第13章　如何实现减法

在你确信继电器可以连接起来以构成二进制加法器后，你可能会问：“减法器如何实现呢？”本章将会为你解答这个问题，且提出这个问题也表明你有了一定的理解力。减法和加法在某些方面是互为补充的，但两种计算的机制不同。加法从最右边一列向最左边一列计算，每一列的进位都加到下一列中去。减法不用进位，相反，要用到借位——一种本质上与加法不同的机制。

例如，让我们看一道典型的不断借位的减法题目：

```
  253
- 176
-----
  ???
```

要做这道题，从最右边一列开始。首先，6比3大，所以需要从5借1，这样就变成了13减6，结果是7。由于从5借了1，5就变成了4，4比7小，所以继续从2借1，14减7等于7。2被借1后成为1，1减1为0，所以最后结果是77：

```
  253
-176
-----
   77
```

如何用逻辑门来实现这看似不合常理的逻辑呢？

我们不会直接用这种方法，代替的是用一个小技巧，使不通过借位来实现减法。这会是一个使大家都满意的好办法。详细地了解减法的完成是很有用的，因为它和用二进制编码在计算机中存储负数的机制有很大联系。

为解释这样的工作，需要清楚地指明两个操作数，即减数和被减数。减数从被减数中去掉后，结果是二者之差：

```
  被减数
-  减数
-------
    差
```

要想不借位，首先将减数从999中减去：

```
  999
-176
-----
  823
```

这里用999是因为操作数是3位，如果是4位数，就用9999。把一个数从一串9中减去得到的结果称为9的补数或补码。176的9的补数是823，反之，823的9的补数是176。这样做的好处在于，无论减数是什么，计算9的补数永远不需要借位。

在计算出减数的9的补数之后，把它加到原来的被减数上：

```
  253
+ 823
------
 1076
```

最后，你再加１并且减去1000：

```
  1076
+    1
-1000
------
   77
```

这样就得到结果了。答案和以前一样，且你根本不用借位。

这是什么原理呢？原来的减法题目是：

253－176

表达式加一个数再减同一个数得到的结果是一样的。所以先加上 1000，再减去1000：

253－176 + 1000－1000

这个式子等同于下面的式子：

253－176 + 999 + 1－1000

再按如下方式重新组合：

253+（999-176）+1－1000

这与前面描述过的用9的补数进行的计算是一致的。虽然用了两个减法和两个加法来代替一个减法，但是也因此省去了讨厌的借位。

但是，如果减数比被减数大怎么办呢？例如如下计算：

```
 176
-253
-----
 ???
```

通常情况下，你看到这个式子后可能会说：“减数比被减数大只需交换两数位置，再做减法，然后给结果取个相反数。”于是你在脑子里交换了它们的位置，并求出了答案：

```
 176
-253
-----
-77
```

要省去借位来做这道题和前面的例子有所不同。首先你要求出 253的9的补数，即

```
 999
-253
-----
 746
```

再把该补数和原来的被减数相加：

```
 176
+746
-----
 922
```

这时候，按照上一道题的步骤，你应该对其加1再减去1000，但在本题中，这种方法不会生效。如果你还按这种步骤做，就需要从923中减去1000，这又导致了借位。

既然实际上前面已经加了999，这里再减去999：

```
  922
-999
-----
  ???
```

当做到这一步时，可看出结果是个负数，故需要交换两数位置，不过这样再做减法时已不需要借位，答案如预期所料：

```
  922
-999
-----
 -77
```

同样的方法可用于二进制数减法，而且会比十进制数减法来得简单。让我们看看该如何做。

原来的减法题目是：

```
  253
-176
-----
  ???
```

当把这些数转化为二进制数时，问题变成：

```
  11111101
-10110000
---------
  ????????
```

步骤1 用11111111减去减数：

```
  11111101
- 10110000
----------
  01001111
```

当计算十进制数减法时，减数是从一串9中减去，得到称为9的补数的结果。对于二进制数减法，减数从一串1中减去，差称为1的补数。但请注意，求1的补数实际上并不需要做减法，因为1的补数中，原来的0变成1，原来的1变成0，所以，1的补数有时也称为相反数或反码。（你是否还记得第11章中反向器的作用是把0变成1，把1变成0。）

步骤2 把步骤1中求得的补数和被减数相加：

```
  11111101
+ 01001111
----------
 101001100
```

步骤3 对结果加1：

```
 101001100
+         1
----------
 101001101
```

步骤4 减去100000000（256）：

```
  101001101
- 100000000
-----------
    1001101
```

该结果就是十进制数77。

现在把两数颠倒位置后再做一遍。在十进制中，减法题目对应于：

```
  176
-253
-----
  ???
```

而在二进制中，即是：

```
  10110000
- 11111101
----------
  ????? ???
```

步骤1　从11111111中减去减数。得到补数：

```
  11111111
- 11111101
----------
  00000010
```

步骤2　把步骤1中的补数和被减数相加：

```
  10110000
+ 00000010
----------
  10110010
```

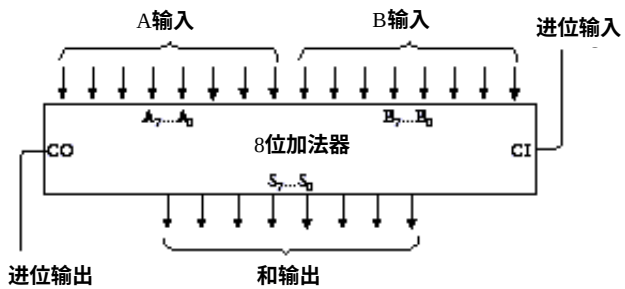
现在，11111111必须再从结果中减掉。当减数比被减数小时，可以通过先加　1再减去100000000来达到此目的。但现在这样做却会用到借位。所以，我们先用　11111111减去步骤2中的结果：

```
  11111111
-10110010
---------
  01001101
```

这实际上是对步骤2中得到的结果取反。最后的结果是77，而真正的答案应该是－77。

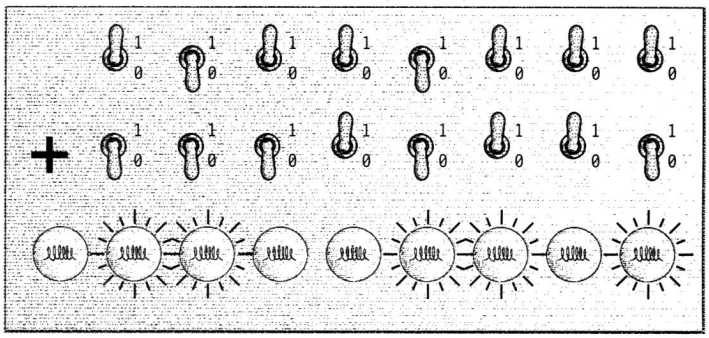
现在，已经可以改进加法机使它既能执行加法操作亦能执行减法操作。为使简便起见，这个加/减法机只执行被减数大于减数的减法操作，即差为正数的操作。

该加法机的核心部件是由逻辑门集成的8位全加器：

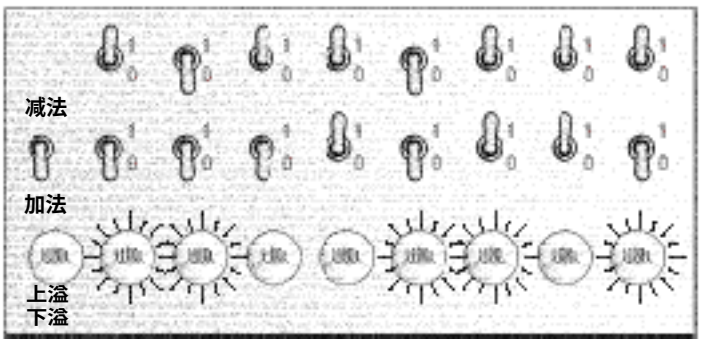


前面讲过输入 A_0～A_7及B_0～B_7连接到开关上，用于表示 8位操作数。进位输入端接地。S_0～S_7连接8个灯泡，用于表示加法的和。由于和可能会是 9位数，进位输出端也连了一个灯泡。

控制面板如下图所示：

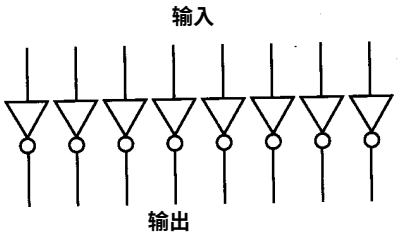


上图中，开关被设为 183（或10110111）和22（或00010110），产生的结果是 205或11001101)。用于加/减法的新的控制面板有一点儿修改，它包含了一个用于选择做加法还是做减法的额外开关。

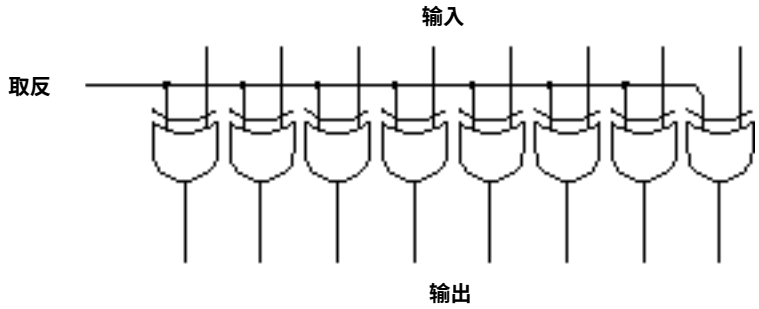


如图所示，当这个开关向下时表示选择加法运算，反之是选择减法运算。此外，只有最右边的8个灯泡用于表示结果，第九个灯泡用来标识上溢 /下溢，它指明了一个不能用 8个灯泡表示的数。当加法操作得到的和大于 255（称为上溢）或减法计算中出现一个负数（下溢）时，这个灯泡就会亮。减数比被减数大时，结果就是一个负数。

这个加法机主要增加了一个求 8位二进制数的补数的电路。由于一个数的补数就是取其每一位的相反数，所以这个电路看起来很简单，就是 8个反向器而已。



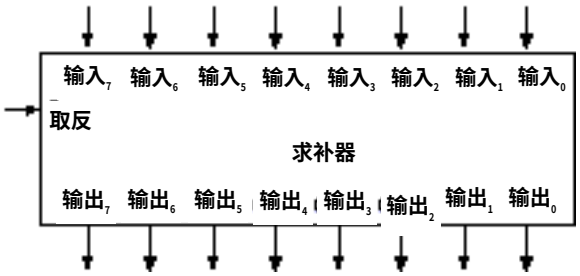
该电路存在一个问题，就是它不分情况地对输入求反。我们需要一台既能做加法又能做减法的机器，而此电路只有做减法时才取反。对它进行一下改进，如下图所示：



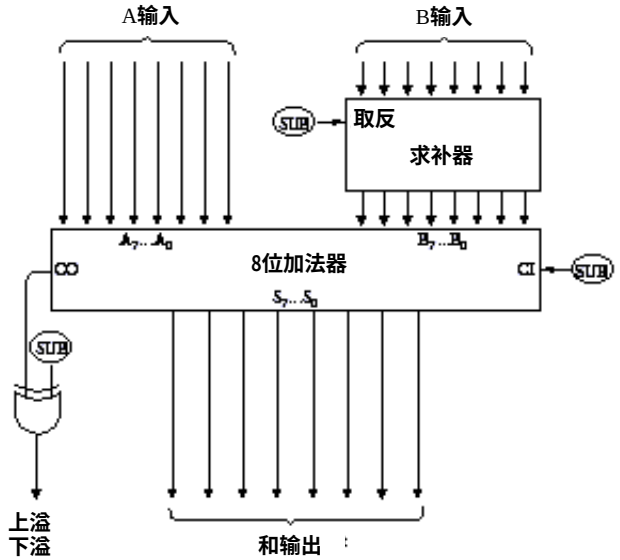
图中标识为“取反”的信号输入到每一个异或门中。回忆一下异或门的功能：

XOR	0	1
0	0	1
1	1	0

如果“取反”信号为 0，则异或门的 8个输出和 8个输入是相同的。例如，如果输入是01100001,则输出也是 01100001；若“取反”信号为 1，则输出取反。例如，当输入是01100001时，输出为10011110。让我们把8个异或门集成到一个盒子里，称为求补器：



求补器、8位加法器及一个异或门可按下图连接：



注意上图中有3个信号都标识为“ SUB ”，这是加/减法转换开关。当该信号为0时做加法，为1时做减法。做减法时， B输入在送入加法器之前先求补。此外，做减法时，通过设置加法器的进位输入端 (CI)为1，使由加法器得到的结果加 1。对加法而言，求补电路没有起作用，CI输入也就是0。

“ SUB ”信号及加法器的CO输出作为异或门的输入来控制表示上溢 /下溢的小灯泡。如果“ SUB ”信号为0（表示做加法），则当CO输出为1时灯泡点亮，这表示加法的和大于255。

当做减法时，如果被减数大于减数，则加法器的 CO端正常输出1，这表示在减法的最后一步中要减去100000000。所以，只有当加法器的 CO输出为0时，上溢/下溢灯泡才被点亮。这时减数大于被减数，差是个负数。上面这个加 /减法器现在还不能表示负数。

你一定兴致勃勃地想知道该如何实现减法了。

本章一直在谈论负数，但没有指出二进制负数的表示方法。你可能会认为它的表示和十进制负数一样，只需在数的前面加个负号。例如，－ 77在二进制中写成－1001101。你当然可以这么表示，但别忘了用二进制数的目的在于只用 0和1表示所有的东西，当然也包括一个小小的负号了。

你可以用某一位代替负号，当该位为 1时就表示负数，为0时表示正数，这似乎也是可行的。但还有一种方法，它不仅能表示负数，而且还很适于把正数和负数相加到一起。这种方法的不足之处是你必须提前决定数字需要多少位。

通常用来表示正、负数的方法的好处是这种方法能表示所有的正数、负数。我们把 0想象成向一个方向延伸的无穷的正数流和向另一个方向延伸的无穷的负数流的中点：

… －1 000 000 -999 999… －3 －2 －1 0 1 2 3… 999 999 1 000 000 …

但是,如果并不需要无限大或无限小的数，而是完全可以确定计算中所遇到的数的范围，情况便有所不同了。

下面来看看帐户的例子，人们有时可以在帐户上看到负数。假设帐户上从来没有超过$500的存款，而银行给我们的预支额是$ 500，这就意味着帐户上的数字在$499～－$ 500之间。假设我们不会一次取出 $500，也不会写一张超过 $500的支票，同时我们只处理美元，而不考虑到更小的货币单位 —— 美分。

这些假设表明帐户能处理的数字范围是从－ 500～499，总共1000个数。这个限制暗示我们只能用3位十进制数，且可不用负号来表示这 1000个数。其中的关键在于我们不需要 500～999之间的正数，所以它们就可以用来表示负数。下面是其工作原理：

用500表示－500
用501表示－499
用502表示－498
⋮
用998表示－2
用999表示－1
用000表示0
用001表示1
用002表示2
⋮

用497表示 497

用498表示498

用499表示499

换句话说，以5、6、7、8、9开头的3位数实际上都表示负数。不用如下的表示法：

－500 －499 －498 … －4 －3 －2 －1 0 1 2 3 4 … 497 498 499

而用这样的表示法：

500 501 502 … 996 997 998 999 000 001 002 003 004 … 497 498 499

注意这样形成了一个环形排序，最小的负数（500）看上去是最大的正数（499）的延续。数字999是比零小的第一个负数。如果给999加上1，通常得到1000。但由于只处理3位数，所以实际上是000。

这种处理称为10的补数。要把3位负数转换成10的补数，需从999中减去它再加1。换句话说，10的补数是9的补数再加1。例如，要把－255写成10的补数，应先从999中减去255得到744，再加上1后得到745。

你可能听说过“减法不过是负数的加法”，你也可能回答过“其实还是不得不做减法”。然而，通过使用10的补数，就不用去做减法了，全部都可以用加法来计算。

假设你有余额为$143的帐户，并写了一张$78的支票，这表明你要把－78加到143上。－78的补数是999－78+1，即922。所以新的余额是143＋922（忽略上溢），即65。若我们再写一张$150的支票，则必须减去150，用补数表示就是850。先前的余额065加上850等于915，所以，新的余额实际上是－$85。

二进制中对应的系统称为2的补数。假设我们用8位二进制数工作，范围从00000000～11111111，对应于十进制的0～255。这时如果你想要表达负数，则以1开头的每个8位数都表示一个负数，如下所示：

二进制数	十进制数
10000000	－128
10000001	－127
10000010	－126
10000011	－125
⋮	
11111101	－3
11111110	－2
11111111	－1
00000000	0
00000001	1
10000010	2
⋮	
01111100	124
01111101	125
01111110	126
01111111	127

你可以表示的数的范围从－128～127。最左边的一位称为符号位，1表示负数，0表示正数。

要计算2的补数得先求出1的补数再加上1，这等同于先求反再加1。例如，十进制数125是01111101，要用2的补数来表示－125，可先取反得10000010，再加1就得到10000011。可用上表来验证这个结果。要回到原来的数只需同样的操作：取反后加1。

这个系统使不用负号就能表示正、负数，它也使我们只用加法规则就可以随意进行正、负数运算。例如，计算－127+124，利用上表即得

```
  1 0 0 0 0 0 0 1
+ 0 1 1 1 1 1 0 0
-----------------
  1 1 1 1 1 1 0 1
```

和是十进制的－3。

这里要注意上溢或下溢，即结果大于127或小于－128的情况。例如，125加125：

```
  0 1 1 1 1 1 0 1
+ 0 1 1 1 1 1 0 1
-----------------
  1 1 1 1 1 0 1 0
```

因为最高位是1，结果代表一个负数：－6。再看－125加上它自已：

```
  1 0 0 0 0 0 1 1
+ 1 0 0 0 0 0 1 1
-----------------
1 0 0 0 0 0 1 1 0
```

由于限制了只取8位数，所以最左边的1被扔掉，剩下的8位表示6。

一般而言，若两个操作数的符号相同，而结果的符号与操作数的符号不相同时，这样的加法是无效的（即加法运算产生了溢出！）。

现在，二进制数可以有两种不同的使用方法。二进制数可以是无符号的或有符号的，无符号的二进制8位数的表示范围从0～255，有符号的二进制8位数的表示范围从－128～127。这些数本身不会告诉你它们是否带有符号。例如，假设有人问：“10110110对应于十进制数的几？”这时，你必须先问清楚它是无符号数还是有符号数？它可能是182或－74。

这就是二进制数的麻烦：它们仅仅是一些0和1而没有告诉它们的任何含义。