这种类型的加法器需要让进位位以波纹方式通过电路。因此，我们称这个电路为行波进位加法器。每个进位位传递到下一个加法器都会引入一个小延迟，所以扩展这种设计来处理更多位会让电路变慢。在全部进位位都传播到位之前，电路的输出是不准确的。

7400系列IC提供了多个版本的4位加法器。如果你需要4位加法器，那么可以使用这样的IC，不必用单个逻辑门来构建加法器。

现在暂停一下，考虑考虑刚才讨论的内容的更广泛的含义。你已经学习了如何构建4位加法器，但这和计算有什么关系？回想一下，计算机是可以通过编程来执行一组逻辑指令的电子设备。这些指令包含了算术运算指令，而且我们刚刚看到了如何把用晶体管构建的逻辑门组合起来执行其中一种运算（加法运算）的。我们把加法作为计算机运算的一个具体例子进行了介绍，尽管本书不做详细讨论，但是你也可以用逻辑门实现其他基本的计算机运算。这就是计算机的工作方式——让简单的逻辑门协同工作来完成复杂的任务。

5.5 有符号数

到目前为止，本章只关注了正整数，但是如果我们也想处理负整数，该怎么办呢？我们需要考虑怎样在计算机这样的数字系统中表示负整数。计算机中的所有数据都被表示为0/1序列。负号既不是0也不是1，所以我们需要使用一种约定来表示数字系统中的负值。在计算机中，有符号数（signed number）是一个位序列，它可以用来表示负数或正数，具体取决于这些位的值。

设计数字系统时必须定义用多少位来表示一个整数。通常，我们用8位、16位、32位或64位来表示整数。这些位中的一个可以被分配用于表示负号。例如，如果最高有效位为0，那么该数为正数；如果最高有效位为1，那么该数为负数。其余的位则用来表示该数的绝对值。这种方式被称为原码表示。这是可行的，但是为了考虑有特殊含义的位，它会增加系统设计的复杂度。例如，考虑到符号位，我们之前构建的加法器电路就需要修改。

在计算机中，表示负数的更好方法被称为补码。在这种情况下，一个数的补码表示的是这个数的负数。确定数的补码的最简单方式是：把每个1用0代替，把每个0用1代替（即按位翻转），然后再加1。这里请注意：乍看之下，这似乎过于复杂，但如果你按照每一步进行，便会发现很简单。

让我们以4位数（数字5）为例，该数字用二进制表示为0101。图5-16展示了确定该数补码的过程。

图5-16 确定0101的补码

我们首先按位翻转，然后再加1，得到二进制的1011。因此，在这个系统中，5表示为0101，-5表示为1011。请记住，1011只是在4位有符号数的环境下才表示-5。稍后我们会看到，在不同的环境中，这个二进制序列有不同的解释。反过来，如果要确定负值的补码，该怎么做呢？过程是一样的，如图5-17所示。

图5-17 确定1011的补码

如同在图5-17中所看到的，求取-5的补码便又得到了原来的值5。这是有道理的，因为-5的负数是5。

练习5-2：确定补码

确定数字6的补码。

现在我们知道了如何用补码把一个数表示为正值或负值，但这有什么用呢？我认为了解这个系统好处的最简单的方法是尝试一下。假设我们想

把7与-3相加（即7减3）。我们的期望结果为+4。我们先确定输入的二进制表示，如图5-18所示。

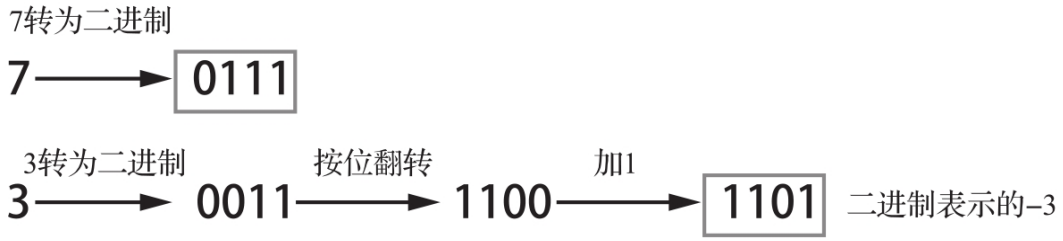

图5-18 确定7和-3的4位补码

我们的两个二进制输入是0111和1101。现在，暂时忘记我们处理的是正值和负值，只把两个二进制数相加。不用担心各个位代表的是什么，把它们相加就好，然后准备迎接惊喜吧！完成二进制运算后，请看图5-19。

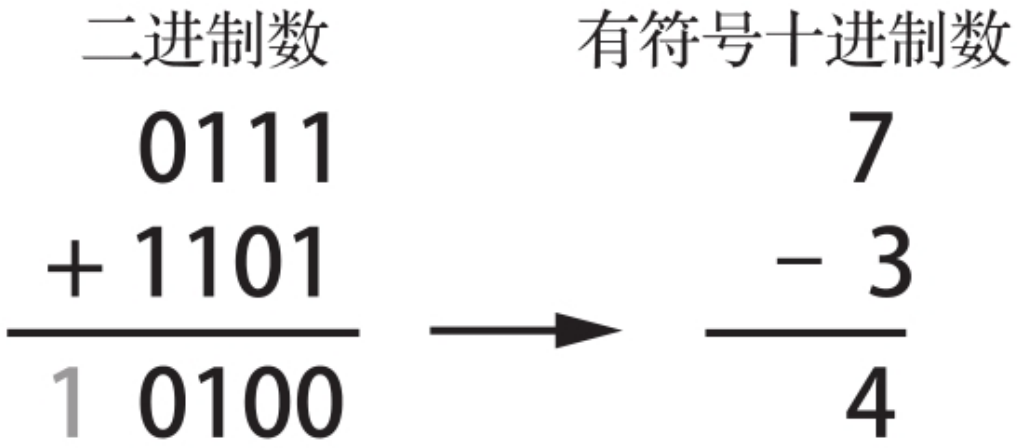

图5-19 将两个二进制数的加法解释为有符号十进制数的加法

如图5-19所示，这个加法运算产生的进位超出了4位数能表示的范围。稍后将更详细地解释这一点，但是现在，我们先忽略这个进位位。因此，便得到4位的结果0100，这就是我们期望的数字4！这就是补码表示法的美妙之处。在加法和减法运算过程中，我们不需要任何特殊的处理，它就能正常工作。

我们在这里暂停一下，先想想这件事的意义。还记得我们之前构建的那些加法器电路吗？它们也适用于负值！任何用于处理二进制加法的电路都可以使用补码作为处理负数或减法的手段。对所有这些工作原理的详细的数学解释超出了本书的范围，如果你对此感兴趣，网上有很好的解释。

补码

术语"补码"实际上是指两个相关的概念。补码是表示正整数和负整数的一种符号形式。例如，数字5用4位补码表示为0101，而-5则表示为1011。同时，补码也是一种运算操作，用于对以补码格式存储的整数取反。例如，取0101的补码就得到1011。

还有一种看待补码的方法：最高有效位的权重等于该位权重的负值，其他所有位的权重等于相应位的权重。因此，对于4位数而言，每位的权重如图5-20所示。

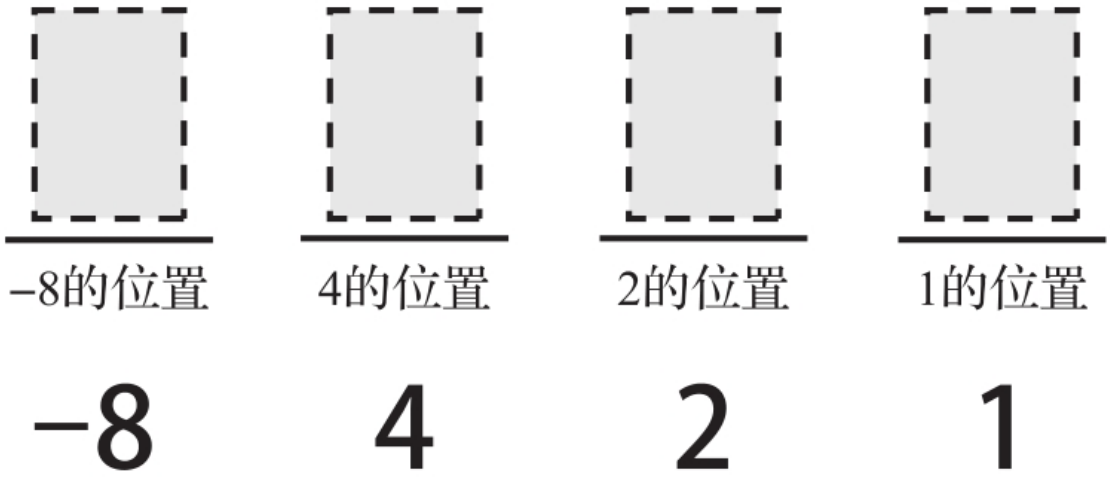

图5-20 用补码表示的有符号4位数的位值权重

如果我们把这种补码表示方法应用到1101（-3）上，我们就能计算出对应的十进制值，如图5-21所示。

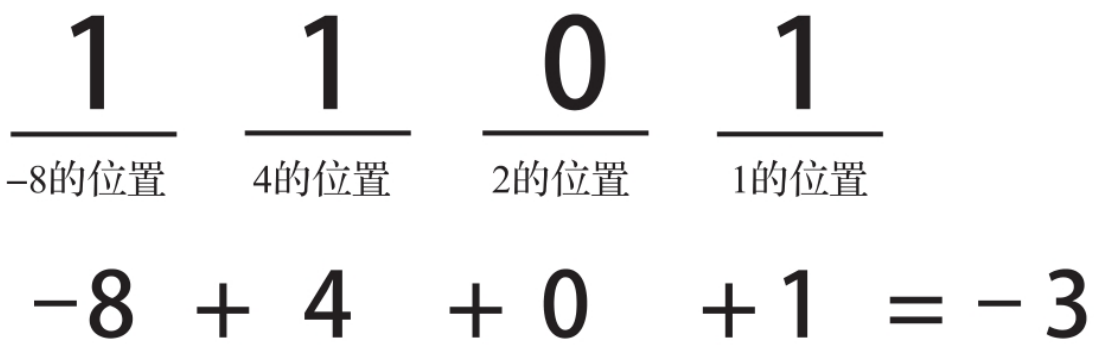

图5-21 使用补码的位值确定1101的有符号十进制值

在处理补码时，我发现把最高有效位的权重看作该位权重的负值是一种思维捷径。现在，我们已经讨论了4位有符号数中所有位置的权重，接下来我们可以检查用这样的数表示的全部值的范围，如表5-3所示。

表5-3 4位有符号数全部可能的值

二进制数	有符号十进制数	二进制数	有符号十进制数	二进制数	有符号十进制数
0000	0	0110	6	1100	–4
0001	1	0111	7	1101	–3
0010	2	1000	–8	1110	–2
0011	3	1001	–7	1111	–1
0100	4	1010	–6		
0101	5	1011	–5		

根据表5-3，我们可以观察到：对于4位有符号数，最大正值是7，最小负值是-8，一共有16个可能的值。请注意，当最高有效位是1时，数值为负。对于*n*位有符号数，我们可以概括如下：

-1

❑最大正值为（2^n）-1；

-1

❑最小负值为-（2^n）；

❑数值的总数为2^n。

对于8位有符号数，我们发现：

❑最大正值为127；

❑最小负值为-128；

❑数值总数为256。

5.6 无符号数

有符号数用补码表示负值，是处理负数的一种便捷方式，它不需要专门的加法器硬件。之前我们介绍的加法器对负值和正值都有效。但是，在计算机中有一些场景根本就不需要负值，把数字当成有符号数只会浪费大约一半的取值范围（所有的负值都不会用到），同时，还把最大值限制为可能可以表示值的大约一半。因此，在这种情况下，我们希望把数字看作无符号的，这意味着位序列总是表示正值或零，而绝不会是负值。

再看一下4位数，当我们把它解释为有符号数或无符号数时，表5-4给出了每个4位二进制值的含义。

表5-4 4位有符号或无符号数全部可能的值

二进制数	有符号十进制数	无符号十进制数
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	–8	8
1001	–7	9
1010	–6	10
1011	–5	11
1100	–4	12
1101	–3	13
1110	–2	14
1111	–1	15

对于*n*位无符号数，我们概括如下：

❑最大值为（2^n）-1；

❑最小值为0；

❑数的总数为2^n。

我们举个4位数的例子，如1011。查查表5-4，这个值代表什么？代表的是-5还是11？答案视情况而定！它既可以代表-5，也可以代表11，这取决于上下文。从加法器电路的角度来看，这无关紧要。对加法器来说，它就只是1011而已。无论怎样，加法运算都是按相同的方式来执行的，唯一的区别是我们怎么解释其结果。让我们看个例子。在图5-22中，我们把两个二进制数1011和0010相加。

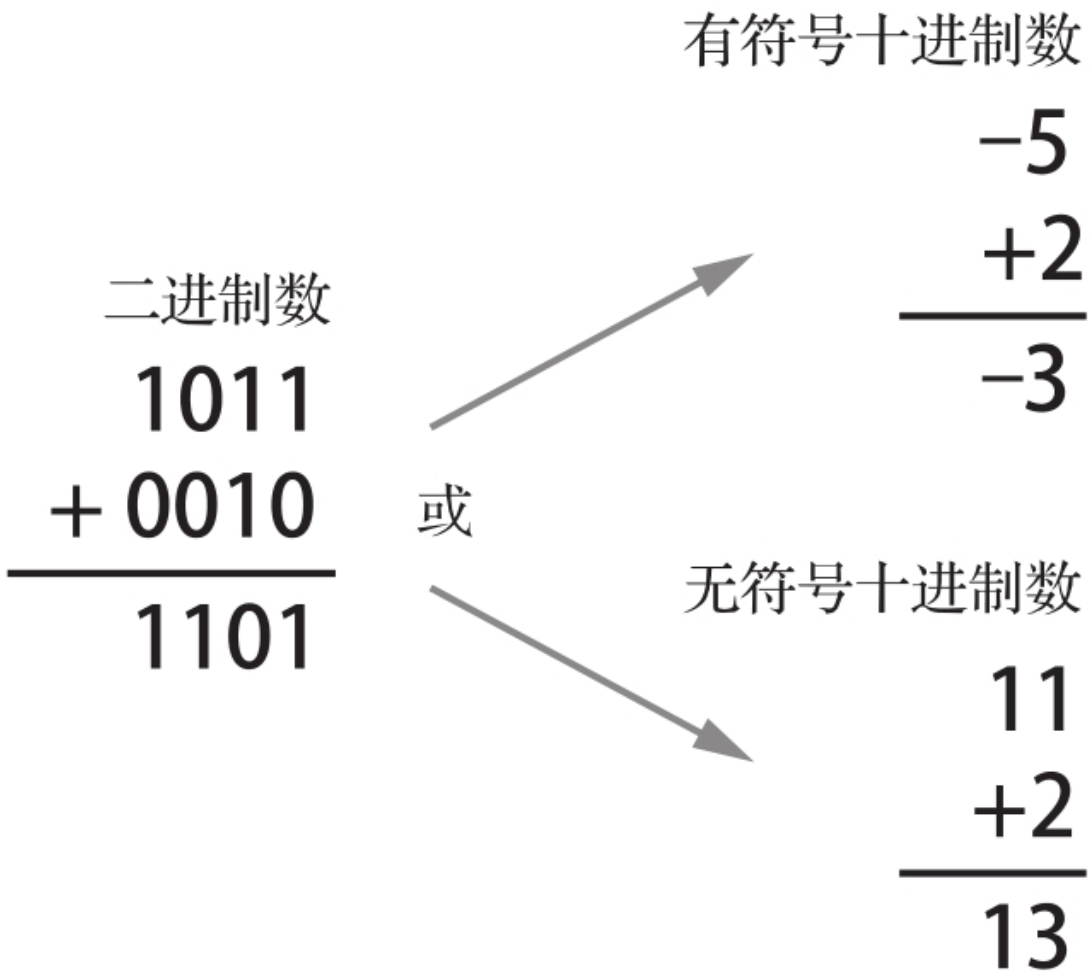

图5-22 两个二进制数相加（分别将它们解释为有符号数和无符号数）

如图5-22所示，无论使用的是有符号数还是无符号数，这两个二进制数相加的结果都是1101。在计算完成后，我们要决定怎样解释结果。要么是-5加2得到-3，要么是11加2得到13。任何一种情况下，算术运算都是对的，就是解读的问题而已！在计算机环境中，由在计算机上运行的程序负责正确解释加法运算的结果是有符号的还是无符号的。

练习5-3：两个二进制数相加并解释为有符号数和无符号数

将1000和0110相加。先把它们解释为有符号数，然后再解释为无符号数。结果说得通吗?

到目前为止，我们基本上忽略了最高位的进位，但我们应该理解它的含义。对于无符号数来说，进位位为1意味着发生了整数溢出。换句话说，就是结果太大，无法用分配的表示整数的位数来表示。对于有符号数，如果最高有效位的进位输入不等于其进位输出，那么发生溢出。同样，对于有符号数来说，如果最高有效位的进位输入等于最高有效位的进位输出，那么就没有发生溢出，可以忽略进位位。

整数溢出是计算机程序错误的来源。如果程序不检查是否发生溢出，那么加法运算的结果可能会被错误解读，从而导致意外行为。整数溢出错误的一个著名例子出现在街机游戏Pac-Man中。当玩家达到256级时，屏幕右侧就全是混乱的图形。之所以出现这种情况是因为等级数是用8位无符号整数表示的，其最大值255再加1就会导致溢出。游戏的逻辑没有考虑到这一点，因而会出现故障。

5.7 总结

本章以加法运算为例来说明计算机是如何基于逻辑门来执行复杂任务的。你学习了如何执行二进制加法运算，如何基于逻辑门构建硬件来执行二进制数的加法。你看到了半加器是怎样把两个位相加并产生一个和数位和一个进位位的，明白了全加器可以实现两个位和一个进位位的加法运算。我们讨论了如何组合针对单个位的加法器以执行多个位的加法运算。你学到了如何在计算机中用有符号数和无符号数表示整数。

第6章将介绍时序逻辑电路。通过时序逻辑，硬件可以有存储器，以进行数据的存储和检索。你将看到存储器电路是如何构建的。我们还将讨论时钟信号，这是一种同步计算机系统中多个组件状态的方法。

设计5：搭建半加器

在本设计中，你将用XOR门和AND门来搭建一个半加器。输入由开关或按钮控制。输出连接到LED以方便观察它们的状态。对于这个设计，你需要下列组件：

❑面包板；

❑两个LED；

❑和LED一起使用的两个限流电阻（约为220Ω）；

❑跨接线；

❑7408 IC（含4个AND门）；

❑7486 IC（含4个XOR门）；

❑两个适合面包板的按钮或开关；

❑两个470Ω的电阻；

❑5V电源。

有关7408 IC引脚编号的提示，参见图4-14。7486 IC之前没有讨论过，所以我在图5-23中给出了它的引脚图。

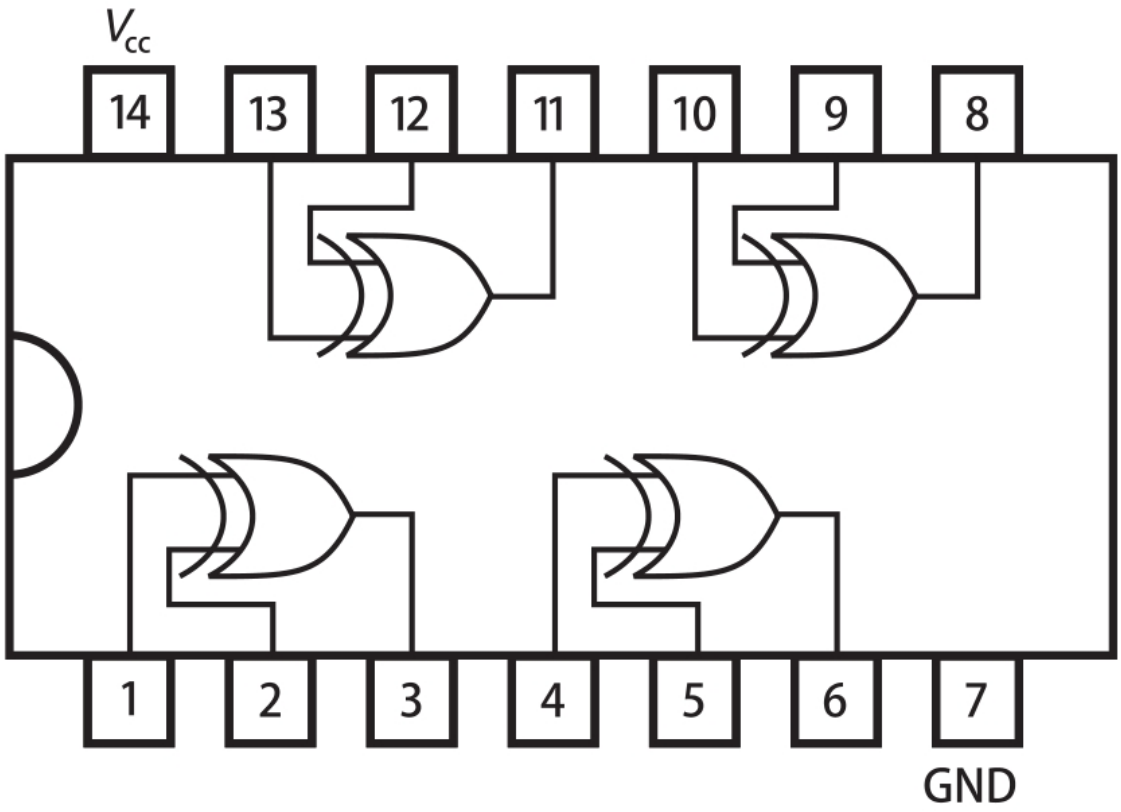

图5-23 7486 XOR IC引脚图

图5-24提供了半加器的连线图。关于如何搭建这个电路的更多细节请参阅之前的图。

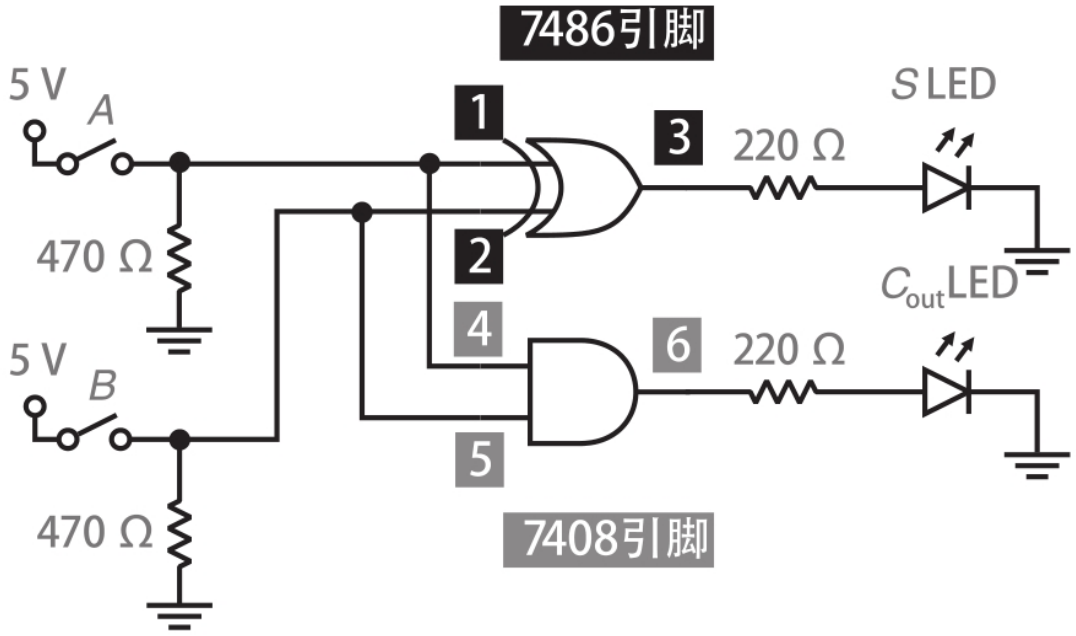

图5-24 用XOR和AND门搭建半加器的电路图

图5-24展示了带有下拉电阻的开关和带有限流电阻的LED的连接。还要注意7486和7408 IC的引脚编号——用方块表示。注意把*A*和*B*连接到电阻和IC的连线上的黑点。这些黑点代表连接点，例如，开关A、470Ω电阻、7486 IC的引脚1以及7408 IC的引脚4都是连接起来的。不要忘记通过引脚14和引脚7（图5-24中未显示）把7486和7408 IC分别连接到5V和接地。

图5-25展示了在面包板上实现的电路。

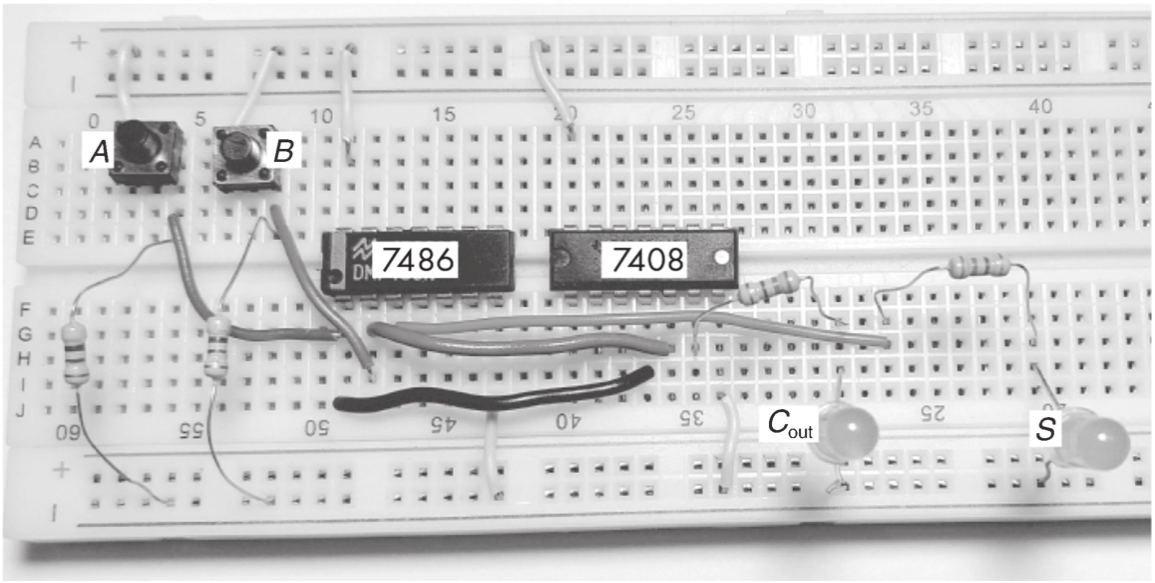

图5-25 用XOR和AND门搭建的半加器的实物图

电路搭建完成后，请尝试输入*A*和*B*的所有组合，以确认输出与半加器真值表（见表5-1）给出的预期值相匹配。

第6章

存储器和时钟信号

在前面的章节中，我们看到了如何组合数字逻辑门以产生有用的组合逻辑电路，其输出是输入的函数。本章将讨论时序逻辑电路。这些电路有存储功能，能够保存过去的记录。我们将介绍一些特殊类型的存储设备：锁存器和触发器。我们还将介绍时钟信号，它是一种同步多个电路元件的方法。

6.1 时序逻辑电路和存储器

现在，我们来看一种被称为时序逻辑电路的数字电路。时序逻辑电路的输出不仅取决于其当前的一组输入，还取决于该电路过去的输入。换句话说，时序逻辑电路对自己以前的状态有一定的了解。数字设备在存储器中存储过去状态的记录，存储器是用来保存和检索二进制数据的组件。

让我们考虑一个简单的时序逻辑：投币式自动贩卖机。自动贩卖机至少有两个输入：投币口和自动贩卖按钮。为了简单起见，我们假设自动贩卖机只出售一种商品，且该商品价格为一枚硬币。除非插入硬币，否则自动贩卖按钮不会执行任何操作。如果自动贩卖机是基于组合逻辑的，其状态只取决于当前输入，那么就必须在插入硬币的同时按下自动贩卖按钮。

幸运的是，自动贩卖机不是这样工作的！它们有存储器，能跟踪硬币是否已经插入。当我们按下自动贩卖按钮时，自动贩卖机中的时序逻辑会检查其存储器以查看之前是否有硬币插入，如果有，机器就会分配商品。本章后面的内容将以这个时序逻辑示例为基础。

由于有存储器，因此实现时序逻辑是可能的。存储器保存二进制数据，其存储容量用位或字节来衡量。现代计算机设备（比如智能手机）通常具有至少1GB的存储器。这是超过80亿位！我们从更简单的具有1位存储容量的存储设备开始介绍。

6.2 SR锁存器

锁存器是一种记忆一位的存储设备。SR锁存器有两个输入——*S*（用于置位）和*R*（用于复位），以及一个被称为*Q*的输出——"被记忆的"单个位。当*S*被设为1时，输出*Q*也变为1。当*S*变为0时，*Q*还是等于1，因为锁存器记得这个之前的输入。这就是存储器的本质——存储器组件记得之前的输入，即使该输入发生了变化。当*R*被设为1时，指示复位/清除该存储器位，所以输出*Q*变为0。就算*R*变回0，*Q*也将保持为0。

我们在表6-1中总结了SR锁存器的行为。

表6-1 SR锁存器的行为

S	*R*	*Q*（输出）	行为
0	0	保持之前的值	保持
0	1	0	复位
1	0	1	置位
1	1	X	无效

根据设计，同时把*S*和*R*设置为1，输入无效，此时，*Q*的值未定义。实际上，尝试这样做会导致*Q*变为1或者0，但我们不能确定是哪一个。此外，尝试同时置位和复位锁存器是没有意义的。SR锁存器的电路图符号如图6-1所示。