2} + b_{u-2}(2)^{u-3} + \cdots + b_1(2)^0$

再用2除Q_1，又得到整数商Q_2和余数b_1。即

$$Q_1 = 2Q_2 + b_1$$

其中，$Q_2 = b_{u-1}(2)^{u-3} + b_{u-2}(2)^{u-4} + \cdots + b_2(2)^0$

这个过程一直要继续到整数商Q_n为0或数的精确度已合乎要求为止。转换过程所产生的余数项就是二进制整数部分的系数，它等效于给定的十进制数。

为了说明上述过程，现令$(I)_{10}=25$。由于除数是2，任何数除2时，它的余数不是0就是1，因此，很快就能确定b_0值。将商数作为新的被除数，再用2除，又可确定b_1的值。依此类推，直到商数为0，便可得到各个系数b_i的值，其过程如下：

$$
\begin{array}{lll}
(I)_{10}=2(12)+1 & Q_1=12 & b_0=1 \\
Q_1=2(6)+0 & Q_2=6 & b_1=0 \\
Q_2=2(3)+0 & Q_3=3 & b_2=0 \\
Q_3=2(1)+1 & Q_4=1 & b_3=1 \\
Q_4=2(0)+1 & Q_5=0 & b_4=1
\end{array}
$$

因此，得到$(I)_2=11001$，它同$(I)_{10}=25$等效。

根据上面讨论的方法，我们可用下列形式很方便地将十进制整数转换成二进制数。

$$
\begin{array}{l|ll}
2 & 25 & \\
2 & 12 & b_0=1 \\
2 & 6 & b_1=0 \\
2 & 3 & b_2=0 \\
2 & 1 & b_3=1 \\
 & 0 & b_4=1
\end{array}
$$

因而，　$(I)_2=b_4b_3b_2b_1b_0=11001$。

完成整数部分的转换后，下一步是把十进制数的小数部分$(F)_{10}=d_{-1}d_{-2}\cdots d_{-m}$转换成二进制数的小数部分$(F)_2=b_{-1}b_{-2}\cdots b_{-v}$。

将十进制小数转换成二进制小数的方法与整数转换所用的方法相似，只是以相乘代替相除，并且以相乘所得的整数部分作为各位的系数。必须指出：由于十进制数和二进制数的分辨间隔不同，有限位的十进制小数不一定能用有限位的二进制小数表示，因而，应选定一个数作为二进制小数的最大位数。假若用v来表示二进制小数的最大位数，于是

$$(F)_{10}=b_{-1}(2)^{-1}+b_{-2}(2)^{-2}+\cdots+b_{-v}(2)^{-v}$$

用2乘$(F)_{10}$，得到

$$2(F)_{10}=b_{-1}+b_{-2}(2)^{-1}+\cdots+b_{-v}(2)^{-v+1}=b_{-1}+C_1$$

$(F)_{10}$为纯小数，同2相乘后，$2(F)_{10}$可能大于1，而出现整数部分，也可能仍是纯小数，其整数部分为0。这样，b_{-1}就可能是1或0。因此，用2来乘被转换的十进制小数，所得结果的整数部分为b_{-1}，小数部分为C_1。再以2乘小数部分C_1，又可得到

$$2C_1=b_{-2}+b_{-3}(2)^{-1}+\cdots+b_{-v}(2)^{-v+2}=b_{-2}+C_2$$

根据同样的道理，b_{-2}为0或1，而C_2小于1。这个过程一直要继续到小数部分为0或数的精确度已合乎要求时为止。即

$$2C_{v-1}=b_{-v}+C_v$$

如果$C_v=0$，这时，十进制小数对二进制小数的转换是精确的。否则，转换以后得到的二进制小数是给定十进制小数的一个近似值。假设转换过程所允许的最大误差为ε，近似转换须满足$2^{-v}C_v\leqslant\varepsilon$。由此可见，十进制小数转换成二进制小数时，可能出现两种情况：精确转换和近似转换。

第一种情况：精确转换

假若给定$(F)_{10}=0.125$，十进制小数$(F)_{10}$转换成二进制小数$(F)_2$的转换过程如下：

$2(0.125)=0+0.25$　　$b_{-1}=0$　　$C_1=0.25$

$2(0.25)=0+0.5$　　$b_{-2}=0$　　$C_2=0.5$

$2(0.5)=1+0.0$　　$b_{-3}=1$　　$C_3=0$

转换结果为精确值，$(F)_2=(0.001)_2$

第二种情况：近似转换

假若给定$(F)_{10}=0.3$，令$(\varepsilon)_{10}=0.001$，十进制小数$(F)_{10}$转换成二进制小数$(F)_2$的转换过程如下：

$2(0.3)=0+0.6$　　$b_{-1}=0$　　$C_1=0.6$

$2(0.6)=1+0.2$　　$b_{-2}=1$　　$C_2=0.2$

$2(0.2)=0+0.4$　　$b_{-3}=0$　　$C_3=0.4$

$2(0.4)=0+0.8$　　$b_{-4}=0$　　$C_4=0.8$

$2(0.8)=1+0.6$　　$b_{-5}=1$　　$C_5=0.6$

$2(0.6)=1+0.2$　　$b_{-6}=1$　　$C_6=0.2$

$2(0.2)=0+0.4$　　$b_{-7}=0$　　$C_7=0.4$

$2(0.4)=0+0.8$　　$b_{-8}=0$　　$C_8=0.8$

$2(0.8)=1+0.6$　　$b_{-9}=1$　　$C_9=0.6$

$2(0.6)=1+0.2$　　$b_{-10}=1$　　$C_{10}=0.2$

由于　$2^{-9}(0.6)>0.001$

但是　$2^{-10}(0.2)<0.001$

所以　$(F)_2\cong(0.0100110011)_2$

转换结果为近似值，这种转换是近似转换。

通过上面的讨论，我们已经熟悉了十进制数对二进制数的转换。用同样的方法，可以将十进制数转换成任意进制数。现举例说明。

例1.5　给定十进制数$(N)_{10}=(75.12)_{10}$，将它转换成五进制数。

转换过程如下：

整数部分		小数部分
5 | 75		$5(0.12)=0+0.6$
5 | 15	0	$5(0.6)=3+0.0$
5 | 3	0	
0	3	

转换结果：　$(N)_5=(300.03)_5$

例1.6　已知十进制数$(N)_{10}=(25.3)_{10}$，将它转换成十六进制数。

转换过程如下：

整数部分		小数部分
16 | 25		$16(0.3)=4+0.8$
16 | 1	9	$16(0.8)=12+0.8$
0	1	$16(0.8)=12+0.8$
		⋮

可以看出，转换后得到的十六进制数只能是一个近似值。若取三位小数就能满足精确度

要求，则

$$(N)_{16}\cong(19.4CC)_{16}$$

1.2.3 任意两种进位制之间数的转换

前面我们介绍了任意进位制数转换成十进制数和十进制数转换成任意进制数，对于两个任意进制数的转换也可以采用类似方法来实现。但是，一种方便的方法是利用十进制作桥梁，首先把r_1进制数$(N)_{r_1}$转换成十进制数$(N)_{10}$，然后再将十进制数$(N)_{10}$转换成r_2进制数$(N)_{r_2}$。现举例说明。

例1.7 将四进制数$(1023.231)_4$转换成等效的五进制数。

首先，将数$(1023.231)_4$转换成十进制数，即

$$(1023.231)_4=1(4)^3+0(4)^2+2(4)^1+3(4)^0+2(4)^{-1}+3(4)^{-2}+1(4)^{-3}$$
$$=64+8+3+0.5+0.1875+0.015625=(75.703125)_{10}$$

然后再将数$(75.703125)_{10}$转换成五进制数，即

整数部分		小数部分
5 | 75		5(0.703125) = 3 + 0.515625
5 | 15	0	5(0.515625) = 2 + 0.578125
5 | 3	0	5(0.578125) = 2 + 0.890625
0	3	5(0.890625) = 4 + 0.453125
		⋮

因而 $(1023.231)_4=(300.3224\cdots)_5$

虽然，数字系统广泛采用二进制，但它书写起来很长，非常麻烦，也容易出错。为此，人们常以八进制和十六进制作为二进制的速写形式。

八进制的基数为8，由于$2^3=8$，所以一位八进制数相当于三位二进制数。这样，从八进制转换成二进制时，只要把每位八进制数用三位二进制数表示；而从二进制转换成八进制时，只需把每三位二进制数用一位八进制数表示。同样，由于$2^4=16$，因此，一位十六进制数相当于四位二进制数，这样，十六进制与二进制之间的转换也很方便。二进制、八进制和十六进制之间具有整指数倍数的关系，因而可直接进行转换。例如

八进制	2		5		7	.	0		5		5	4
二进制	0 1 0		1 0 1		1 1 1	.	0 0 0		1 0 1		1 0 1	1 0 0
十六进制		A			F	.		1		6		C

则 $(257.0554)_8=(10101111.0001011011)_2=(AF.16C)_{16}$

由此可知，二进制转换成八进制（或十六进制）时，二进制数的整数部分从低位开始，每三位（或四位）分成一组，若最左边一组不够三位（或四位），可在有效位左边添0，然后，将每组二进制数转换为八进制（或十六进制）数。八进制（或十六进制）转换成二进制时，情况与上述相反。用二进制作桥梁，就可方便地实现八进制与十六进制之间的转换。

1.3 带符号数的代码表示

前面讨论的数都没有考虑符号，一般认为是正数，但在算术运算中总会出现负数。不带

符号的数是数的绝对值，在绝对值前加上表示正负的符号就成了带符号数。一个带符号的数由两部分组成：一部分表示数的符号，另一部分表示数的数值。数的符号是一个具有正、负两种值的离散量信息，它可以用一位二进制数来表示。习惯上以0表示正数，而以1表示负数。对于一个n位二进制数，如果数的第一位为符号位，则剩下的$n-1$位就表示数的数值部分。一般，直接用正号“+”和负号“-”来表示符号的二进制数，叫做符号数的真值。数的真值形式是一种原始形式，不能直接用于数字计算机中。但是，当使符号数值化以后，就可在计算机中使用它。计算机中使用的符号数叫做机器数。

在数字系统中，表示带符号二进制数的方法很多，目前常用的有原码、反码和补码三种表示形式。由于它们的形成规则不同，所以其运算方法也不同。

1.3.1 原 码

原码表示又称为符号—数值表示法。在以原码形式表示的正数和负数中，第一位表示符号位，对于正数，符号位记作0，对于负数，符号位记作1，其余各位表示数值部分。

假如两个带符号的二进制数分别为N_1和N_2，其真值形式为：

$$N_1 = +10011 \qquad N_2 = -01010$$

则N_1和N_2的原码表示形式为：

$$[N_1]_{原} = 010011 \qquad [N_2]_{原} = 101010$$

根据上述原码形成规则，一个n位的整数N（包括一位符号位）的原码一般表示式为

$$[N]_{原} = \begin{cases} N & 0 \leqslant N < 2^{n-1} \\ 2^{n-1} - N & -2^{n-1} < N \leqslant 0 \end{cases}$$

对于定点小数，通常小数点定在最高位的左边，这时，数值小于1。定点小数原码一般表示式为

$$[N]_{原} = \begin{cases} N & 0 \leqslant N < 1 \\ 1 - N & -1 < N \leqslant 0 \end{cases}$$

从原码的一般表示式中可以看出：当N为正数时，$[N]_{原}$和N的区别只是增加一位用0表示的符号位，由于在数的左边增加一位0对该数的数值并无影响，所以$[N]_{原}$就是N本身；当N为负数时，$[N]_{原}$和N的区别是增加一位用1表示的符号位。必须指出，在原码表示法中，有两种不同形式的零，即正零和负零。正零和负零的值是相等的。

原码表示法简单，但是加、减运算较复杂。例如，若两个数相加，应首先比较两个数的符号：若两数的符号相同，就将两个数的数值相加，最后给结果附上相应的符号；若两数的符号不同，就得进一步比较两数的数值相对大小，然后将数值较大的数减去数值较小的数，最后确定结果的符号。用逻辑电路来实现这样一个过程，所需电路较复杂，运算速度也较慢。为了简化减法运算和方便逻辑处理，通常把带符号数表示成补码和反码形式。

1.3.2 反 码

在有的文献里，反码又称为“对1的补数”。用反码表示时，左边第一位也为符号位，符号位为0代表正数，符号位为1代表负数。对于负数，反码的数值是将原码数值按位求反，即原码的某位为1，反码的相应位就为0，或者原码的某位为0，反码的相应位就为1。而对于正数，反码和原码相同。所以，反码数值的形成与它的符号位有关。

假如两个带符号的二进制数分别为N_1和N_2，其真值形式为：

$$N_1 = +10011 \qquad N_2 = -01010$$

则N_1和N_2的反码表示形式如下：

$$[N_1]_{反} = 010011 \qquad [N_2]_{反} = 110101$$

根据上述的反码形成规则，一个n位的整数N（包括一位符号位）的反码一般表示式为：

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

同样，对于定点小数，若小数部分的位数为m，它的反码一般表示式为：

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

从反码的一般表示式可以看出：正数N的反码$[N]_{反}$与原码$[N]_{原}$相同；对于负数N，其反码$[N]_{反}$的符号位为1，数值部分是将原码数值按位变反。必须指出，在反码表示法中，零的表示不是唯一的，有正零和负零两种形式。

1.3.3 补 码

补码表示又称为“对2的补数”。在补码表示法中，正数的表示同原码和反码的表示是一样的，而负数的表示却不同。对于负数，从原码转换到补码的规则是：符号位不变，仍为1，数值部分变反加1，即按位变反，最低位加1。

假如两个带符号二进制数分别为N_1和N_2，其真值形式为：

$$N_1 = +10011 \qquad N_2 = -01010$$

则N_1和N_2的补码表示形式为

$$[N_1]_{补} = 010011 \qquad [N_2]_{补} = 110110$$

根据上述补码形成规则，一个n位的整数N（包括一位符号位）的补码一般表示式为：

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

同样，对于定点小数，补码一般表示式可写成：

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

由补码的一般表示式可以看出：正数N的的补码$[N]_{补}$、反码$[N]_{反}$和原码$[N]_{原}$是相同的。对于负数，补码$[N]_{补}$的符号位为1，其数值部分为反码数值加1。必须指出，在补码表示法中，零的表示是唯一的，即只有一种形式。

综上所述，带符号二进制数可以表示成原码、反码和补码，表1.2和1.3列举了带符号二进制整数和二进制小数的真值与它的原码、反码和补码之间的对照，这里假设二进制数共有四位，并包括一位符号位。

1.3.4 原码、反码和补码的加、减运算

上面介绍了带符号数的三种表示法，它们的形成规则不同，加、减运算的规律也不相同。

表 1.2 带符号二进制整数的真值、原码、补码和反码

$+N$	正整数 原码、补码、反码	$-N$	负整数		
			原码	补码	反码
+000	0000	−000	1000	—	1111
+001	0001	−001	1001	1111	1110
+010	0010	−010	1010	1110	1101
+011	0011	−011	1011	1101	1100
+100	0100	−100	1100	1100	1011
+101	0101	−101	1101	1011	1010
+110	0110	−110	1110	1010	1001
+111	0111	−111	1111	1001	1000
		−1000	—	1000	—

表 1.3 带符号二进制小数的真值、原码、补码和反码

$+N$	正小数 原码、补码、反码	$-N$	负小数		
			原码	补码	反码
+0.000	0.000	−0.000	1.000	—	1.111
+0.001	0.001	−0.001	1.001	1.111	1.110
+0.010	0.010	−0.010	1.010	1.110	1.101
+0.011	0.011	−0.011	1.011	1.101	1.100
+0.100	0.100	−0.100	1.100	1.100	1.011
+0.101	0.101	−0.101	1.101	1.011	1.010
+0.110	0.110	−0.110	1.110	1.010	1.001
+0.111	0.111	−0.111	1.111	1.001	1.000
		−1.000	—	1.000	—

用原码进行加、减运算时，由于加、减法有不同的规则，所以使用起来不方便。而用反码和补码进行加、减运算时，可以把减去一个正数看成是加上一个负数。即先把负数用反码和补码表示，然后按加法规则进行运算。在运算中，符号位也被看成一位数，它与数值部分一样按加法进位规则进行运算。运算所得的符号位就是正确结果的符号。反码运算和补码运算的差别在于：反码运算时，符号位产生的进位要加到数值的最低位；而补码运算时，不考虑符号位产生的进位，这个进位将自动丢失。

下面举例说明原码、补码和反码运算的方法。

例1.8 已知两个带符号二进制整数N_1和N_2，其真值形式为 $N_1=+10011$和$N_2=+01010$，求 $N=N_1-N_2$。

现分别以原码、反码和补码进行运算。

原码运算：

据题知，N_1和N_2的原码分别为010011和001010，且N_1的数值大于N_2的数值，所以应将N_1作为被减数，N_2作减数，其差值为正，即有