国外电子与通信教材系列

逻辑电路设计基础

Introduction to Logic Design

[美] Alan B. Marcovitz 著
殷洪玺 杨匡汉 李正斌 等译
徐安士 审校

Mc Graw Hill
PHEI
電子工業出版社
Publishing House of Electronics Industry
www.phei.com.cn

国外电子与通信教材系列

逻辑电路设计基础

Introduction to Logic Design

[美] Alan B. Marcovitz 著

殷洪玺 杨匡汉 李正斌 等译

徐安士 审校

電子工業出版社

Publishing House of Electronics Industry

北京·BEIJING

内 容 简 介

本书是计算机科学、计算机工程和电气工程等专业的学生学习逻辑电路设计的入门教程。全书共7章和一个附录，前4章介绍数制、开关代数、真值表和卡诺图，并讲解了逻辑函数的化简以及组合系统的分析与设计；后3章介绍时序系统的分析与设计、移位寄存器和计数器、可编程逻辑器件、用列表法和状态分割法进行状态化简和状态分配；附录部分介绍了4个实验操作平台及25个实验室作业。

要学好逻辑电路设计这门课程，需要掌握好三个环节：理论、习题和实验。本书紧紧抓住这些教学环节，系统地阐述了逻辑设计的核心内容，尤其突出了系统的分析和设计方法。对于需要学生通过练习进一步巩固的重点内容，书中均布置了适量作业。在每章讲述内容之后专门安排了一节解题实例和一节习题。本书是学习逻辑电路设计难得的一本好教材，既可作为计算机、电气工程和通信、电子等专业学生的教材或教学参考书，也可供相关专业工程技术人员参考。

Alan B. Marcovitz: **Introduction to Logic Design**.

ISBN: 0-07-112399-7

版权贸易合同登记号：图字：01-2001-5386

图书在版编目(CIP)数据

逻辑电路设计基础/(美)马科维奇(Marcovitz A.B.)著；殷洪玺等译. －北京：电子工业出版社，2002.5
(国外电子与通信教材系列)

书名原文：Introduction to Logic Design

ISBN 7-5053-7584-9

Ⅰ.逻… Ⅱ.①马… ②殷… Ⅲ.逻辑电路-电路设计 Ⅳ.TN791

中国版本图书馆 CIP 数据核字(2002)第 028034 号

责任编辑：窦 昊 罗翀翀

印 刷：北京东光印刷厂

出版发行：电子工业出版社 www.phei.com.cn

北京市海淀区万寿路 173 信箱 邮编：100036

经 销：各地新华书店

开 本：787×1092 1/16 印张：28.75 字数：718 千字

版 次：2002 年 5 月第 1 版 2002 年 5 月第 1 次印刷

定 价：39.00 元

凡购买电子工业出版社的图书，如有缺损问题，请向购买书店调换；若书店售缺，请与本社发行部联系。联系电话：(010) 68279077

前　言

本书是计算机科学、计算机工程和电气工程等专业的学生学习逻辑电路设计的入门教程。学习本教程无需任何先决条件，虽然先学习一门工程导论课或者初级程序设计课会有所帮助。本书强调对基础知识的讲解，并通过大量实例进行讲授。作者认为，学习逻辑设计的惟一途径便是做大量的设计例题。因此，除了在正文中列举大量例题之外，每章都另有一些解题实例，既有问题，也有答案，同时还附有大量练习。此外，还安排有一系列实验，以便把理论和实践紧密联系起来。附录中列举了进行这些实验所需的实验室标准硬件配置（芯片、开关、灯和导线），适用于 PC 机或苹果机的面包板模拟器和两个制作原理图的工具。本课程可以单独讲授，不做实验，但是，如果能配合做 8 到 10 个精心选择的实验，学生将会受益匪浅。

虽然计算机辅助工具已经广泛用于大型系统的设计，但是学生必须先打好基础。作为一门导论课，本书所提供的基础知识已是绰绰有余。本书的原理图制作练习和第 6 章有关硬件设计语言的相关知识，可以使学生顺利过渡到基于计算机辅助设计工具的下一门课程。

出于本书叙述的需要，第 1 章首先对数制进行简单扼要的回顾。已经学过这部分内容的学生，可跳过第 1 节，直接学第 2 节。然后着手讨论组合系统设计过程中的各个步骤，以及如何编制真值表。

第 2 章介绍开关代数和利用通用门——与门、或门、非门、与非门、或非门和异或门（AND，OR，NOT，NAND，NOR，Exclusive - OR）实现开关函数。我们关心的是它们的逻辑性能，而非它们的电子实现。

第 3 章讨论两种化简方法：卡诺图法和迭代合意法（Iterated Consensus）。本章提供多种解题方法（用卡诺图时，可多达 6 个变量）。题目中既有单输出的系统，也有多输出的系统。

第 4 章是关于较大规模组合系统的设计。本章介绍了几种市场上可以买到的较大规模器件，包括加法器、译码器、编码器和优先权编码器，以及数据选择器。接着讨论逻辑阵列的应用——只读存储器（ROM），可编程逻辑阵列（PLA），可编程阵列逻辑（PAL），从而实现中规模组合系统。最后，本章有两个较大系统的设计。

第 5 章介绍时序系统。首先介绍闩锁和触发器的特性，然后介绍时序系统的设计过程。在进行时序系统设计之前，先分析了几个典型的系统。接着研究计数器的特殊情况。最后，便是对文字问题的求解，详细介绍如何对文字表述的问题，编制状态表和状态图。

第 6 章讨论较大规模的时序系统。首先研究移位寄存器和计数器的设计，然后介绍可编程逻辑器件（PLD）。接着讨论用于设计较复杂系统的两种方法，即算法状态机图（ASM，algorithmic state machine diagram）和硬件设计语言（HDL，Hardware Design Language）。最后，给出两个较大规模时序系统的例子。

第 7 章研究状态化简和状态分配问题。首先介绍用列表法进行状态化简。然后，介绍利用分割法进行状态化简和状态分配，从而简化所用的组合逻辑。

本书的一大特点是提供了大量的解题实例。每一章都有大量例题，用以阐明书中介绍的各种方法。对于每一个例题都进行了详细的求解。学生应在不参考本书答案之前，自行求解

每一个例题,然后才将自己的答案和本书的答案相比较。

本书每一章还带有一整套练习,这些练习的答案可从网上获得。

本书的另一个特点便是在附录中提供了许多实验室作业。有四个操作平台:基于硬件的逻辑实验室(利用芯片、线路等);硬件实验室模拟器,使学生可以在计算机屏幕上"连接"导线;还有两个电路制作程序,LogicWorks IV 和 Altera Max + plus II。关于这两个程序,书中有详尽的说明,使学生可以完成多种实验。本书还含有 25 个实验室作业,其中一些有多个选项,教师可以任意采用某些选项,以改变作业的细节。

我们使用本教材作为一门 4 学分的课程。每周授课 3.5 小时,外加 8 个实验室作业。在授课过程中,我们选讲了如下章节:

第 1 章:全部。

第 2 章:全部(2.11 节除外)。

第 3 章:只讲 3.1 节。

第 4 章:全部(4.8 节除外)。但有一个打分设计题需要参考这部分内容。分值占 10%,学生通常分成 2 ~ 3 人一组来完成。

第 5 章:全部,但有时删去 5.6 节。

第 6 章:6.1 节、6.2 节和 6.3 节。有时根据 6.6 节安排第二个作业。

第 7 章和 3.2 节:有时候可能剩一点时间来讲其中一部分,但是从来未能二者兼顾。

如果学时不够,2.10 节可以压缩,3.1.5 节可以略去。3.1.6 节的内容用到 4.7.2 节中讨论 PLA。第 4 章和其他各章联系不大,但这部分内容对学生今后会有用处。5.5 节和 5.6 节,亦可删去,它们对课程影响不大。和第 4 章的情况一样,教师可在第 6 章的内容上自行取舍。如果时间有限,可以只讲 7.1 节;如果时间多一些,可以跳过 7.1 节,讲 7.2 节和 7.3 节。

目　录

第1章 导 论

本书是一本关于数字系统设计的书，这里的设计是指逻辑电路设计。数字系统是一个其中所有信号均用离散值表示的系统，计算机和计算器就是明显的例子，但是大多数电子系统都包含大量的数字逻辑。在数字系统内部，通常用二值信号工作，该二值信号可以记为 0 和 1。这样的系统如图 1.1 所示，它可有任意个输入（$A, B, \cdots$）和输出（$W, X, \cdots$）。

除了所示的数据输入之外，一些电路还要求有定时信号，称为时钟（实际上只是一种按一定速率在 0 和 1 之间交替改变的输入信号）。第 5 章将对时钟信号进行详细讨论。

数字系统的一个简单例子如图 1.1 所示。

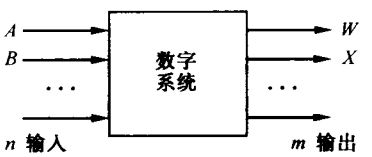

图 1.1 一个数字系统

例 1.1 一个系统有三个输入 A，B，C 和一个输出 Z。当且仅当[①] 两个输入为 1 时，$Z = 1$。

数字系统的输入和输出表示真值。有时，如在例 1.1 中，这些值是二进制，即它们取两个值中的一个。另外一些情况下，它们可能是多值的，例如，在本课程中，输入可能是十进制数，而输出可能用字母表示。每个数都可用一组二进制数字来表示（经常称为“位”）。这个过程是指把输入和输出编码为二进制（在后面将详细讨论）。

这些二进制量的物理意义可能代表两种电压之一，例如在实验室实现时，逻辑 0 代表 0 V 或接地，逻辑 1 代表 5 V。这方面的问题，将在附录中讨论。它也可以是两个方向的一个磁场（像在磁带中那样），一个开关在上、下两个位置（对于输入来讲），或一个灯的开关（对于输出来讲），除了专门在实验室的实验讨论和要把文字描述变为更正式的表述之外，在行文时将只关心 0 和 1，而与物理表示无关。

可以用表格的形式描述一个物理系统的行为，如例 1.1。因为只有 8 种可能的输入组合，可以把它们全列出来，并给出每一种组合对应的输出。这样的表（指真值表）如表 1.1 所示。真值表（及其类似的问题）将留待后面章节研究。

另外两个例子见例 1.2 和例 1.3。

---

① “当且仅当”（if and only if，iff），意思是：“只有满足条件时输出才是 1，如果条件不满足那么输出不是 1（意味着一定是 0）”。

表 1.1 例 1.1 的真值表

A	B	C	Z
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

例 1.2 一个系统有 8 个输入，代表 2 个四位二进制数。该系统有 1 个五位的输出，代表 2 个数之和(每个输入数的可能范围从 0 到 15，输出数的范围从 0 到 30)。

例 1.3 一个系统有 1 个输入 A、1 个时钟及 1 个输出 Z，当且仅当在连续三个时钟时间里，输入是 1 时输出才为 1。

前两个例子是组合系统，即输出仅取决于现时刻输入的值。在例 1.1 中，假如知道 A，B，C 现时刻的值，就能确定现在的 Z 值①。例 1.3 是一个时序系统，即它要求有存储器，因为需要知道此前输入的某些情况(先前的时钟时间里)。

在本书的前半部分，将集中讨论组合系统，时序系统留待后面讨论。正像所看到的，时序系统由存储器和组合逻辑两部分构成。因此，在能开始设计时序系统之前，需要能够设计组合系统。

这里要特别提一下，一般自然语言，特别是英语，不是一种非常精确的语言。前面给出的例子留有一些解释的余地。在例 1.1 中，是指如果三个输入都是 1 时，输出为 1 呢？还是恰好仅有两个输入是 1 时，输出为 1 呢？两种理解都可以。当写真值表时，就必须确定下来，我们解释为两个或两个以上。因此，当所有的三个输入都是 1 时，输出也是 1。在本书的例题中，将尽可能地精确。但是即便如此，不同的人还是有可能以不同的方式来理解这个例题。

归根到底，需要对逻辑系统进行更精确的描述。在前两章讨论组合系统和第 5 章讨论时序系统时，将努力做到这一点。

1.1 数制的简单回顾

本节将介绍数制中与本书内容有关的几个问题，并将只讨论整数问题。如果在其他课程中已经学习过这方面的内容，可以跳过本节，直接学习 1.2 节。

整数通常用位数系统书写，其中每一个数字表示一个幂级数中的一个系数。

$$N = a_{n-1}r^{n-1} + a_{n-2}r^{n-2} + \cdots + a_2 r^2 + a_1 r + a_0$$

式中 n 为数字的位数，r 为数制的基或基数，a_i 为系数，系数皆为整数，其范围为：

$$0 \leqslant a_i < r$$

① 在实际系统中，输入和输出之间稍有延迟。就是说，如果输入在某一瞬间发生变化，输出的变化将稍为滞后。其延迟时间一般在纳秒范围内(10^{-9}秒)。我们几乎总是不考虑这些延时。第 4 章将再讨论此问题。

对于十进制来说，$r = 10$，而 a_i 为0至9中的任一数字。对于二进制来说，$r = 2$，而 a_i 为0或1。在计算机文件中通常还用八进制（$r = 8$）和十六进制（$r = 16$）。在二进制中，数字通常称为“位”（又称比特，*bit*），是 binary digit 的缩写。

因此，十进制数7642（有时写作 7642_{10}，以强调其基数为10，即十进制）代表：

$$7642_{10} = 7 \times 10^3 + 6 \times 10^2 + 4 \times 10 + 2$$

而二进制数：

$$\begin{aligned} 101111_2 &= 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2 + 1 \\ &= 32 + 8 + 4 + 2 + 1 = 47_{10} \end{aligned}$$

从最后这个例子[①] 可以清楚看出如何从二进制转换成十进制：只要求出幂级数的值即可。为了转换快捷，要记住2的各个幂次，这样就不必每一次都去计算。要是能够记住2的前10个幂次，就会节约大量时间。表1.2列出了2的前20个幂次。

表1.2 2的幂次

n	2^n	n	2^n
1	2	11	2 048
2	4	12	4 096
3	8	13	8 192
4	16	14	16 384
5	32	15	32 768
6	64	16	65 536
7	128	17	131 072
8	256	18	262 144
9	512	19	524 288
10	1 024	20	1 048 576

经常会用到前16个二进制正整数，有时还会用到前32个正整数，如表1.3所列（和十进制一样，打头的0一般都省略，但在前16个整数中，仍然列出4位，包括打头的0）。当一个二进制正整数的存储位置的位数确定之后，加上打头的0才能得到正确的位数。

表1.3 前32个二进制整数

十进制	二进制	4位	十进制	二进制
0	0	0000	16	10000
1	1	0001	17	10001
2	10	0010	18	10010
3	11	0011	19	10011
4	100	0100	20	10100
5	101	0101	21	10101
6	110	0110	22	10110
7	111	0111	23	10111
8	1000	1000	24	11000
9	1001	1001	25	11001
10	1010	1010	26	11010
11	1011	1011	27	11011
12	1100	1100	28	11100
13	1101	1101	29	11101
14	1110	1110	30	11110
15	1111	1111	31	11111

① 1.6节为解题实例，其中有本章所讨论的每种类型的问题的更多例题。以后各章也都有一节解题实例。

请注意，比 2^n 少 1($2^n - 1$)的二进制数由 n 个 1 构成，例如：

$$2^4 - 1 = 1111 = 15;\ 2^5 - 1 = 11111 = 31$$

一个 n 位二进制数可表示 0 到 $2^n - 1$ 之间的一个正整数。因此，一个 4 位的二进制数可表示 0 至 15 间的任一整数。一个 8 位的二进制数可表示 0 至 255 间的任一整数。一个 16 位的二进制数可表示 0 至 65 535 间的任一整数。

把一个十进制数转换成二进制数时，要考虑十进制数字的幂次，然后把其中的每个数字转换成二进制，即：

$$746 = 111 \times (1010)^{10} + 0100 \times 1010 + 0110$$

但是这样做需要用二进制乘法，也颇费时间。

有两种利用十进制的便捷算法。第一种算法是把该十进制数减去小于该数的 2 的最大幂次，并在二进制数的相应位置记入 1。对余数作同样处理，其 2 的幂次大于余数时，在相应位置记入 0。

例 1.4 对于十进制数 746，$2^9 = 512$ 是小于或等于 746 的最大幂次，因此在 2^9 位置上记入 1。然后由 $746 - 512 = 234$。2 的下一个幂次为 $2^8 = 256$，大于 234，因此在 2^8 位置上记入 0。下一步，$234 - 128 = 106$，在 2^7 位置上记入 1(此时的二进制数，前三位为 101)。继续用 106 减去 64，余 42，在 2^6 的位置上记入 1。因为 42 大于 32，在 2^5 的位置上记入 1。$42 - 32 = 10$。由于 $2^4 = 16$ 大于 10，而 10 的二进制数为 1010，因此，

$$\begin{aligned} 746_{10} &= 1 \times 2^9 + 0 \times 2^8 + 1 \times 2^7 + 1 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 \\ &\quad + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2 + 0 \\ &= 1011101010_2 \end{aligned}$$

另一种方法是用 2 连续除十进制数，每次所得余数即为二进制数字，但需从最低位(a_0)算起。去掉余数后，重复运算过程。

例 1.5 将十进制数 746 转换成二进制数，746/2 = 373，余数为 0；373/2 = 186，余数为 1；因此，该二进制数最末两位数字应为 10。继续，186/2 = 93，余数为 0；93/2 = 46，余数为 1；46/2 = 23，余数为 0；至此已有 01010。23/2 = 11，余数为 1，11/2 = 5，余数为 1，5/2 = 2，余数为 1，2/2 = 1，余数为 0，1/2 = 0，余数为 1。故所得二进制数为 1011101010，和前一种方法结果相同。在这种方法中，当已经知道余数等效的二进制量时，就可以停下来，直接将它转换为二进制数。因此，当计算得到 23 时，由于知道其对应的二进制数是 10111(根据表 1.3)，就可以把它放在已得到的二进制数之前，结果得到 1011101010。

例 1.6 把 105 转换为二进制数

105/2 = 52，余 1	产生	1
52/2 = 26，余 0		01
26/2 = 13，余 0		001
而 13 = 1101		1101 001

这个方法之所以有效，是因为在幂级数中，除了最后一项以外的所有项都除以 2，因此：

$$\begin{aligned} &746/2 = 373，余数为\ 0 \\ &\quad = 1 \times 2^8 + 0 \times 2^7 + 1 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 \end{aligned}$$

$$+1\times2^2+0\times2+1+\text{余数 }0$$

最后一位为余数,如果重复这个过程,得到:

$373/2 = 186$,余数是 1

$$=1\times2^7+0\times2^6+1\times2^5+1\times2^4+1\times2^3$$
$$+0\times2^2+1\times2+0+\text{余数 }1$$

该余数是从右边数的第二位,再除以 2 余数是 0,为第三位,这个过程继续下去,直到得到最后一位。

本小节作业:

1. 看解题实例例 1,2;
2. 做习题 1,2。

1.1.1 八进制数和十六进制数

八进制和十六进制,经常指 *Octal*($r = 8$)和 *hex*($r = 16$),通常用在计算机文件中,它们都仅仅是二进制数的速记符号。在八进制中,将二进制数分成三位一组(从最低位开始)。例如,一个 9 位数:

$$\begin{aligned}N &= (b_8 2^8 + b_7 2^7 + b_6 2^6) + (b_5 2^5 + b_4 2^4 + b_3 2^3)\\ &\quad + (b_2 2^2 + b_1 2^1 + b_0)\\ &= 2^6(b_8 2^2 + b_7 2^1 + b_6) + 2^3(b_5 2^2 + b_4 2^1 + b_3)\\ &\quad + (b_2 2^2 + b_1 2^1 + b_0)\\ &= 8^2 o_2 + 8 o_1 + o_0\end{aligned}$$

这里 o_i 代表八进制数,它一定是一个落在 0 到 7 范围内的数。圆括号内的每一项仅被转换成十进制数。如果该二进制数的位数不是 3 的倍数,则在前边添 0。

例 1.7 (由例 1.4 和例 1.5 而来)

$$1011101010_2 = 001\ 011\ 101\ 010_2$$
$$= 1352_8$$

为了将八进制数转换为二进制数,必须将每一位八进制数用三位相等的二进制数代替,正好与例 1.7 中的最后一步相反。

为了将八进制数转换为十进制数,可以计算幂级数(这里 8 的幂次可由表 1.2 得到,因为 $8^i = 2^{3i}$)。

例 1.8

$$\begin{aligned}1352_8 &= 1\times8^3+3\times8^2+5\times8+2\\ &= 512+3\times64+40+2\\ &= 746_{10}\end{aligned}$$

为了将十进制数转换为八进制数,可以先把它转换为二进制数,或者改用将十进制转换为二进制的第二种算法(更容易),用除以 8 代替除以 2。

例 1.9

$746/8 = 93$	余 2	产生 2
$93/8 = 11$	余 5	52
$11/8 = 1$	余 3	352
$1/8 = 0$	余 1	1352_8

因为将十进制转换为八进制比转换为二进制的工作量小，所以经常先将其转换为八进制，然后再转换为二进制，因此：

$746_{10} = 1352_8 = 001\ 011\ 101\ 110_2$

十六进制数($r = 16$)按4位分成一组，每一位数在从0到15的范围内，这里大于9的数字用字母表中的前六个大写字母表示：

10 A
11 B
12 C
13 D
14 E
15 F

例1.10 $1011101010_2 = 0010\ 1110\ 1010_2$
$= 2EA_{16}$

通过计算幂级数，可以将十六进制转换为十进制。

例1.11 $2E\ A_{16} = 2 \times 16^2 + 14 \times 16 + 10$
$= 512 + 224 + 10 = 746_{10}$

最后，为了将十进制数转换为十六进制数，重复除以16，其余数即为十六进制数字。

例1.12
746/16 = 46 余 10 产生 A
46/16 = 2 余 14 EA
2/16 = 0 余 2 $2EA_{16}$

本小节作业：

1. 看解题实例例3,4；
2. 做习题3,4。

1.1.2 二进制加法

在计算机和其他数字系统中，经常需要的一种运算是两个数的加法，在这一节，将描述二进制数相加的过程。

表1.4 二进制加法

0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 10(2，和为0，给下一位的进位为1)

为了计算两个二进制数的和，例如：

0 1 1 0 6
0 1 1 1 +7

一次加一位数(和十进制一样)，得到相加之和，和进到下一位的进位。正如十进制有加法表一

样，二进制也要有一个加法表，当然，它短得多(见表 1.4)。例 1.13 显示了二进制加法的分步演算过程。

例 1.13

```
   0
  0110
  0111
     1
```

下一步，右起第二个数字相加：

$$0 + 1 + 1 = 0 + (1 + 1) = 0 + 10 = 10$$

(和为 0，进位为 1)

或 $(0 + 1) + 1 = 1 + 1 = 10$

(相加的先后顺序无关紧要)

相加过程如下：

```
   10
  0110
  0111
    01
```

其最后两步是：

```
  11          01
  0110        0110
  0111        0111
   101        1101
```

注意，第三位相加时，共有三个 1 相加。其中一个是进位 1，另外两个 1 是原有的。所得结果为 3(二进制数字为 11)，即本位和为 1，进位为 1。两数相加之和，自然是十进制数 13。在本例中，最后一次相加时，产生一个进位，答案是 4 位。如果相加数更大，比方说是 13 + 5，答案就需要 5 位，如下面所示。最后的进位写为和的一部分。这和十进制加法一样，两个 4 位数相加，既可得 4 位数，也可得 5 位数。

```
   101
  1101    13
  0101     5
 10010    18
```

在有着几位(n 比特)字的计算机中，在运算过程中，若所得结果超出 n 位(两个 n 位正整数相加产生 n + 1 位得数)，这种情况称为“溢出”。两个 4 位正整数相加，其和大于或等于 16(即 2^4)时，便出现“溢出”现象。上面这个例子，相加之和为 18，大于 4 位正整数的最大值 15，所以出现溢出。

最低位相加(只有两个数)之后，随后的加法运算都有 3 个操作数。把输入进位称为 c_{in}，而相加所得进位称为 c_{out}。因此，加法问题便可表述如下：

c_{in}

a

b

c_{out} S

完整的加法过程见表 1.5。

表 1.5 一位加法器

a	b	c_{in}	c_{out}	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

用于一位运算的器件称为全加器。要是 4 位数相加，可以把 4 个全加器连接起来，如图 1.2 所示。注意，第 0 位加法器的进位输入为 0，因为在这一位没有进位。有时，对于这个位用一个较简单的电路(称为半加法器)。在后面设计全加器时，将再来讨论这个问题。

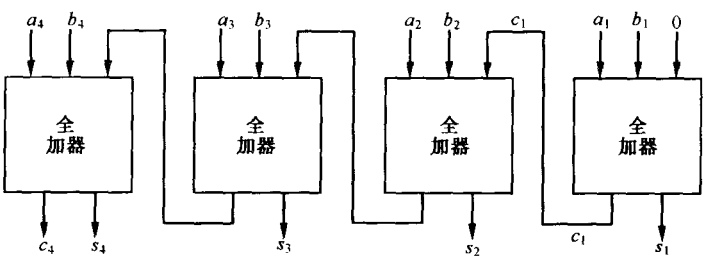

图 1.2 一个四位加法器

本小节作业：

1: 看解题实例例 5；

2. 做习题 5。

1.1.3 有符号数

迄今为止，只考虑正整数，有时称之为无符号数。计算机需要处理有符号数，即既有正数，又有负数。十进制中，人们熟识的记数法为有符号量值，如 +5 或 −3。这种办法也适用于计算机，用数字的第一位作符号标记(通常以 0 表示正，1 表示负)，其余各位为量值。因此，在 4 位系统中，正负可以表示如下：

+5 → 0101 −5 → 1101 −3 → 1011

因为只有 3 位可表示量值，可得数字在 −7 和 +7 之间。当然，大多数计算机使用更多位来存储数，因而其数值范围要大得多。注意，这种表示法既有正 0(0000)，也有负 0(1000)。虽

然这会引起混乱(起码是使计算机内部逻辑复杂化),但有符号数的主要问题是运算复杂。试看下列加法运算:

+5	-5	+5	-5	-3	+3
+3	-3	-3	+3	+5	-5
+8	-8	+2	-2	+2	-2

在头两个算式中,两个操作数的符号相同,只需将两个量值相加,保留符号不变。在其余例子中,首先得确定哪一个操作数量值大,可能是第一个操作数,也可能是第二个操作数。然后,用量值大的操作数减去量值小的操作数。最后,把量值大的那个操作数的符号加到所得数之前。在这四个例子中,是5-3。虽然这些运算都能完成,但所涉及的复杂硬件(一个加法器、一个减法器和一个比较器)带来另外一种解决方案。

二进制有符号数几乎总是存储成2的补码的形式。第一位仍然是符号位(0代表正)。正数(和0)以通常的二进制数存储,可存储的最大数为$2^{n-1}-1$($n=4$时,最大数为7)。因此,在一个4位系统中,+5可存储为0101。

负数$-a$,在n位系统中,则存储为相当于二进制的2^n-a。例如,-3存储为16-3=13,即二进制的1101。

可存储的最大负数为-2^{n-1},在4位系统中为(-8)。位数相同的情况下,2的补码最大数约为无符号数的一半,因为2^n个数中,有一半用于表示负数。除二进制外,这个方法也适用于其他基数,称为基的补码。负数$-a$,在n位数字系统中,存储为r^n-a。例如,在十进制中,就称为10的补码。在两位数时,-16可存储为84。0至49间各数为正数,50至99间各数为负数。

利用下列三步算法,比较容易得到作为2的补码的各个负数的存储方式。

1. 求得该量值等效的二进制数。
2. 对每一位求补(即把所有的0变为1,1变为0)。
3. 最后加1。

例1.14

	(a)	(b)	(c)
1.	5: 0101	1: 0001	0: 0000
2.	1010	1110	1111
3.	1	1	1
	-5: 1011	-1: 1111	0000

注意,没有负0;对+0求补码,所得结果为0000。在2的补码加法中,最高位的进位被略去。

表1.6列出了全部4位二进制数作为正数(无符号数)和作为2的补码的有符号数的实际意义。