计算机是如何做加法的

该文章为网友“国栋”撰写，发表在开源中国的个人博客中。个人仅作记录重编，学习使用。

其讲解思路比教科书上的更自然，逻辑也很紧密，很适合计算机学科上的入门读物。

一、构建多位加法器

原文地址：http://my.oschina.net/goldenshaw/blog/411992

计算机做加法是对人做加法的模拟。那么人是怎么做加法的呢？让我们来考察一下。

人做加法的过程

从一般的情况出发，比如怎么计算“24+35”呢？

| 我们把个位与个位相加，4+5=9，再把十位与十位相加，2+3=5，再合起来得到 59。

这就是所谓的分而治之（divide and conque）了，用打仗的话来说，也可以说是各个击破。

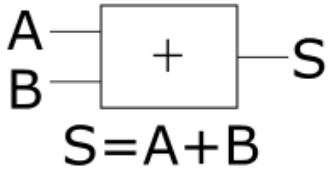
显然，会做两个多位数加法的基础是会做两个一位数加法。那么，问题又来了，如何做两个一位数的加法呢？

| 其实，你靠的是记忆！我们会在后面再去深入探讨这一问题。

自顶向下

目前，我们考虑的是如何构造多位加法器，重点是如何去模拟多位加法器的工作过程。

对于两个一位数加法的问题，我们暂且认为：经过反复学习，在我们的大脑中形成了一个关于一位数加法的神经网络，它的功能可由以下的一个抽象接口模型来描述：



它能接受两个一位数，并快速给出相加的结果。以此为基础，多位数的加法可以分解成多次一位数的加法，并把结果合并起来。

先不考虑一位加法器的实现细节，相反，假定它已经有了我们想要的功能，我们现在先考虑如何用它构建高级的多位加法器。

| 这种思路我们称之为“自顶向下（top down）”，先把目光聚焦在高层的结构上。

| 一位加法器是为多位加法器服务的，从服务对象考虑起能让我们更好地确定底层接口的规范。

| 假定底层已有我们想要的功能，这种思路又称为“wishful thinking（按愿望思维）”。

| 它避免了我们过早地纠结在底层实现的细节上去。

关于进制与表示的问题

有人可能会想，人做加法是用十进制，计算机或者说电路做加法是用二进制，这样去类比可行吗？是不是应该先讲讲二进制的知识先呢？

| 进制差异自然会带来影响，特别越到底层，影响越显著。

| 不过，我们将看到，进制的选择暂时不会影响高层的结构。

| 对于我们而言，十进制更加自然，到后面，会逐步转到二进制上去讨论。

至于数字的表示问题，10 进制的一大麻烦就是要表示的状态太多了，既然要用电路来实现，首先很容易想到用电压来表示数字。比如数字 1 就用 1V 的电压来表示，同理，数字 9 就用 9V 来表示。

| 自然，这种表示不是唯一的，你也可以用 0.1V 来表示 1，那么相应地用 0.9V 来表示 9，

| 关键或许在于保持这种比率关系。

两种策略

考虑前面人做加法的例子：24+35。

串行

一种策略就是只用一个一位数加法器，分成多次，每一次把同一位上的两个数扔给这个一位加法器去运算：

| 比如先把个位的 4 和 5 扔进去，得出一个结果 9，存起来；

| 再把十位上的 2 和 3 扔进去...

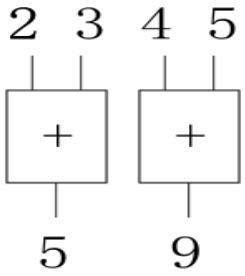
当然，这种方式的挑战在于如何存储中间结果，如何依次地把数据送进去，另外还有速度的问题等等。

| 打造出一个如此的加法器几乎相当于一个初级计算机的雏形了。

显然，人做多位加法的过程也是串行式的。

并行

那么，一种更加简单粗暴的策略就是把几个一位加法器拼接起来，同时进行计算，一方面无需关注存储，依次送入数据等问题，另外速度上也更快了。



| 24+35，按照我们的习惯，左边是高位（十位），右边是低位（个位）。

如果需要更多的位数，那么简单放置更多的一位加法器就完了。这种方式需要更多的硬件，但我们也因此得到了好处。

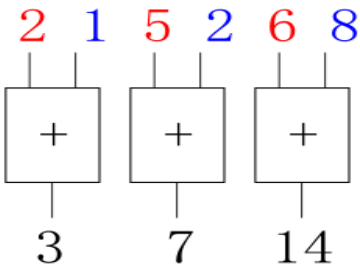
当然了，无限的并行下去也是不行的，最终我们要的是一种中庸之道，平衡之术，盲目推崇或排斥某种方式往往都是不可取的。比如一个 32 位机如何做 64 位的加法呢？最终还是靠串行做 2 次 32 位加法来实现。

进位的处理

当然，有一个很重要的问题被有意无意地忽略了，那就是进位。

考虑以下多位数的加法：“256+128”，

那么，通过组合三个一位数加法器，分别处理个位，十位与百位数字的相加，结果如下：



显然，3714 不是我们想要的结果，因为没有考虑到进位的影响。正确的结果应该是 384。

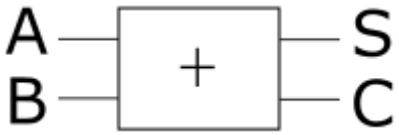
到目前为止，我们关于一位加法器的构想还是很粗糙的。首先，从最终结果的表示层面而言，按前面约定的表示，输出应该限制在 0V-9V 的区间。

如果是 0V，按电路上一般的约定，可用一个接地符号来表示：

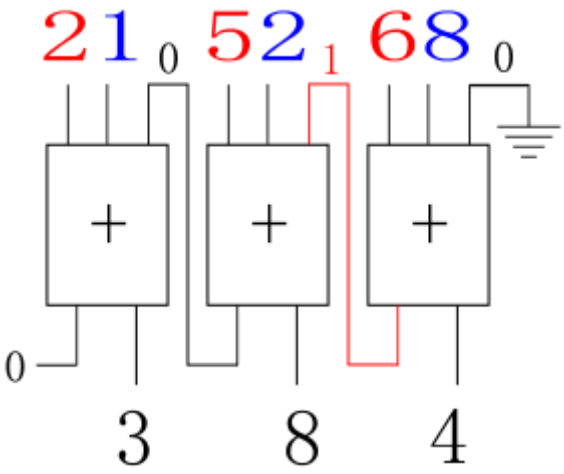
对于上述个位数加法而言，那么就是输入一个 6V 和一个 8V 的电压，结果输出的 14V 电压，这不是我们想要的结果。

| 显然，上述例子中，个位中只需要输出 4V 即可，而不是 14V，14 需要拆分成 1 和 4。

现在需要修正一位加法器的原型。首先至少要两个输出，一个是加位（sum，S）的输出，一个进位（carry，C）的输出：



其次，进位的输出还要参与到下一级的输入中去，也即要有三个输入，输出则只要显示加位的结果即可：



| 对于右边最低位，不存在进位，使用一个接地符号表示输入为 0V.

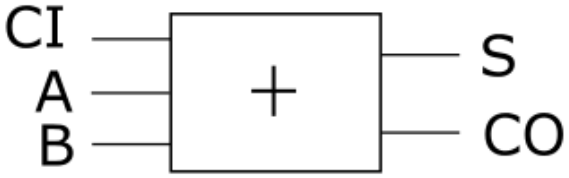
| 其余的进位输入则来自低位的进位输出。

| 图中红色的线表示产生了进位。对于中间的十位数加法，实际是 5+2+1=8

经过这样的调整，多位加法器就能正常工作了，但这样一来，模型也复杂多了。

全加器

综合进位及显示的需要，那么，一个具有普遍性的一位加法器至少要三个输入，其中有一个进位输入（Carry In）；以及两个输出（其中，CO 指 Carry out，进位输出）：



| 这样的一个东西，我们称之为全加器（Full Adder，FA）。

前面说到，如果从高层考虑起，能让我们更好地确定底层接口的规范，我们现在的确得到了一个更明确的接口。

| 如果一开始就一头猛扎进去实现最初设想的一位加法器原型，

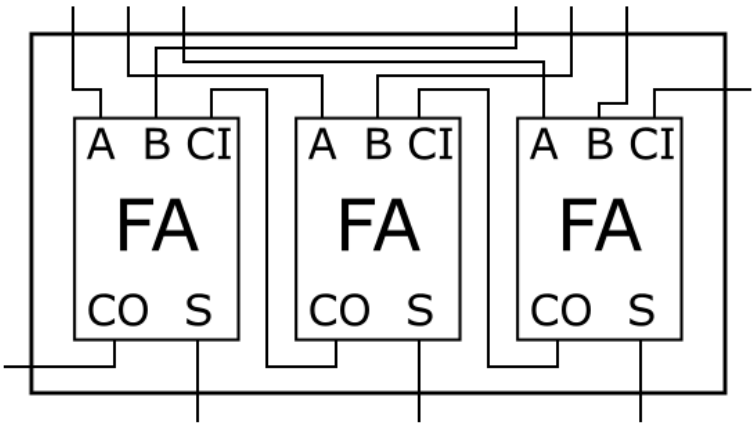
| 恐怕到构建多位加法器时又不得不返工了。

现在尽管我们不知道怎么去实现它，但我们却更清楚它应该是怎样的。

| 你心中有个 Big Picture（大图景），你更加不容易走偏。

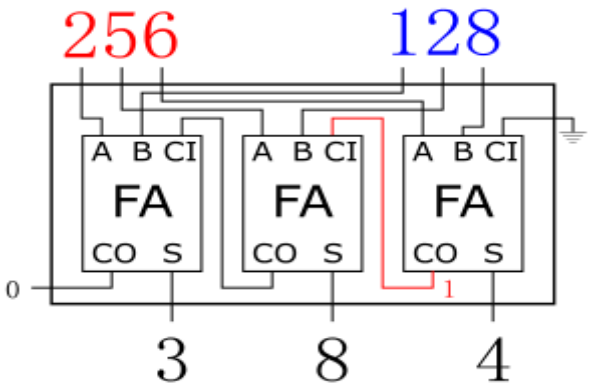
多位加法器

然后，以此为基础，把三个一位加法器封装在一起，把同一个数的三个位放在一起，得到以下一个三位加法器的原型：



| 走线就有点随意了，我也不是什么专业绘电路图的，大家能看明白原理就行了。

执行上述加法的过程如下：

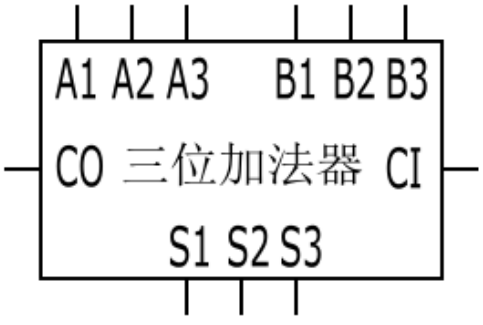


| 通过梳理线路的走向，就无须像前面那般两个数字交叉在一起很不直观。

| 线未必非得要走到一块，有种说法是“接口要对用户友好”，

| 无论你是设计硬件还是软件的接口，都应该记住这一点。

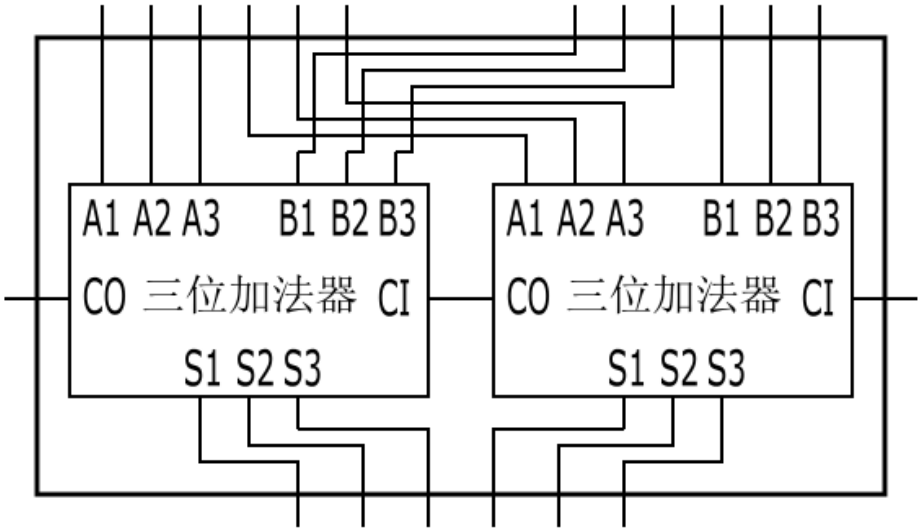
明白了它的内部实现，之后，再提到它时，只需要一个简化的接口模型：



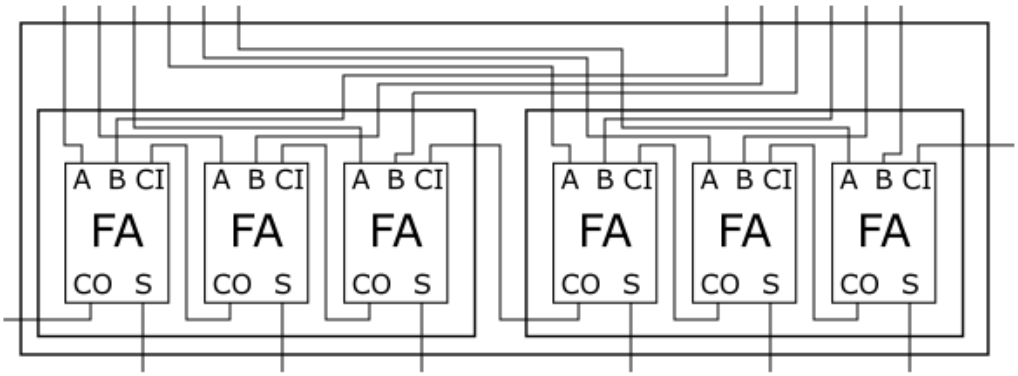
那么，这就是搞软件的人口中所谓的抽象呀，封装呀，提供接口呀，隐藏实现呀，blabla...

模块化

以它为基础，我们又可以构建比如一个 6 位加法器，在它里面封装了两个三位加法器：



如果想呈现全部的细节，那么就像下面这样了，线很多，但道理还是一样的：



| 当然也可以跳过三位加法器，直接构建在 6 个全加器基础上。

| 本质上还是一样，不过少了层封装。

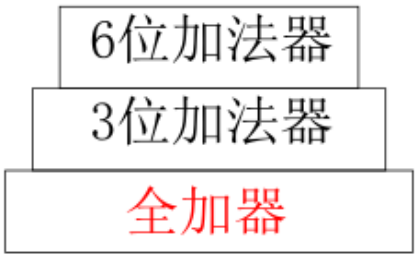
自然，按照同样的思路，你还可以打造 12 位加法器。当然你也可以先造个 2 位的加法器，然后再一层一层弄出 4 位，8 位的等等。你甚至可用一个 2 位加一个 3 位来造一个 5 位的加法器，有何不可呢？随便你怎么去组合。

| 搞软件的同学是不是又想到了什么分层次呀，模块化呀，然后又是一大堆理论，大道理，blabla...

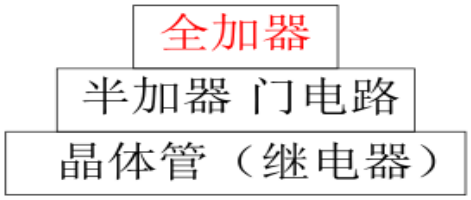
| 这些原则其实是软硬通吃的。

层次性

当然，以上这些最终都建立在全加器功能的基础上。



而后面我们将看到，全加器它又是建立在半加器等的基础之上。



| 构造一层又一层的抽象，这就是我们应付复杂性的重要手段。

当不知道全加器是如何工作时，我们可能会觉得不够踏实，似乎一切建立在无源之水，无本之木之上一样。我们将在下一篇再探讨如何去构建全加器。

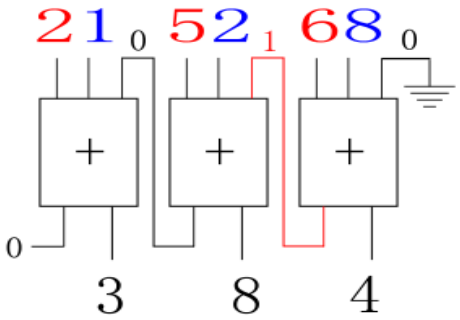
二、构建一位加法器

原文地址：http://my.oschina.net/goldenshaw/blog/411993

在上一篇中，通过对多位加法器结构的分析，我们得到了一位加法器的原型，也即所谓的全加器（Full Adder, FA），现在需要考虑如何去实现它。

分解

观察一下所谓的进位输入，还是以前述的十位上的加法为例：



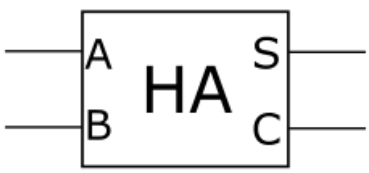
其实就是做了一次三个一位数的加法“5+2+1”而已。

| 怎么做三个数加法呢？很简单，依然是运用分而治之的原则。分两次来做，5+2=7，7+1=8。

进位可完全视作是一个特殊的加数（只能为 1 或者为 0），与其余两个加数等价，要把三个数相加，显然你需要知道如何把两个数相加，之后再把结果与第三个数相加。

半加器

因此，还是要回到两个数相加的情况，我们需要一个更为基础的能处理两数相加的构件：



| 这样的一个东西，我们称之为半加器（Half Adder，HA）。

| 它能接收两个输入，并产生两个输出：一个加位（Sum），一个进位（Carry）

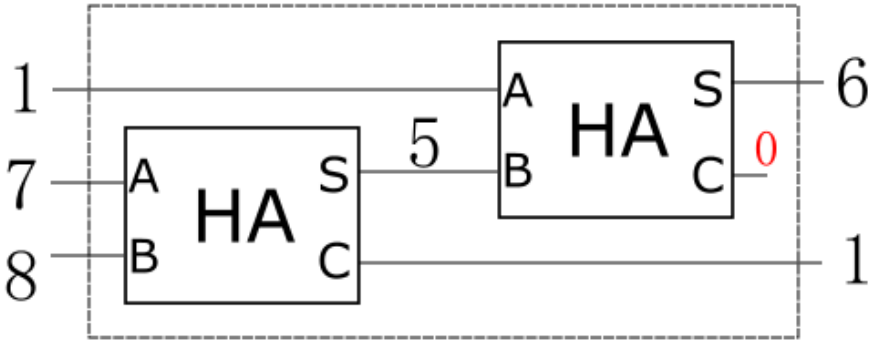
用半加器构建全加器

与前面类似，先不考察半加器的细节，假定它已经有我们想要的功能，我们如何用它来构建全加器？

考察一个例子：7+8+1（1 代表一个进位输入），最终结果是 16，要得到一个 6 和一个新的进位。

| 还记得前面关于串行与并行策略的讨论不？

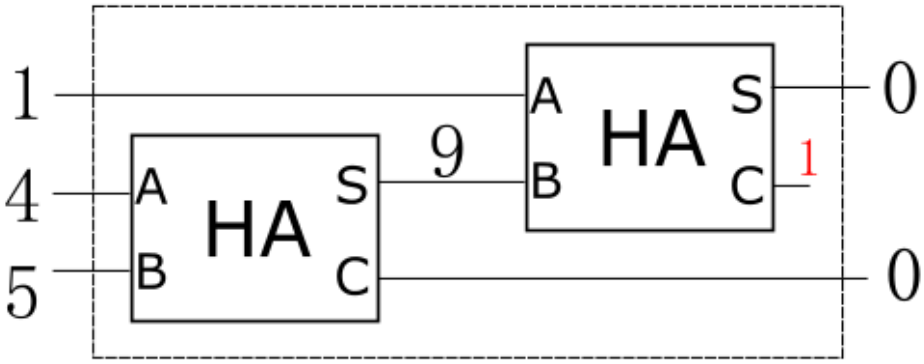
既然实质就是做两次两个数的加法，那么作为一种简单粗暴的方式，依旧是弄它两个半加器，然后连起来，代表两次加法：



在这里，第二个半加器的进位输入被我们丢弃了，只取了第一个的进位作为最后输出。

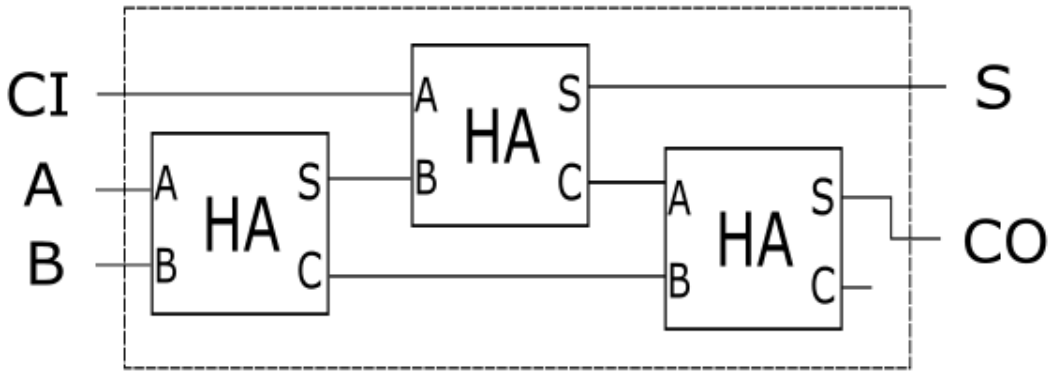
对上述例子而言，目前的做法显然是 OK 的，但能否适应更一般的情况呢？

让我们来看另一个例子：4+5+1



结果悲剧了，应该是 10，但输出却是 00，在第二个半加器里才出现了进位，但被丢弃了，导致了错误的结果。

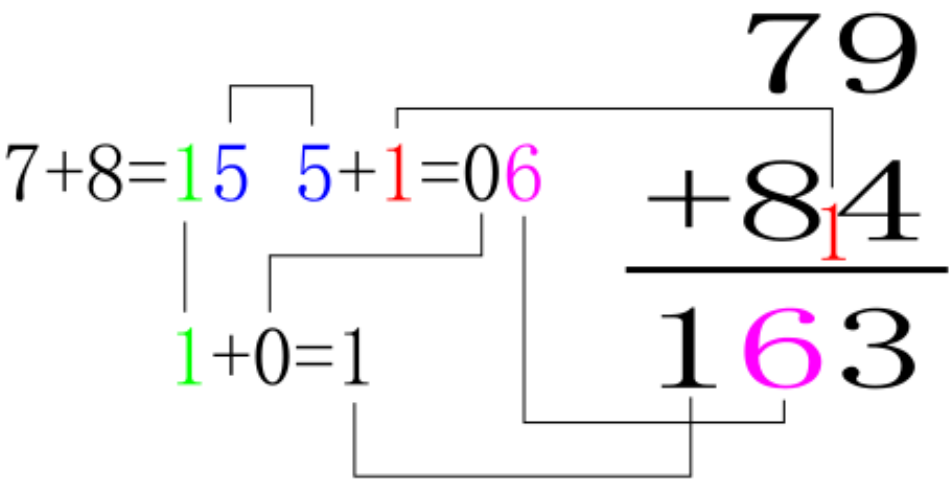
由于无法预料会在哪里产生进位，丢弃哪一个都不行。简单的处理就是再引入一个半加器，把这俩再加一下：



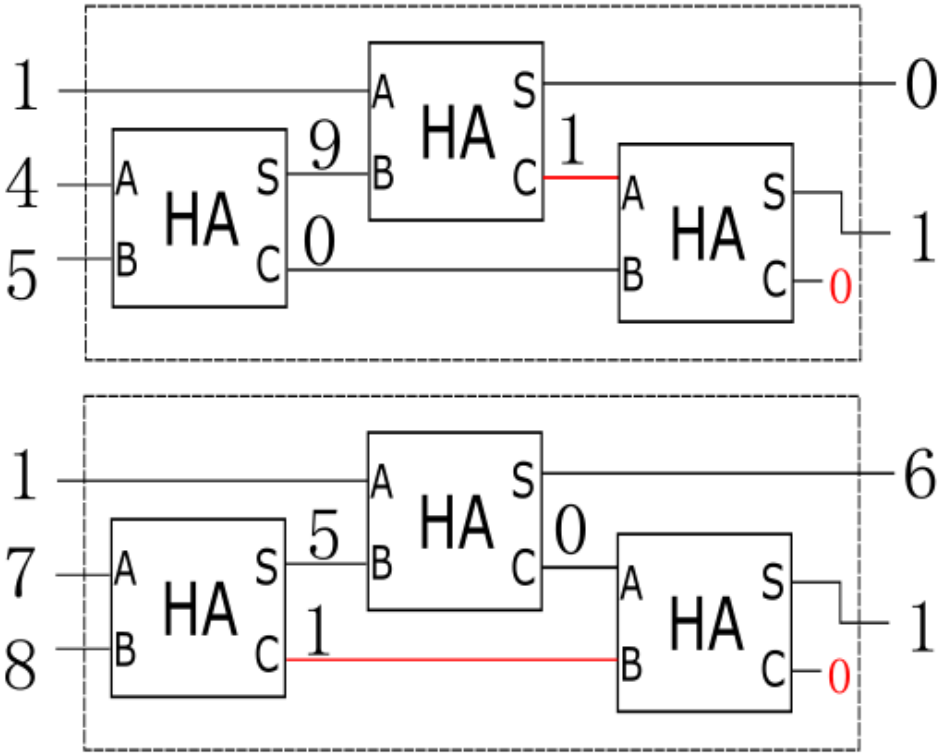
| 注：是把第三个半加器的加位作为最终的进位输出，进位则丢弃

|（最终只需要两个输出，势必要丢掉一个）。

| 如果我们观察一下人做加法的一般过程，那么我们的确是在不知不觉中做了三次两个数的加法：



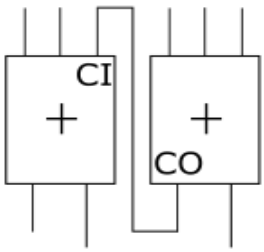
现在考察以下两种情况，都能够满足要求：



两次进位的问题

现在，即便前面两个都产生了进位，1+1=2，第三个半加器结果最大也不过是 2，要满 10 才能产生进位，因此，丢弃它的进位是安全的。

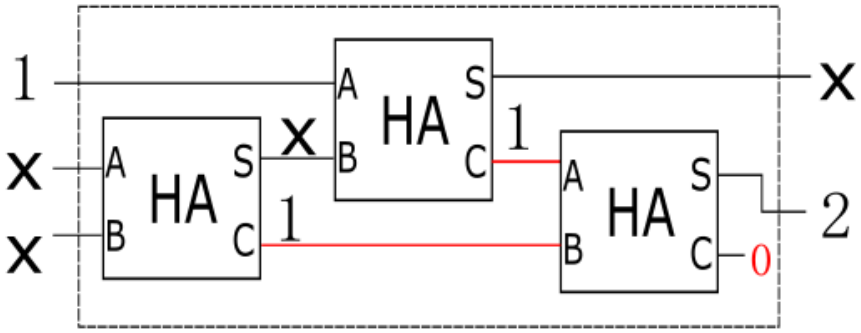
但另一方面，最终的进位输出（CO）可能会参与到下一级的进位输入（CI）中去，比如前面构建多位加法器时：



如果真的产生了两次进位，则违反了我们关于进位输入只能为 1 或 0 的假设。

仔细审查一下，我们可以断言，如果进位输入满足了假定，两个半加器都产生进位的情况是不存在的。

| 因为这意味着三个数之和至少达到了 20！



| 考虑其中的进位输入最大也就是 1，三个数最极端的情况也就是 9+9+1=19，不可能超过 20。

因此，只要一开始没有违反进位输入（CI）的假定，那么进位输出（CO）也是可控的，因此级联多个一位全加器组成多位加法器是可行的。

对进位逻辑的再思考

综上所述，进位或者来自第一个半加器，或者来自第二个半加器；又或者都没有进位，但不可能同时产生进位！

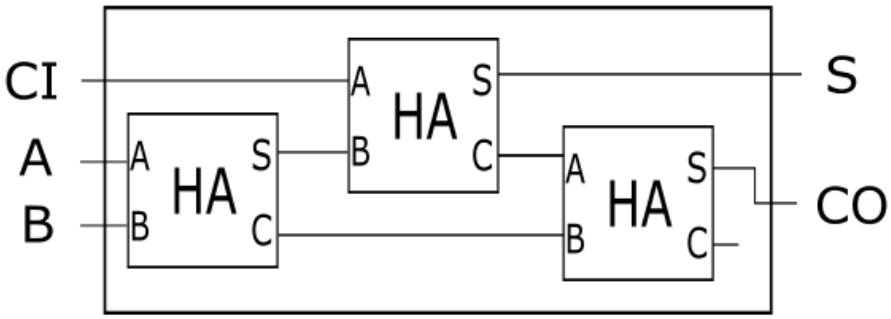
概括地讲，第三个半加器只需处理这三种情况：0+0，0+1，1+0.

| 它甚至不用处理我们通常认为最简单的 1+1。

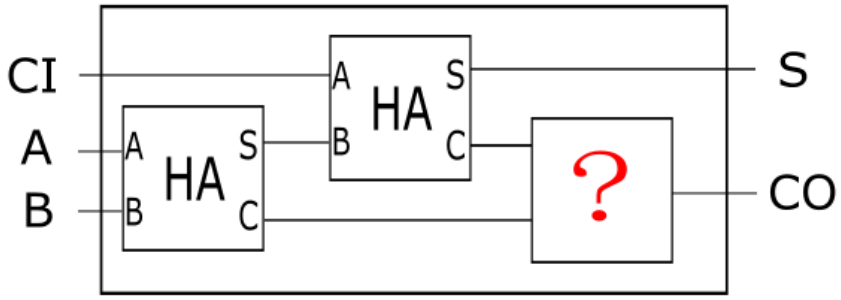
这对于一个我们设想能够处理任何两个个位数相加的半加器来说，的确是有点屈才了，从它的进位输出被丢弃也可看出它没有得到充分利用。

| 事实很明显：这里的逻辑其实无需用一个半加器去实现。

到目前为止，我们的全加器由三个半加器构成：



它是能够满足我们的要求的，但我们希望或许能够简化并用一个更简单的器件代替第三个半加器：



到目前为止，讨论都在十进制的基础上，但不难注意到，对于进位逻辑而言，它只涉及 0 和 1，这或许是开始逐步引入二进制的一个不错的契机。

三、改进全加器

原文地址：http://my.oschina.net/goldenshaw/blog/413591

在前面，我们谈到，希望能够简化对进位逻辑的处理。

虽然已经谈论不少的内容，但一直都还是停留在较为抽象的阶段。现在面临一个较为简单的问题，也应该是时候深入到电路底层去弄出点实际的东西来了。

从简(柿)单(子)的(先)问(拿)题(软)入(的)手(捏)，这应该是要始终贯彻的原则。

进位的逻辑

概括一下，要满足的逻辑可形式化地表达如下 ：

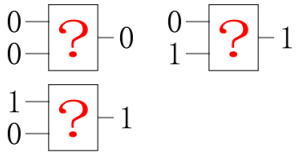
| f(0, 0) –> 0

| f(0, 1) –> 1

| f(1, 0) –> 1

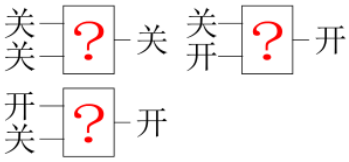
| f(1, 1) ：违反了我们的约定，不考虑。

用一张图来表示，那就是：



因为只涉及 0 和 1，我们甚至可以用二值逻辑来实现。

| 比如开和关，其中开代表 1，关代表 0；



| 又或者一个低电平和一个高电平，其中低电平代表 0，高电平代表 1。总之，有很多方式。

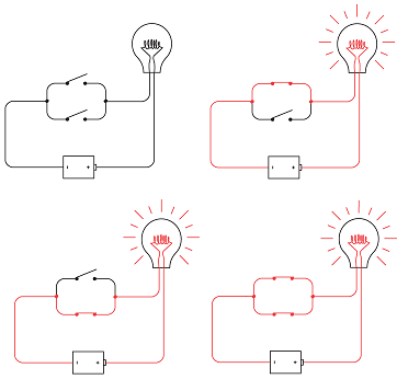
以上逻辑，如果我们用开和关来概括，那就是：

| 两个都关(0)时，结果为关(0)；（即 f(0, 0)=0）

| 有一个为开(1)，结果为开(1)。（即 f(0, 1)=1，f(1, 0)=1）

并联电路

以上逻辑是否让你想起了熟悉的并联电路呢？



| 图片来自《编码：隐匿在计算机软硬件背后的语言》（Code: The Hidden Language of Computer

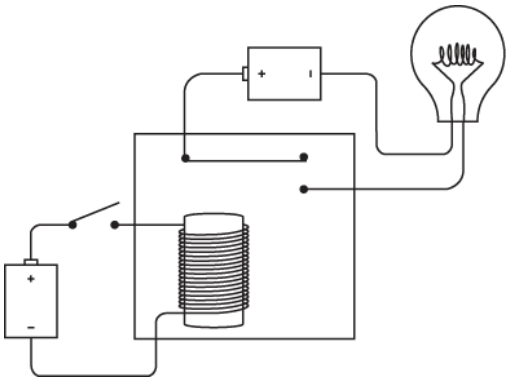
| Hardware and Software）一书。下面类似的截图或来自此书，或参考原书中的例图画成。

很明显，我们要表达的逻辑其实就是一个并联的逻辑。

| 严格地讲，是它的一个子集。我们有三条规则，上图中有四条规则，但最后一条用不上，
| 违反了我们的假定。

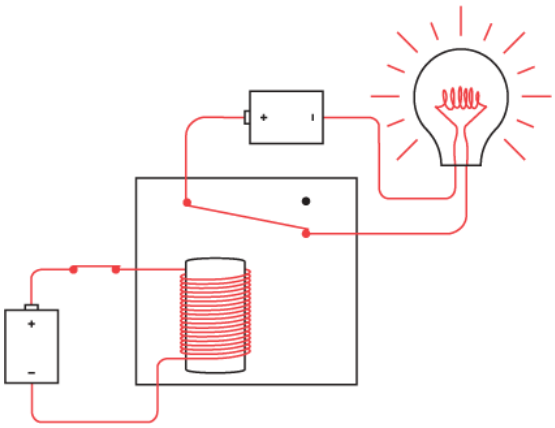
继电器（Relay）

如果我们用继电器来作为一种控制手段：



| 希望你还记得继电器是什么，其实就是个电磁铁和一个弹性开关的综合体，
| 上面的开关是个有弹性的小铁片，当通电时，线圈中产生电磁感应，形成一个磁场，
| 好像一块磁铁一样，因此能把铁片吸引下来，从而控制电路的通断。
| 好吧，这些东西多少涉及了一些物理知识，希望你还能有点印象。

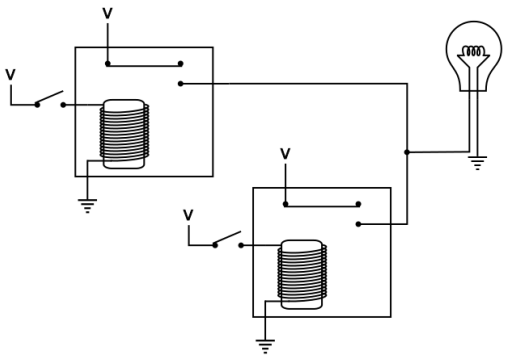
那么可以达到用输入来控制输出（图中形象地用灯泡的亮灭来表示）的目的：



| 当左边的开关合上时，电磁感应使得开关铁片被吸引下来从而连通了灯泡的电路。

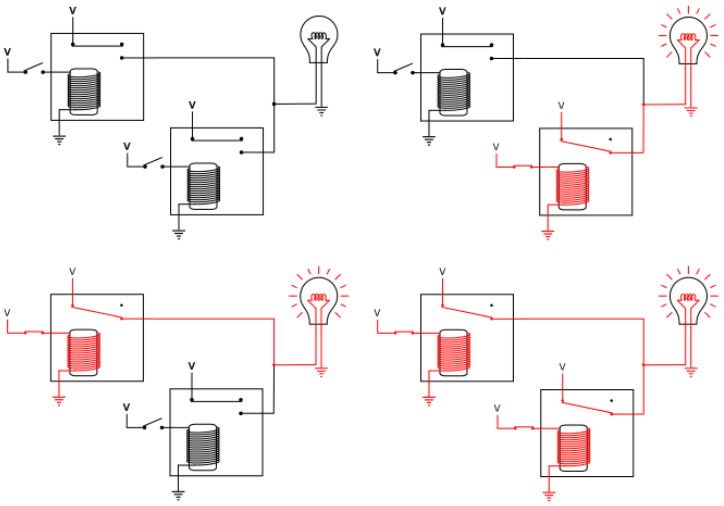
或门（Or Gate）

当然了，只是这样的话，用处并不大。但当以并联的方式连接两个继电器时：



| 注：图中简化了电池的画法，所有的 V（Voltage，电压）代表连接着电池的正极，
| 接地符号则表示连接电池的负极。电池及之间的连线均已经省略。

那么，左边两个开关的状态（输入）将影响右边灯泡的状态（输出）



这样的一个东西就是所谓的或门了，如果用一个竖线符号”|”来表示一个或操作，把 0 和 1 想像成上面的关和开，那么

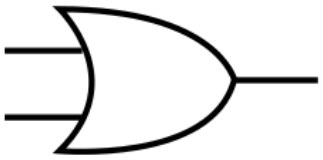
| 0 | 0 –> 0

| 0 | 1 –> 1

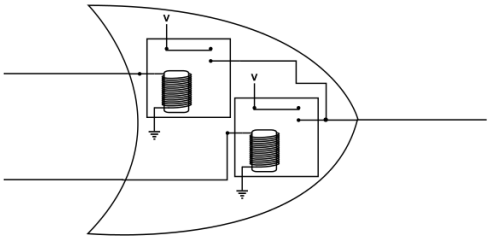
| 1 | 0 –> 1

| 1 | 1 –> 1

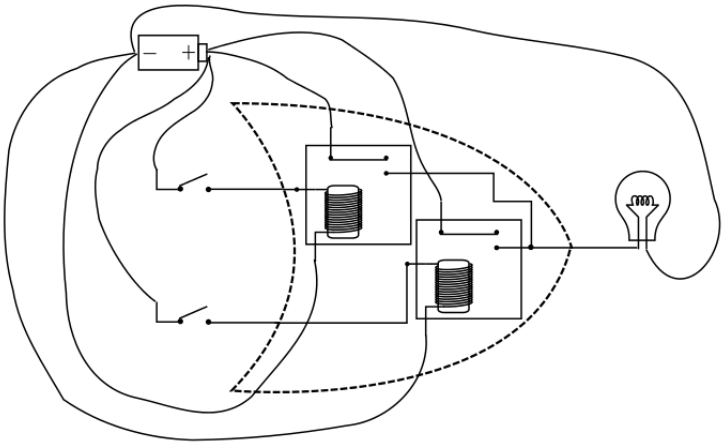
或门用以下符号来表示，它有两个输入一个输出。



它实际是对这样一种功能的抽象：

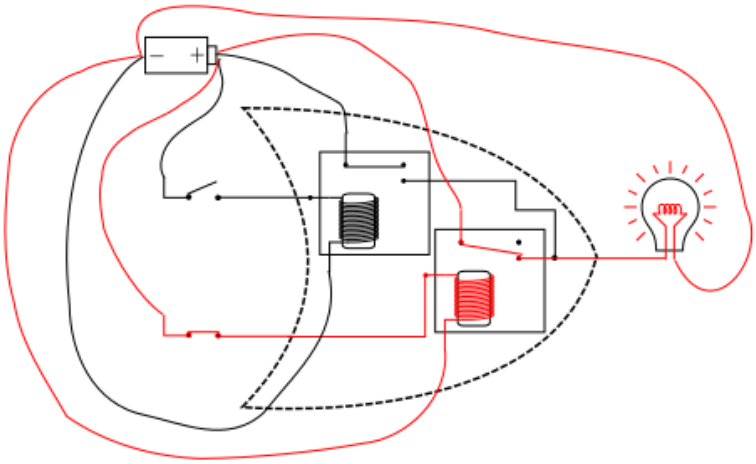


当然，如果真心要真刀真枪地连出一个或门来控制灯泡的亮灭来玩下，大概是下面这个样子



| 图中的走线也很随意，通常，或门内部会统一引出一条电源和一条接地线，这样会整洁一些。

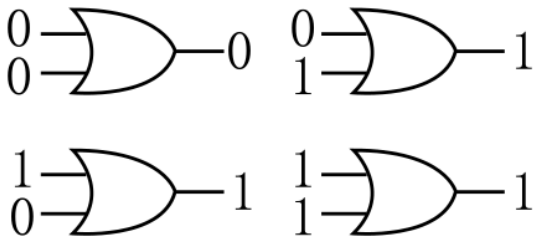
当我们按下其中一个开关时，就能控制灯泡了，情形如下：



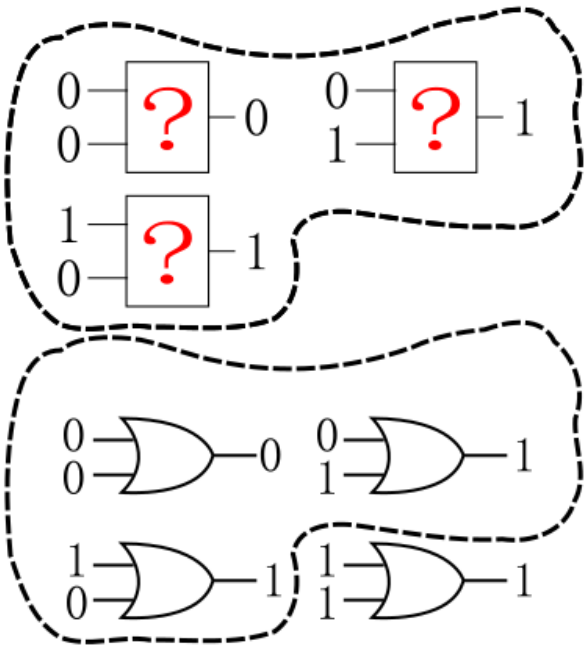
或门是一个有源电路，这里的源指电源，当然我们通常会省去关于电源方面的部分，只关心功能方面的抽象。

| 另：现在通常是用晶体管来制造或门，而不是这些笨重的继电器，
| 不过它们表达的逻辑是一致的。
| 一个晶体管从功能逻辑上讲相当于一个继电器，关于晶体管的具体原理后面再谈。
| 所谓功能逻辑大概也就是软件中的接口（interface），至于用继电器还是晶体管
| 则是具体实现（implement）上的细节。

把其中的开和关用 1 和 0 来表示，那么一个或门的逻辑如下：



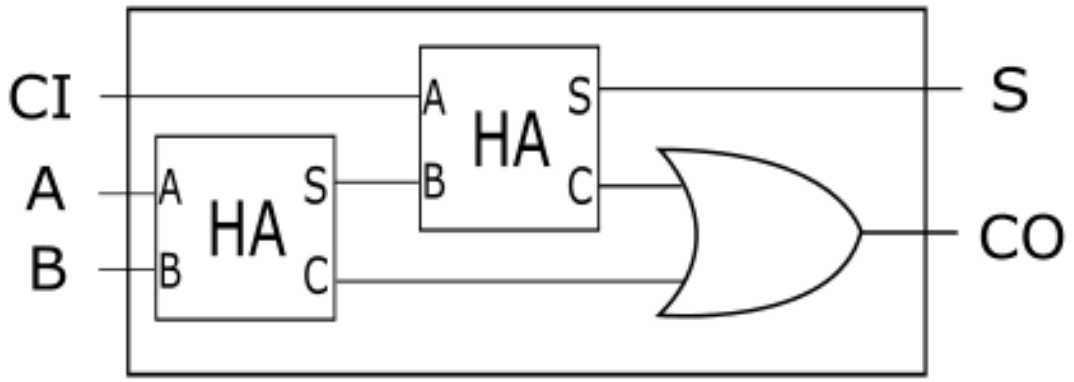
对比我们的需求



就不难发现，这正是我们想要表达的逻辑。

改进的全加器

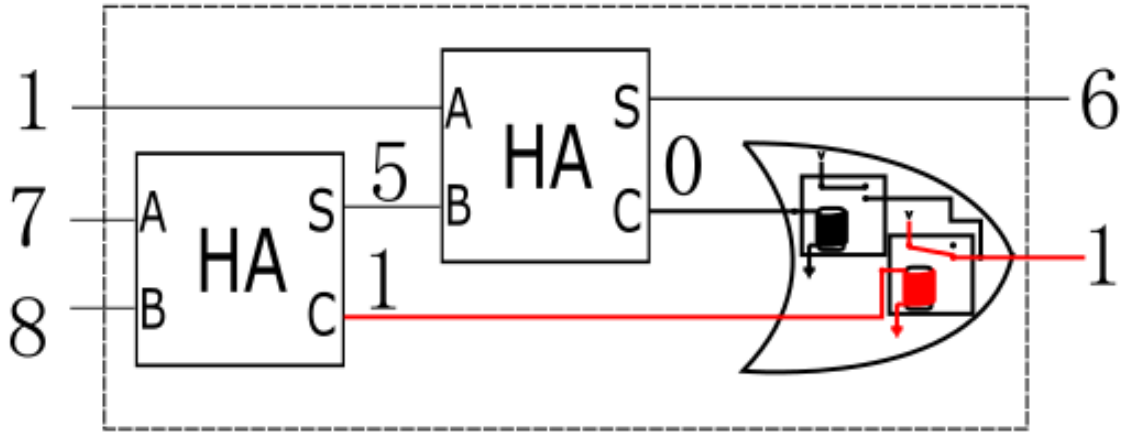
至此，用或门来替换最后的一个半加器，据此可以简化得到全加器的模型如下：



它的内部由两个半加器再加一个或门组成。

当然，由于我们目前依旧还是从十进制角度考虑的，或门的输出还是要受到一定约束的，因为它的值要参与到下一级的输入。比如按前面的约定，电源方面应该选用 1V 的电压。

下图是执行前述一个计算的原理示意图：



注意这里，输入的 1 是作为驱动继电器用的，而输出的 1 是从被控制的电源部分取得的，所以前面说电源的电压要选用 1V 的。

| 显然，只要电源电压是 1V，输出永远也只会是这个值，或者为 0。

| 这也保证不会输出错误的值到下一级。

| 另外，即便有产生两次进位的错误，从上面的原理图也能看出，输出依旧是 1。

另一方面，一个或门其实并不计较输入的是 1V 还是 2V，或者是其它更大的值。只要这个电压足以驱动继电器，那么它就产生输出。

| 严格地讲，这也就是所谓的二值逻辑，一个或门仅能表示两种状态，开或关，

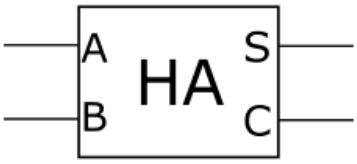
| 它对数值其实是不敏感的。到后面全面转向二进制后，就无需对电压的具体值作出规定了。

现在，我们部分地实现了全加器的功能，也即是对最简单的进位逻辑的处理，并初步地引入了二进制的思想。目前，还有一个更为重要的部件，我们还是靠着 wishful thinking 来考虑它的，这就是半加器（HA），我们将在下一篇讨论它的设计与实现。

四、构建半加器的初步设想

原文地址：http://my.oschina.net/goldenshaw/blog/415274

在前面的篇章中已经讨论了如何在半加器的基础上构建全加器，那么现在是考虑如何去构建这样一个半加器（Half Adder，HA）了。



进一步分解

虽然已经少了一个输入，但输入输出还是不少，因此要考虑能否进一步的简化。

| 始终，分而治之是一以贯之的一个策略，如果要处理的问题比较棘手，

| 那么就考虑如何进行功能抽象并把它们分解，这是应付复杂性的一个重要手段。

比如，输入能否分解呢？

| 你有两个输入，但一次只考虑一个输入的话，你既不能确定进位的结果，

| 也不能确定加位的结果，所以你是不能分的。

| 通常而言都不会考虑分解输入。

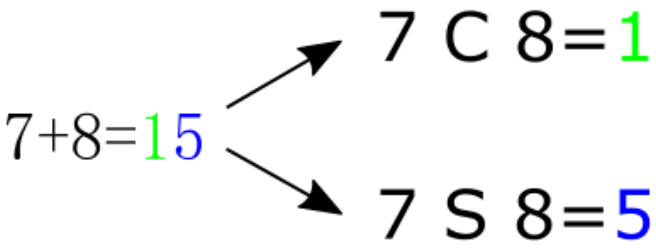
那么，两个输出又能否分解呢？

我们注意到两个输出都可以表示成输入的函数：

| S=f(A, B)

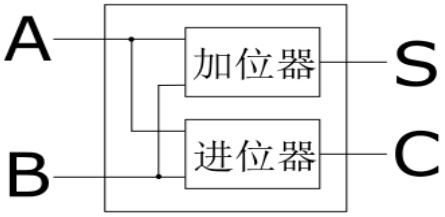
| C=f(A, B)

完全由输入所决定，S 与 C 无关，C 也与 S 无关，它们均由 A，B 决定。



| 其中，C 代表进位运算，S 代表加位运算。

据此，我们可以在内部把输入同时传递给两个部件并因此拆解两个输出，分成一个加位器和一个进位器两部分：

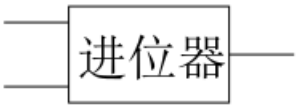


如果用软件语言来表达，大概是这样：

```
1|  public Output halfAdd(Input a, Input b) {
2|      Sum s = sum(a, b);
3|      Carry c = carry(a, b);
```

```
4|          return    new Output(s, c);
5|    }
6|    public Sum sum(Input a, Input b) {
7|          // TODO
8|          return null;
9|    }
10|   public Carry carry(Input a, Input b) {
11|         // TODO
12|         return null;
13|   }
```

尽管我们还不知道它们工作的细节，但已经可以单独拿出比如“进位器”模块来分析：



进位器

这样，问题再一次简化，只有两个输入一个输出。对于一个进位器，需要满足的约束如下：

```
| 输入-->输出
|   0+0 –> 0
|   0+1 –> 0
|   …
|   5+4 –> 0
|   5+5 –> 1
|   …
|   9+9 –> 1
```

请注意，以上的加号请不狭义地去理解，这里用“+”这个符号仅是想表达进位器它接受两个输入（参数）。

```
| 可以像之前那样引入一个符号 C 来表达进位器所代表的运算，那么”9 C 9 = 1”,“5 C 4 = 0”等。
| 把 C 想像成一个两个参数的函数，那么 C(9, 9)=1；其实加法也可以写” +(9, 9)=18”,
| 只是我们可能觉得不习惯。
| 想要更好的可读性，那么可写成 Carry(9, 9)=1。 总之，这些不过是表达形式上的差异。
```

显然，输出不是 0，就是 1，不可能超过 1.

但输入的情况则太多了，暂时来讲，我们还是从十进制上来考虑的。如何把众多的输入情况映射到这两种输出中去呢？这的确是个挑战。

首先，我们能否像之前在全加器里那样用继电器来设计这里的进位器呢？

那么，很遗憾，如果我们还是十进制的形式，那么我们有十种状态要表示，在之前，之所以能巧妙地引入继电器，是因为在那种特殊的情况下，输入和输出都只有两种状态。

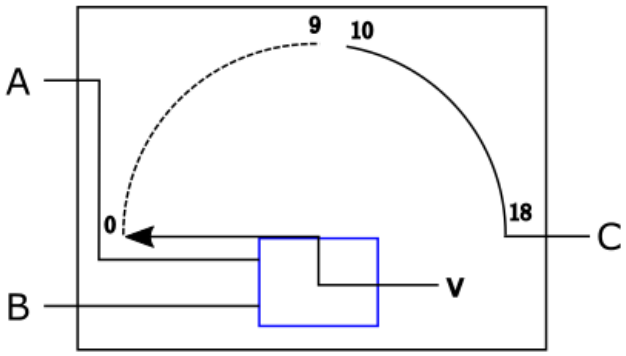
```
| 如果你想知道一根导线有电还是没电，你也许只要有一枝简单的电笔就行了；
| 如果你想知道电压的具体值，你则需要一个复杂的电压表！
```

当然，这不是说打造一个十进制的加法器就是完全不可能的事。

| 个人来说，我不太喜欢教科书上那种，一来就说加法器你要用二进制来做，

| 然后直接就把结果展现给你，完全没有思考的空间！

我们可以从比如电压表上得到一些启发，设想一下如下的原型：



关于蓝色的框，姑且叫它“旋转器”吧，我们还是再次发挥 wishful thinking（按愿望思维）的光荣传统。

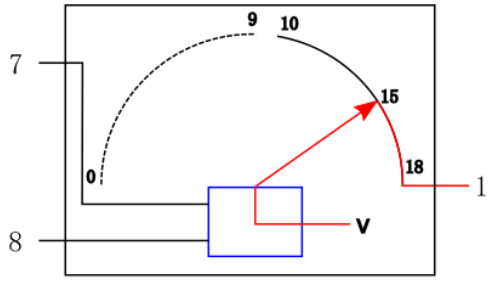
| 比起继电器中上下跳动的开关，这是一种更加高级的开关形式了。

| 它能根据两个输入电压的和值进行旋转，就像电压表测量电压那样，指针就是一个旋转的开关。

| 初步估计还是很复杂的，可能要一些轴承呀，弹簧呀，电磁铁之类的，

| 还要考虑产生的力如何合在一起。还好我们有抽象这一手段，细节就放到后面再考虑了。

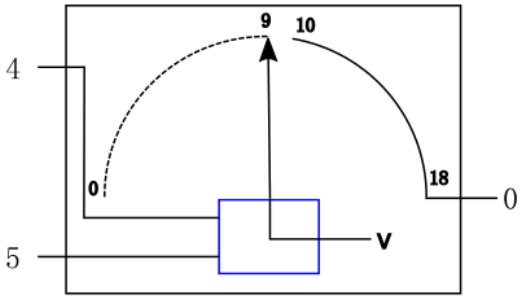
然后设想以下的一个示例：



如果两个电压足够，那么指针可以偏转到能连通电路的程度，就可以对外输出电压了。

| 当然，图上的电源 V 须设置为 1V 的电压。

如果两个电压和较小呢？那么就不足以连通，输出自然就是 0 了：



自然，要考虑的问题还很多，比如指针开关如何与弧形的导线形成良好接触同时又不能有太大的阻力等等。总体而言，不简单。

| 以上是个人的一些设想，如果大家觉得 low 也希望别见笑，

| 各位也不妨脑洞大开想想其它的方式，自然，你需要一定的电路基础。

然后还要考虑设计“旋转器”，这玩意可就复杂了，那么我们或许还是先看看加位器吧！是否会简单一些呢？

加位器

如果是加位器，输入的种类一样，而输出则更多，可以是 0-9 任何一个值：

| 输入-->输出

| 0+0 –> 0

| 0+1 –> 1

| …

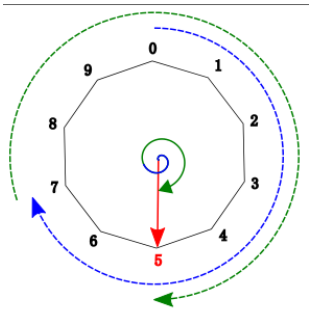
| 5+4 –> 9

| 5+5 –> 0

| …

| 9+9 –> 8

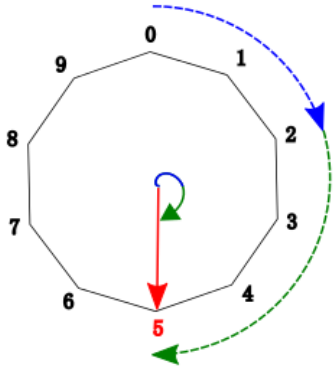
那么，很遗憾，它的情况或许还要复杂过进位器，而且它的输出还不是单调的变化，那么我们可能要考虑一个类似钟表一样的结构原型，比如还是以“7+8”为例：



7 S 8=5

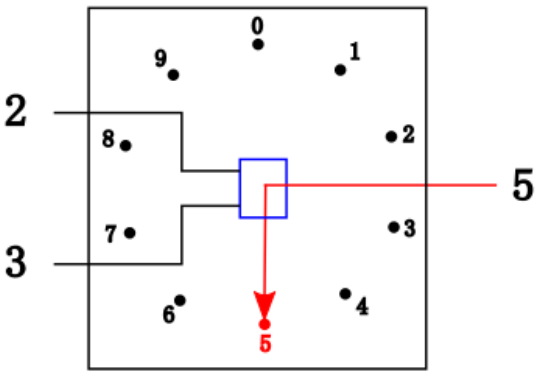
| 同样要涉及一个“旋转器”，它还要满足能转差不多 720 度（两圈）。上图中转了一圈半。

以下为另一种输出为 5 的情况，这次转了半圈，但结果是一样的：



2 S 3=5

那么可以这样去考虑，在各个点上布置不同的电压输出，旋到不同点上就对外输出不同的电压：



自然，你还得考虑误差出现的可能，也许应该把点换成小弧段。

| 当然了，以上也仅是个人一些不成熟的构想，你可能会有更好的想法！

如果你此时觉得，这是不是也太复杂了，且不说我们还差一个重要部件“旋转器”呢！那么，或许也正是这种复杂性，迫使我们去考虑其它的可能性，也即是打造二进制的电路。这样我们就不需要什么精确的旋转开关器了，一个能上下跳动的开关就能满足我们的要求。

二进制：A Great Insight

既然最终是要做二进制的加法器，那么为什么浪费时间脑洞大开去讲这些十进制的设计呢？

那么首先十进制对我们而言是更自然的方式。

| 事实上，历史上就很多人曾经设计了许多的十进制的机械加法器。

而意识到可以用二进制来代替，这在认识上是一个飞跃，对计算的本质也有了更加深刻的理解。

在《计算机科学哲学》（Philosophy of Computer Science）一书上，作者谈到四个计算机科学中的伟大洞见（great insight），第一条就：

| All the information about any computable problem can be represented using only

| two nouns: 0 and 1.

| 所有关于可计算问题的信息都可以用 0 和 1 两个名词来表示。

作者把它称之为“Bacon’s, Leibniz’s, Morse’s, Boole’s, Ramsey’s, Turing’s, and Shannon’s Insight”

| 那么这些洞见来自于谁呢？它来自于培根，莱布尼茨，莫尔斯，布尔，拉姆齐，图灵，香农。

没错，是以上几头牛的头脑上才最终慢慢将这些思想孵化成熟。

| 而教科书中对此常常要么是轻描淡写地一笔带过，要么压根就不谈，

| 一上来就跟你讲二进制，好像这些是多么自然的想法！

| 好吧，二进制对我们这些通常有“十个指头”的人而言是不自然的！

我们会在后面再谈谈进制的问题，并最终用二进制的方式完成半加器的设计。

五、从十进制到二进制

原文地址：http://my.oschina.net/goldenshaw/blog/416154

在前面我们看到，用十进制来设计半加器是有不少困难的，因为它的状态太多了。如果想用简单的开和关去表达想要的计算，那么就要转向二进制。不过话说回来，你是否想过，为啥我们喜欢十进制呢？

为什么是十进制？

人类在大约 20 万年前走出非洲，分散到各地形成了各种文明，但为何在计数这一问题上，常常不约而同选用了十进制呢？

没错，答案就在我们的双手上，我们的十根指头与我们的十进制之间绝不是巧合。

| 设想一下原始人类的一次狩猎大丰收，一个首领兴奋地用指头在数着他们捕获的羊，

| 一只，两只...哇，全部手指用光了还没数完，只好先在地上标识一下先，

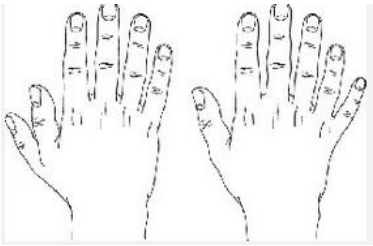
| 然后再把指头全部收回继续清点。

那么，这么一个标识的动作就是进位的雏形了，而我们最终形成对十进制的偏好也就不足为奇了。

其它的可能

据说非洲有部落是采用 20 进制，有学者去考察，发现他们数数时把脚指头也用上了，难怪呢！

而人其实有时会患上一种称为多指畸形的遗传性疾病，其中以六指症最为多见：



其实说是畸形是因为他们是少数派，如果他们成为主流，我们可能就会习惯使用十二进制了！

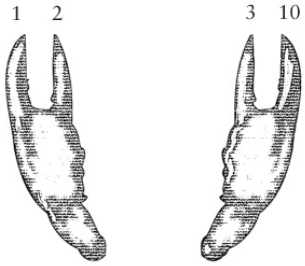
而怕老鼠的喵星人--哆啦 A 梦则可能喜欢二进制：

| 难道二进制真的是未来的趋势？据作者设定，它的生日是 2112 年 9 月 3 日。

| 掐指头一算，哎呀，还不到 100 年了！话说小时候看书时觉得 22 世纪是件很遥远的事~

不过跟别人猜拳时就悲剧了：

而龙虾们如果有一天也能建立自己的文明，那么它们就可能是用四进制了：



| 截图来自《Code》一书。

广义的进制

如果我们把标准放松一点，那么就不难发现人类社会中是存在很多的进制的。

比如以时间为例，它混合了各种的进制。小时是 12 或者 24 进制，而分秒则是 60 进制。只不过我们还是用十进制方式来书写它们。23:59:59，再加一秒是 00:00:00，因为满 60 或者满 24 就要进位了，进位后就归零了。

| 如果注意到 12 可以被 2，3，4，6 整除，而 60 则能被 2，3，4，5，6 整除。

| 那么采用这样的进制是有助于去更细地划分它们的。

而货币方面，比如英国曾经是这样的：1 英镑等于 20 先令，而 1 先令又等于 12 便士。

| 1 先令等于 12 便士自然也能使得它易于分割。

而我们中国古代的秤，斤两之间曾经采用过 16 进制。

| 你以为“半斤八两”是怎么来的？

| 16 进制自然可以使得对一斤多次取半还能取整。

进制的总结

下面对各种进制做些小结。

1. **N 进制有 N 个不同的数字（符号）。**

| 所以，十进制就有十个不同数字，0，1，2，3，4，5，6，7，8，9；

| 那么，二进制就有两个不同数字，0，1；（借用了十进制的符号）

| 如果是十六进制呢？那么即便借了十进制的全部符号还是不够用，

| 此时要么发明一些新的符号，要么就再借用一些别的符号了。对于十六进制，

| 实际借用了 6 个大写字母，0，1，2，3，4，5，6，7，8，9，A，B，C，D，E，F。

2. **N 进制逢 N 进 1。**

| 十进制逢十进一，二进制就是逢二进一。

| 对于二进制，1+1 就要进位了，因为结果是 2，但二进制根本不存在 2 这个符号，

| 最大也就是 1。进位后加位归零，结果是 10（二进制）.

十进制	**二进制**
5 + 5 = 10	1 + 1 = 10
逢十进一	**逢二进一**

| 反复加一，可以得到更多的值，红色的小 1 表示有进位：

二进制	1 + 1 = 10	10 + 1 = 11	11 + 1 = 100	100 + 1 = 101	···
	⋮	⋮	⋮	⋮	
十进制	2	3	4	5	···

| 以下为十进制与二进制及十六进制的一个对应表：

十进制	二进制	十六进制
00	0000	0
01	0001	1
02	0010	2
03	0011	3
04	0100	4
05	0101	5
06	0110	6
07	0111	7
08	1000	8
09	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

3. N 进制一个位最多只能表示 N 种不同状态，要想表示更多的状态，就要用更多的位。两个位能最多表示 N×N 种状态，三个位能表示 N×N×N 种状态，依次类推。

| 一位 16 进制能表示 16 种状态，两位则能表示 16×16=256 种状态。

| 一位十进制能表示 10 种状态，两位则能表示 10×10=100 种状态。

| 同理，一位二进制能表示 2 种状态，两位则能表示 2×2=4 种状态。

| 自然，要表示同样多种状态，二进制比十进制需要更多的位数。

| 正如上面显示，至少 4 位才能表示。

| 但尽管如此，任何一个十进制数都是可以用二进制来表示的。

| 用康托尔的话来说就是两个无穷集合是一一对应的。

进制的转换

对于一个十进制的数，比如说 368，它的实质是什么呢？

| 其实是：$3\times10^2+6\times10^1+8\times10^0$

| $=300+60+8$

| $=368$

| 对于 N 进制而言，N 就是这个指数的底；指数则是右边最低位为 0，

| 然后向着左边的高位递增。

二进制转十进制

所以对于一个二进制数，比如 1010 来说，它对应的十进制数为：

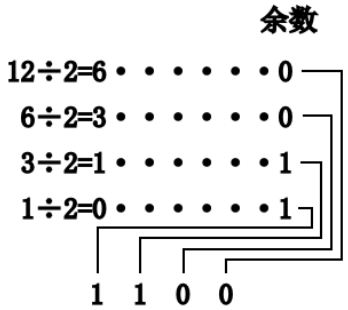
| $1\times2^3+0\times2^2+1\times2^1+0\times2^0=8+0+2+0=10$

| 所以 $1010_2=10_{10}$。

| 下标为相应进制。

十进制转二进制

而对于十进制到二进制的转换，则通过反复对 2 整除直到商为 0，然后取对应余数来获取。比如以 12 为例：



那么我们有：1210=11002

二进制的优势

以加法表为例，如果使用的是十进制，严格地讲，有 10×10=100 种情况需要考虑。

| 想想我们曾经背过的九九乘法表，它已经去掉了 0，以及结合了交换律去掉了一些重复的情况，

| 却还依然有高达 45 个条目，作为加法表，也是如此。

如果采用二进制呢？这个数目就大大减少了。

| 对于 N 进制而言，有 N×N 种规则需要考虑。因此，对二进制而言，只有四条规则。

二进制的加法表只有以下四条规则：

| 0+0=00，

| 0+1=01，

| 1+0=01，

| 1+1=10。

乘法表甚至更简单，因为不涉及进位：

| 0×0=0，

| 0×1=0，

| 1×0=0，

| 1×1=1。

如果不考虑与 0 相乘的情况，则只有一个条目：1×1=1

对比一下二进制的“一一乘法表”与十进制的“九九乘法表”：

十进制九九乘法表

1×1=1								
1×2=2	2×2=4							
1×3=3	2×3=6	3×3=9						
1×4=4	2×4=8	3×4=12	4×4=16					
1×5=5	2×5=10	3×5=15	4×5=20	5×5=25				
1×6=6	2×6=12	3×6=18	4×6=24	5×6=30	6×6=36			
1×7=7	2×7=14	3×7=21	4×7=28	5×7=35	6×7=42	7×7=49		
1×8=8	2×8=16	3×8=24	4×8=32	5×8=40	6×8=48	7×8=56	8×8=64	
1×9=9	2×9=18	3×9=27	4×9=36	5×9=45	6×9=54	7×9=63	8×9=72	9×9=81

VS

二进制一一乘法表

1×1=1

啊，如果从一开始就采用二进制该多好，这画面想想也要醉了，再笨的熊孩子估计也能分分钟就掌握它们！

| 二进制的代言人，“蔡 10”小姐就在一首歌里感慨地唱道：

|“那画面太美我不敢看...”(《布拉格广场》，蔡依林 & 周杰伦)

我们现在觉得用二进制不方便，看着不习惯，那大概是因为没有从小就接受这样的方式。

如果你天天接触，估计你也会习惯了。

| 有这么一种说法：世界有 10 种人，一种是懂二进制的，一种是不懂的。

你懂了吗？

| 国家需要你这种懂二进制的！

关于进制的问题，就说到这里。有了以上基础，在下一篇将介绍如何构建最终的二进制半加器。

六、最终的半加器

原文地址：http://my.oschina.net/goldenshaw/blog/416599

在上一篇，我们谈到了进制的问题，如果用二进制的话，加法表只有四条规则：

| 0+0=00，

| 0+1=01，

| 1+0=01，

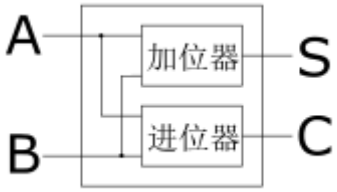
| 1+1=10

以此为需求，再次尝试构建半加器。

分解

首先，大的结构还是稳定不变的：

| 这点不会因为采用不同的进制而有所不同。



把上述四种情况，一横一坚弄成一张表，如下：

A \ B	0	1
0	00	01
1	01	10

再按加位和进位分成两部分，如下：

A \ B	0	1
0	00	01
1	01	10

A \ B	0	1
0	0	0
1	0	1

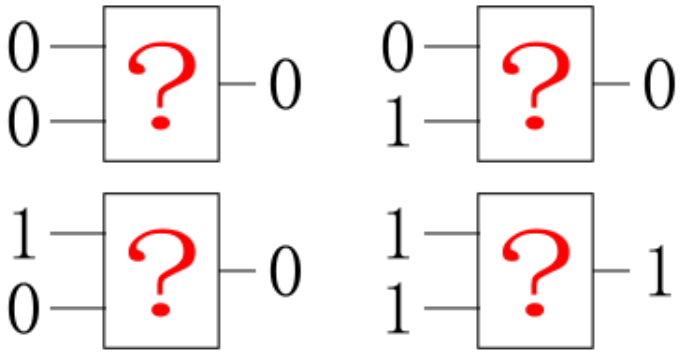
进位

A \ B	0	1
0	0	1
1	1	0

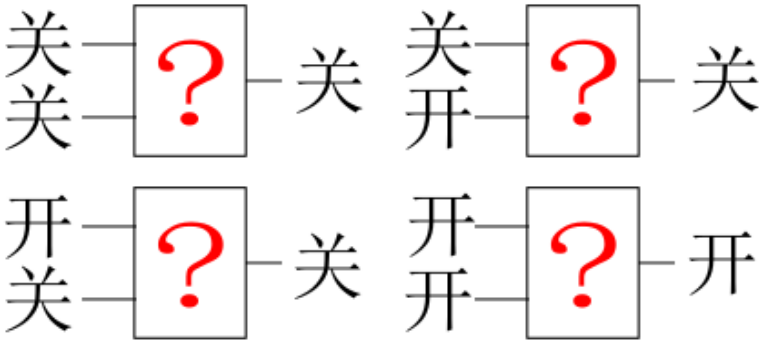
加位

进位器

先考虑进位的情况。根据上表，进位器它要满足以下几种情况：



跟之前类似，令“关=0，开=1”，那么

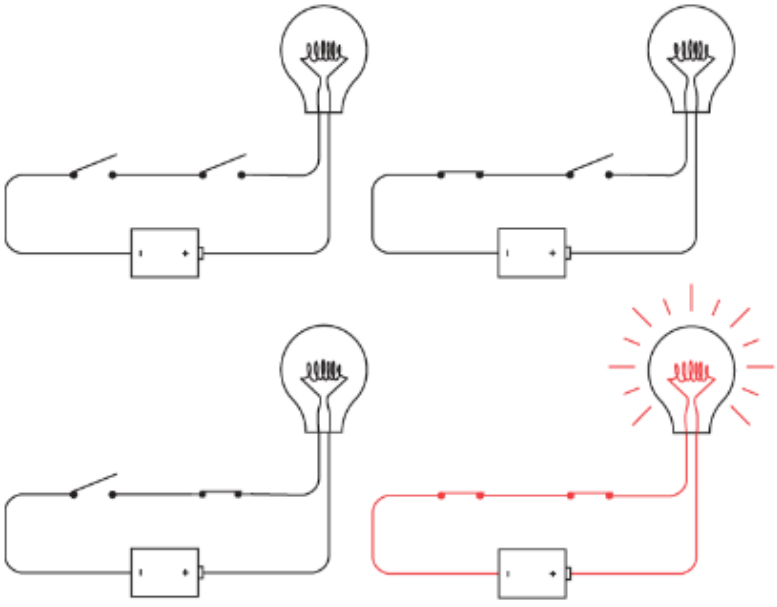


用一句话来概括就是：“两个都是开，结果才是开，任意一个没开，结果就是关。”

| 你还记得有什么电路有如此的性质呢？

串联电路

还记得串联电路不？

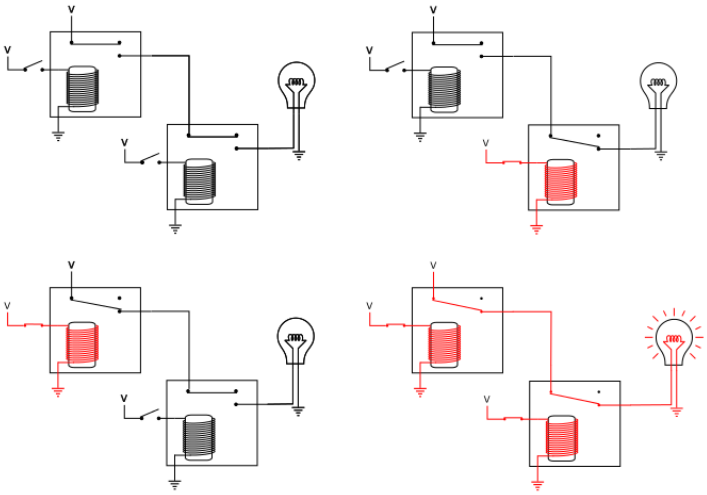


| 注：由来自《编码：隐匿在计算机软硬件背后的语言》（Code: The Hidden Language of Computer Hardware and Software）一书中的图片整合而成。

| 下面类似的截图或来自此书，或参考原书中的例图画成。

与门

那么，当以串联的方式连接两个继电器时：



这样的一个东西就是所谓的与门（And Gate）了，如果用一个符号”&”来表示一个与操作，把 0 和 1 想像成上面的关和开，那么

| 0 & 0 –> 0

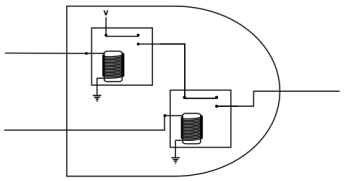
| 0 & 1 –> 0

| 1 & 0 –> 0

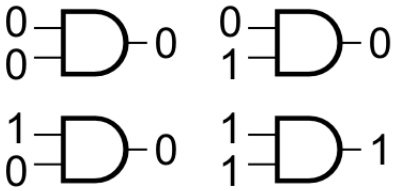
| 1 & 1 –> 1

与门用以下符号来表示，它有两个输入一个输出：

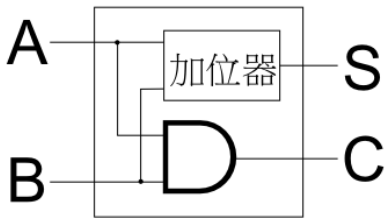
它实际是对这样一种功能的抽象：



与门满足以下逻辑：



那么，这也正是进位器要表达的逻辑。对半加器而言，所谓进位器，就是一个与门，因此到目前为止我们的半加器是这样的：



接下来再来看加位器的设计。

加位器

加位器的逻辑是这样的：

A \ B	0	1
0	0	1
1	1	0

到目前为止，我们已经有了或门和与门，它们的逻辑如下：

或门

A \ B	0	1
0	0	1
1	1	1

与门

A \ B	0	1
0	0	0
1	0	1

很遗憾，现有这些门的逻辑都不符合要求。

当然我们也不难注意到或门已经很接近加位器的逻辑了，除了右下角有些出入外：

加位器

A \ B	0	1
0	0	1
1	1	0

或门

A \ B	0	1
0	0	1
1	1	1

而且我们特别注意到，对加位器而言，当两个输入为 1 时输出反而是 0.（1+1 会导致进位从而使得加位归零）

| 观察加位器的逻辑，如果把 0 和 1 看成是正和负，倒是有点像我们所说的“负负得正”的情况：

A \ B	0	1
0	0	1
1	1	0

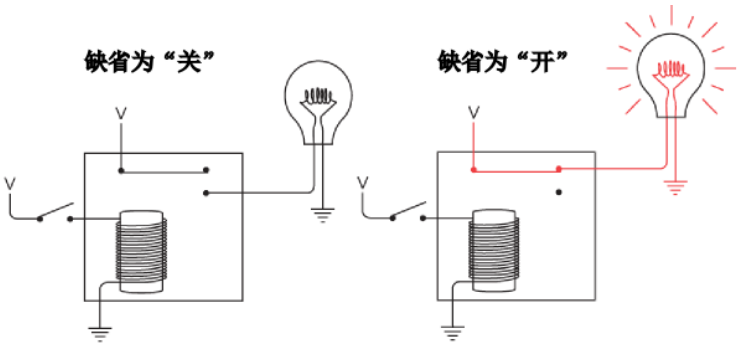
A \ B	正	负
正	正	负
负	负	正

但显然，无论是对于并联（或门）还是串联（与门）电路，是不可能存在什么“负负得正”的，当输入都为 1 时，灯泡没理由不亮。

想要达到输入都为 1 时，输出反而为 0，或门和与门都不可能满足要求，这里我们需要一点“反向”的思维。

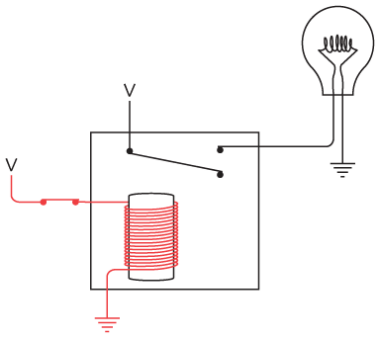
反向器，非门

在之前，继电器的开关缺省都是处于断开的状态，但这一点并不是必须的，我们也完全可以让它缺省处于打开的状态：

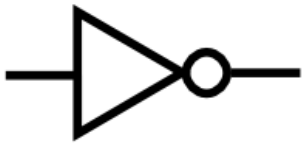


| 显然，把灯泡接在上面一个触点上就行了。

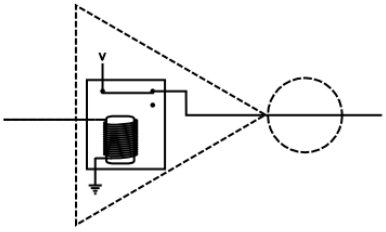
那么对于第二种情况，当有输入时，开关反而被吸引下来导致电路断开：



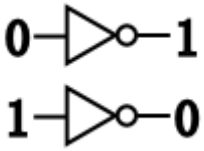
那么这样一种电路我们称之为反向器（Inverter），又称之为非门（Not Gate），用以下的符号表示：



它只有一个输入，一个输出。它是对这样一个功能的抽象：



它能表达以下逻辑：

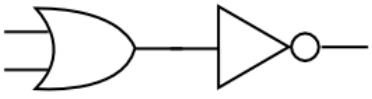


或非门

当把或门的输出又接在一个非门上时，或门的输出都被取反：

A \ B	0	1
0	0	1
1	1	1

A \ B	0	1
0	1	0
1	0	0



通过这么一个反向，我们可以在右下角也得到输出为 0 的情况了，但这么一来，其它三个原本 OK 的逻辑现在反而又不行了。

或非门

A \ B	0	1
0	1	0
1	0	0

加位器

A \ B	0	1
0	0	1
1	1	0

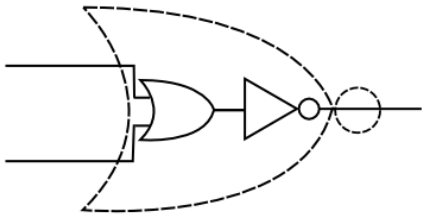
或门

A \ B	0	1
0	0	1
1	1	1

| 正所谓“按下葫芦起了瓢”。

这样一种组合我们称之为“或非门（Not Or Gate，又称为 Nor Gate）”，用以下符号表示：

它是对以下组合的一种抽象：



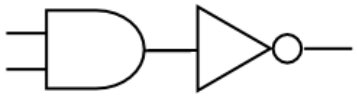
当然它不能满足加位器的要求，我们可以再试试其它的组合。

与非门

同理当把与门的输出又接在一个非门上时，与门的输出都被取反：

A \ B	0	1
0	0	0
1	0	1

A \ B	0	1
0	1	1
1	1	0



这一次，对比加位器的要求，右下角 OK 了，但左上角又不行了。

加位器

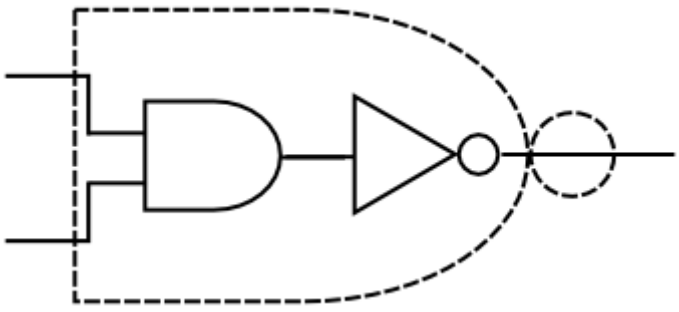
A \ B	0	1
0	0	1
1	1	0

与非门

A \ B	0	1
0	1	1
1	1	0

这样的一种组合，我们称之为与非门（Not And Gate，或 Nand Gate）。用以下符号表示：

它是对以下组合的一种抽象：



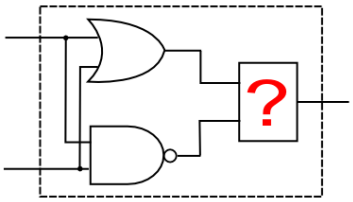
显然，与非门还是差了一点点从而无法满足加位器的要求。

异或门

如果观察与非门以及之前的或门与加位器之间的逻辑：

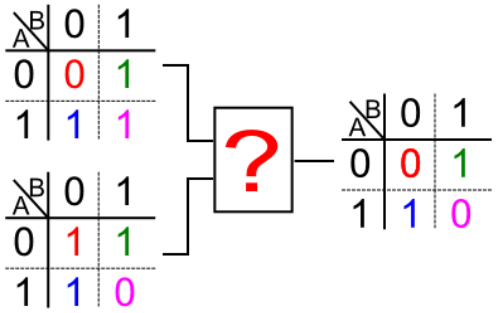
显然，它们各自覆盖了加位器逻辑的一大部分，如果它们能够以某种方式再次组合起来，是不是能全面覆盖加位器的逻辑呢？

换句话说，我们希望在两者基础之上再去综合，也即考虑如下的加位器内部结构：



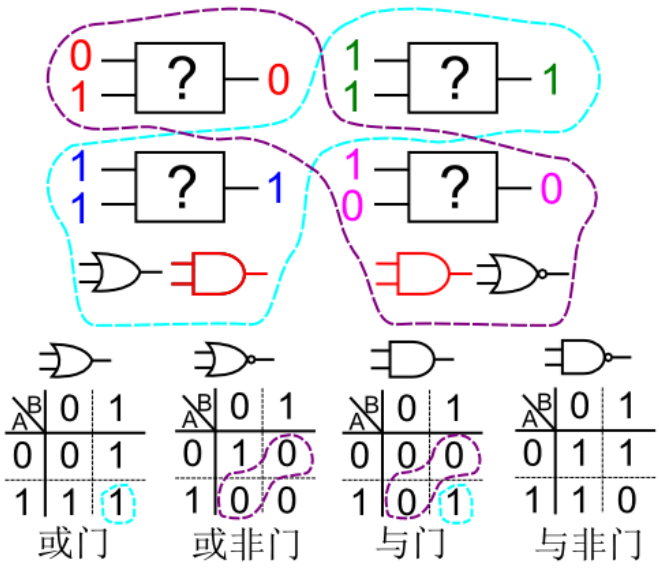
加位器

对于图中未知的部件，它要满足以下逻辑：



| 它接受前述或门和与非门的输入，并产生最终的进位器输出。

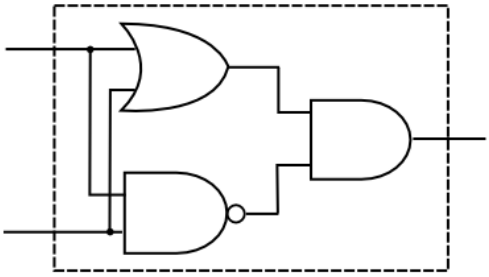
一一分解四种情况，并对比现有的四种两个输入一个输出的门电路



那么不难发现，或门跟与门均能满足上图中蓝色部分的要求，而与门跟或非门都能满足图中紫色部分的要求，只有与门是它们的交集，也即能满足所有四种情况。所以这里的未知部件可以用一个与门代替。

| 可能有人会问你为何就要这样去划分上图中的两部分呢？的确，这有点事后诸葛亮的味道，
| 实际中我们确实是需要一定的尝试，才能最终找到正确结果，另外我们也要对门电路的逻辑
| 特性非常熟悉才行。应该说我们的运气还是不错的，这里只需要一个简单的门电路就 OK 了，
| 更复杂的情况我们可能需要继续组合多个门电路才能达到要求。

至此，我们得到了加位器的结构如下：



加位器

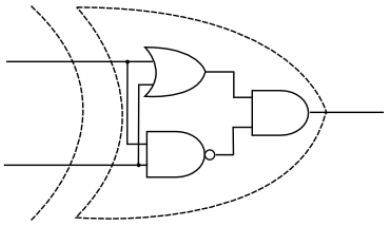
当然了，实际上没有加位器这种叫法，这样一个东西就叫“异或门（Xor Gate）”。

| Xor Gate： Exclusive Or Gate，这里强调了它与“或门”的联系及区别。

它用这样的符号去表示：

| 这个图也体现了它跟或门的渊源。

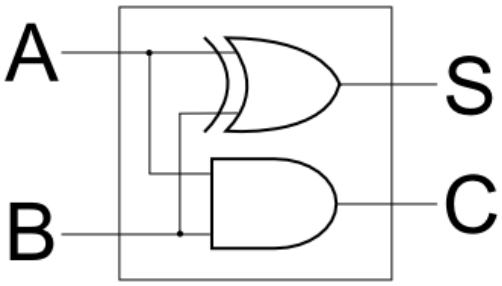
它是对以下组合的抽象：



尽管异或门跟或门的输出仅仅有一个地方不同，但最终结构上两者相差是很大的，异或门要复杂得多，更不用说异或门内部就包含一个或门。

半加器

那么，我们最终的半加器的结构是这样的：



它由一个异或门和一个与门构成，分别负责加位与进位的输出，这就是最终的二进制半加器。

在下一篇，我们再做些回顾与总结。

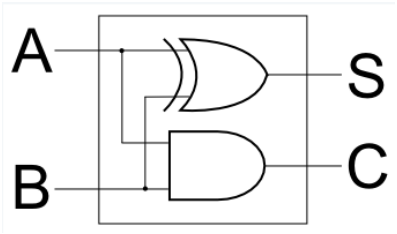
七、回顾与总结

原文地址：http://my.oschina.net/goldenshaw/blog/469162

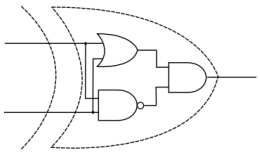
在前面，运用自顶向下的方式，我们明白了计算机做加法的方式。

继电器的网络

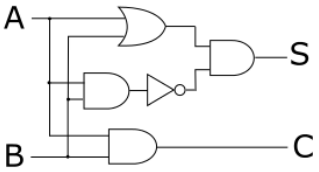
我们最终得到了半加器的模型如下：



而其中的异或器内部又是如下的结构：

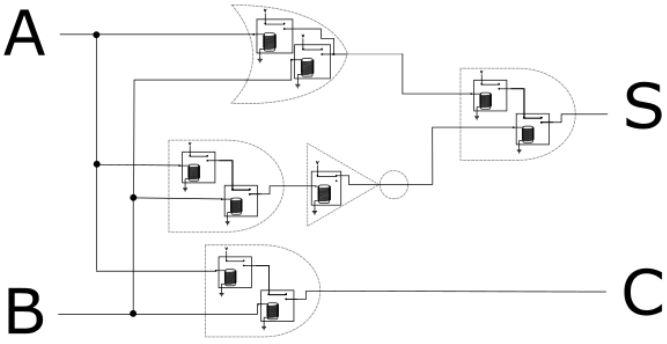


如果只用与门，或门以及非门三种电路来表示，则半加器是这样的：



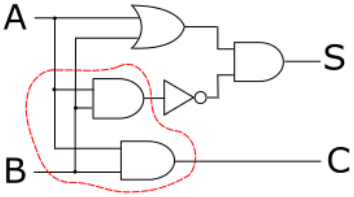
| 它由三个与门，一个或门再加一个非门组成。

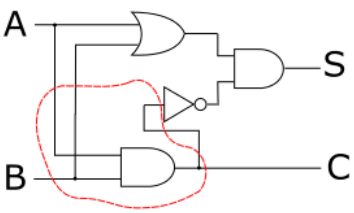
如果把其中的继电器也画出来：



显然，这是一个继电器构成的网络。清点一下，我们发现有 9 个继电器。

但实际上，注意左下角的两个与门是有点重复的，可以用一个与门来取代。





所以综合来看，只需用 7 个继电器就可以达到目的了。

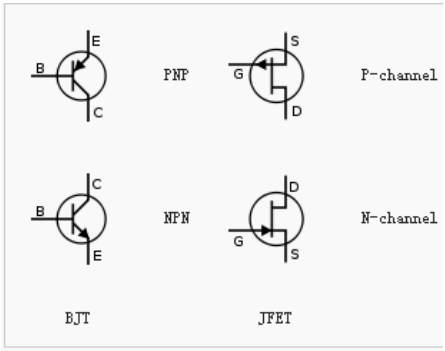
| 当我们分开分析时，可以简化分析的过程，但也可能引入一些不必要的重复，

| 这些需要在综合的时候进行取舍。

| 另一方面，由于有良好的接口抽象，内部实现细节的调整不会冲击到整体的设计。

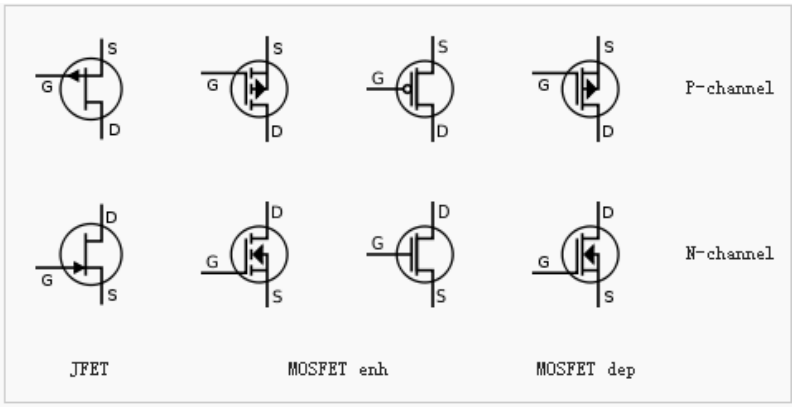
当然，对于现代的 IC 芯片，这里的继电器要换成晶体管。（这同样属于内部实现细节的问题）

| 晶体管的符号如下（来自 wiki，下同）：



BJT and JFET symbols

| 又或者是更加先进的场效应管（场效应晶体管）：

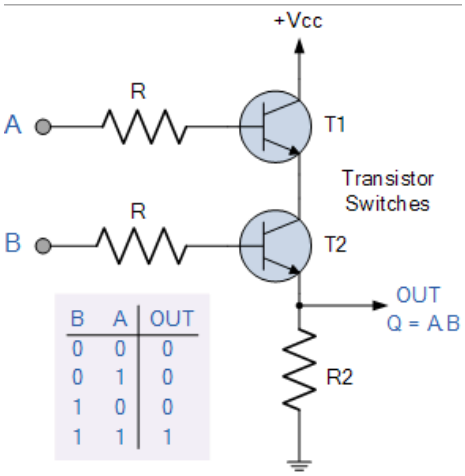


JFET and IGFET symbols

| 当然了，功能逻辑还是一样的，都是用输入去控制输出，继电器用的是电磁感应，

| 晶体管则用了半导体的特性。

| 一个与门的示例（来自 http://www.electronics-tutorials.ws/logic/log43.gif?81223b）：

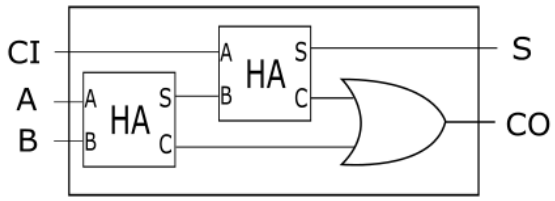


| 半导体的通断特性无法像继电器开关那般直观地展示，

| 简单地讲，AB 上无电压时，T1，T2 就像是绝缘体；有电压时则变成导体。

| 如果你对它的原理感兴趣，可自行搜索了解更多。

而前面又说到，一个全加器由两个半加器和一个或门构成：



那么，它需要 7×2+2=16 个继电器。

| 那么，比如一个四位的加法器就需要 16×4=64 个继电器开关。

| 从这里大家也可以感受一下一个简单全加器它里面基本功能元件的数量的一个大概规模。

| 注：我们倾向于认为与门，或门和非门这些为基础的门电路，

| 又认为比如“与非门”是这些门电路的组合，但实际上有可能对某些晶体管的实现而言，

| 与非门可能是更基础的构件。

人如何做一位数的加法？

在前面，我们曾经留下一个问题：人是如何做两个一位数的加法呢？

记忆之网

你靠的是记忆！

| 是的，事实就是这样。

| 你其实没有在算，当你看到 7+8 时，你的脑海里就浮现出了 15 这个数字。

如果这不能让你信服，不妨看看乘法的情况。

| 比如问你 7×8 等于多少，你脱口而出 56，你怎么能算得这么快呢？

| 这相当于要做 7 次或者 8 次加法，你一定背过乘法口诀！

| 如果再问你 12×13 等于多少？傻眼了吧，你还能脱口而出吗？出不来了吧？这是为什么呢？

可是，我们好像没有背过加法口诀？

| 但其实你要是看看小学一年级的学生的数学作业本，

| 老师会让他们反复的练习一位数的加法，直到形成条件反射！

| 虽然我们没有主动去背过什么加法表，

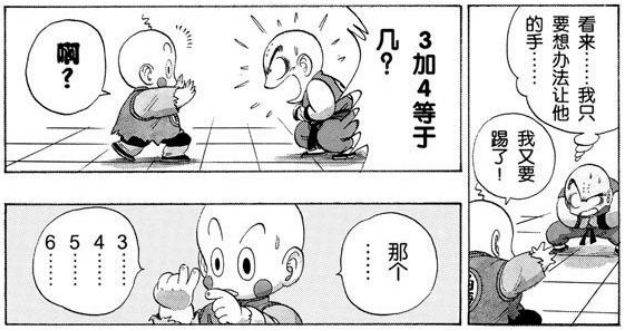
| 但反复的练习让我们在不知不觉中已经记住了所有的规则！

太初有道

那么，我们什么时候才是不靠记忆来做加法的呢？

在最初，我们先学会数数，1，2，3，4，5，6，7，8，9，10.反复念叨直至滚瓜烂熟，以此建立数的先后次序，也即所谓大小的概念。

然后，我们怎么做加法呢？



| 以上截图来自腾讯动漫《龙珠》http://ac.qq.com/ComicView/chapter/id/505436/cid/121

没错，我们靠数指头来做加法！十个指头就是我们的终极武器。这很可能是你小时做 2+3 时的情形：

| 你看到一个 2，于是你数“1，2”，你边数边伸指头，看到 3，你继续数“1，2，3”；

| 然后你再次从一开始清点所有伸出的指头，1，2，3，4，5，

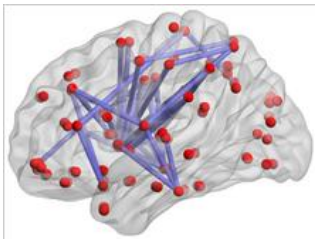
| 你边清点边把指头缩回去，最终你得出了 5 是这次加法的结果。

当然，做得多了之后，你就无需再数了，因为你已经记住了。把一位加法弄得滚瓜烂熟之后，多数人甚至都可以轻松地心算两位数加法了：

| 在算数这个问题上，库林（图中头上零根毛的那位）完全是碾压饺子（头上一根毛的）的节奏。好卑鄙的库林！:）

一种猜测

经过反复的演算，可以这么猜测，我们其实也是在我们脑海中构建了一个一位数加法的网络：

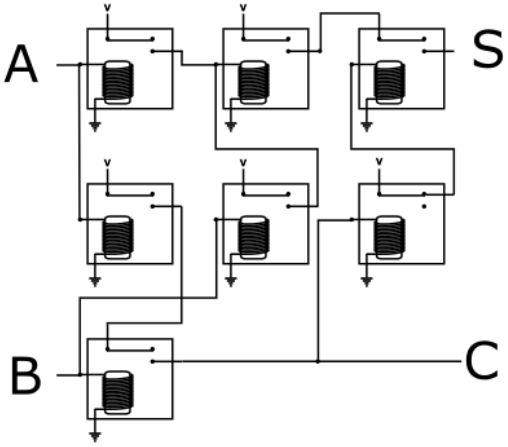


当我们看到比如一个加式：“2+3=”时，这些符号经眼睛传入我们大脑之后引发了相应神经元的一系列电化学反应，最终我们得出了“5”这个结果。

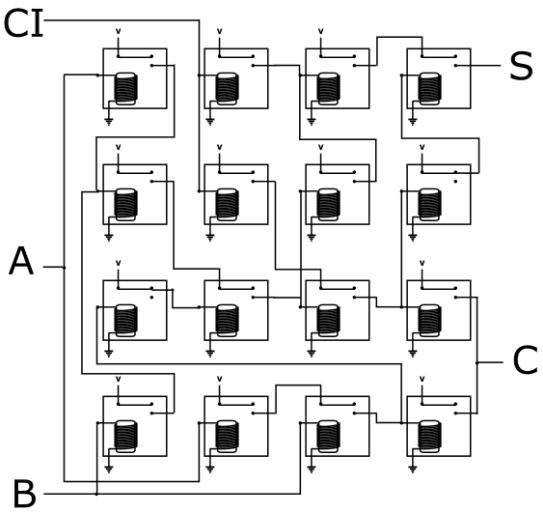
| 2+3=5，这是一个我们熟知的事实，与其说我们在“运算”，不如说我们只是在记忆之网中把这个结果提取出来。

究竟做了什么？

下图是前述 7 个继电器构成的半加器网络的一个等价连接：



如果以上还不让你觉得复杂，那么再看下如下全加器的一个等价连接，这次有 16 个继电器开关：



在这里，半加器及门电路级的抽象层全部被丢弃了。如果一开始就给你一堆这样的连接，你能意识到它们是在做二进制的加法吗？

| 与此类似，在软件层面，如果代码缺少抽象层级，把过多的细节无情地暴露出来，
| 通常也会导致代码难以理解。

对每个继电器（晶体管）而言，它们显然不知道自己在做什么，每个继电器只不过是简单根据输入的情况输出一个响应而已。

| 一切不过是机械地通通断断而已，
| 所谓“做加法”，只不过我们精心按着加法的逻辑凑出来的一个结果。
| 显然，你换种方式去连接它们，只要你能凑出加法的逻辑，你都可以说你实现了一个加法器。
| 当然我们肯定希望用最简单经济的方式去实现它。

网络从哪里来？

那么对于我们人来说呢？我们不妨问自己一个问题：当我们在做加法的时候，我们究竟在做什么？

| 数字及运算符号经眼睛进入大脑（输入），引发了神经元的一系列充放电反应，
| 最终出来一个结果（输出），看上去与上述过程也很像。

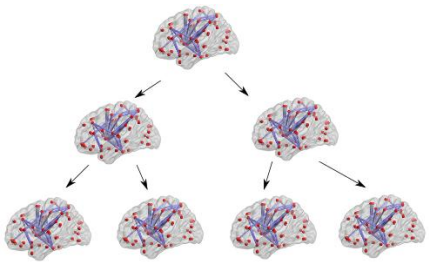
我们脑海中为何会有这样一个对抽象符号作出反应的网络呢？

| 当我们长到六七岁时，幸福的童年突然就结束了，我们觉得被送到了一所
| “监狱”（大人们称之为“小学”）一样的地方，学习那些来自阿拉伯的符号，
| 以及一种称为“加法”的运算。用的还是十进制！
| 自然，这些抽象的有那么一点点反人类的东西让不少小盆友叫苦不迭。

于是乎，经过老师们（或许还有家长们）的威逼利诱（你也可称之为“努力“），这些叫“加法”的东西终于扎根小朋友的脑海。

| 事实上，如果我们愿意承认的话，我们接受这些知识完全是被动的。

而老师头脑中的网络则可能是来自老师的老师，然后又来自老师的老师的老师...



| 进化生物学家道金斯在他的《自私的基因》一书中把这种现象与基因的遗传与变异作了对比，
| 他用了一个词“模因（meme）”来描述这种文化遗传因子，模因在人脑中“繁殖”与“变异”，
| 同时也以其它形式存在如书本，电脑上等。（未来也许不再需要人脑这种介质？）

所谓的”智能“

关于算术的知识能在人脑中不断传承下来，并得到发展，也许是因为我们一度认为这样的东西是人这种聪明的，有智能的生物才能去掌握的。

| 阿猫阿狗们好像就不能做到这些，或者说做得不好，

| 以至于如果真能做到很简单的一些算术也会当成一条新闻。

但现在，连一堆电路也能做到这些，甚至做得比人类还快还好；另一方面，当我们深入分析做加法的过程时，到了最底层，我们甚至有些疑惑：加法的意义究竟在何处？我们只看到一堆的连线和一些通通断断的器件，意义似乎消解了，只剩下一些“机械的（mechanic）”东西，这或许正是我们能用机器（machine）去做它们的原因。

这一过程真的体现了所谓的“智能”吗？

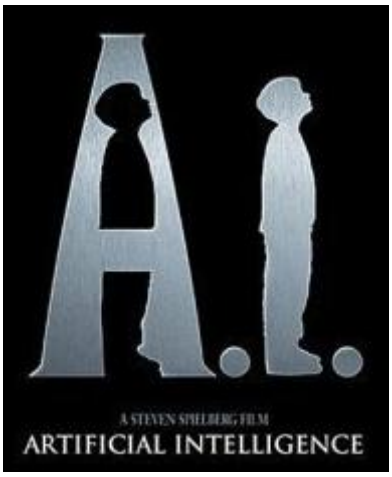
| 我们人类或许只是更复杂“机器”而已？又或者说，所谓的”智能“究竟是什么呢？

量变到质变？

据估计，人脑中可能有高达一千亿左右的神经元细胞，然后彼此间以非常复杂的方式连接在一起。

而现在那些超大规模的集成电路，比如 cpu 的核心，指甲那么大小一片上可以集成高达 20 亿个晶体管，彼此间的连接同样也是非常复杂，而这一切还在不断的发展中，那么我们会面临怎样的可能性呢？

如果你愿意仔细去思考的话，这些或许是一个很深奥的问题。



| 以上为大导演史蒂文·斯皮尔伯格（Steven Spielberg）的电影《人工智能》（Artificial Intelligence）的海报。