普通高等院校电气电子类规划系列教材

数字逻辑与数字系统

主编　付智辉　裴亚男

SHUZI LUOJI
YU SHUZI XITONG

西南交通大學出版社

普通高等院校电气电子类规划系列教材

数字逻辑与数字系统

主编　付智辉　裴亚男

SHUZI LUOJI
YU SHUZI XITONG

西南交通大学出版社
·成　都·

内 容 简 介

本书从数字电路的基础知识出发，介绍了数制和码制、逻辑代数、门电路、组合逻辑、时序逻辑、半导体存储器、可编程器件（PLD，CPLD，HDPLD，FPGA）、在系统编程技术（ISP）、硬件描述语言（VHDL）、Quartus II 开发系统及 EDA 技术的设计思想等内容。每章结尾有一定数量的习题。

本书可作为高等院校计算机、通信、电子信息、自动化等专业的相关课程的教材，也可作为相关技术人员的参考书。

图书在版编目（CIP）数据

数字逻辑与数字系统 / 付智辉，裴亚男主编. —成都：西南交通大学出版社，2015.1
普通高等院校电气电子类规划系列教材
ISBN 978-7-5643-3393-5

Ⅰ. ①数… Ⅱ. ①付… ②裴… Ⅲ. ①数字逻辑－高等学校－教材②数字系统－高等学校－教材 Ⅳ. ①TP302.2

中国版本图书馆 CIP 数据核字（2014）第 204812 号

普通高等院校电气电子类规划系列教材
数字逻辑与数字系统
主编 付智辉 裴亚男
*
责任编辑 李芳芳
助理编辑 张少华
封面设计 墨创文化
西南交通大学出版社出版发行
四川省成都市金牛区交大路 146 号 邮政编码：610031 发行部电话：028-87600564
http: //www.xnjdcbs.com
四川森林印务有限责任公司印刷
*
成品尺寸：185 mm × 260 mm 印张：15.75
字数：392 千字
2015 年 1 月第 1 版 2015 年 1 月第 1 次印刷
ISBN 978-7-5643-3393-5
定价：33.00 元

课件咨询电话：028-87600533
图书如有印装质量问题 本社负责退换

前　言

数字逻辑与数字系统是高等院校计算机科学与技术、电子工程、通信类等专业的重要基础课程。对于从事计算机（包括单片机）系统开发与应用的广大科技工作者来说，熟练掌握数字逻辑设计的理论和方法是十分必要的。

数字逻辑设计的物质基础是逻辑器件。随着微电子技术的飞速发展，数字逻辑器件，特别是可编程逻辑器件及相关技术发展迅猛。为此，数字逻辑课程的教学内容也必须不断改进，才能使教学与科技发展相适应。

本书在编写过程中，注重以下几点：

1. 在内容、体系上，从小规模数字集成电路入手，重点介绍中、大规模集成电路。

2. 注意工程能力的培养，对于功能级器件（中规模数字集成电路）均以实际产品为例，着重讨论器件的功能真值表和外特性，进而讨论该器件的应用，具有实用性的特点。

3. 注意吸取国内外的先进技术，重点介绍大规模数字集成电路原理和应用，特别是运用EDA技术对电子电路进行分析和设计的新方法。

4. 为贯彻理论联系实际的原则，书中以不同的方式安排了一定数量的电路应用实例并注重阅读电子电路图和电子器件手册训练以提高学生的实际应用能力。

5. 正文、例题、习题紧密配合。例题是对正文的补充，而某些内容则需要学生通过习题来巩固，有利于学生的理解和深化。

6. 为方便读者用EDA软件作电路图时选取元器件，本书最后附录了数字集成电路功能分类/品种代号对照表。

本书的第1章和第7章由付智辉编写，第2章和第3章由徐敏道编写，第4章和第6章裴亚男编写，第5章和附录由钟化兰编写。全书由付智辉统稿并审核。

限于编者水平，疏漏和不足之处在所难免，敬请读者指正。

编　者

2014.6

目　录

第 1 章　数制与码制

在数字逻辑系统中，所有的信息单元都是用二元符“0”和“1”表示的，它们代表的可能是数值量，也可能是逻辑量，还可能是数字、字符等其他任何信息。用一串“0”和“1”表示的数值量我们称其为二进制数。数有正有负，不带符号的数默认为正数；数还有大小之分。用一串“0”和“1”表示的数字、字符等其他任何信息我们都称其为二进制码。码既没有符号也没有大小之说。用一位“0”和“1”可以表示逻辑量“是”与“非”、“真”与“假”等。本章讨论二进制数与二进制码。

1.1　数制与数制转换

1.1.1　数　制

用数值量表示物理量的大小时，仅用一位数码往往不够用，因此经常需要用进位计数的方法组成多位数码来使用。我们把多位数码中每一位的构成方法以及从低位到高位的进位规则称为数制。

在数字系统中经常使用的计数进制除了十进制以外，还有二进制和十六进制。

1. 十进制

十进制是日常生活和工作中最常使用的进位计数制。在十进制数中，每一位可能是 0 ~ 9 十个数码中的一个，所以计数的基数是 10。超过 9 的数必须用多位数表示，其中低位和相邻高位之间的关系是“逢十进一”，故称为十进制。例如

$$163.78 = 1 \times 10^2 + 6 \times 10^1 + 3 \times 10^0 + 7 \times 10^{-1} + 8 \times 10^{-2}$$

所以任意一个十进制数 D 均可展开为

$$D = \sum k_i \times 10^i \tag{1-1}$$

其中，k 是第 i 位的系数，它可以是 0 ~ 9 这十个数码中的任何一个。若整数部分的位数是 n，小数部分的位数为 m，则 i 包含从 $n-1$ 到 0 的所有正整数和从 -1 到 $-m$ 的所有负整数。

若以 N 取代式（1-1）中的 10，即可得到任意进制（N 进制）数展开式的普遍形式

$$D = \sum k_i \times N^i \tag{1-2}$$

式中，i 的取值与式（1-1）的规定相同；N 称为计数的基数；k_i 为第 i 位的系数；N^i 称为第 i 位的权。

2. 二进制

目前在数字系统中应用最广的是二进制。在二进制数中，每一位仅有 0 和 1 两个可能的数码，所以计数基数为 2。低位和相邻高位间的进位关系是“逢二进一”，故称为二进制。

根据式（1-2），任何一个二进制数均可展开为

$$D=\sum k_i\times 2^i \tag{1-3}$$

并由此式计算出它所表示的十进制数的大小。例如

$$(1101.11)_2=1\times2^3+1\times2^2+0\times2^1+1\times2^0+1\times2^{-1}+1\times2^{-2}=(13.75)_{10}$$

式中，分别使用下标注 2 和 10 表示括号里的数是二进制和十进制数。有时也用 B（Binary）和 D（Decimal）代替 2 和 10 这两个下标。

3. 十六进制

十六进制数的每一位有十六个不同的数码可能，分别用 0～9、A（10）、B（11）、C（12）、D（13）、E（14）、F（15）表示。因此，任意一个十六进制数均可展开为

$$D=\sum k_i\times 16^i \tag{1-4}$$

并由此式计算出它所表示的十进制数值。例如

$$(\mathrm{B5.7D})_{16}=11\times16^1+5\times16^0+7\times16^{-1}+13\times16^{-2}=(181.48828125)_{10}$$

式中，下标注 16 表示括号里的数是十六进制，有时也有 H（Hexadecimal）代替这个下标。

由于计算机多为 8 位、16 位和 32 位机，而 8 位、16 位和 32 位的二进制数用 2 位、4 位和 8 位的十六进制数表示更加简便。

1.1.2 数制转换

1. 二-十转换

把二进制数转换为等值的十进制数称为二-十转换。转换时只要将二进制数按式（1-3）展开，然后把所有各项的数值按十进制数相加，就可以得到等值的十进制数了。例如

$$(1011.01)_2=1\times2^3+0\times2^2+1\times2^1+1\times2^0+0\times2^{-1}+1\times2^{-2}=(11.25)_{10}$$

2. 十-二转换

所谓十-二转换，就是把十进制数转换成等值的二进制数。

1）整数部分的转换

假定 $(N)_{10}$ 为十进制整数，等值的二进制数为 $(k_nk_{n-1}\cdots k_0)_2$，则依式（1-3）可知

$$\begin{aligned}(N)_{10}&=k_n2^n+k_{n-1}2^{n-1}+\cdots+k_12^1+k_02^0\\&=2(k_n2^{n-1}+k_{n-1}2^{n-2}+\cdots+k_1)+k_0\end{aligned} \tag{1-5}$$

式（1-5）表明，若将 $(N)_{10}$ 除以 2，则得到的商为 $k_n2^{n-1}+k_{n-1}2^{n-2}+\cdots+k_1$，而余数为 k_0。

同理，将式（1-5）中的商除以 2 得到新的商，即

$$k_n 2^{n-1} + k_{n-1} 2^{n-2} + \cdots + k_1 = 2(k_n 2^{n-2} + k_{n-1} 2^{n-3} + \cdots + k_2) + k_1 \tag{1-6}$$

由式（1-6）不难看出，若将 $(N)_{10}$ 除以 2 所得的商再次除以 2，则所得余数为 k_1。

依此类推，反复将每次得到的商再除以 2，就可求得二进制数的每一位了。

例如，将 $(67)_{10}$ 化为二进制数可用短除式进行，步骤如下：

	余数	k_i
2 | 67		
2 | 33	……1	…… $k_0=1$
2 | 16	……1	…… $k_1=1$
2 | 8	……0	…… $k_2=0$
2 | 4	……0	…… $k_3=0$
2 | 2	……0	…… $k_4=0$
2 | 1	……0	…… $k_5=0$
0	……1	…… $k_6=1$

倒取余数可得

$$(67)_{10} = (1000011)_2$$

2）小数部分的转换

若 $(N)_{10}$ 是一个十进制的小数，对应的二进制小数为 $(0.k_{-1}k_{-2}\cdots k_{-m})_2$，则据式（1-3）可知

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

将上式两边同乘以 2 得到

$$2(N)_{10} = k_{-1} + (k_{-2}2^{-1} + k_{-3}2^{-2} + \cdots + k_{-m}2^{-m+1}) \tag{1-7}$$

式（1-7）说明，将小数 $(N)_{10}$ 乘以 2 所得乘积的整数部分即 k_{-1}。

同理，将乘积的小数部分再乘以 2 又可得到

$$2(k_{-2}2^{-1} + k_{-3}2^{-2} + \cdots + k_{-m}2^{-m+1}) = k_{-2} + (k_{-3}2^{-1} + \cdots + k_{-m}2^{-m+2}) \tag{1-8}$$

即乘积的整数部分就是 k_{-2}。

依次类推，将每次乘 2 后所得乘积的小数部分再乘以 2，便可求出二进制小数的每一位。

例如，将 $(0.8125)_{10}$ 化为二进制小数时可连续乘以 2，步骤如下所示：

	整数	k_{-i}
$0.8125 \times 2 = 1.625$	1	$k_{-1}=1$
$0.625 \times 2 = 1.25$	1	$k_{-2}=1$
$0.25 \times 2 = 0.5$	0	$k_{-3}=0$
$0.5 \times 2 = 1.0$	1	$k_{-4}=1$

顺序取整数得

$$(0.8125)_{10} = (0.1101)_2$$

运算时，式中的整数不参加连乘。在十进制的小数部分转换过程中，有时连乘 2 不一定

能使小数部分等于 0，这说明该十进制小数不能用有限位二进制小数表示。这时只要取足够多的位数，使其误差达到所要求的精度就可以了。

例如，将十进制数 0.16 转换成二进制数，精确到小数点后 4 位。

十进制数 0.16 连续四次乘 2 后，其小数部分等于 0.56，仍不为 0。由于要求精确到小数点后 4 位，因此将 0.56 再乘一次 2，小数点后第 5 位为 1，得 $(0.16)_{10}=(0.00101)_2$。

3. 二-十六转换

把二进制数转换成等值的十六进制数称为二-十六转换。

由于 4 位二进制数恰好有 16 个状态，而把这 4 位二进制数看作一个整体时，它的进位输出又正好是逢十六进一，所以整数部分只要从低位到高位（小数部分从高位到低位）将每 4 位二进制数分为一组并代之以等值的十六进制数，即可得到对应的十六进制数。

例如，将 $(1011110.1011001)_2$ 化为十六进制数时可得

$$\begin{array}{cccccc} (& 0101 & 1110. & 1011 & 0010 &)_2 \\ & \downarrow & \downarrow & \downarrow & \downarrow & \\ (& 5 & E. & B & 2 &)_{16} \end{array}$$

如果二进制数整数部分（或小数部分）的位数不是 4 的整数倍，在转换为十六进制数的过程中将每 4 位二进制数分组时，整数部分在最高位添 0（小数部分在最低位添 0），以补足每组 4 位二进制数的要求同时又不改变原二进制数。

4. 十六-二转换

十六-二转换是指把十六进制数转换成等值的二进制数。转换时只需将十六进制数的每一位用等值的 4 位二进制数代替就行了。

例如，将 $(7FB.A6)_{16}$ 化为二进制数时得到

$$\begin{array}{cccccc} (& 7 & F & B. & A & 6)_{16} \\ & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ (0111 & 1111 & 1011. & 1010 & 0110)_2 & \end{array}$$

5. 十六-十转换

在将十六进制数转换为十进制数时，可根据式（1-4）将各位按权展开后相加求得。在将十进制数转换为十六进制数时，可以先转换成二进制数，然后再将得到的二进制数转换为等值的十六进制数。也可采用整数部分连除以 16，倒序取余数；小数部分连乘以 16，顺序取整数的方法。这两种转换方法上面已经讲过了。

1.2 机器数及机器数的加、减运算

1.2.1 机器数

前面讨论的数都没有考虑符号，对于不带符号的数，一般默认为正数。对于带符号的

数，我们称其为符号数。一个符号数由两部分组成：一部分表示数的符号，另一部分表示数的数值。直接用“+”或“-”号表示符号的二进制数，称作符号数的真值。在数字系统中，除数值位用二进制数表示外，符号也必须进行数值化处理。一般用“0”表示符号“+”，用“1”表示符号“-”。将符号数值化后的二进制数，称作机器数。在数字系统中，所有的算术运算都用机器数进行，针对不同的硬件算法，机器数通常又有三种数码形式：原码、反码和补码。

1. 原　码

将真值的符号数值化，数值位保持不变所构成的数码，称作原码。对于正数，符号位记作“0”；对于负数，符号位记作“1”。原码的第一位表示符号位，其余各位表示数值位。

例如，两个带符号的二进制小数分别为 N_1 和 N_2，它们的真值形式为

$$N_1 = +0.1000\,,\quad N_2 = -0.1001$$

假设机器数字长为 5 位，则 N_1 和 N_2 的原码表示形式为

$$[N_1]_{原码} = 0.1000\,,\quad [N_2]_{原码} = 1.1001$$

在原码表示形式中，0 有两种不同的形式，即

$$[+0]_{原码} = 0.0000\,,\quad [-0]_{原码} = 1.0000$$

可以看出，要得到一个带符号二进制数的原码，只要将其符号按“+”→“0”、“-”→“1”的方法将符号位数值化即可。需要说明的是，小数点不占字位，且符号位右边紧跟着数值位的小数部分。

2. 反　码

机器数用反码表示时，左边的第一位也为符号位，符号位为 0 代表正数，符号位为 1 代表负数。对于正数，即符号位为 0 的数，反码的数值位和原码的数值位相同；对于负数，即符号位为 1 的数，反码的符号位仍为 1，反码的数值位是将原码数值位按位求反（原码的某位为 1，反码的相应位就为 0；原码的某位为 0，反码的相应位就为 1）。

还是以上面两个带符号的二进制数 N_1 和 N_2 为例，它们的真值形式为

$$N_1 = +0.1000\,,\quad N_2 = -0.1001$$

假设机器数字长为 5 位，则 N_1 和 N_2 的反码表示形式为

$$[N_1]_{反码} = 0.1000\,,\quad [N_2]_{反码} = 1.0110$$

在反码表示形式中，0 也有两种不同的形式，即

$$[+0]_{反码} = 0.0000\,,\quad [-0]_{反码} = 1.1111$$

显然，反码的数值位的形成和它的符号位有关。

3. 补　码

在补码表示法中，正数的补码和原码及反码的表示相同，而负数的补码却不同。对于负

数，即符号位为 1 的数，其补码的符号位仍为 1，但数值位是将原码的数值位按位求反还要在最末位加上 1。

仍然以上面两个带符号的二进制数 N_1 和 N_2 为例，它们的真值形式为

$$N_1 = +0.1000\,,\quad N_2 = -0.1001$$

假设机器数字长为 5 位，则 N_1 和 N_2 的补码表示形式为

$$[N_1]_{补码} = 0.1000\,,\quad [N_2]_{补码} = 1.0111$$

在补码表示形式中，0 只有一种形式，即

$$[+0]_{补码} = 0.0000\,,\quad [-0]_{补码} = 0.0000$$

由于字长是由系统硬件决定的，所以一个系统的字长是固定的。在求负数的补码过程中，加 1 操作可能会产生进位，由于字长固定这个进位自然丢失。显然，补码的数值位的形成也和它的符号位有关。

1.2.2 机器数的加、减运算

当两个二进制数码表示两个数量大小时，它们之间可以进行数值运算，这种运算称为算术运算。二进制算术运算和十进制算术运算的规则基本相同，唯一的区别在于二进制数是逢二进一而十进制数是逢十进一。上面介绍了带符号数的 3 种表示法，它们的形成规则不同，加、减运算的规律也不相同。

1. 原码运算

原码中的符号位仅用来表示数的正、负，不参加运算，进行运算的只是数值部分。原码运算时，应首先比较两个数的符号，若两数的符号相同，则两数相加就是将两个数的数值相加，结果的符号不变；若两数的符号不同，于是就要进一步比较两数数值的相对大小，然后两数相加是将数值较大的数减去数值较小的数，结果的符号与数值较大的数的符号相同。下面举例说明。

例 1.1 已知 $N_1 = -0.0011$，$N_2 = +0.1011$，求 $[N_1 + N_2]_{原码}$ 和 $[N_1 - N_2]_{原码}$。

解： $[N_1 + N_2]_{原} = [(-0.0011) + (+0.1011)]_{原}$

由于 N_1 和 N_2 的符号不同，并且 N_2 的绝对值大于 N_1 的绝对值，因此，要进行 N_2 减 N_1 的运算，其结果为正。运算结果为原码，即

$$[N_1 + N_2]_{原} = 0.1000$$

故其真值为

$$N_1 + N_2 = +0.1000$$

又

$$[N_1 - N_2]_{原} = [(-0.0011) - (+0.1011)]_{原} = [(-0.0011) + (-0.1011)]_{原}$$

由于 N_1 和 N_2 的符号相同，因此，实际上要进行 N_1 加 N_2 的运算，其结果为负。运算结果为原码，即

$$[N_1-N_2]_{原}=1.1110$$

故其真值为

$$N_1-N_2=-0.1110$$

2. 反码运算

反码运算比原码运算要方便。两数和的反码等于两数的反码之和，两数差的反码可以用两数反码的加法来实现。值得注意的是，反码运算时，符号位和数值位一样参加运算，如果符号位产生了进位，则此进位应加到和数的最低位，称之为“循环进位”。反码加、减运算规则是

$$[N_1+N_2]_{反}=[N_1]_{反}+[N_2]_{反}+符号位产生的进位$$

$$[N_1-N_2]_{反}=[N_1]_{反}+[-N_2]_{反}+符号位产生的进位$$

运算结果符号位为 0 时，说明是正数的反码，与原码相同；运算结果符号位为 1，说明是负数的反码，应对结果再求反码才得原码。下面举例说明。

例 1.2　已知 $N_1=+0.1100$，$N_2=+0.0010$，求 $[N_1+N_2]_{反}$ 和 $[N_1-N_2]_{反}$。

解：（1）求 $[N_1+N_2]_{反}$

$$[N_1]_{反}=0.1100,\quad [N_2]_{反}=0.0010$$

$$\begin{array}{rr} & 0.1100 \\ + & 0.0010 \\ \hline & 0.1110 \end{array}$$

由于符号位运算结果并未产生进位，或者说进位为 0，所以有

$$[N_1+N_2]_{反}=[N_1]_{反}+[N_2]_{反}=0.1100+0.0010=0.1110$$

故其真值为

$$N_1+N_2=+0.1110$$

（2）求 $[N_1-N_2]_{反}$

$$[N_1-N_2]_{反}=[N_1+(-N_2)]_{反}$$

$$[N_1]_{反}=0.1100,\quad [-N_2]_{反}=1.1101$$

$$\begin{array}{rr} & 0.1100 \\ + & 1.1101 \\ \hline & \boxed{1}\,0.1001 \\ + & \quad\longrightarrow 1 \\ \hline & 0.1010 \end{array}$$

由于符号位产生了进位，因此要进行“循环进位”，则

$$[N_1-N_2]_{反}=[N_1]_{反}+[-N_2]_{反}+符号位产生的进位$$
$$=0.1100+1.1101+0.0001=0.1010$$

其真值为

$$N_1-N_2=+0.1010$$

3. 补码运算

补码加、减运算规则为

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

补码的加、减运算规则表明：两数和的补码等于两数的补码之和，而两数差的补码也可以用加法来实现。运算时，符号位和数值位一样参加运算，如果符号位产生进位，则只需将此进位“丢掉”。运算结果的符号位为 0 时，说明是正数的补码；运算结果的符号为 1 时，说明是负数的补码，应对结果再求补码才得原码。下面举例说明。

例 1.3 已知 $N_1=-0.1100$，$N_2=-0.0010$，求 $[N_1+N_2]_{补}$ 和 $[N_1-N_2]_{补}$。

解：（1）求 $[N_1+N_2]_{补}$

$$[N_1]_{补}=1.0100,\quad [N_2]_{补}=1.1110$$

$$\begin{array}{rr} & 1.0100 \\ + & 1.1110 \\ \hline \text{丢掉}\longleftarrow\boxed{1} & 1.0010 \end{array}$$

$$[N_1+N_2]_{补}=[N_1]_{补}+[N_2]_{补}=1.0100+1.1110$$

由于符号位产生了进位，因此，要将此进位丢掉，即

$$[N_1+N_2]_{补}=1.0010$$

运算结果的符号位为 1，说明是负数的补码，应对运算结果再求补码才得到原码，即

$$[N_1+N_2]_{原}=\{[N_1+N_2]_{补}\}_{补}=1.1110$$

故其真值为

$$N_1+N_2=-0.1110$$

（2）求 $[N_1-N_2]_{补}$

$$[N_1-N_2]_{补}=[N_1+(-N_2)]_{补}$$
$$[N_1]_{补}=1.0100,\quad [-N_2]_{补}=0.0010$$

$$\begin{array}{rr} & 1.0100 \\ + & 0.0010 \\ \hline & 1.0110 \end{array}$$

即

$$[N_1 - N_2]_{补} = 1.0110$$

由于运算结果的符号位为 1，是负数的补码，应再求结果的补码才得原码，即

$$[N_1 - N_2]_{原} = \{[N_1 - N_2]_{补}\}_{补} = 1.1010$$

故其真值为

$$N_1 - N_2 = -0.1010$$

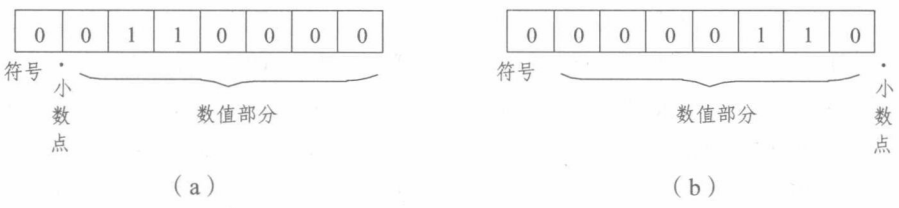
通过上面的讨论可以看出，原码、反码和补码各有优缺点。原码表示法简单直观，但进行加、减运算较复杂。原码减法必须做真正的减法，不能用加法来代替。这样，实现原码运算所需的逻辑电路也较复杂，并增加了机器的运算时间。由于用反码和补码进行加、减运算时，减法运算也是用加法完成，所以反码和补码表示法可以使加、减运算变得简单，且容易用逻辑电路实现。但用反码进行减法运算时，若符号位产生进位就需进行两次算术相加。用补码进行减法运算较方便，因为它只需进行一次算术相加。因此，在近代计算机中，加、减法几乎都采用补码运算。此外，我们也不难发现：乘法运算可以用加法和移位两种操作实现，而除法运算可以用减法和移位操作实现。因此，二进制数的加、减、乘、除运算都可以用加法运算电路完成，这就大大简化了运算电路的结构。

1.3　数的定点表示和浮点表示

在计算机中，处理小数点的方法有两种：一种是用数定点表示，另一种是用数浮点表示。数的这两种表示方法，在计算机中都得到了应用。并且通常把使用定点数的计算机称为定点机，把使用浮点数的计算机称为浮点机。

1.3.1　数的定点表示

所谓定点表示是指数中的小数点位置固定不变。一般说来，小数点可固定在任何位置，但通常都是把小数点固定在数值部分的最高位之前或固定在最低位之后。前者将数表示成纯小数，后者将数表示成整数。在机器中，小数点实际上是不表示出来的，而是在一个约定的位置。在定点机中，对于 n 位定点数，符号位占一位，总字长为 $n+1$ 位，则该定点机为 $n+1$ 位机。例如 $N_1 = +0.0110$ 是小数，在 8 位定点小数机中的表示如图 1.1（a）所示。$N_2 = +0110$ 是整数，在 8 位定点整数机中的表示如图 1.1（b）所示。



（a）　（b）

图 1.1　定点机中数的表示形式

当定点数的小数点位置确定后，它的数域也就随之而定了。

对于 n 位的定点小数 N，它的数域是

$$2^{-n} \leqslant |N| \leqslant 1-2^{-n}$$

在定点机中，一切参加运算的数及最后的结果，都不能超出数域范围，否则就会出现错误的结果。如果运算数小于 2^{-n}，计算机作为 0 处理；如果运算数大于 $1-2^{-n}$，计算机作为“溢出”处理，迫使计算机停止运行或转入出错处理程序。

对于 n 位的定点整数 N，它的数域是

$$0 \leqslant |N| \leqslant 2^{n}-1$$

当需要表示的数的绝对值在较小数域范围内时，数的定点表示是可行的。然而，某些数值可能需要更多位数来表示，在这种情况下，通常采用浮点表示法。浮点表示法可用较少的位数表达较大的数值域内的数值。

1.3.2 数的浮点表示

浮点表示是指数中小数点的位置不固定，或者说是浮动的。浮点数的一般表示形式为

$$N = 2^{J} \times S$$

式中，S 为数 N 的尾数，J 表示数 N 的阶码，2 为阶码的基数。

浮点数由两部分组成：第一部分是指数部分表示小数点浮动的位置；第二部分是尾数部分，表示数的符号和有效数位。

在计算机中，小数点的移动是通过尾数的移动及阶码的相应变化来实现的。

在浮点机中，数的表示形式如图 1.2 所示。

图 1.2 浮点机中数的表示形式

在计算机运算过程中，为了提高运算精度，避免有效数字丢失，二进制浮点数可以表示成规格化的形式。所谓浮点数的规格化，是使尾数最高位为 1，也就是使尾数满足 $1/2 \leqslant |S| < 1$。

例如，二进制数 1010，若表示成 $2^{+100} \times 0.1010$，则是规格化的数；若表示成 $2^{+101} \times 0.0101$ 就是非规格化的数。计算机对浮点数进行规格化操作，是通过对尾数移位，同时对阶码作相应变化来实现的。

数的定点表示和浮点表示各有其优缺点，而且这两种表示方式在不同场合都得到了应用。在位数相同的情况下，浮点数的表示范围比定点数的表示范围大。浮点数的运算精度一般比定点数高，所以，浮点计算机的使用要比定点计算机方便。但浮点运算规则比定点运算规则复杂，浮点运算需要考虑阶码和尾数两部分的运算。当两个浮点数相加时，首先要使两数的阶码相等，然后尾数相加；当两个浮点数相乘时，阶码相加，尾数相乘。因而，实现浮点数运算的逻辑电路和控制也比较复杂。因此，一般大、中型计算机采用浮点表示，而小型和专用计算机多采用定点表示。

1.4　码　制

前面我们讨论的是数，数有正负和大小之分。例如，可比较两个数的大小，可对数进行加、减等算术运算。下面讨论码。在数字系统中，码的形式和机器数相同，也是由若干位二进制数组成，所以也称作数码。但两者意义完全不同，一般来说，码已没有表示数量大小的含义，只是表示不同事物的代号而已。例如可以用数码代表某个数字、某个字符或者某件事情等信息，当然也可以代表某个数。不同的数码不仅可以表示数量的不同大小，而且能用来表示不同的事物。在后一种情况下，这些数码称为代码。

例如在举行长跑比赛时，为便于识别运动员，通常给每个运动员编一个号码。显然，这些号码仅仅表示不同的运动员，已失去了数量大小的含义。

为便于记忆和处理，在编制代码时总要遵循一定的规则，这些规则就叫作码制。

1.4.1　十进制数的二进制编码

一位十进制数的二进制编码简称为二—十进制码或 BCD 码，所谓 BCD 码是指用若干位二进制数来表示一位十进制数。这种编码即具有二进制数的形式，又具有十进制数的特点，所以也称作数码。

十进制数有 0～9 共 10 个数字，所以表示 1 位十进制数，至少需要 4 位二进制教。但应指出的是，4 位二进制数可以产生 $2^4=16$ 种组合，用 4 位二进制数表示 1 位十进制数，有 6 种组合是多余的。因而，十进制数的二进制编码可以有许多种方法，即有许多种不同的编码方案，每种编码都有它的特点。表 1.1 列举了目前常用的几种编码方案。

表 1.1　十进制数字的二进制代码

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

下面分别介绍几种常用编码。