高等院校计算机教材系列

数字逻辑

胡家宝 编著

机械工业出版社
China Machine Press

高等院校计算机教材系列

数字逻辑

胡家宝 编著

机械工业出版社
China Machine Press

本书系统介绍了数字逻辑电路分析和设计的基本理论与基本方法，包括数制系统及其编码、布尔代数基础、组合逻辑电路和时序逻辑电路的基本概念、组合逻辑电路和时序逻辑电路的分析与设计方法、可编程逻辑器件和数字系统设计方法。本书还介绍了 VHDL 硬件描述语言和 MAX + PLUS Ⅱ软件的内容，便于应用计算机软件设计数字逻辑电路。

本书可以作为计算机科学与技术、自动控制、电子信息等专业的本科生教材，也可以作为数字系统设计相关科研人员的参考书。

图书在版编目（CIP）数据

数字逻辑/胡家宝编著．－北京：机械工业出版社，2006.2
（高等院校计算机教材系列）
ISBN 7-111-17805-X

Ⅰ.数…　Ⅱ.胡…　Ⅲ.数字电路:逻辑电路－高等学校－教材　Ⅳ.TN79

中国版本图书馆 CIP 数据核字(2005)第 132506 号

机械工业出版社(北京市西城区百万庄大街 22 号　邮政编码　100037)
责任编辑：傅志红
北京诚信伟业印刷有限公司印刷·新华书店北京发行所发行

2006 年 2 月第 1 版第 1 次印刷
787mm×1092mm 1/16·14.5 印张
印数：0 001-4000 册
定价：23.00 元

凡购本书，如有倒页、脱页、缺页，由本社发行部调换
本社购书热线：(010)68326294

前　言

“数字逻辑”是研究数字逻辑电路分析和设计的学科。它是计算机科学与技术专业的一门专业基础课程，也是核心课程之一。随着微电子技术和信息技术的不断发展，数字逻辑电路的分析与设计，以及可编程逻辑器件都有了很大的进展，它为数字逻辑电路的知识体系增添了新的内容和活力。作者结合教学实践，将数字逻辑与该领域的新技术相结合，系统地介绍了数字逻辑的理论与方法。本书的总体思路考虑如下：

保证基本理论的完整性。注重数字逻辑中的基本理论、基本技术和基本内容，目的是帮助读者掌握逻辑电路设计的基础理论和方法。

贯彻理论联系实际的原则。在注重理论论述的基础上，结合作者的研究和教学实践，精练讲述内容，精选例题和习题，帮助读者掌握逻辑电路的分析和设计方法。

在保证课程大纲内容的前提下，对有关概念、原理、方法的阐述力求准确、精练。在写作风格上力求通俗易懂，语言朴实，深入浅出。

内容上采用数制系统及其编码、布尔代数基础、组合逻辑电路、同步时序逻辑电路、异步时序逻辑电路、可编程逻辑器件、数字系统设计基础的编排顺序。如果需要可以把可编程逻辑器件移到组合逻辑电路后面讲授。

为了适应逻辑电路分析和设计新技术发展的需求，在附录中增加了 VHDL 硬件描述语言和 MAX + PLUSⅡ软件的内容，用于“数字逻辑”的实验教学或者课程设计，从而进一步拓展“数字逻辑”课程的分析和设计方法，帮助读者掌握逻辑电路设计的新技术。

本书共有 7 章。第 1 章介绍数制系统及其编码。主要包括数制及数制转换、有符号数的编码表示、数字系统中的 BCD 编码、可靠性编码和字符编码。第 2 章介绍布尔代数基础。主要包括逻辑代数基础，逻辑函数的概念，逻辑代数的公理、定理和规则，逻辑函数的标准形式，逻辑函数的化简。第 3 章介绍组合逻辑电路。主要包括组合逻辑电路的基本概念、组合逻辑电路的分析、组合逻辑电路的设计、含有无关最小项的组合逻辑电路的设计、逻辑函数中反变量的处理、组合逻辑电路的险象、常用组合逻辑集成电路。第 4 章介绍同步时序逻辑电路。主要包括时序逻辑电路的结构模型与分类、同步时序逻辑电路的结构模型和基本概念、同步时序逻辑电路的分析和设计方法、集成计数器。第 5 章介绍异步时序逻辑电路。主要包括异步时序逻辑电路的结构模型、脉冲异步时序逻辑电路的分析和设计、电平异步时序逻辑电路的分析和设计。第 6 章介绍可编程逻辑器件。主要包括可编程只读存储器、采用 ROM 阵列图设计数字逻辑电路、可编程逻辑阵列 PLA/PAL/GAL、在系统可编程技术和器件。第 7 章介绍数字系统设计。主要包括数字系统基本概念、数字系统设计的一般过程、数字系统设计工具和实现方法、数据子系统和控制子系统的设计方法。

附录 A 给出了美国信息交换标准码(ASCII)。

附录 B 介绍 VHDL 语言的基本语句和采用 VHDL 语言描述数字逻辑电路的例子，以便读者能够初步使用 VHDL 硬件描述语言。

附录 C 介绍 MAX + PLUSⅡ软件，给出了 2 个实例说明使用 MAX + PLUSⅡ软件设计和

分析逻辑电路的方法。

附录B与附录C结合起来，就可以使用VHDL语言编写组合逻辑电路和时序逻辑电路的VHDL程序，并对组合逻辑电路和时序逻辑电路进行仿真，实现组合逻辑电路和时序逻辑电路的功能，不需要用门电路和触发器连接组合逻辑电路和时序逻辑电路。这样能够帮助读者对组合逻辑电路和时序逻辑电路有更深入的认识。另外，结合实验课和课程设计，设计实际的组合逻辑电路和时序逻辑电路，就能够达到理论与实践相结合的目的。

本书中的逻辑电路图由胡莹绘制。

为了方便教学，我们使用PowerPoint制作了本书的课件，供教师在多媒体教学环境中使用。该课件可以在机械工业出版社华章分社网站 www.hzbook.com下载。另外我们还使用Web技术制作了多媒体教学软件，可满足在局域网和Internet中进行“数字逻辑”教学的需要。

由于作者水平有限，书中疏漏和错误之处在所难免，恳请广大读者指正。

胡家宝
于武汉理工大学

目　录

第 1 章　数制系统及其编码

计算机技术的发展促进了科学技术的进步和生产力的发展。目前，计算机技术广泛应用于科学计算、数据处理和生产过程控制等领域。计算机中处理的量是数字量。数字量是使用“0”和“1”表示现实世界中的物理量，例如使用数字量表示时间、颜色、图形图像等。随时间变化的物理量称为模拟量，例如压力表、温度计、交流电压表等表示的量是模拟量。使用这些仪表表示模拟量时，如果要求有较高的精度，则很难对被测的模拟量进行运算和存储。这是因为这些仪表中处理的是模拟量。当前对模拟量的处理正在越来越多地被数字量处理所替代。例如数字压力表，可以以较高的精度测量压力，测得的压力数据可以容易地进行运算和存储，因为数字压力表中处理的是数字量。

对数字量进行传输、处理的系统称为数字系统，例如电子钟、电子手表都是数字系统。电子计算机是典型的数字系统。

数字逻辑主要研究使用数字量进行逻辑处理的逻辑电路。逻辑电路的输入是数字量，数字量在电路中进行“与”、“或”、“非”等逻辑运算，逻辑处理后的数字量可以被传输和存储。研究数字逻辑电路是从分析和设计两个方面进行的。分析研究是使用分析方法，得到数字逻辑电路的逻辑功能；设计研究是使用设计方法，设计满足要求的数字逻辑电路。数字逻辑电路包括组合逻辑电路与时序逻辑电路。时序逻辑电路包括同步时序逻辑电路与异步时序逻辑电路。

本章介绍数字系统中使用的二进制数和其他进位计数制，二进制数同其他进位计数制之间的转换方法，有符号的二进制数在数字系统中的表示方法，二－十进制编码，可靠性编码和字符代码。

1.1　数制系统

1.1.1　位置数制系统

采用十进制计数表示数值，使用的是一种称为位置数制的表示数值方法。也就是每位十进制数符的数值与它的位置有关。每位数符的数值是该数符与它的进位基数幂的乘积。例如，$759.24_{10} = 7\times10^2 + 5\times10^1 + 9\times10^0 + 2\times10^{-1} + 4\times10^{-2}$。

使用二进制计数表示数值，每位数符的数值也是该数符与它的进位基数幂的乘积。例如

$$1101.101 = 1\times2^3 + 1\times2^2 + 0\times2^1 + 1\times2^0 + 1\times2^{-1} + 0\times2^{-2} + 1\times2^{-3}$$

这种表示数值的方法称为位置数制系统。在位置数制系统中，数符所在位置的进位基数的幂称为该数符的位权。对于一个数值 N，设进位基数为 R，使用位置数制系统可以表示为

$$N_R = a_{n-1}\,a_{n-2}\cdots a_1\,a_0\,a_{-1}\,a_{-2}\cdots a_{-m} = a_{n-1}\times R^{n-1} + a_{n-2}\times R^{n-2} + \cdots + a_1\times R^1 + a_0\times R^0 + a_{-1}\times R^{-1} + a_{-2}\times R^{-2} + \cdots + a_{-m}\times R^{-m} = \sum_{i=-m}^{n-1} a_i R^i$$

式中，a_i 是 R^i 的数字符号，称为数符。a_i 的取值范围是 $0 \leqslant a_i \leqslant R-1$，$R$ 为进位基数，n

为整数数值的位数，m 为小数数值的位数。上述表示也称为数值 N_R 的多项式表示。

当 $R = 2$ 时，称为二进制计数，简称二进制数，它只有 0 和 1 两个数字符号。例如

$$1011.01_2 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2} = 8 + 0 + 2 + 1 + 0 + 0.25 = 11.25_{10}$$

当 $R = 8$ 时，称为八进制计数，简称八进制数，它有 0，1，2，3，4，5，6，7 八个数字符号。例如

$$147.3_8 = 1 \times 8^2 + 4 \times 8^1 + 7 \times 8^0 + 3 \times 8^{-1} = 103.375_{10}$$

当 $R = 16$ 时，称为十六进制计数，简称十六进制数，它有 0，1，2，3，4，5，6，7，8，9，A，B，C，D，E，F 十六个数字符号。例如

$$2F.A_{16} = 2 \times 16^1 + 15 \times 16^0 + 10 \times 16^{-1} = 47.625_{10}$$

表 1-1 列出了十进制数 0～16 所对应的二进制数、八进制数和十六进制数。

表 1-1 十进制数 0～16 的二、八、十六进制数

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
16	10000	20	10

二进制数的运算规则如下：

加法规则 $0 + 0 = 0$ $0 + 1 = 1$ $1 + 0 = 1$ $1 + 1 = 0$(进位是 1)

减法规则 $0 - 0 = 0$ $1 - 0 = 1$ $1 - 1 = 0$ $0 - 1 = 1$(借位是 2)

乘法规则 $0 \times 0 = 0$ $1 \times 0 = 0$ $0 \times 1 = 0$ $1 \times 1 = 1$

除法规则 $0 / 1 = 0$ $1 / 1 = 1$

例 1-1 设 $A = 11110$，$B = 110$，计算：(1)$A + B$；(2)$A - B$；(3)$A \times B$；(4)$A \div B$。

解：(1) $A + B = 11110 + 110 = 100100$

$$\begin{array}{r} 11110 \\ +\quad 110 \\ \hline 100100 \end{array}$$

(2) $A-B = 11110 - 110 = 11000$

$$\begin{array}{r} 1\ 1\ 1\ 1\ 0 \\ -\quad\ 1\ 1\ 0 \\ \hline 1\ 1\ 0\ 0\ 0 \end{array}$$

(3) $A\times B = 11110 \times 110 = 10110100$

$$\begin{array}{r} 1\ 1\ 1\ 1\ 0 \\ \times\quad\ 1\ 1\ 0 \\ \hline 0\ 0\ 0\ 0\ 0 \\ 1\ 1\ 1\ 1\ 0 \\ +\ 1\ 1\ 1\ 1\ 0 \\ \hline 1\ 0\ 1\ 1\ 0\ 1\ 0\ 0 \end{array}$$

(4) $A \div B = 11110 \div 110 = 101$

$$\begin{array}{r} \quad\ 1\ 0\ 1 \\ 110\overline{)\ 1\ 1\ 1\ 1\ 0} \\ -1\ 1\ 0 \\ \hline \quad\ 1\ 1\ 0 \\ -\quad 1\ 1\ 0 \\ \hline \quad\ 0 \end{array}$$

从二进制数的运算规则可以看到，二进制数运算比较简单。二进制数的物理实现很容易。例如，可以用二进制数的“0”和“1”表示电子器件中的无脉冲和有脉冲。另外，二进制数的物理实现比其他进位计数制的物理实现节省电子器件。再如，晶体管导通和截止时的电平分别用“0”和“1”表示。用“0”和“1”表示的脉冲波形如图1-1所示。使用触发器存储表示“0”和“1”也比较方便。因此在数字系统中都是采用二进制计数。

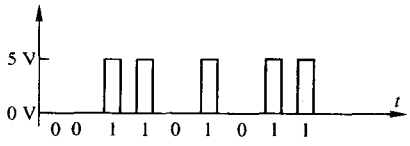


图 1-1 二进制数“0”和“1”表示脉冲波形

二进制数的缺点是在表示一个数值时，它的位数比八进制数、十进制数的位数要多，不方便书写、阅读和记忆。

由于八进制数和十六进制数转换成二进制数比较方便，因此常用于表示数字系统中的输入、输出和显示。

1.1.2 数制转换

1. 二进制数与十进制数的转换

(1) 二进制数转换成十进制数

二进制数转换成十进制数可采用多项式表示法进行。将二进制数使用位权展开成多项式，然后采用十进制数运算法则进行计算。例如

$$11010.101 = 1\times 2^4 + 1\times 2^3 + 0\times 2^2 + 1\times 2^1 + 0\times 2^0 + 1\times 2^{-1} + 0\times 2^{-2} + 1\times 2^{-3}$$

$$= 16 + 8 + 0 + 2 + 0 + 0.5 + 0 + 0.125 = 26.625$$

二进制数转换成十进制数的方法可以进行推广。任何一种进位计数制要转换成十进制数，都可以使用多项式表示法来完成。

(2) 十进制数转换成二进制数

十进制数可能既有整数数值又有小数数值，要把它转换成二进制数，需要对整数数值和小数数值分别进行转换，然后把转换的两部分合并起来。

1) 整数转换。整数转换采用"除2取余"方法。该算法是设十进制数的整数为 N，将 N 除2，它的余数"0"或者"1"是转换成的二进制数整数数值最低位 a_0 的数符；再将前次 N 除2的商继续除2，它的余数是 a_1 的数符。按照这样的方法进行下去，直到商是0时为止，最后一位余数是 a_{n-1} 的数符。所有余数数符组成的序列，即是 N 转换成的二进制数的整数数值，即 $N_{10} = a_{n-1}\, a_{n-2} \cdots a_1\, a_0$。必须注意，在除2的运算中，第一次 N 除2的余数是转换成的二进制数的最低位，最后一位余数是转换成的二进制数的最高位。例如，将十进制数23转换成二进制数，整数转换过程一般使用下面的形式进行：

		余数	
2	23	1	低位
2	11	1	
2	5	1	
2	2	0	
2	1	1	高位
	0		

所以，十进制数23对应的二进制数是10111。

上述方法可以进行推广。设十进制数的整数为 N，R 为要转换的进位基数。将 N 除以 R，写下 N 除以 R 的余数，再将前次 N 除以 R 的商除以 R，写下余数。反复进行下去，即可把 N 转换成 R 进位计数制的整数。

2) 小数转换。小数转换采用"乘2取整"方法。该算法是设十进制数的小数为 N，将 N 乘2，它的积的整数部分"0"或者"1"是转换成的二进制数小数数值最高位 a_{-1} 的数符。再将前次乘2以后积的小数部分继续乘2，它的积的整数部分是 a_{-2} 的数符。按照这样的方法进行下去，直到积的小数部分是0时为止。所有整数的数符"0"或者"1"组成的序列，即是转换成的二进制数小数，即 $N_{10} = 0.a_{-1}\, a_{-2} \cdots a_{-m}$。例如，将十进制数0.6875转换成二进制数，转换过程一般使用下面的形式进行：

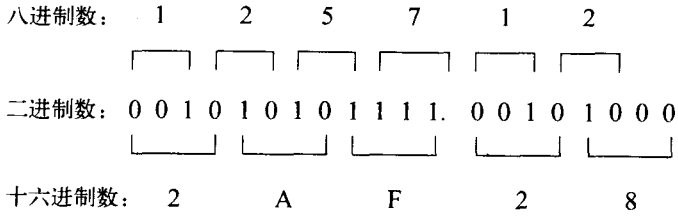
		0.6875	
		× 2	
小数高位	$a_{-1}=1$	1.3750	
		× 2	
	$a_{-2}=0$	0.7500	
		× 2	
	$a_{-3}=1$	1.5000	
		× 2	
小数低位	$a_{-4}=1$	1.0000	

所以，十进制数 0.6875 转换成二进制数的小数是 0.1011。但是，有的时候十进制小数数值 N 经过 K 次乘 2 后，积的小数部分始终不能为“0”，这表明该十进制小数 N 不能用有限位数的二进制数小数数值表示。这时根据要求转换到规定的小数位数即可。确定最低位数符是 0 或者 1 的方法是，根据要求把十进制数转换成一定位数的二进制小数数值 $0.a_{-1}a_{-2}\cdots a_{-m}$ 后，再求出二进制小数 $a_{-(m+1)}$ 位的数符，然后对 $a_{-(m+1)}$ 作“0”舍“1”入的处理。也就是，如果 $a_{-(m+1)}=0$，则 a_{-m} 位的数值不变，即把 $a_{-(m+1)}=0$ 舍弃，如果 $a_{-(m+1)}=1$，则在 a_{-m} 位的数值上加 1。

上述方法可以进行推广。设十进制数的小数数值为 N，R 为要转换的进位基数。将 N 乘以 R，写下 N 乘以 R 整数部分的数符。再将前次 N 乘以 R 的小数数值乘以 R，写下 N 乘以 R 整数部分的数符。反复进行下去，即可把 N 转换成 R 进制小数。

2. 二进制数与八进制数和十六进制数间的转换

八进制数的基数是 $8(2^3)$，十六进制数的基数是 $16(2^4)$，八进制数与十六进制数的基数都是 2 的整数幂。由于三位二进制数的不同组合分别对应着八进制数中的每一个数符，把二进制数转换成八进制数时，以二进制数的小数点为中心，分别向左、右两边将每三位二进制数分为一组。在分组时，如果最左边出现不足三位二进制数，则在它的前面添 0，补足成三位二进制数；如果最右边出现不足三位二进制数，则在它的后面添 0，补足成三位二进制数，然后写出每一组对应的八进制数的数符，即可把二进制数转换成八进制数。同样，由于四位二进制数的不同组合分别对应着十六进制数中的每一个数符，这样把二进制数转换成十六进制数时，以二进制数的小数点为中心，分别向左、右两边将每四位二进制数分为一组。在分组时，如果最左边出现不足四位二进制数，则在它的前面添 0，补足成四位二进制数；如果最右边出现不足四位二进制数，则在它的后面添 0，补足成四位二进制数，然后写出每一组对应的十六进制数的数符，即可把二进制数转换成十六进制数。例如



所以二进制数 $001010101111.00101000_2=1257.12_8=2AF.28_{16}$。

要将八进制数、十六进制数转换成二进制数，可按上述方法的相反过程进行。

1.2 有符号二进制数的编码表示

在前面讨论进位计数制时，没有涉及到符号。本节介绍在数字系统中，有符号的二进制数如何表示，以及这样的表示给二进制数运算带来的方便。人们经常在二进制数前用“－”、“＋”符号表示二进制数负数和正数。这样表示的二进制数称为*真值*。

在数字系统中，真值由表示真值的“－”、“＋”符号和真值的数值两个部分组成。一般用“0”表示真值的“＋”符号，用“1”表示真值的“－”符号，这样真值的符号就数字化了。将真值的符号部分数字化以及真值的数值部分采用编码表示，这样表示的值称为*机器数*。真值的符号部分在机器数中称为*符号位*，真值的数值部分在机器数中称为*尾数*。例如，真值

+ 0.1011001的机器数表示如图 1-2a 所示。真值 - 0.1011001 的机器数表示如图 1-2b 所示。

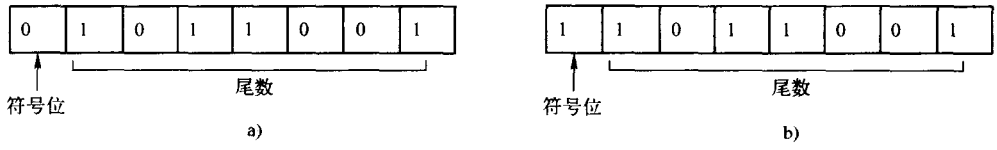


图 1-2 二进制数的机器数表示

在机器数中，如果默认小数点固定在符号位的右边，这样机器数的尾数部分只能表示小数，这种表示二进制数的方法称为数的定点小数表示。如果数字系统使用定点小数表示，则数字系统中的数值以小数形式存储。当从数字系统输出数值时，再转换成原来形式的数值。如果默认小数点固定在尾数的右边，这样尾数部分只能表示整数 - 这种表示二进制数的方法称为数的定点整数表示。

机器数有 3 种形式，即原码、补码和反码。下面分别进行介绍。

1.2.1 原码

采用原码表示有符号的二进制数时，符号位部分用“0”表示二进制正数，用“1”表示二进制负数，尾数部分与真值的数值部分相同。因此采用原码的形式表示二进制数时，仅是二进制数的符号位数字化了。下面对二进制数整数和小数的原码分别进行讨论。

1. 二进制小数的原码

设二进制小数 $N=\pm 0.a_{-1}a_{-2}\cdots a_{-m}$，它的原码定义为

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

例如，$N_1=+0.1011$，$N_2=-0.1011$，N_1和 N_2 的原码是

$$[N_1]_{原}=0.1011$$

$$[N_2]_{原}=1-(-0.1011)=1.1011$$

根据定义，小数“0.000…00”用原码表示有两种形式，即

$$[+0.000\cdots00]_{原}=0.000\cdots00$$

$$[-0.000\cdots00]_{原}=1.000\cdots00$$

2. 二进制整数的原码

设二进制整数 $N=\pm a_{n-1}a_{n-2}\cdots a_0$，它的原码定义为

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

其中，n 为二进制整数N 的位数。例如，$N_1=+1101$，$N_2=-1101$，N_1 和 N_2 的原码是

$$[N_1]_{原}=0\ 1101$$

$$[N_2]_{原}=2^4-(-1101)=1\ 0000+1101=1\ 1101$$

根据定义，整数“000…00”用原码表示也有两种形式，即

$$[+000\cdots00]_{原}=0\ 000\cdots00$$

$$[-000\cdots00]_{原}=1\ 000\cdots00$$

采用原码形式表示有符号二进制数的物理实现很简单。但是，在数字系统中采用原码运

算不方便，因为，当两个原码进行加法或者减法运算时，需要根据两个原码的符号位“0”或者“1”来决定是做加法运算还是做减法运算。如果是做减法运算，还需要根据尾数的绝对值确定哪一个作为被减数，哪一个作为减数，并确定运算结果的符号位是“0”还是“1”。这些会增添数字系统中运算的复杂性。

1.2.2 反码

采用反码表示有符号的二进制数时，符号位部分用“0”表示二进制正数，用“1”表示二进制负数。反码的尾数部分与符号位有关。符号位是“0”时，尾数同真值的数值部分相同；符号位是“1”时，尾数是把真值的数值部分各位取反，也就是“0”写成“1”，“1”写成“0”。

1. 二进制小数的反码

设二进制小数 $N = \pm 0.a_{-1}a_{-2}\cdots a_{-m}$，它的反码定义为

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

其中，m 为二进制小数 N 的位数。例如，$N_1 = +0.1011$，$N_2 = -0.1011$，N_1 和 N_2 的反码是

$$[N_1]_{反} = 0.1011$$

$$[N_2]_{反} = 2 - 2^{-4} + N_2 = 1.1111 - 0.1011 = 1.0100$$

根据定义，小数“0.000…00”用反码表示有两种形式，即

$$[+0.000\cdots00]_{反} = 0.000\cdots00$$

$$[-0.000\cdots00]_{反} = 1.111\cdots11$$

2. 二进制整数的反码

设二进制整数 $N = \pm a_{n-1}a_{n-2}\cdots a_0$，它的反码定义为

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

其中，n 为二进制整数 N 的位数。例如，$[N_1] = +1101$，$N_2 = -1101$，N_1 和 N_2 的反码是

$$[N_1]_{反} = 0\ 1101$$

$$[N_2]_{反} = (2^5 - 1) + N_2 = 1\ 1111 - 1101 = 1\ 0010$$

根据定义，整数“000…00”用反码表示也有两种形式，即

$$[+000\cdots00]_{反} = 0\ 000\cdots00$$

$$[-000\cdots00]_{反} = 1\ 111\cdots11$$

可以使用反码实现两个二进制数的加法与减法运算。设 N_1 和 N_2 是两个二进制数，运算规则如下：

$$\text{计算 } N_1 + N_2\text{：}[N_1]_{反} + [N_2]_{反} = [N_1 + N_2]_{反}$$

$$\text{计算 } N_1 - N_2\text{：}[N_1]_{反} + [-N_2]_{反} = [N_1 - N_2]_{反}$$

当使用反码计算 $N_1 + N_2$ 时，首先分别求出 N_1 和 N_2 的反码，即 $[N_1]_{反}$ 和 $[N_2]_{反}$。然后计算 $[N_1]_{反} + [N_2]_{反}$，便是 $[N_1 + N_2]_{反}$ 的结果。当使用反码计算 $N_1 - N_2$ 时，首先分别求出 N_1 和 $-N_2$ 的反码，即 $[N_1]_{反}$ 和 $[-N_2]_{反}$。然后计算 $[N_1]_{反} + [-N_2]_{反}$，便是 $[N_1 - N_2]_{反}$ 的结果。在这里，使用反码计算 $N_1 - N_2$，是通过计算 $[N_1]_{反} + [-N_2]_{反}$ 实现的。使用反码运算可以把真值的减法运算转化为加法运算。

必须注意，由于是使用反码计算 N_1+N_2 或 N_1-N_2，因此，其结果仍然是反码，也就是$[N_1+N_2]_{反}$或$[N_1-N_2]_{反}$。另外，如果$[N_1]_{反}+[-N_2]_{反}$结果的符号位是“0”，表示 N_1-N_2 的结果是正数；如果符号位是“1”，表示 N_1-N_2 的结果是负数，要写出这个结果的真值，则把$[N_1]_{反}+[-N_2]_{反}$结果的尾数部分求反码一次。

使用反码进行运算时，符号位和尾数一起参加运算。如果符号位在相加时产生进位，要将进位“1”加到尾数运算结果的最低位。

例如，$N_1=+0.1110$，$N_2=+0.0101$，采用反码计算 N_1-N_2，得到

$$[N_1-N_2]_{反}=[N_1]_{反}+[-N_2]_{反}=0.1110+1.1010$$

$$\begin{array}{r} 0.1110 \\ +\quad 1.1010 \\ \hline 10.1000 \\ +\ \text{└}\!\!\rightarrow 1 \\ \hline 0.1001 \end{array}$$

所以，$[N_1-N_2]_{反}=0.1001$。由于$[N_1-N_2]_{反}$结果的符号位为“0”，表示结果是正数，因此 $N_1-N_2=+0.1001$。

1.2.3 补码

采用补码表示有符号的二进制数时，符号位用“0”表示二进制正数，用“1”表示二进制负数。补码的尾数部分与符号位有关。符号位是“0”时，尾数同真值的数值部分相同。符号位是“1”时，尾数是把真值的数值部分各位取反，然后在尾数的最低位上加 1。

1. 二进制小数的补码

设二进制小数 $N=\pm 0.a_{-1}a_{-2}\cdots a_{-m}$，它的补码定义为

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

例如，$N_1=+0.1011$，$N_2=-0.1011$，N_1 和 N_2 的补码是

$$[N_1]_{补}=0.1011$$

$$[N_2]_{补}=2+N_2=10.0000-0.1011=1.0101$$

根据定义，小数“0.000…00”用补码表示只有一种形式，即

$$[+0.000\cdots00]_{补}=0.000\cdots00$$

$$[-0.000\cdots00]_{补}=0.000\cdots00$$

2. 二进制整数的补码

设二进制整数 $N=\pm a_{n-1}a_{n-2}\cdots a_0$，它的补码定义为

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

其中，n 为二进制整数N的位数。例如，$N_1=+1101$，$N_2=-1101$，N_1 和 N_2 的补码是

$$[N_1]_{补}=0\ 1101$$

$$[N_2]_{补}=2^5+N_2=10\ 0000-1101=1\ 0011$$

根据定义，整数“000…00”用补码表示只有一种形式，即

$$[\ +\ 000\cdots00\ ]_{补} = 0\ 000\cdots00$$

$$[\ -\ 000\cdots00\ ]_{补} = 0\ 000\cdots00$$

可以使用补码实现两个二进制数的加法与减法运算。设 N_1 和 N_2 是两个二进制数，运算规则如下：

计算 $N_1 + N_2$：$[N_1]_{补} + [N_2]_{补} = [N_1 + N_2]_{补}$

计算 $N_1 - N_2$：$[N_1]_{补} + [-N_2]_{补} = [N_1 - N_2]_{补}$

当使用补码计算 $N_1 + N_2$ 时，首先分别求出 N_1 和 N_2 的补码，即$[N_1]_{补}$和$[N_2]_{补}$。然后计算$[N_1]_{补} + [N_2]_{补}$，便是$[N_1 + N_2]_{补}$的结果。当使用补码计算 $N_1 - N_2$ 时，首先分别求出 N_1和 $-N_2$ 的补码，即$[N_1]_{补}$和 $[-N_2]_{补}$。然后计算$[N_1]_{补} + [-N_2]_{补}$，便是$[N_1 - N_2]_{补}$的结果。在这里，使用补码计算 $N_1 - N_2$，是通过计算$[N_1]_{补} + [-N_2]_{补}$实现的。使用补码运算也可以把真值的减法运算转化为加法运算。

必须注意，由于是使用补码计算 $N_1 + N_2$ 或 $N_1 - N_2$，因此，其结果仍然是补码，也就是$[N_1 + N_2]_{补}$或$[N_1 - N_2]_{补}$。另外，如果$[N_1]_{补} + [-N_2]_{补}$结果的符号位是“0”，表示 $N_1 - N_2$ 的结果是正数。如果符号位是“1”，表示 $N_1 - N_2$ 的结果是负数，要写出这个结果的真值，则把$[N_1]_{补} + [-N_2]_{补}$结果的尾数部分求补码一次。

使用补码进行运算时，符号位和尾数一起参加运算。如果符号位在相加时产生进位，则将该进位“1”丢掉。

例 1-2 $N_1 = +1001$，$N_2 = +0011$。采用补码计算：(1) $N_1 + N_2$；(2) $N_1 - N_2$；(3) $N_2 - N_1$。

解：(1) 计算 $N_1 + N_2$

$$[N_1 + N_2]_{补} = [N_1]_{补} + [N_2]_{补} = 0\ 1001 + 0\ 0011$$

$$\begin{array}{r} 0\ \ 1\ 0\ 0\ 1 \\ +\ \ 0\ \ 0\ 0\ 1\ 1 \\ \hline 0\ \ 1\ 1\ 0\ 0 \end{array}$$

$[N_1]_{补} + [N_2]_{补}$运算结果为 0 1100，它的符号位为“0”，表示结果是正数。因此，$N_1 + N_2 = +1100$。

(2) 计算 $N_1 - N_2$

$$[N_1 - N_2]_{补} = [N_1]_{补} + [-N_2]_{补} = 0\ 1001 + 1\ 1101$$

$$\begin{array}{r} 0\ \ 1\ 0\ 0\ 1 \\ +\ \ 1\ \ 1\ 1\ 0\ 1 \\ \hline \text{丢掉①}\ \ 0\ \ 0\ 1\ 1\ 0 \end{array}$$

$[N_1]_{补} + [-N_2]_{补}$运算结果为 0 0110，它的符号位为“0”，表示结果是正数。符号位在相加时产生进位，将该进位“1”丢掉。因此，$N_1 - N_2 = +0110$。

(3) 计算 $N_2 - N_1$

$$[N_2 - N_1]_{补} = [N_2]_{补} + [-N_1]_{补} = 0\ 0011 + 1\ 0111$$

$$\begin{array}{r} 0\ \ 0\ 0\ 1\ 1 \\ +\ \ 1\ \ 0\ 1\ 1\ 1 \\ \hline 1\ \ 1\ 0\ 1\ 0 \end{array}$$

$[N_2]_{补}+[-N_1]_{补}$运算结果为 1 1010，它的符号位为“1”，表示结果是负数。$[N_2]_{补}+[-N_1]_{补}=1\ 1010$是$[N_2]_{补}+[-N_1]_{补}$运算后负数的补码形式。如果要把“1 1010”表示成真值，那么对“1 1010”求一次补码，即最左边的一位“1”是真值的符号位“-”，然后把尾数每位求反后，在尾数的最低位上加 1，因此，$N_2-N_1=-0110$。

在数字系统中，求补码的实现比较容易，且补码运算比反码运算要简单一些，所以补码运算广泛使用在数字系统中。

1.3 数字系统中的编码

在数字系统中，使用机器数表示二进制数，也使用二进制数表示字符。如果要在数字系统中使用二进制数字表示字符，就涉及对字符进行编码。另外，在数字系统中二进制数传输、处理的速度很高，为了防止出错需要使用可靠性编码。下面介绍在数字系统中常用的几种编码。

1.3.1 十进制数的二进制编码

如前所述，在数字系统中除了采用机器数表示二进制数以外，有时需要用若干位二进制数表示一位十进制数，以便能在数字系统中表示和使用十进制数，适应处理十进制数的需要。目前通常使用四位二进制数对十进制数的每一个数符进行编码，称为二-十进制编码，简称 BCD 码。下面介绍 3 种 BCD 码。

1. 8421 BCD 码

8421 BCD 码是将每个十进制数的数符用四位二进制数表示，即用 0000～1001 这 10 个不同的四位二进制数分别表示十进制数的 0～9 这 10 个数符。它的编码如表 1-2 中所示。在 8421 BCD 码中，每一位二进制数符从左到右的位权分别是 2^3、2^2、2^1、2^0。因此，8421 BCD 码称为有权码。把十进制数转换成 8421 BCD 码是将该十进制数的每一位数符换成 8421 BCD 码。8421 BCD 码转换成十进制数是将 8421 BCD 码的每四位二进制数写出它所对应的十进制数的数符。

例如，写出十进制数 238 的 8421 BCD 码。

$$238_{10}=0010\ 0011\ 1000\ (\ 8421\ \text{BCD 码}\ )$$

例如，写出 1001 0100 0001(8421 BCD 码)的十进制数。

$$1001\ 0100\ 0001(\ 8421\ \text{BCD 码}\ )=941_{10}$$

8421 BCD 码与十进制数中的 10 个数符所表示的二进制数完全相同，因此十进制数转换成 8421 BCD 码很容易实现。8421 BCD 码具有奇偶特性。当十进制数符是奇数时，它所对应的 8421 BCD 码的最低位数符为 1；当十进制数符是偶数时，它所对应的 8421 BCD 码的最低位数符为 0。在 8421 BCD 码中不使用 1010～1111 这 6 个二进制数的组合。

2. 2421 BCD 码

2421 BCD 码是将每个十进制数的数符用四位二进制数表示，即用 0000～0100、1011～1111 这 10 个不同的四位二进制数分别表示十进制数的 0～9 这 10 个数符。在 2421 BCD 码中 0000～0100 这 5 个编码与 8421 BCD 码相同，分别表示十进制数的 0～4 这 5 个数符。注