第 1 章　数字系统与二进制数

1.1　数字系统

人类已经进入到数字时代，数字系统在我们日常生活中起着越来越重要的作用，并广泛应用于通信、商贸、交通控制、航天器制导、医疗、天气监测、因特网等领域以及其他许多商业、工业和科研部门，人们从而拥有了数字电话、数字电视、数字通用光盘、数字相机、手持(便携式)设备，当然也包括数字计算机等。有些人喜欢将音乐下载到便携式媒体播放器(例如，iPod Touch™)和其他高分辨率显示的手持设备中。这些设备具有图形用户接口(GUI，Graphical User Interfaces)。通过接口让设备执行命令，这种方式对用户来说简便易用，但实际上这些命令涉及一系列复杂内部指令的精确执行问题。大多数这样的设备内部均嵌入了特殊用途的数字计算机。数字计算机最具挑战的特性是其通用性，它可以执行一系列的指令(也称为程序)，对给定数据进行操作和处理。用户可以根据特定的要求对程序或数据进行修改。正因为有这种灵活性，通用数字计算机才可以完成各种各样的信息处理任务，从而得到非常广泛的应用。

数字系统的另一个特性是它具有描述和处理离散信息的能力。我们知道，任何一个取值数目有限的元素集都包含着离散信息，如十进制数的各个数、字母表中的 26 个字母、扑克牌中的 52 张牌以及国际象棋盘中的 64 个方格等。早期的数字计算机主要用于数值计算，它处理的离散信息是各种各样的数字，因此就出现了“数字计算机”这个术语。数字系统中的离散信息元素可以用一类称为“信号”的物理量表示，而最常见的信号就是电压和电流，它们一般由晶体管构成的电路产生。目前，在各种数字电子系统中的电信号只有两个离散值，因而也被称为二进制。一个二进制数又称为一个比特(bit)，它有两个基本的数值：0 和 1。离散信息单元可以用一组比特表示，称为二进制码。例如，在数字系统中，十进制数 0 到 9 可以用一个 4 位码组表示(例如，0111 表示十进制数 7)。一组码字对应的数值取决于它所在的编码系统。为了便于说明，我们将二进制系统中的 0111 写成$(0111)_2$，十进制系统中的 0111 写成$(0111)_{10}$，显然$0111_2 = 7_{10}$，它并不等于0111_{10}或一百一十一。这里，下标只是表示码字的进制数。通过使用多种技术，用一组比特表示各种离散符号(不仅是数字)，从而可以用数字的方式研究系统。因此，数字系统就是处理二进制离散信息单元的系统。在当今的技术领域中，正如我们所看到的，二进制系统是最实用的，它们可以采用电子元器件来实现。

离散信息量或者来源于被处理数据的本质，或者可能来自连续过程的量化。比如，工资表就是一个自然的离散信息处理过程，它包含了雇员姓名、社会保险号、周薪和所得税等。员工工资单可以用字母(姓名)、数字(薪水)以及一些特殊符号(如$)等离散数值来处理。再比如，进行研究的科学家在观察连续过程时，一般都是以表格形式记录特定的数值。科学家就是这样对连续数据进行量化的，并将表中的每一个数赋予离散量。在很多情况下，量化处理可以由模数转换器自动完成，通过模数转换器将模拟(连续)量转换为数字(离散)量。

通用数字计算机就是最典型的数字系统，其主要组成部分是存储器单元、中央处理单元

以及输入/输出单元。存储器单元用于存储程序、输入/输出数据以及中间数据。中央处理单元依照特定的程序执行算术运算和其他数据处理的操作。用户通过键盘这种输入设备将程序和数据输送到存储器中。打印机这种输出设备主要用于接收计算结果,并把结果打印出来提供给用户。数字计算机可以有多个输入/输出设备。通信单元是非常有用的设备,它可以通过因特网与其他用户实现互联。数字计算机功能非常强大,不仅能执行算术运算,也能执行逻辑操作。另外,用户还可以根据内部和外部条件,采用编程方式做出决策。

商用产品采用数字电路实现有其根本原因。与数字计算机一样,大多数数字设备都是可编程的。通过改变可编程设备中的程序,相同的硬件条件可以实现多种不同的用途。如此低廉的开发成本使客户群越来越广泛。随着数字集成电路技术的进步,数字设备成本得以大幅度下降。因为单个硅片上集成的晶体管数目不断增加,数字电路的功能越来越复杂,每片的成本不断下降,价格越来越便宜。由数字集成电路构成的设备每秒钟可以进行数百万次操作。通过纠错编码,数字系统的工作非常可靠。一个典型例子就是数字通用光碟(DVD, Digital Versatile Disk),它可以将表示视频、音频的数字信息以及其他的一些数据毫无损失地记录并保存下来。DVD 中以此方式记录的数字信息在播放前由数字采样来检查码元,任何错误都会被自动检测出来并得到纠正。

数字系统由数字模块构成。为便于理解每个数字模块的功能,有必要介绍数字电路与逻辑功能的基础知识。本书前 7 章主要介绍数字设计的基本工具,如逻辑门结构、组合和时序电路以及可编程逻辑器件等知识;第 8 章介绍如何使用现代硬件描述语言(HDL, Hardware Description Language)在寄存器传输级(RTL, Register Transfer Level)描述数字设计;第 9 章是使用数字电路进行的实验。

数字设计方法发展的主要趋势是采用硬件描述语言(HDL)描述和模拟数字电路的功能。HDL 类似于一种编程语言,非常适合于以文本形式描述数字电路。利用 HDL 可以在硬件电路建立之前模拟和验证数字系统的功能。HDL 也可以和逻辑综合工具一起,用于数字系统的自动设计过程。因此,熟悉和掌握基于 HDL 的设计方法对于学生而言是非常重要的。数字电路的 HDL 描述将贯穿全书,书中这些例子不仅有助于描述 HDL 的特性,也是描述 HDL 工业应用的最好实践。有的情况要提醒读者不能轻视。例如,HDL 模型可以模拟一种现象,但不能被设计工具综合;建立的 HDL 模型将会造成芯片资源的浪费,综合到硬件后不能正确工作等。

如前所述,数字系统处理二进制形式表示的离散信息值。用来计算的操作数可以表示成二进制数的形式。其他离散元素,包括十进制数和字母表中的字母,也可以用二进制码来表示。数字电路(又称为逻辑电路)中的数据处理主要通过二进制逻辑单元(逻辑门)来实现,而数字量则存储在二进制(2 个数值)存储单元(触发器)中。本章主要介绍各种二进制概念,为后续章节的进一步学习提供基本参考。

1.2 二进制数

十进制数 7392 代表一个数值,该值等于 7 个千加上 3 个百加上 9 个十加上 2 个一。千、百等分别是 10 的不同幂次,幂次由各个系数所在位置确定。更确切地讲,7392 可以写成

$$7 \times 10^3 + 3 \times 10^2 + 9 \times 10^1 + 2 \times 10^0$$

然而，按照惯例一般只写系数，幂次从右到左递增，系数的位置决定对应 10 的幂次。总之，带小数点的十进制数可以表示成一串系数形式：

$$a_5a_4a_3a_2a_1a_0.\ a_{-1}a_{-2}a_{-3}$$

系数 a_j 是十个数字(0，1，2，…，9) 中的某个数，下标值 j 为对应的幂次，系数必须要和 10 的幂次相乘。因此，上式可以展开成：

$$10^5a_5 + 10^4a_4 + 10^3a_3 + 10^2a_2 + 10^1a_1 + 10^0a_0 + 10^{-1}a_{-1} + 10^{-2}a_{-2} + 10^{-3}a_{-3}$$

这里，$a_3 = 7$，$a_2 = 3$，$a_1 = 9$，$a_0 = 2$。

由于只使用 10 个数字，每个系数均要与 10 的幂次相乘。因此，十进制的基数(radix 或 base)为 10。二进制是与十进制不同的数制，其系数只有两种取值：0 或 1，每个系数 a_j 都要乘以基数的幂 2^j，结果相加后就可以得到等效的十进制数。小数点(如十进制小数点)用来区分 10 的正幂次和 10 的负幂次。例如，与二进制数 11010.11 相对应的十进制数是 26.75，系数和 2 的幂次相乘后再展开，结果如下：

$$1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2} = 26.75$$

数字系统的种类很多，由此可以推广到以 r 为基数的任何进制，即：

$$a_n \cdot r^n + a_{n-1} \cdot r^{n-1} + \cdots + a_2 \cdot r^2 + a_1 \cdot r + a_0 + a_{-1} \cdot r^{-1} + a_{-2} \cdot r^{-2} + \cdots + a_{-m} \cdot r^{-m}$$

系数 a_j 的取值范围是 0 到 $r-1$。为区分不同进制数，一般将按位置记数法所表示的数用括号括起来，并在其右下角标注该进制的下标(十进制数在明显可以看出其进制时可省略下标)。例如某个五进制数为：

$$(4021.2)_5 = 4 \times 5^3 + 0 \times 5^2 + 2 \times 5^1 + 1 \times 5^0 + 2 \times 5^{-1} = (511.4)_{10}$$

由于基数为 5，各系数的取值只可能为 0、1、2、3、4。八进制数的基数为 8，有八个数字符号 0、1、2、3、4、5、6、7。如某个八进制数为 127.4，为了得到与其相等的十进制数，将其按 8 的幂次展开：

$$(127.4)_8 = 1 \times 8^2 + 2 \times 8^1 + 7 \times 8^0 + 4 \times 8^{-1} = (87.5)_{10}$$

注意：8 和 9 这两个数字不能出现在八进制中。

当基数小于 10 时，通常是从十进制系统中借用需要的 r 个数字当系数。而当基数大于 10 时，就用字母表中的字母作为 10 个十进制数的补充。例如，在十六进制(基数为 16)系统中，前十个数字(0～9)来自于十进制系统，而字母 A、B、C、D、E 和 F 分别用来表示数 10、11、12、13、14 和 15 这六个数字。例如，某个十六进制数为：

$$(B65F)_{16} = 11 \times 16^3 + 6 \times 16^2 + 5 \times 16^1 + 15 \times 16^0 = (46\,687)_{10}$$

十六进制系统常被开发人员用来表示数字系统中长比特串的地址、指令和数据。例如，B65F 表示 1011011001010000。

如前所述，二进制的数字又称为比特。当某个比特等于 0 时，它对转换结果没有影响。因此，将二进制数转换为十进制数，就是把比特为 1 的那些位置所对应的 2 的幂数相加。例如：

$$(110101)_2 = 32 + 16 + 4 + 1 = (53)_{10}$$

该二进制数中有四个 1，其对应的十进制数是四个以 2 为底的幂次之和。表 1.1 中列出了 2 的 n 幂次表示的 24 个数。在计算机系统中，2^{10} 用 K(kilo，千)来表示，2^{20} 用 M(mega，兆)表

示，2^{30} 用 G(giga，吉或千兆)表示，2^{40} 用 T(tera，梯或万兆)表示。因此，$4K = 2^{12} = 4096$，$16M = 2^{24} = 16\ 777\ 216$。计算机的存储容量通常用字节数来表示。一个字节等于八比特，可以表示键盘上的一个字符。4G 硬盘能够容纳 $4G = 2^{32}$ 字节的数据(大约 40 亿字节)。一个 T 等于 1024 个 G，大约一万亿字节。

表 1.1 2 的幂次

n	2^n	n	2^n	n	2^n
0	1	8	256	16	65 536
1	2	9	512	17	131 072
2	4	10	1024(1K)	18	262 144
3	8	11	2048	19	524 288
4	16	12	4096(4K)	20	1 048 576(1M)
5	32	13	8192	21	2 097 152
6	64	14	16 384	22	4 194 304
7	128	15	32 768	23	8 388 608

r 进制数的算术运算规则与十进制数相同。当采用非十进制数时，使用 r 个允许的数字要非常小心。对两个二进制数进行加法、减法和乘法的例子如下：

```
被加数：  101101      被减数：  101101      被乘数：   1011
被加数：  +100111     减数：    −100111     乘数：   × 101
和：      1010100     差：      000110              1011
                                              0000
                                部分积：      1011
                                积：        110111
```

两个二进制数求和的计算法则与十进制数相同，只是和数中任一有效位置上的数字只可以为 0 或 1，给定有效位上的任一进位可以被更高一级有效位上的一对数字使用。减法要稍微复杂一点，但法则仍然和十进制数相同，只是给定有效位上的借位相当于给被减数加 2(十进制中的借位相当于给被减数加 10)。乘法非常简单，乘数不是 1 就是 0，因此，部分积要么等于移位后的被乘数，要么等于 0。

1.3 数制的转换

不同基数的数如果对应相同的十进制数，则认为它们是相等的。例如，$(0011)_8$ 和 $(1001)_2$ 均等于十进制数 9。基数为 r 的数转换为十进制数比较简单，通常是将该数按幂次展开再求和，这一点前面已经做了介绍。现在，我们来介绍将十进制数转换为 r 进制数的一般操作步骤。如果这个数是带小数点的，那么必须将整数部分和小数部分分别进行转换，然后将它们的转换结果合并起来。整数部分的转换方法是：把待转换的十进制整数除以 r，取其余数，所得之商再除以 r，再取其余数，如此反复。下面的例子可以很好地说明这个步骤。

例 1.1 将十进制数 41 转换为二进制数。

首先，把 41 除以 2 得到整数商 20 和余数 1，将这个商再继续除以 2，得到新的商和余数，如此重复，直到最后的整数商变为 0，从余数中可以得到想要的二进制数的系数，具体步骤如下：

	整数商		余数	系数
41/2 =	20	+	1	$a_0 = 1$
20/2 =	10	+	0	$a_1 = 0$
10/2 =	5	+	0	$a_2 = 0$
5/2 =	2	+	1	$a_3 = 1$
2/2 =	1	+	0	$a_4 = 0$
1/2 =	0	+	1	$a_5 = 1$

因此，最后结果为：$(41)_{10} = (a_5a_4a_3a_2a_1a_0)_2 = (101001)_2$

其算术过程可以更方便地表述如下：

整数	余数
41	
20	1
10	0
5	0
2	1
1	0
0	1　101001 = 结果

从十进制数到任意 r 进制数的转换与上例类似，只需将除数 2 用 r 来代替就可以了。

例 1.2　将十进制数 153 转换为八进制数。

八进制数的基数 r 为 8。首先，将 153 除以 8 得到整数商 19 和余数 1；然后，将 19 再除以 8 得到整数商 2 和余数 3；最后，将 2 除以 8 得到的商为 0，余数为 2。操作过程如下：

153	
19	1
2	3
0	$2 = (231)_8$

十进制小数转换为二进制数的方法与整数部分的转换是类似的。不过，在进行转换时用的是乘法而不是除法，每次取的是整数部分而不是余数。具体方法最好还是通过下面的实例来说明。

例 1.3　将 $(0.6875)_{10}$ 转换为二进制数。

首先，将 0.6875 乘以 2 得到一个整数和一个小数，这个新的小数再乘以 2 又得到一个新的整数和一个新的小数，如此重复这个过程，直到小数部分变为 0 或者数值达到了足够的精度。最终的二进制数系数可以由这些整数得出：

	整数		小数	系数
$0.6875 \times 2 =$	1	+	0.3750	$a_{-1} = 1$
$0.3750 \times 2 =$	0	+	0.7500	$a_{-2} = 0$
$0.7500 \times 2 =$	1	+	0.5000	$a_{-3} = 1$
$0.5000 \times 2 =$	1	+	0.0000	$a_{-4} = 1$

因此，答案是：$(0.6875)_{10} = (0.a_{-1}a_{-2}a_{-3}a_{-4})_2 = (0.1011)_2$

将一个十进制小数转换为 r 进制数的方法与上面的方法类似，乘数 2 用 r 来代替，并且整数中的系数范围值由 0 和 1 变为 0 到 $r-1$。

例 1.4 将 $(0.513)_{10}$ 转换为八进制数。

$$0.513 \times 8 = 4.104$$
$$0.104 \times 8 = 0.832$$
$$0.832 \times 8 = 6.656$$
$$0.656 \times 8 = 5.248$$
$$0.248 \times 8 = 1.984$$
$$0.984 \times 8 = 7.872$$

转换后的 7 个有效数字可从上面这些乘积的整数部分得到:

$$(0.513)_{10} = (0.406517\cdots)_8$$

既有整数部分也有小数部分的十进制数的转换方法是先将整数和小数分别进行转换,再把两部分的转换结果合并在一起,利用例 1.1 和例 1.3 的结果可得:

$$(41.6875)_{10} = (101001.1011)_2$$

根据例 1.2 和例 1.4 的结果有:

$$(153.513)_{10} = (231.406517)_8$$

1.4 八进制和十六进制数

十进制、二进制和十六进制数之间的转换在数字计算机中有着十分重要的作用,因为较短的十六进制字符比长串的 1 和 0 更容易识别。由于 $2^3 = 8$, $2^4 = 16$, 所以,每个八进制数对应三位二进制数,而每个十六进制数则对应四位二进制数。十进制、二进制、八进制和十六进制中的前十六个数的表示见表 1.2。

表 1.2 不同进制的数

十进制数(基为 10)	二进制数(基为 2)	八进制数(基为 8)	十六进制数(基为 16)
00	0000	00	0
01	0001	01	1
02	0010	02	2
03	0011	03	3
04	0100	04	4
05	0101	05	5
06	0110	06	6
07	0111	07	7
08	1000	10	8
09	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

从二进制数到八进制数的转换实现起来比较简单，只要将每三位二进制数分为一组，从小数点开始，分别向左、向右进行，然后给每一组分配相应的八进制数。下面的例子描述了这个转换方法：

$$\begin{array}{cccccccccc}(10 & 110 & 001 & 101 & 011 & \cdot & 111 & 100 & 000 & 110)_2 = (26153.7406)_8\\ 2 & 6 & 1 & 5 & 3 & & 7 & 4 & 0 & 6\end{array}$$

二进制数到十六进制数的转换与此类似，不同的只是将每四位二进制数分成一组：

$$\begin{array}{cccccccc}(10 & 1100 & 0110 & 1011 & \cdot & 1111 & 0010)_2 = (2C6B.F2)_{16}\\ 2 & C & 6 & B & & F & 2\end{array}$$

研究表 1.2 中所列的数值，应该很容易记住每组二进制数对应的十六进制(或八进制)数。

八进制数或十六进制数转换成二进制数的步骤与上面介绍的方法刚好相反。每个八进制数转换成相等的三位二进制数，每个十六进制数转换成相等的四位二进制数。可以通过下面的例子来说明：

$$\begin{array}{cccccccc}(673.124)_8 = (110 & 111 & 011 & \cdot & 001 & 010 & 100)_2\\ 6 & 7 & 3 & & 1 & 2 & 4\end{array}$$

和

$$\begin{array}{cccccc}(306.D)_{16} = (0011 & 0000 & 0110 & \cdot & 1101)_2\\ 3 & 0 & 6 & & D\end{array}$$

二进制数使用起来不太方便，与十进制数相比，二进制数的有效数字是十进制数的 3 ~ 4 倍。例如，二进制数 111111111111 等于十进制数 4095。不过，数字计算机用的却是二进制数，且有时候操作员或用户需要用二进制数直接和机器进行通信。尽管计算机里配置的是二进制系统，但人们考虑降低数字的个数时，可以尽量使用八进制和十六进制，需要和机器直接进行通信时进行相应的二进制转换。如二进制数 111111111111 有 12 个数字，表达成八进制数是 7777(四个数字)，表达成十六进制数是 FFF(三个数字)。人与人之间的通信(计算机中的二进制数)用八进制或十六进制表示更好一些，表达起来更简洁，相同情况下只有二进制个数的三分之一到四分之一。因此，大多数计算机手册一般用八进制数或十六进制数来说明二进制数。尽管在八进制和十六进制之间可以任意选择，但一般选择的是十六进制数，用两个十六进制数就可以表示一个字节。

1.5　补码

补码在数字计算机中用于简化减法运算和逻辑操作，从而使电路更加简单、价格更加便宜。每个 r 进制系统都有两种类型的补码：补码(radix complement)和反码(diminished radix complement)。第一种可看作 r 的补码，第二种为$(r-1)$的补码。这里讨论二进制数的 2 的补码和 1 的补码以及十进制数的 10 的补码和 9 的补码。

反码

给定一个有 n 个数字的 r 进制数 N，其$(r-1)$的补码定义为$(r^n-1)-N$，即反码。对于十进制数来说，$r=10$ 和 $r-1=9$，所以 N 的反码(9 的补码)是$(10^n-1)-N$。在这个例子中，10^n代表一个由 1 后面加上 n 个 0 组成的数。10^n-1 是一个由 n 个 9 表示的数。例如，如果 $n=4$，我们有 $10^4=10\,000$ 和 $10^4-1=9999$。因此，十进制数的反码是用 9 减去其每一个

数字。如:

546700 的反码为 999999 – 546700 = 453299

012398 的反码为 999999 – 012398 = 987601

对于二进制数, $r=2$ 和 $r-1=1$, 所以 N 的反码是 $(2^n-1)-N$。同样, 2^n 是一个由 1 后面加上 n 个 0 组成的二进制数, 2^n-1 是一个用 n 个 1 表示的二进制数。例如, 如果 $n=4$, 有 $2^4=(10000)_2$ 和 $2^4-1=(1111)_2$。因此, 二进制数的反码是用 1 减去其每个数字来得到的。然而, 当用 1 去减二进制数时, 有 $1-0=1$ 或 $1-1=0$, 从而将一位从 0 变为 1 或者从 1 变成 0。因此, 二进制数的反码在形式上就是将 1 改为 0 或将 0 改为 1。如:

1011000 的反码是 0100111

0101101 的反码是 1010010

同样, 八进制或者十六进制数 $(r-1)$ 的反码是用 7 或 F(十六进制数 15)分别减去该数每一位数字来得到的。

补码

一个有 n 个数字的 r 进制数 N, 其 r 的补码定义为 r^n-N(分 $N\neq 0$ 和 $N=0$ 两种情况)。与前面的反码相比, r 的补码实质就是反码加 1, 即 $r^n-N=[(r^n-1)-N]+1$。因此, 十进制数 2389 的补码是 7610 + 1 = 7611, 就是由其反码的值加上 1 得来的。二进制数 101100 的补码是 010011 + 1 = 010100, 就是将它的反码值加 1 得到的。

10^n 是用一个 1 后面加上 n 个 0 表示的数。因此, 十进制数 N 的补码 10^n-N 也可以这样来构成, 保留最低有效位上的 0 不变, 用 10 减去第一个非零有效数字, 再用 9 减去所有更高位上的有效数字。如:

十进制数 012398 的补码是 987602

和

十进制数 246700 的补码是 753300

第一个十进制数的补码 2 是将最低有效位上的 8 被 10 减得到的, 再用 9 减去所有其他数字。第二个十进制数补码的获得方法是保持两个最低有效位的 0 不变, 用 10 减去 7, 再用 9 减去其他三个数字。

类似地, 二进制数的补码是不改变两个低位的 0 和第一个 1, 再将其他更高位的有效数字用相反的数字代替, 即用 0 代替 1, 用 1 代替 0。因此

二进制数 1101100 的补码是 0010100

和

二进制数 0110111 的补码是 1001001

第一个数的补码是通过不改变低两位的 0 以及第一个 1, 而将其他四个高位有效数字取反, 即用 0 代替 1, 用 1 代替 0。第二个数的补码是不改变最低有效位的 1, 而将所有其他位取反。

在前面的定义中, 都是假设那些数没有小数点。如果原始数 N 包含小数点, 应该将小数点临时去掉以计算 r 补码或 $(r-1)$ 反码, 然后再在相同的位置上将小数点恢复。同样值得强调的是, 补码的补码又重新等于它的原码, N 的补码是 r^n-N, 其补码的补码是 $r^n-(r^n-N)=N$, 刚好等于原码。

补码的减法

利用借位概念的直接减法在小学就学过了。这种方法是当被减数比减数小时就从高位借 1。人们用纸和笔进行减法运算时，这种方法确实很好，而要用硬件实现减法运算时，这种方法的效率就不如使用补码的方法了。

两个有 n 个数字的 r 进制无符号数做减法 $M-N$ 可以按以下步骤来进行：

1. 将被减数 M 加上减数 N 的补码，即 $M+(r^n-N)=M-N+r^n$。
2. 如果 $M \geqslant N$，结果的和将产生可以被丢弃的进位 r^n，剩下的就是 $M-N$ 的结果。
3. 如果 $M<N$，那么和就不会产生进位，等于 $r^n-(N-M)$，也就是 $(N-M)$ 的补码。常见的计算结果是取和的补码并在其前面加上一个负号。

下面的例子说明了这个步骤：

例 1.5　使用补码，实现十进制数减法 72532 − 3250。

$$
\begin{aligned}
M &= \quad 72532\\
N\text{的补码} &= +\ \underline{96750}\\
\text{和} &= \quad 169282\\
\text{丢弃最后进位 } 10^5 &= -\ \underline{100000}\\
\text{最终结果} &= \quad 69282
\end{aligned}
$$

注意 M 是 5 位数，而 N 是 4 位数，进行减法的两个数必须具有相同的位数。因此，我们把 N 写成 03250，其补码将在最高有效位上产生一个 9，出现的最后进位就表示 $M \geqslant N$，结果为正数。

例 1.6　使用补码，实现十进制数减法 3250 − 72532。

$$
\begin{aligned}
M &= \quad 03250\\
N\text{的补码} &= +\ \underline{27468}\\
\text{和} &= \quad 30718
\end{aligned}
$$

这里没有最后进位。因此，答案为 −(30718 的补码) = −69282。

注意，因为 3250 < 72532，其结果为负数。由于处理的是无符号数，这种情况下确实无法得到一个无符号结果。当利用补码进行减法时，负的结果一般通过最后进位的出现和补码的结果来识别。当用纸笔进行计算时，可以将结果转换成一个带符号的负数，就是计算结果。

利用补码进行二进制数的相减，其方法和步骤与前面相同。

例 1.7　给定两个二进制数 $X=1010100$ 和 $Y=1000011$，用补码来实现：(a) $X-Y$，(b) $Y-X$。

(a)

$$
\begin{aligned}
X &= \quad 1010100\\
Y\text{的补码} &= +\ \underline{0111101}\\
\text{和} &= \quad 10010001\\
\text{抛弃末端进位 } 2^7 &= -\ \underline{10000000}\\
\text{结果：} X-Y &= \quad 0010001
\end{aligned}
$$

(b)

$$
\begin{aligned}
Y &= \quad 1000011\\
X\text{的补码} &= +\ \underline{0101100}\\
\text{和} &= \quad 1101111
\end{aligned}
$$

没有最后进位。

因此,答案是:$X-Y=-$(1101111 的补码) $=-0010001$

无符号数的减法也可以利用$(r-1)$的反码进行。要知道$(r-1)$的反码只比补码小 1,因此,将被减数和减数的补码相加的结果产生一个和数。当有最后进位时,该和数比正确结果少 1。丢弃该最后进位,并将和数加 1,被称为循环最后进位(end-around carry)。

例 1.8 用反码重做例 1.7。

(a) $X-Y=1010100-1000011$

$$\begin{aligned} X &= 1010100 \\ Y\text{的反码} &= +\ 0111100 \\ \text{和} &= 10010000 \\ \text{循环最后进位} &= +\ \quad\ 1 \\ \text{结果:}X-Y &= 0010001 \end{aligned}$$

(b) $Y-X=1000011-1010100$

$$\begin{aligned} Y &= 1000011 \\ X\text{的反码} &= +\ 0101011 \\ \text{和} &= 1101110 \end{aligned}$$

没有最后进位。因此,答案是:$Y-X=-$(1101110 的反码) $=-0010001$。

注意,此负数是通过求反码得到的,这也是反码的一种应用。具有循环最后进位方法的步骤同样适用于十进制无符号数的减法。

1.6 带符号的二进制数

正整数(包括零)可以表示成无符号数。然而,为了表示负整数,需要表示符号。在普通算术里,一般用“-”号表示负数,用“+”号表示正数。由于硬件限制,计算机必须用二进制数表示任何事物。通常,二进制数最左边的那个位表示其符号,按惯例 0 表示正号,1 表示负号。

要注意,不论是带符号的二进制数还是无符号的二进制数,在计算机中都是用一组比特来表示的,这一点很重要。究竟是无符号数还是带符号数由用户来确定。若二进制数是带符号的,则其最左边的位表示符号,其余位表示数值;而若二进制数被假定是无符号数,则最左边的位是该数的最高有效位。例如,01001 可以被认为是 9(无符号数)或是 +9(带符号数),最左边的那位为 0。11001 作为无符号数时等于 25,而作为有符号数时是 -9,因为此时其最左边位置上的 1 表示负号,而其余的四位表示二进制数 9。如果预先知道数的类型,一般就不会混淆。

上一个例子中,那些带符号数的表示方式称为“符号-数值”表示法。在该表示法中,数由数值和符号(+或-)或表示符号的位(0 或 1)两部分组成,这就是普通算术中带符号数的表示。当算术运算在计算机中实现时,使用“符号-补码”系统表示负数是比较方便的。在这个系统中,负数用它的补码表示。“符号-数值”系统通过改变数的符号对其取反,而“符号-补码”系统则是通过求补码将其取反。因为正数最左边的位总是以 0 开始,所以负数的补码总以 1 开始。“符号-补码”系统可以采用反码(1 的补码)或补码(2 的补码),但最常用的是补码。

例如，我们用八位表示二进制数 9。+9 表示为 00001001，即最左边是符号位 0，跟在后面的是等于 9 的二进制序列。注意，所有八位都必须有一个值。因此，在符号位和第一个 1 之间要插入一些 0。尽管只有一种方法来表示 +9，而用八位表示 −9 却有三种不同的表示方法：

符号 − 数值表示法：　10001001
符号 − 反码表示法：　11110110
符号 − 补码表示法：　11110111

在“符号 − 数值”表示法中，−9 是通过把 +9 最左边位的 0 改为 1 得到的。在“符号 − 反码”表示法中，−9 是通过把 +9 的所有位取反得到的，包括符号位。“符号 − 补码”表示的 −9 是通过取 +9 的二进制补码得到的，也包括符号位。

表 1.3 列出了所有可能的四位带符号二进制数的三种表示法，与此相对应的十进制数也列在其中作为参考。注意，三种表示法中所有正数是完全相同的，且最左边位（最高有效位）为 0。“符号 − 补码”表示法中 0 只有一种表示，且为正数，而其他两种表示法中既有正 0 也有负 0，这在普通算术中是不会遇到的。所有负数的最高有效位都是 1，据此我们将其与正数区分开来。四位可以表示 16 个二进制数，在“符号 − 数值”和“符号 − 反码”表示中，有八个正数和八个负数，包括两个零。在“符号 − 补码”表示中，有八个正数（其中包括一个零）和八个负数。

表 1.3　带符号的二进制数

十 进 制 数	符号 − 补码	符号 − 反码	符号 − 数值
+7	0111	0111	0111
+6	0110	0110	0110
+5	0101	0101	0101
+4	0100	0100	0100
+3	0011	0011	0011
+2	0010	0010	0010
+1	0001	0001	0001
+0	0000	0000	0000
−0	—	1111	1000
−1	1111	1110	1001
−2	1110	1101	1010
−3	1101	1100	1011
−4	1100	1011	1100
−5	1011	1010	1101
−6	1010	1001	1110
−7	1001	1000	1111
−8	1000	—	—

“符号–数值”表示一般用于普通算术运算中，但要用于计算机运算却很麻烦，要分开处理符号和数值。因此，计算机通常采用的是“符号 − 补码”表示。反码存在着一些难点，在算术运算中很少使用。因为将 0 和 1 互换相当于逻辑求反操作，这一点在逻辑运算中很有用，参见下一章的内容。下面讨论带符号的二进制数时只用“符号 − 补码”来表示负数，相同的步骤可应用于“符号 − 反码”表示法，只要如同处理无符号数一样包含最后进位就行了。

算术加法

两个数的加法在“符号 – 数值”表示中遵循普通算术规则。如果两个数的符号相同，我们可以将两个数值相加，而相加之和的符号保持不变。如果两个数的符号不同，就用绝对值大的数减去绝对值小的数，并将绝对值大的那个数的符号赋给运算结果。例如，(+25) + (–37) = –(37 –25) = –12，就是用绝对值大的数 37 减去绝对值小的数 25，运算结果使用数 37 的符号。上述步骤需要比较符号和绝对值，然后再进行加或减。该步骤同样也适用于“符号 – 数值”法所表示的二进制数。不同的是，对于用“符号 – 补码”表示的二进制数，在进行两个数相加时，并不需要比较符号或相减运算，而只有加法运算。该步骤非常简单，对于二进制数的加法将在下面说明。

两个以“符号 – 补码”形式表示的二进制符号数的加法，就是把这两个数相加，包括它们的符号位。符号位上所产生的进位要丢弃。

数的加法举例如下：

+ 6	00000110	– 6	11111010
+13	00001101	+13	00001101
+19	00010011	+ 7	00000111
+ 6	00000110	– 6	11111010
–13	11110011	–13	11110011
– 7	11111001	–19	11101101

注意，负数必须首先转换为补码形式。如果相加后得到的和是负数，则给出的结果也是补码形式。例如，–7 表示为 11111001，也是 +7 的二进制补码。

上述四个例子的每一个所进行的运算都是包括符号位的加法。符号位上所产生的进位都要被丢弃，最后得到的负数结果自动就是 2 的补码形式。

为了得到正确结果，我们必须保证要有足够的位数表示和数。如果两个 n 位的数相加，得到的结果却是 $n+1$ 位的和数，称为产生了溢出。当人们用纸笔进行加法运算时，溢出并不是问题，因为我们没有对纸的宽度做出限制，我们只要再添一个 0 到正数或再添一个 1 到负数作为它们的最高有效位，从而将其扩展成为 $n+1$ 位，然后再进行相加。溢出在计算机中之所以成为问题，是因为计算机保存数的位数(字长)是有限的，结果哪怕超过限定位数 1 位都不行。

用补码形式表示的负数与“符号 – 数值”法所表示的负数是不同的。为了确定以“符号 – 补码”形式表示的负数的值，有必要将其转换为正数，这样的形式我们更为熟悉。例如，带符号的二进制数 11111001 是个负数，因为其最左边位上是 1。该二进制负数的补码是 00000111，等于二进制数 +7，因此原负数等于 –7。

算术减法

当以补码形式表示负数时，两个二进制带符号数的减法是非常简单的，具体步骤如下：

取减数的补码(包括符号位)，将其与被减数(包括符号位)相加，丢弃符号位上所产生的进位。

这种方法的产生是因为在算术运算中，如果改变减数的符号，就可以用加法代替减法运

算。下面的关系式可以说明：

$$(\pm A) - (+B) = (\pm A) + (-B)$$
$$(\pm A) - (-B) = (\pm A) + (+B)$$

把一个正数变为负数是很容易的，只要取它的 2 的补码就行了。反过来也很容易，因为负数补码的补码可产生等值的正数。我们来看减法（－6）－（－13）＝＋7，用八位二进制数可将其写成(11111010－11110011)。通过取减数（－13）的 2 的补码可以得到（＋13），减法变成了加法。在二进制数的算术运算中，就是 11111010＋00001101＝100000111，丢弃最后进位，我们得到正确的答案是 00000111（＋7）。

对于用“符号－补码”法表示的二进制数，如果按照无符号数的基本运算法则进行相同的加法和减法运算，那是没有意义的。因此，计算机只需要普通的硬件电路来处理这两种类型的算术运算。鉴于此，计算机系统中几乎所有的算术单元都采用了“符号－补码”系统。用户或程序员必须要了解的是，加法或减法运算结果的差异将取决于这些数究竟是带符号数还是无符号数。

1.7　二进制码

数字系统中使用的信号有两个不同的值，并且电路元件有两个稳定状态。二进制信号、二进制电路元件以及二进制数字中有直接的对应关系。例如，具有 n 个数字的二进制数可以用 n 个电路元件来表示，并且每个元件都有一个取值为 0 或 1 的输出信号。数字系统不仅能表示和处理二进制数，而且还能够表示和处理其他的离散信息元素。在一组数值中，任何不同的离散信息元素都可以用二进制码(即一种 0 和 1 的组合模式)来表示。码元必须是二进制形式，因为当今的技术，只有那些运行 0 和 1 二值逻辑的电路才能够被用于计算机。不过，还必须认识到二进制码只是改变了记号，并不能改变其所代表的信息元素的含义。如果任意检查一台计算机的数据流，将会发现绝大多数时候它们都代表某种类型的编码信息，而不是二进制数。

一个 n 位二进制码就是一个 n 位的比特组合，它有 2^n 种不同的 0、1 组合，每种组合都代表一个编码。一个四元素集可以用两个比特来编码，每个元素被赋予四种比特组合中的一个：00、01、10、11。八元素集需要三个比特，而十六元素集则需要四个比特。一个 n 位码所表示的二进制数从 0 到 $2^n - 1$ 。每个元素必须分配唯一的比特组合，两个元素不能有相同的编码值，否则码的分配就会产生混乱。

尽管给 2^n 个不同的值编码需要的最小位数是 n，对于一个二进制码来说，所用的位数却没有最大值。例如，10 个十进制数可以用 10 个比特来编码，且每个数都可以被分配一个由九个 0 和一个 1 组成的码组。例如，十进制数 6 对应的编码是 0001000000。

BCD 码

尽管计算机采用的是最普通的二进制数体制，且在现代电子技术中容易表示，但大多数人还是更习惯于十进制。解决此差异的途径是将十进制数转换为二进制数，用二进制实现所有的算术运算，然后再把二进制运算结果变回十进制数。这种方法需要将十进制数存储在计算机中，以便将它们转换为二进制数。因为计算机只能接受二进制数值，因此，我们就必须

用由 0 和 1 组成的编码表示十进制数。只有当十进制数以编码形式存储在计算机里时，才可能直接对其进行数学运算。

如果要编码的元素个数不是 2 的幂次，则二进制编码会有一些码组无处分配，10 个十进制数集就是如此。区分 10 个元素的二进制码至少要有 4 位，而 4 位二进制码可以有 16 种不同的组合。因此，其中有 6 种编码没有被分配。将 4 位排列为 10 种不同的组合就能得到不同的二进制码。表 1.4 列出了十进制数最常用的码对应的二进制数分配，也就是二进制编码十进制数(二－十进制码)，通常简称为 BCD 码。十进制编码还有一些其他方法，本节后面将会介绍几种。

表 1.4 给出了每个十进制数对应的 4 位二进制码。一个 k 位的十进制数用 BCD 码表示需要 $4k$ 个比特。十进制数 396 用 BCD 码表示就是 12 比特：0011 1001 0110，每组四个比特表示一个十进制数。只有当十进制数在 0 到 9 之间时，它的 BCD 码才等于其相应的二进制数。即使都是由 1 和 0 组成，但一个大于 10 的 BCD 码不同于其相应的二进制数。此外，二进制数 1010 ~ 1111 没有使用，在 BCD 码中没有意义。我们来看十进制数 185 以及与其对应的 BCD 码和二进制数值：

$$(185)_{10} = (0001\ 1000\ 0101)_{BCD} = (10111001)_2$$

该 BCD 码有 12 比特，但与其等价的二进制数只需要 8 比特。与相应的二进制数相比，很明显 BCD 码需要更多的位数。不过，使用十进制数依然具有优势，因为人们一般都是用十进制数作为计算机的输入和输出数据的。

表 1.4 二进制编码对应的十进制数(BCD)

十进制符	BCD 码
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

认识到 BCD 数是十进制数而不是二进制数这一点很重要，尽管它们都用比特表示。十进制数和 BCD 码之间唯一的区别是十进制数用符号 0，1，2，…，9 表示，而 BCD 码用的却是二进制码 0000，0001，0010，…，1001，其所表示的十进制数值是完全一样的。十进制数 10 用 8 比特的 BCD 码表示为 0001 0000，十进制数 15 表示为 0001 0101。而与它们相对应的二进制分别为 1010 和 1111，只有 4 个比特。

BCD 加法

我们来看两个用 BCD 码表示的十进制数的加法，包括来自前面低有效位上产生的可能进位。因为每个数字不会超过 9，和数也就不会超过 9 + 9 + 1 = 19，其中所加的 1 是前面的进位。假设把 BCD 码当作二进制数一样相加，那么二进制和数会得到 0 到 19 之间的结果。以二进制数形式表示就是从 0000 到 10011，而在 BCD 码中则为 0000 到 1 1001；其中第一个 1 为进位，而其余四个比特则是两个 BCD 码相加之和。当二进制数之和小于或等于 1001 时(没有进位)，其对应的 BCD 码是正确的。而当二进制数之和大于或等于 1010 时，结果就是无效的 BCD 码。此时，应该把二进制数之和加上 $6 = (0110)_2$ 后才能转换为正确的 BCD 码，并产生需要的进位，这是因为二进制数和的最高有效位的进位和十进制数的进位之间相差 16 － 10 = 6。下面给出了 BCD 码相加的三个例子。

4	0100	4	0100	8	1000
+5	+0101	+8	+1000	+9	1001
9	1001	12	1100	17	10001
			+0110		+0110
			10010		10111

在上面每个例子中，两个 BCD 码就像两个二进制数那样相加。如果二进制数的和大于或等于 1010，就再加上 0110，得到正确的 BCD 和与进位。在第一个例子中，二进制数和等于 9，得到的结果就是正确的和的 BCD 码。第二个例子中，二进制的和产生了一个无效的 BCD 码，加上 0110 后就得到了正确的和的 BCD 码 0010（十进制数 2）以及一个进位。在第三个例子中，二进制的和产生了一个进位。当两数之和大于或等于 16 时就会出现这种情况，尽管其他四个比特小于 1001，由于有进位，该二进制数和仍然需要修正。加上 0110，就可以得到所需的和的 BCD 码 0111（十进制数 7）与一个 BCD 进位。

两个 n 位无符号 BCD 数的加法遵循与上面相同的原则。下面我们来看以 BCD 码进行的加法 184 + 576 = 760：

BCD 进位	1	1		
	0001	1000	0100	184
	+0101	0111	0110	+576
二进制和	0111	10000	1010	
加 6		0110	0110	
BCD 和	0111	0110	0000	760

首先，最低有效位上的一对 BCD 码产生等于 0000 的和的 BCD 码以及一个进位输送到上一相邻位。其次，对 BCD 码与前面的进位一起相加后产生的和数等于 0110，同时产生一个进位输送到下一对 BCD 码。第三对码与进位相加后产生的和为 0111，不需要进行修正。

十进制算术运算

以 BCD 码表示的带符号十进制数与带符号二进制数的表示相类似。我们既可以采用熟悉的“符号 – 数值”表示法，也可以用“符号 – 补码”表示法。十进制数的符号通常也用 4 位表示，以便与十进制数的 4 位编码一致。一般习惯用 0000 表示加号，而用 9 的 BCD 码 1001 表示减号。

“符号 – 数值”表示法在计算机中很少使用。“符号 – 补码”法既可以是 9 的补码也可以是 10 的补码，但 10 的补码是最常用的。为了得到 BCD 数的 10 的补码，我们首先要得到其 9 的补码，并在最低有效位上加 1。9 的补码运算是用 9 减去每个数。

上节中的二进制“符号 – 补码”表示法同样可以应用于十进制数的“符号 – 补码”表示。进行加法运算时，将所有数字相加，包括符号位数字，且要丢掉最后进位，当然这是在假设所有的负数都为补码的情况下。我们来看用“符号 – 补码”法所表示的十进制数加法（ +375）–（ –240）= +135 是如何实现的。

0	375
+9	760
0	135

第二个数最左边位上的 9 表示负号，且 9760 是数 0240 的补码。两个数相加并去掉最后进位后就可以得到结果 +135。当然，包括正负号在内，计算机中的十进制数都必须用 BCD 码表示。用 BCD 码实现的加法前面已经做了阐述。

进行十进制数减法运算时，不管是无符号数还是以补码形式表示的带符号数，都与二进制数的情况一样，就是取减数的补码，并将其与被减数相加。许多计算机有专用的硬件来直接实现以 BCD 码表示的十进制数的算术运算。计算机用户可以通过一些程序指令直接实现十进制数的算术运算，而不需将它们转换成二进制数。

其他的十进制码

一个十进制数编码至少需要四位二进制代码表示。通过四位二进制代码的十种不同可能的排列组合可以形成许多不同的编码。BCD 码以及三种其他编码见表 1.5。每种编码在由四位不同排列所构成的十六种可能组合中只用了其中十种组合，而其余六种组合没有用到，没有任何意义，应该被忽略。

表 1.5 十进制数的四种不同的二进制码

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

BCD 码和 2421 码是有权码的例子。在有权码中，每位都被赋予了一个权值，在这种方式下每个数可以通过将码组中所有为 1 的权值相加来求得。BCD 码的权值是 8、4、2 和 1，分别与每位的权值(2 的幂次)相对应。例如，0110 用权值解释表示 6，因为 $8\times0+4\times1+2\times1+1\times0=6$。当权值分别为 2、4、2、1 时，码组 1001 给出的十进制数等于 $2\times1+4\times1+2\times0+1\times1=7$。注意，有些数字在 2421 码中可能有两种编码方式，如十进制数 4 可以被分配为 0100 或者 1010，因为这两个码组的权值相加后都等于 4。

BCD 加法器直接将 BCD 值按位相加，不需要转换成二进制。但是，如果结果大于 9，需将结果加 6。可见，BCD 加法器需要更多的硬件，不再具备传统二进制加法器的速度优势[5]。

2421 码和余 3 码都是自补码。这种码的特性是其十进制数反码是通过将该码中 0 和 1 变成 1 和 0 直接得到的，即它们的 0 和 9、1 和 8、2 和 7、3 和 6、4 和 5 等均是互为反码。例如，十进制数 395 用余 3 码表示为 0110 1100 1000，而 604 的反码表示为 1001 0011 0111，就是将余 3 码的每位简单取反得到的(就像二进制数 1 的补码一样)。

因为其自补特性，余 3 码曾在一些老式计算机中使用。余 3 码是一种无权码，每个码由

对应的二进制数值加上 3 而来。注意，BCD 码不是自补码。

8、4、-2、-1 码是一种具有正、负权值的十进制码。例如，代表 0110 表示十进制数 2，数值计算为 $8\times0+4\times1+(-2)\times1+(-1)\times0=2$。

格雷码

许多物理系统产生的输出数据是连续的，这些数据在应用于数字系统之前必须要先转换为数字形式。连续或模拟信息一般是通过模数转换器转换为数字量。有时用表 1.6 中的格雷码表示从模拟数据转换为数字数据比较方便。与直接二进制序列相比，格雷码的优点是从一个数转换到下一个数时只有一位发生变化。例如，由 7 变为 8 时，格雷码相应地由 0100 变为 1100，只有第一位从 0 变成 1，而其他三位保持不变。比较直接采用二进制数，7 变为 8 就是由 0111 变成 1000，四位都发生了变化。

格雷码克服了由硬件生成的普通二进制序列中当一个数转变到下一个数时可能会出错或者产生模糊的问题。如果使用二进制数，在最右边位的转换时间大于其他三位的情况下，从 0111 到 1000 的转换可能会产生中间错误的数 1001，这可能对机器带来严重后果。格雷码会避免这种情况的发生，因为在两个数转换过程中只会有一位发生变化。

格雷码的典型应用是当用轴上连续变化的角坐标表示模拟数据时，轴被分成一些段，每段被赋予一个数值。如果相邻段采用相应格雷码序列表示，在线检测时就会很容易区分开，不会产生混淆。

表 1.6　格雷码

格　雷　码	十 进 制 值
0000	0
0001	1
0011	2
0010	3
0110	4
0111	5
0101	6
0100	7
1100	8
1101	9
1111	10
1110	11
1010	12
1011	13
1001	14
1000	15

ASCII 码

在许多应用场合中，数字计算机需要处理的不仅是数字数据，还包括字母字符。例如，一家拥有好几千员工的高科技公司，为了表示员工名字和其他相关信息，有必要对字母表中的字母赋予二进制码。此外，二进制码必须能表示一些数字和特殊字符(比如$)。字母数字字符集包括 10 个十进制数、26 个字母表中的字母以及许多特殊字符。如果只包含大写字符，这样的字符集包含 36 至 64 个元素。而如果既包含大写字母，也包含小写字母，则该字符集的元素数目会在 64 至 128 之间。第一种情况需要 6 位二进制码，而第二种情况则需要 7 位二进制码。

字母数字字符的标准二进制码是 ASCII 码(美国信息交换标准代码)，它用七位二进制给 128 个字符编码，如表 1.7 所示。该码的 7 位用 b_1 到 b_7 表示，b_7 为最高有效位。例如，字母 A 用 ASCII 码表示为 1000001(列 100，行 0001)。ASCII 码包含 94 个可打印的图形字符以及 34 个用于不同控制功能的不可打印字符。图形字符包括 26 个大写字母(A 到 Z)、26 个小写字母(a 到 z)、10 个数字(0 到 9)、32 个特殊的可打印字符如%、*和$等。

34 个控制字符在 ASCII 表中用缩写名表示，在表格下面列出的是这些控制字符所表示的功能含义。控制字符主要用于路由选择数据以及将打印文档安排成规定的格式。控制字符一般有三种类型：格式控制符、信息分隔符和通信控制符。格式控制符用来控制打印排版，主要包括大家熟悉的退格(BS)、横向列表(HT)和回车(CR)等打印机控制符；信息分隔符用来

将数据分成不同的段落和页面等，主要包括记录分隔符(RS)和文件分隔符(FS)；通信控制符主要用于远程终端的文件传输，它不同于同一通信信道中前后的其他信息，STX(start of text，文本的起始)和 ETX(end of text，文本的结束)是通信控制符的典型例子，用来给通信信道传输制定文本信息框架。

ASCII 码长有 7 位，但大多数计算机处理的是称为一个字节的 8 位单元。因此，ASCII 字符通常要用一个字节来存储，多出的一个字符有时可以用作它用，这取决于实际应用。例如，将最高有效位设置为 0，有些打印机可以确认其为 8 位的 ASCII 字符。将最高有效位设置成 1，另外的 128 个 8 位字符可以用来表示其他诸如希腊字母或斜体字等符号。

表 1.7　美国信息交换标准代码(ASCII 码)

	$b_7b_6b_5$							
$b_4b_3b_2b_1$	000	001	010	011	100	101	110	111
0000	NUL	DLE	SP	0	@	P	`	p
0001	SOU	DC1	!	1	A	Q	a	q
0010	STX	DC2	"	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	′	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	|
1101	CR	GS	–	=	M	]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	_	o	DEL

控制字符

NUL	空　　白	DLE	数据链路换码(换义)
SOH	标题开始	DC1	设备控制 1
STX	正文开始	DC2	设备控制 2
ETX	正文结束	DC3	设备控制 3
EOT	传输结束	DC4	设备控制 4
ENQ	询问	NAK	否认
ACK	确认	SYN	同步空传
BEL	响铃(告警)	ETB	块结束
BS	退格	CAN	取消
HT	横向列表	EM	纸尽
LF	换行	SUB	替换
VT	纵向列表	ESC	脱离
FF	换页(走纸)	FS	文件分隔符
CR	回车	GS	字组分隔符
SO	移出	RS	记录分隔符
SI	移入	US	单元分隔符
SP	空格	DEL	删除

检错码

为了检测数据传输过程中的差错，有时候在 ASCII 码中再增加 1 位(一般位于信息码组之前)来显示它的奇偶性。奇偶校验位是一个附加位，它指示整个码组具有奇数个 1 或偶数个 1 的信息。我们来看下面具有偶校验和奇校验的两个字符的情况：

	偶校验	奇校验
ASCII 码 A = 1000001	01000001	11000001
ASCII 码 T = 1010100	11010100	01010100

在上面例子中，我们在每个 ASCII 码的最左边位置上添加一个附加位，使得整个码组在奇校验时具有奇数个 1，在偶校验时具有偶数个 1。总之，要么采用奇校验，要么采用偶校验。相比之下偶校验用得更多一些。

奇偶校验位有助于检测信息传输过程中出现的差错。该操作是在每个传送字符的末端产生一位偶校验位，包含奇偶校验位的 8 位字符被传送到目的地后，在接收端检测每个到达字符的奇偶性。如果接收到的字符不是偶数，就意味着至少有一位在传输过程中发生了改变。这种方法可以发现奇数个码元错误，对于偶数个码元错误就无能为力了。要检测偶数个码元错误，可能需要采用另外的检错码。

检测到错误之后再做什么取决于特定的应用场合。有一种可能是重新传输信息，但前提是该错误是随机的且不会再次发生。因此，如果接收器检测到奇偶校验错误，就回送一个 ASCII 码 NAK(否认)控制字符，这是一个偶性的八位码 10010101。如果没有检测到错误，接收器就回传一个 ACK(确认)控制字符 00000110。发送端对 NAK 的响应是重新传输信息，直到接收到正确的奇偶性为止。如果经过多次努力后仍然发生传输错误，操作员会收到需要检测信道故障的信息。

1.8　二进制存储与寄存器

数字计算机中的二进制信息必须用一些能够存储的介质来存放。二进制单元是能保持两种稳定状态并能存储一位信息的器件。该单元接收的输入激励信号将其置为两种稳定状态之一，输出是区分这两种状态的物理量。二进制单元中存储的信息在一种稳定状态下是 1，在另一种稳定状态下就是 0。

寄存器

寄存器是一组二进制单元。n 位的寄存器可以存储包含 n 位信息的任何离散值。寄存器状态是由 0 和 1 组成的 n 元素组，其中每位表示该寄存器的一个单元状态。寄存器内容可以是对其存储信息进行解释的函数。例如，我们来看一个 16 位寄存器，该寄存器具有如下内容：

1100001111001001

寄存器的 16 个单元可以有 2^{16} 种可能的状态。如果寄存器内容代表二进制整数，则该寄存器可以存储任意一个从 0 到 $2^{16}-1$ 的二进制数。对于上面这个特定例子，寄存器内容就是数值等于十进制数 50 121 的二进制数。如果寄存器存储的是八位字符码，那么该寄存器内容就是

两个具有一定含义的字符。对于偶校验的 ASCII 码，该寄存器内容是两个字符：C(左八个字符)和 I(右八个字符)。另一方面，如果把寄存器内容理解为四个以 4 位二进制码表示的十进制数，则该寄存器内容是一个 4 位十进制数。在余 3 码中，该寄存器保存的是十进制数 9096。在 BCD 码中，这个寄存器内容毫无意义，因为 1100 不代表任何十进制数。从这个例子可以很清楚地看到，寄存器可以存储离散的信息单元，相同配置根据具体应用可被理解成不同类型的数据。

寄存器传输

数字系统的特性由寄存器和数据处理部件决定。寄存器传输是数字系统的基本操作，其含义是将二进制信息从一组寄存器传输到另一组寄存器中。信息可能是从一个寄存器直接传输到另一个寄存器，也可以通过一些数据处理电路实现某种运算。图 1.1 说明了信息在寄存器间的传输过程，并且还用图例描述了从键盘到存储单元寄存器之间的二进制信息传输。假设输入单元有一个键盘、一个控制电路以及一个输入寄存器，每次按下键盘，控制电路就会将一个等价的八位字符码提供给输入寄存器。假设该字符码用的是 ASCII 编码方式，而且是奇校验，则来自于输入寄存器的信息会被传送到处理寄存器的 8 个最低有效单元。在所有传输完成后，输入寄存器被清零，使控制单元通过键盘被再次按下后可以插入新的八位码。每个被传送到处理寄存器的八位字符紧跟着其前面的、被左移到下八个单元的字符。当四个字符传输完毕后，处理寄存器满了，其内容就被传送到存储寄存器中。当“J”、“K”、“O”、“H”这些字符被按下后，存储寄存器中的内容如图 1.1 所示。

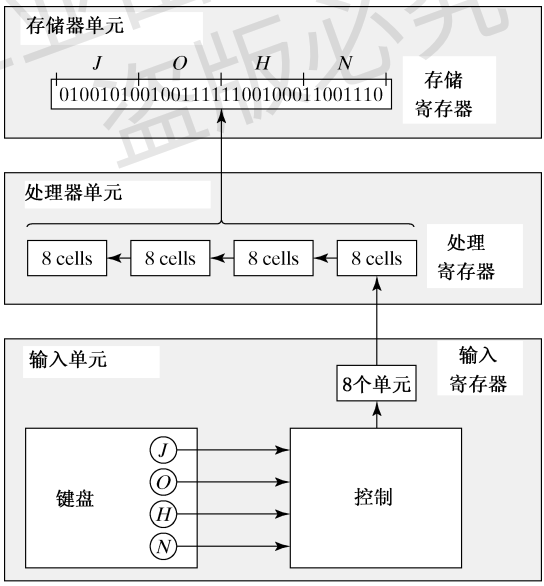


图 1.1　存储寄存器间的信息传输

为了处理二进制形式的离散信息值，必须要给计算机提供能保存处理数据的设备以及能处理单个信息的电路单元。通常用来保存数据的设备就是寄存器。用数字逻辑电路可以实现对二进制逻辑变量的处理。图 1.2 说明了两个 10 位二进制数相加的过程。存储器单元一般由几百万个寄存器组成，在图中只用了三个寄存器来表示。处理器单元部分由三个寄存器

R1、R2、R3 和数字逻辑电路组成，其中数字逻辑电路用来实现对寄存器 R1 和 R2 的位处理，并将其算术和的二进制数传送至寄存器 R3 中。存储寄存器只用于存储信息，不能处理两个操作数，但存储在其中的信息可以被传送至处理寄存器中。处理寄存器所得到的处理结果可以被回传给存储寄存器中保存起来，以等待下一次操作。图中显示了从存储寄存器传送到寄存器 R1 和 R2 的两个操作数内容。数字逻辑电路进行求和运算，并将结果传送至 R3，此时 R3 的内容可以被回传给某个存储寄存器。

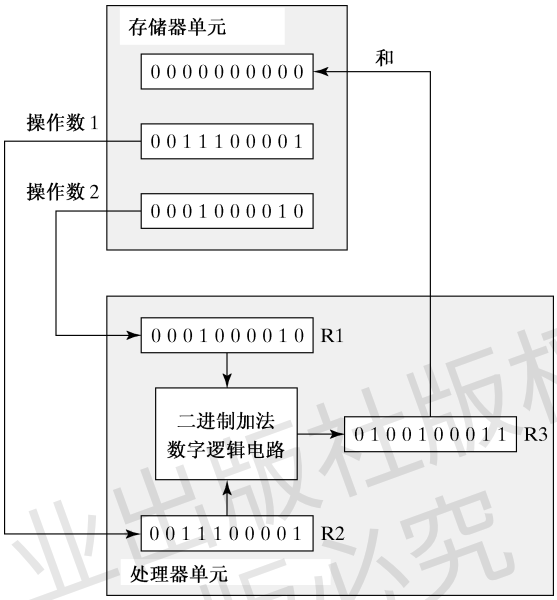


图 1.2　二进制信息处理举例

上面两个例子以一种非常简单方式描述了数字系统中的信息流动。数字系统中的寄存器是存储和保持二进制信息的最基本单元，数字逻辑电路处理存储在寄存器中的二进制信息。第 2 章到第 6 章主要介绍数字逻辑电路与寄存器，第 7 章主要介绍存储器，第 8 章主要介绍寄存器传输级的描述和数字系统设计。

1.9　二进制逻辑

二进制逻辑处理具有两个离散值的变量和具有某种逻辑意义的运算。变量的这两个逻辑值可以用不同的名称(如真和假、是和非等)来称呼。但对于我们而言，用逻辑值 0 或 1 表示比较方便。这一节介绍的二进制逻辑相当于一种代数，也称为布尔代数。布尔代数的详细描述将在第 2 章中给出。本节的目的是以启发式方式介绍布尔代数，使其与数字逻辑电路和二进制信号发生联系。

二进制逻辑定义

二进制逻辑包括二进制变量和一组逻辑运算。逻辑变量一般用字母表中的 A、B、C、x、y、z 等表示，每个逻辑变量只有 1 和 0 两种可能的取值。基本逻辑运算有三种：与(AND)、或(OR)和非(NOT)。每种运算结果也是一个二进制数，用字母 z 表示。

1. 与：与运算符用一个小圆点(·)表示，有时也可以省略。例如，$x \cdot y = z$ 或 $xy = z$，都读做“x 与 y 等于 z”。逻辑运算与的含义是：当且仅当 x 和 y 同时为 1 时，才有 $z = 1$；否则，$z = 0$。(记住：x、y、z 都是二进制变量，只能等于 1 或 0，不能是其他任何值。)运算 $x \cdot y$ 的结果是 z。
2. 或：或运算符用加号(+)表示。例如，$x + y = z$ 读做“x 或 y 等于 z”，其含义是：如果 $x = 1$ 或 $y = 1$，或同时 $x = 1$ 且 $y = 1$，则 $z = 1$；如果同时有 $x = 0$ 且 $y = 0$，则 $z = 0$。
3. 非：非运算符用一个撇号($'$，有时用一个上横杠)表示。例如，$x' = z$(或者 $\bar{x} = z$)，读做“x 的非等于 z”，其含义是：z 是 x 的非。换言之，如果 $x = 1$，则 $z = 0$；而若 $x = 0$，则 $z = 1$。非运算也相当于反码运算，它将 1 变成 0，将 0 变成 1，即 1 的反码是 0，0 的反码是 1。

二进制逻辑类似于二进制算术运算，逻辑运算与、或分别与乘法、加法相类似。实际上，与运算和或运算所用的符号与乘法和加法所用符号是一样的。然而，二进制逻辑与二进制算术运算不能相混淆。应该认识到算术变量所代表的数可能有多种取值，而逻辑变量只能取 1 或 0 这两个值。例如，在二进制算术运算中，1 + 1 = 10(读做：“一加一等于二”)。而在二进制逻辑中，1 + 1 = 1(读做：“一或一等于一”)。

对于 x 和 y 逻辑取值的每一种组合，都有一个由逻辑运算所定义的 z 值与之相对应，该定义可以用真值表这种简洁的方式列出来。真值表是一种将输入变量的各种可能取值组合与相应运算结果一起列成的表。与和或运算的真值表是将一对变量 x 和 y 的所有可能取值列在一起，每种组合的运算结果又被单独列在一起。表 1.8 列出了与、或及非运算的真值表，这些表清晰地描述了三种逻辑运算的定义。

表 1.8　逻辑运算的真值表

与			或			非	
x	y	$x \cdot y$	x	y	$x + y$	x	x'
0	0	0	0	0	0	0	1
0	1	0	0	1	1	1	0
1	0	0	1	0	1		
1	1	1	1	1	1		

逻辑门

逻辑门是对一个或多个输入信号进行运算，并产生输出信号的电子电路。诸如电压或电流之类的电信号，模拟信号的电压范围是 0 ~ 3 V。但在数字系统中，一般要变换成两种可识别的值 0 或 1 存在。电压运算电路要对两个离散的电压电平做出响应，这两个电压电平所代表的二进制变量为逻辑 1 或逻辑 0。例如，某个专用数字系统可能会定义逻辑 0 对应于 0 V 电压信号，而逻辑 1 则对应于 3 V 的电压信号。在实际应用中，每个电压电平都有一个可接受的范围，如图 1.3 所示。数字电路输入端接收的是在允许范围内的二进制信号，而输出端对落在特定范围内的二进制信号做出响应。只是在过渡状态期间，需要跨越两个

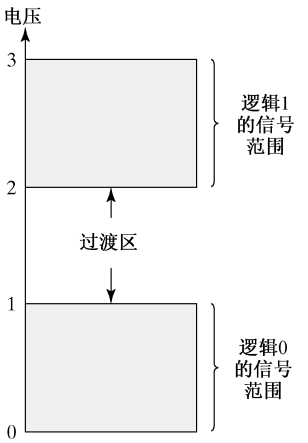
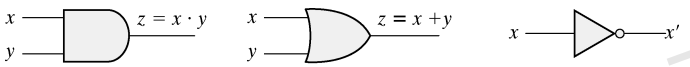


图 1.3　二进制逻辑值的信号电平

允许范围之间的中间区域。任何需要计算和控制的信息都可由当前的二进制信号通过各种逻辑门组合来进行运算，其中每个信号代表一个特定的二进制变量。当物理信号在一个特定范围内时，可以看成 0 或 1。

用来表示三种逻辑门的图形符号如图 1.4 所示，这些门都由硬件模块组成。当输入逻辑满足一定要求时，它们会产生相当于逻辑 1 或逻辑 0 的输出信号。与门和或门中的输入信号 x 和 y 可能有四种状态：00、10、11 和 01。这些输入信号与每个门相应的输出信号的时序如图 1.5 所示，时序图描述了每个门对四种输入信号组合的响应，其横坐标表示时间，纵坐标显示了信号在两个允许电压电平之间的改变。现实中，两个逻辑值的转变非常迅速，但不是瞬间。低电平表示逻辑 0，高电平表示逻辑 1。只有当两个输入信号都为逻辑 1 时，与门的输出才是逻辑 1。只要任何一个输入信号为逻辑 1 时，或门的输出就是逻辑 1。非门通常相当于一个反相器，取这个名称的理由可以从其时序图中的信号响应体现出来，该图显示输出信号与输入信号是逻辑反相的。



(a) 二输入与门　(b)二输入或门　(c)非门或反相器

图 1.4　数字逻辑电路的符号

与门和或门的输入信号可能超过两个。如图 1.6 所示为一个 3 输入的与门和一个 4 输入的或门。3 输入与门是当三个输入同时为逻辑 1 时，其输出才为逻辑 1；而其中只要有一个输入为逻辑 0，其输出就是逻辑 0。对于 4 输入或门，只要有一个输入为逻辑 1，其输出就是逻辑 1；只有当四个输入都为 0 时，其输出才为逻辑 0。

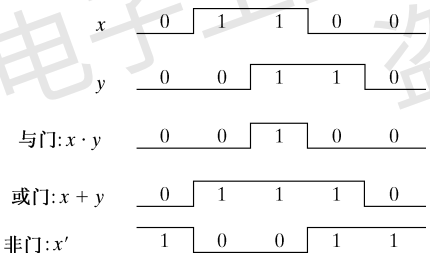
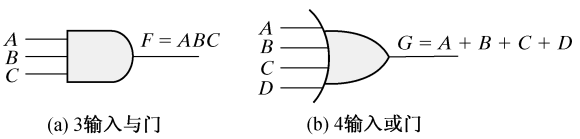


图 1.5　逻辑门的输入-输出信号



(a) 3输入与门　(b) 4输入或门

图 1.6　多输入逻辑门

习题

(＊号标记的习题解答在本书的最后。)

1.1　列出从 16 到 32 的八进制数和十六进制数。用 A 和 B 作为最后两个数字，试列出从 8 到 28 的十二进制数。

1.2＊　分别指出下列三种系统中所包含的确切字节数是多少？

(a)32 kB；(b)64 MB；(c)6.4 GB

1.3　将下列各数转换为十进制数。

(a) ＊$(4310)_5$　　(b) ＊$(198)_{12}$

(c) $(435)_8$　　(d) $(345)_6$

1.4　用 16 位可以表示的最大二进制数是多少？与其等值的十进制和十六进制数各是多少？

1.5* 确定下面每种情况下这些数的基数,使得运算结果是正确的。

(a) 14/2 = 5　　(b) 54/4 = 13　　(c) 24 + 17 = 40

1.6* 一元二次方程 $x^2 - 11x + 22 = 0$ 的解为 $x = 3$ 和 $x = 6$。数的基数是多少?

1.7* 先将十六进制数 64CD 转换成二进制数,再从二进制数转换为八进制数。

1.8 用以下两种方式将十进制数 431 转换成二进制数:(a)直接转换为二进制数;(b)先转换为十六进制数,再从十六进制数转换为二进制数。哪种方法更快?

1.9 用十进制数表示以下各数:

(a)* $(10110.0101)_2$　　(b)* $(16.5)_{16}$

(c)* $(26.24)_8$　　(d) $(DADA.B)_{16}$

(e) $(1010.1101)_2$

1.10 将以下二进制数转换为十六进制和十进制数:(a)1.10010;(b)110.010。解释为什么(b)的结果对应的十进制数是(a)的 4 倍。

1.11 实现以下二进制除法:111011 ÷ 101。

1.12* 不转换为十进制数,将下面的数直接进行加法与乘法运算。

(a)二进制数 1011 和 101。

(b)十六进制数 2E 和 34。

1.13 进行以下的进制转换:

(a)把十进制数 27.315 转换为二进制数。

(b)计算出等于 2/3 的 8 位二进制数,然后再把二进制数转换为十进制数,结果与 2/3 接近程度如何?

(c)把(b)中的二进制结果转换为十六进制数,再将该十六进制数转换为十进制数,答案是否相同?

1.14 求下面二进制数的反码和补码:

(a) 00010000　　(b) 00000000

(c) 11011010　　(d) 10101010

(e) 10000101　　(f) 11111111

1.15 找出以下十进制数的反码和补码:

(a) 25 478 036　　(b) 63 325 600

(c) 25 000 000　　(d) 000 000 00

1.16 (a)求十六进制数 C3DF 的补码。

(b)将十六进制数 C3DF 转换为二进制数。

(c)求(b)结果的补码。

(d)将(c)的结果转换为十六进制数,并和(a)的结果进行比较。

1.17 实现以下十进制无符号数的减法,其中减数用补码表示。如果结果为负,求其补码,并在前面加一负号。验证自己的答案。

(a)4637 - 2579　　(b)125 - 1800

(c)2043 - 4361　　(d)1631 - 745

1.18 实现以下二进制无符号数的减法,其中减数用补码表示。如果结果为负,求其补码,并在前面加负号。

(a)10011 - 10010　　(b)100010 - 100110

(c)1001 - 110101　　(d)101000 - 10101

1.19* 用"符号 - 数值"法表示的十进制数为 +9286 和 +801。将它们转换为带符号的补码形式,并进行以下运算:(注意和是 +10 627,需要六个数字和一个符号)。

(a)(+9286) + (+801)　　(b) (+9286) + (-801)

(c)(-9286) + (+801)　　(d) (-9286) + (-801)

1.20　将十进制数 +49 和 +29 转换为用“符号 – 补码”法表示的二进制数，用足够的位数来表达。然后，用二进制形式实现以下运算：(+29) + (–49)，(–29) + (+49)和(–29) + (–49)。将结果转换回十进制数，并验证其是否正确。

1.21　如果数$(+9742)_{10}$和$(+641)_{10}$是“符号 – 数值”表示方式，它们的和是$(+10\ 383)_{10}$，需要 5 个数字和 1 个符号。将下面的十进制数转换为“符号 – 补码”形式，并求出它们的和。

(a)(+9742) + (+641)　　(b)(+9742) + (–641)

(c)(–9742) + (+641)　　(d)(–9742) + (–641)

1.22　把十进制数 6514 转换为 BCD 码和 ASCII 码。对于 ASCII 码，在其左边添加一位偶校验位。

1.23　把十进制无符号数 791 和 658 表示成 BCD 码形式，并给出求和的必要步骤。

1.24　用以下的权将十进制数表示为含权的二进制码。

(a)*6, 3, 1, 1　　(b) 6, 4, 2, 1

1.25　将十进制数 6248 表示成：(a)BCD 码；(b)余 3 码；(c)2421 码；(d)6311 码。

1.26　找出十进制数 6248 的反码，并将其用 2421 码表示。其结果是习题 1.25 中(c)结果的反码，从而说明了 2421 码具有自补特性。

1.27　按照一定的顺序给 52 张扑克牌分配二进制码，要求使用最少的位数。

1.28　用八位的 ASCII 码表达“G. Boole”，包括句点和空格。把每个字符最左边的位当作奇偶校验位。每个 8 位码都应该有偶校验。(George Boole 是 19 世纪的数学家，下一章介绍的布尔代数就是以他的名字命名的。)

1.29*　对下面的 ASCII 码进行译码：

1010011　1110100　1100101　1110110　1100101　0100000　1001010　1101111 1100010　1110011

1.30　下面是 ASCII 码字符串，为了表达紧凑，该字符串被转换成十六进制：73 F4 E5 76 E5 4A EF 62 73。每对数字有 8 位，其最左边位为奇偶校验位，其余位是 ASCII 码。

(a)将它们转换成二进制数，并对其进行 ASCII 解码。

(b)确定使用的是奇校验，还是偶校验。

1.31*　ASCII 码中有多少个打印字符？其中有多少特殊字符(不是字母也不是数字)？

1.32*　将 ASCII 大写字母转换为小写字母，哪一位必须取反？反之又如何？

1.33*　一个 12 位寄存器的状态是 100010010111。如果按照下面的要求，它表示的内容是什么？

(a)三个 BCD 码表示的十进制数；

(b)三个余 3 码表示的十进制数；

(c)三个 84-2-1 码表示的十进制数；

(d)一个二进制数。

1.34　列出 10 个十进制数的 ASCII 码，最左边位是偶校验位。

1.35　用类似于图 1.5 所示的时序图方法，试画出图 P1.35 中相对于三个输入变量 a、b、c 的输出函数 f 和 g 的信号波形，输入变量 a、b、c 有 8 种可能的组合。

1.36　用类似于图 1.5 所示的时序图方法，画出图 P1.36 中相对于两个输入变量 a、b 的输出函数 f 和 g 的信号波形，输入变量 a、b 有 4 种可能的组合。

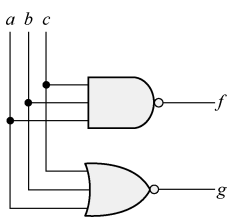


图 P1.35

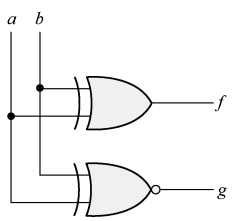


图 P1.36

参考文献

1. Cavanagh, J. J. 1984. *Digital Computer Arithmetic.* New York: McGraw-Hill.
2. Mano, M. M. 1988. *Computer Engineering: Hardware Design.* Englewood Cliffs, NJ: Prentice-Hall.
3. Nelson, V. P., H. T. N agle, J. D. I rwin, and B. D. C arroll. 1997. *Digital Logic Circuit Analysis and Design.* Upper Saddle River, NJ: Prentice Hall.
4. Schmid, H. 1974. *Decimal Computation.* New York: John Wiley.
5. Katz, R. H. and Borriello, G. 2004. *Contemporary Logic Design*, 2nd ed. Upper Saddle River, NJ: Prentice-Hall.

网络搜索主题

BCD 码
ASCII
存储寄存器
二进制逻辑
BCD 加法
二进制码
二进制数
余 3 码