



# **But** How Do It **Know?**

*The Basic Principles of Computers for Everyone*

**J. Clark Scott**

但它怎么知道？

计算机的基本原理  
对每个人来说

依据

克拉克·斯科特

版权所有©2009 By John Clark Scott

计算机设计并入本文版权所有©200

9由John Clark Scott

保留所有权利

作者:John C.Scott, Oldsmar, FL 34677

ISBN 978-0-615-30376-5

buthowdoitknow.com

封面艺术, 摄影和设计, 亚历山大C。斯科特三世, Art

byAlexScott.com

在美利坚合众国印刷

第一版:2009年7月

10 9 8 7 6 5 4 3 2 1

# 目录目录

[目录目录](#)

[导言](#)

[事实就是这样](#)

[速度](#)

[语言](#)

[就一点点](#)

[怎么...?](#)

[简单的变化](#)

[图表](#)

[记得什么时候](#)

[我们能用一点做什么?](#)

[别名玫瑰](#)

[八个就够了](#)

[代码](#)

[返回字节](#)

[MAGIC总线](#)

[m矿门组合](#)

[计算机的前半部分](#)

[数字](#)

[地址](#)

[电脑的另一半](#)

[m矿石闸门](#)

[我正在使用字节](#)

[左移和右移](#)

[记分员](#)

[安德尔](#)

[奥雷尔](#)

[独家Orer](#)

[加法器](#)

[比较器和零](#)

[逻辑](#)

[算术逻辑单元](#)

我精通处理器

时钟

做些有用的事

一样一样来

一切都在掌控之中

做一些有用的事，重新审视

接下来是什么？

第一个伟大的发明

说明书

算术或逻辑指令

加载和存储指令

数据指令

第二大发明

跳跃的另一种方式

第三大发明

清除标志指令

塔达！

关于算术的几个字

外面的世界

键盘

显示屏

另一个代码

关于代码的最后一句话

磁盘

对不起，我是

这就是所有的人

硬件和软件

程序

操作系统

语言

文件系统

错误

电脑疾病？

固件

靴子

数字与模拟

我撒了谎-有点

全面披露  
哲学

## 导言

这本书的书名是这样一个古老笑话的妙语：

乔是个很好的人，但总是有点迟钝。

他走进一家商店，一个推销员站在一群人面前的肥皂盒上。

推销员正在推销一项奇迹般的新发明保温瓶。

他说：“它能使热的食物保持热，使冷的食物保持冷。 . . .”

乔想了一会儿，惊讶于这个新发明能够

根据你放进去的食物的种类，决定它应该做哪两件不同的事情。

他无法抑制自己的好奇心，他跳上跳下，在空中挥舞手臂，说：“但是，但是，但是，但是……”最后他脱口而出他那燃烧的问题：“但是它怎么知道呢？”

你可能对这个笑话笑了，也可能没有笑，但关键是乔看了看这个保温瓶能做些什么，决定它必须能够感应到里面的东西，然后相应地进行加热或冷却操作。

他想它一定有加热器和冰箱。

他不知道它实际运行的更简单的原理，即热量总是试图从一个更热的区域移动到一个更冷的区域，而保温瓶所做的就是

放慢这个动作。

有了冷的内容物，外界的热量在进入的过程中就会减慢，而有了热的内容物，热量在离开的过程中就会减慢。

瓶子不需要“知道”来完成它的使命，也不需要加热或冷却任何东西。

最终，里面的东西，不管是热的还是冷的，最终都是在室温下。

但是乔关于瓶子是如何工作的概念要比事实复杂得多。

所以这本书的标题，是因为当谈到电脑，人们看他们，看看他们能做什么，想象各种各样的事情，必须在这些机器。

或者他们想像他们必须基于的各种原则，因此他们可能能够做到什么。

人们可以把人的品质赋予机器。

很多人发现自己陷入了尴尬的境地，就像我们在笑话中的朋友乔一样。

但是计算机其实很容易理解。

当然，计算机比保温瓶有更多的部件，但是每一个部件都非常简单，它们都是按照非常简单、非常容易理解的原理操作的。

在计算机中，它的工作原理与电有关，但这并不意味着它很难理解。

如果你观察到当你打开一个电灯开关时，一个灯泡亮起来，当你关闭开关时，灯变暗，那么

你已经遵守了计算机的工作原理。这就是你了解电脑所需要的关于电的全部知识。

## 只是事实，夫人

这本书的主要目的不是要成为一本教科书。在每一章的结尾都没有问题要做。它的目的仅仅是为那些想知道盒子里到底发生了什么的人揭开电脑的神秘面纱。当然，它也为最终获得计算机科学博士学位的年轻人提供了一个完美的计算机入门。不过，对于家庭主妇、长者和阅读能力良好的儿童来说，这是很容易理解的。它应该在水管工和清洁工都能理解。它不需要以前的技术教育。它只需要你能读懂语言，你能打开和关闭一个灯泡，你能做非常简单的加法，数量级为 $8+5=13$ 。

这本书介绍了构成计算机的全部要素。它以适当的顺序呈现每一件作品和每一部分，以便每一件作品都有意义，并且能够被理解。每一部分都得到了充分的解释，每一个新词在首次使用时都得到了彻底的定义。任何进一步简化该主题的尝试都会在大图中留下空白，在大图中，仍然需要有人猜测各部分如何协同工作，而您永远不会听到“啊哈，我明白了！”这一刻，我认为您很快就会有的。

这本书不是什么大学教科书的“呆板”版本。这是对计算机基本原理的完整解释。这是一本技术书籍，但也是一本烹饪书，司机教育手册。本书从一开始就定义了理解机器所需的每一项内容。不管某人对计算机已经了解多少，这将填补任何缺失的部分，并把它们放在一起成为有意义的东西。就连我们的朋友乔也能用功地读懂这本书。有成千上万的单词和想法与计算机领域有关，使整个主题似乎一团糟。但它们背后的基本概念很简单。在这本书中，将不会有大量关于计算机的构造或历史的琐事，仅仅是基本要素，不多也不少。计算机的每一部分都有一个简单的功能，当它们连接在一起时，你就会得到一台叫做计算机的有用的机器。

这本书里没有什么值得记住的。

每一章都是为了给你一个你以前没有的新想法，或者如果这是你以前听说过的东西，它似乎总是令人困惑。每一个想法都很简单，一件事导致下一件事。

每一章都提出了一个想法。每个想法都很简单，容易理解。

后面的章节将在前几章的基础上提出一些想法。

如果有人写一本关于如何盖房子的书，可能会有不同层次的细节。

最简单的书会说：“打地基，筑墙，盖屋顶，

把水管和电器装进去，你就完成了，“对于那些还没有使用锤子、锯、安装水龙头和接电线灯开关经验的人来说，这还不够详细。

我们将展示计算机是由一个简单的部件组成的，然后将它们连接在一起，直到我们构建了一台完整的计算机。这将比你想象的简单得多。

## 速度

电脑看起来神秘而神奇。他们怎么能做他们该做的？

他们玩游戏，画画，他们“知道”你的信用评级。这些机器能做各种各样奇妙的事情。但它们很简单。他们只能做很少，很简单的事情。

而且，他们一次只能做一件简单的事情。

他们似乎在做复杂的事情，只是因为他们在很短的时间内一个接一个地做了大量简单的事情。结果，如

一款电子游戏，外表很复杂，但实际上很简单，只是非常非常快。

计算机被设计成做少量特定的简单的事情，并且一个接一个地快速地做这些事情。

哪些简单的事情被完成，按照什么顺序，决定了计算机在任何给定的时间内完成什么样的任务，但是计算机所做的任何事情都不包含超出其有限能力的任何内容。

一旦你看到一台电脑是由什么组成的，你就会意识到他们是如何做到他们所做的，确切地说，他们能做什么，也不能做什么。

因此，计算机的秘密不在于它们的复杂程度，而在于它们的速度。

让我们看看他们的速度到底有多快。

由于计算机是靠电工作的，所以它们的速度与电的速度有关。

你可能还记得听说过光速是每秒186,000英里。真快啊。

光可以在一秒钟内绕地球七圈，或者在大约一秒半内从地球到月亮。

根据物理学家的说法，电与光有许多共同的性质，当它在电线中行进时，速度会减慢到光速的一半左右。但是，Goi

NG在一秒钟内绕地球三周半是非常快的。

作为一个比较，想象这是一个炎热的一天，你有一个电风扇坐在桌子上吹冷风对你。风扇旋转得如此之快，以至于叶片模糊不清，但它每秒只旋转40次左右。

其中一个叶片边缘上的一个点在那秒内只会移动大约150英尺，这个点只需要35秒就能移动1英里。

由于风扇叶片已经是一个模糊的，很难想象它们只会快十倍。

如果是这样的话，那扇风扇就会吹出一阵微风。

如果你能让它跑得更快一百倍，它几乎肯定会自我毁灭，风扇叶片会断裂，卡在天花板上。但是，在同一圈中运行的电流在一秒钟内会绕1亿圈，比风扇叶片快250万倍。真快啊。

一百万是一个很大的数字。

如果你拿一张40英寸见方的大纸，拿把尺子放在纸的上边缘，沿着纸的上边缘每英寸画25个点，你就必须画1000个点才能穿过那张纸。

如果你把尺子向下移动 $1/25$ 英寸，再画一千个点，然后继续这样做，你就必须把尺子向下移动一千次，每次画一千个点。如果你

如果你能完成这么无聊的任务，你就会得到一张上面有一百万个点的纸。

有很多点什么的。

最后，如果你能找到一千个人，他们每人画一百万张圆点纸，然后把那一千张纸堆成一堆，你就会得到十亿个圆点。

现在让我们假设，在计算机内部移动的电流可以通过移动一英尺来完成一些简单的任务。这意味着计算机在一秒钟内可以做5亿件简单的事情。

相比之下，桌上的风扇旋转7个小时，只转100万次，大约需要整整6个月的时间才能转5亿次。

当你谈论电流在计算机内部的部件之间移动的速度时，你可以看到一些部件相距一英尺，一些更近，一英寸，十分之一英寸。

在这些部分里面，有更多的部分彼此非常接近，有的仅相隔千分之一英寸。

而且电的传播距离越短，它就越快到达那里。

说今天的电脑一秒钟能做多少事是没有意义的，因为那会使这本书过时。计算机制造商继续生产速度是过去两三年最快计算机的两倍的新计算机。他们的速度在理论上是有限的，但是工程师们不断地寻找实际的方法来绕开理论，制造出越来越快的机器。

在计算机变得更快、更小、更便宜的这段时间里，计算机做的事情，自20世纪40年代首次被发明以来，确实没有改变。

他们仍然做同样的一些简单的事情，只是更快，更便宜，更可靠，在一个更小的包。一台计算机只有几个部分，它们都是由相同的部件组成的。

每个部分都有一个特定的任务，把这些部件组合成一台机器是一个真正了不起的发明。但这并不难理解。

### 语言

在本书中，我们将需要定义一些用于描述计算机内部部件的单词。

每一个单词在第一次使用时都会被详细描述。

尽管从整个计算机行业来看，有成千上万的单词和缩略语在使用，但理解计算机本身只需要十几两个单词。

你可能以前听过一些这样的词，并从它们的用法中理解了它们的含义，但现在你将得到正确和完整的定义。在许多情况下，你可能会发现它们比你想象的要简单。

## 就一点点

电脑里有什么？

它显示你的静止图片，动态图片，音乐，你的支票簿，你写的信，它玩电子游戏，沟通世界各地，以及更多。但是电脑里有图片吗？

如果你拿出显微镜，知道到哪里去看，你能在电脑里找到一些小图片吗？

你会看到“A”和“B”和“8”和“12”在某处移动吗？

答案是否定的，电脑里没有图片、数字或字母。                          电脑里只有一种东西。

这种东西很多，但里面只有一种。这叫做一点点。

当你把一枚硬币抛向空中，让它落在地上时，它会以两种可能的状态中的一种落在地上——要么是正面朝上，要么是尾巴朝上。

客厅里的灯（假设你有开关而不是调光器）可以开也可以关。

你前门上的锁可以上锁也可以解锁。

所有这些事情都有什么共同点？  它们都包含一个可以处于两种可能状态之一的东西。  
这是位的定义。

比特是一种在空间中具有一定大小和位置的物理物体，它本身具有某种性质，在任何给定的时间都可以处于两种可能的状态中的一种，并且可以在这两种状态之间来回变化。

一块粘土一点儿也不。

它可以被模塑成一个球，一个立方体，一个薄煎饼，一个戒指，一个圆木，一张脸或任何你能想到的东西。

它有一个大小和空间位置，但有太多的状态，它可以被称为有点。

如果你把那块黏土弄平，一边刮“是”，另一边刮“不是”，然后放进窑里烧到硬为止，那么你也许可以称之为“是”。它可以坐在桌子上

“是”或“否”显示。那么它将只有两个状态。

你以前可能听说过与计算机有关的比特，现在你知道它们是什么了。

在计算机中，比特不像硬币或锁，它们最像光。

也就是说，计算机中的位不是有电就是没有电。

在计算机中，比特非常非常小，并且有非常大量的比特，但是这就是所有的比特。

就像客厅里的灯一样，这部分要么亮着要么关着。在客厅里，墙上的电通到开关里。当你打开开关，电流从开关，通过电线在墙上和天花板，进入灯插座，然后进入灯泡。所以客厅里的这个部分有几英尺长，包括开关、电线、插座和灯泡。

在计算机中，比特大多是微小的，实际上是微观的。

而且，计算机位的一端没有机械开关，另一端没有灯泡。

如果你把灯泡从客厅的插座上取下来，开关在打开的时候仍然会把电送到插座上，而且还会有点小——你只是不能通过看一个灯泡就看出它是开着还是关着。

你的电脑有一些类似开关的东西，比如键盘上的按键，还有一些类似灯泡的东西，比如屏幕上的小点，

但大部分碎片都在里面，看不见。

这基本上是所有的电脑位元。

它们有很多很多，它们以各种方式排列和连接起来，我们将随着本书的进展详细研究，但这是所有计算机内部的东西——比特。

比特总是处于它的两种可能状态中的一种，或者关闭或者打开，并且当它们被告知这样做时，它们在打开和关闭之间变化。

计算机位不像硬币那样，必须在物理上翻转才能从一种状态转换到另一种状态。

位不CH

橙色的形状或位置，它们看起来没有任何不同，它们不会移动或旋转，也不会变得更大或更小。计算机位只是一个地方，如果那个地方没有电，那么位就关了。

当有电时，钻头就接通了。

如果你想把一枚硬币从正面变成反面，你必须把它从正面翻过来，这需要一定的时间。

因为唯一需要在计算机位中移动的是电，所以将它的状态从OFF切换到ON或ON切换到OFF比任何需要物理移动的事情都要快得多。

所以我希望这能为你简化电脑的话题。电脑里只有一件事，比特。

有很多是肯定的，但当你理解了一些东西，你就会明白里面是什么。

## 怎么...?

想象一下，这是一个阳光明媚的日子，你走进一个窗户开着的房间。你注意到天花板上的灯亮着。你认为这是浪费，你要把灯关掉。你看紧挨着门的墙，看到一个带两个开关的开关板。所以你假设离门近的是天花板上的灯。但是您注意到交换机已经关闭。另一个开关也关了。所以你会想“也许有人把开关装上了倒过来，“所以你还是决定打开开关。你翻来覆去，但什么也没发生，天花板上的灯一直亮着。所以你决定它一定是另一个开关，然后你打开，关闭，打开，关闭。天花板上的灯光继续照耀着你，什么也没发生。你环顾四周，没有其他的门，没有其他的开关，没有明显的方式关闭这该死的灯。一定是这两个开关中的一个，到底是谁建了这个疯狂的房子？所以你拿了一小杯用每只手握住它们，然后开始疯狂地翻转它们。然后突然你注意到天花板上的灯瞬间熄灭了。所以当天花板上的灯熄灭时，你就会放慢开关的开关。两个开关都显示“ON”（接通），灯现在熄灭了。你关掉一个开关，然后打开，灯亮了，然后后退。这是倒退。关一次灯等于开一次灯？然后你关掉另一个开关，然后打开，同样的事情，灯亮了，然后后退。搞什么鬼？不管怎样，你终于弄明白它是怎么运作的。如果两个开关都接通，灯就会熄灭。如果一个或另一个或两个开关都关闭，则天花板灯亮。有点傻，但是你达到了你的目的，你打开了两个开关，灯熄灭了，你就离开了这个疯狂的房间。

这个关于奇怪的电灯开关的小故事的目的是什么？答案是，在这一章中，我们将介绍计算机最基本的组成部分。这部分的工作原理和那个奇怪房间的照明系统一模一样。这个计算机部件是一个简单的设备，它有三个连接点，可能有电，也可能没有电。其中两个连接点是可以将电放入设备中的地方，第三个连接点是可以将电从设备中出来的地方。

在这三个连接中，有两个被称为“输入”，因为电可以从其他地方发送到它们。

第三种连接被称为“输出”，因为电流可以从其中出来，然后被送到其他地方。

此计算机部件是一种使用位执行某些操作的设备。

如果有两个位，并且将这两个位连接到输入端，则该器件会“查看”这两个位，并“决定”是否打开或关闭一个输出位。

它“决定”的方式非常简单，而且总是一样的。

如果两个输入均为ON，则输出将为OFF。

如果一个或两个输入都关闭，则输出将打开。

这就是那个有奇怪电灯开关的房间的工作原理。

请记住，计算机内除了位之外什么也没有。这个简单的设备就是比特的来源和去向。

这种设备的“决定”是如何打开和关闭计算机中的位。

两位进入器件，一位流出。

两位来自其它地方，由器件检查，并生成新的第三位，以便它可以转到其它地方。

如果你非常善于观察，你可能会问自己这样一个问题：“当两个输入都关闭时，输出就会打开，所以……如果两个输入端都有电，如何在输出端获得电

这是一个很好的问题，很好的答案是，这些设备中的每一个都与电源相连。

就像你家的每一个电器或台灯，每个都有一个插头，插头上有两个插头，这个设备有一对电线，其中一个连接到一个地方，电源总是开着，另一个连接到一个地方，电源总是关着。这就是输出电流的来源。当有人制造一台计算机时，他们已经

E将所有这些电源连接到每一个器件以使其工作，但当我们绘制器件的关系图、它们的连接方式以及它们将执行的操作时，我们不会费心绘制电源线，它们只会使绘图变得杂乱。据了解，每个部分都有自己的电源连接，我们不担心它。

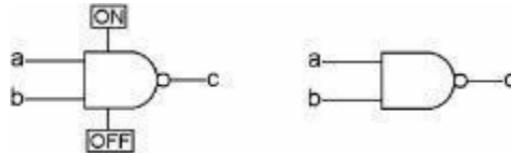
只要明白它就在那里，我们就不会再在本书的其余部分再提它了。要不是我在这里

我猜你迟早会问自己这个问题。

现在我知道我说过，你不需要了解很多关于电的知识就能理解电脑。这是最复杂的了。

这款设备内部实际上有六个电子部件，但我们不打算在本书中讨论这些部件。

有电子学背景的人可以看看里面有什么，大约30秒后会说：“哦，是的，如果两个输入都打开，输出就会关闭，对于任何其他组合，输出就会打开，



就像书上说的，“然后那个人就可以继续读这本书，而不必再去想书里的内容。

不懂电子学的人会错过这几秒钟的理解，但这本书对每个人来说都是一样的。

在正常的室内布线中，一个开关打开和关闭一个灯。

在计算机中，需要两个开关，这是一种倒退，因为它们都必须打开才能关灯。

但是如果你接受这样的事实，你就能理解计算机中的一切是如何工作的。

这种类型的计算机部件实际上是在构建计算机所需的唯一类型的部件。当然，构建一台完整的计算机需要大量的计算机，但是有了足够的计算机，您就可以制造任何类型的计算机。你又来了，看看电脑有多简单？它只是充满了这种小类型的东西——很多是肯定的，但这是所有的。	它被称为“门”，我找不到一个很好的理由为什么它被称为“门”，栅栏中的门在打开时让人通过，在关闭时阻止人通过。计算机门从另外两个位产生第三个位，它不打开和关闭，也不停止或让任何东西通过。这个计算机术语“门”的含义似乎不符合这个词的一般含义，但很抱歉，这个名字不是我编的，这就是它的名字。你会习惯的。至少这不是什么骗局
g古希腊语单词。	在接下来的几章中，我们将展示如何通过将几个

好的。右边的图纸显示了我们需要的一切：

	<p>门连接在一起 做一些有用的事 情。</p> <p>我们将使用如下 图纸。</p> <p>顶端有一个小圆 圈的“D”形代表我 们所描述的设备 ，线条代表连接 到计算机其他部 分的电线进出。 左边的图片显示 了一个门及其电 源线，但正如所 承诺的，我们不 会关心他们的其 余部分</p>	
<p>这是一个大门的 画像。</p> <p>左边的两条线（ A和B）是输入 ，右边的线(C) 是输出。</p> <p>所有三条线都是 位，这意味着它 们要么接通要么 断开。</p> <p>每个输入位都来 自计算机中的其 他位置，并且根 据它来自何处发 生的情况而开启 或关闭，然后该 门根据其两个输 入的状态来设置 其输出开启或关 闭。</p>		
b	c	F的
F的	打开	F的

每一行都显示了输入的一种可能的组合，以及在这些情况下的输出。

把这张小图和用两个电灯开关的奇数房间的经验进行比较。

如果一个开关叫做'a'，另一个开关叫做'b'，天花板上的灯叫做'c'，那么这个小图表就完整而准确地描述了那个房间里的设备是如何工作的。

把灯关掉的唯一方法是同时打开开关'a'和'b'。

## 简单的变化

如前所述，这个网关是构建计算机所需要的唯一东西，但是您需要大量的网关，它们必须以智能的方式连接在一起，以便能够使它们做一些有用的事情。

我们在这里要做的是展示两件简单的事情，这两件事情在任何一台计算机上都要做很多次。

第一个很简单。取上面的栅极，取两条输入线'a'和'b'，并将它们连接在一起。

因此，'a'和'b'总是相同的。它们仍然可以断断续续地更改，但'a'和'b'永远不会不同。

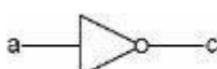
'a'和'b'可以都打开，也可以都关闭。

因此，这种组合的图表上只有两条线，两种可能性：



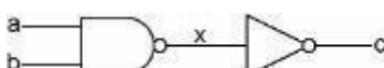
a	b	c
F的	F的	打开
打开	打开	F的

实际上，由于列'a'和'b'是相同的，所以实际上只有一个输入，它可以用三角形而不是'd'形状简单地绘制。它的图表也非常简单：



a	c
F的	打开
打开	F的

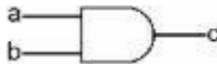
对于我们的第二个变体，让我们将一个原始类型的门与我们刚刚发明的新门结合起来，如下所示：



我们会把它们的工作原理结合起来。'a'、'b'和'x'就像第一扇门，'x'和'c'就像第二扇门。

a	b	x	c
F的	F的	打开	F的
F的	打开	打开	F的
打开	F的	打开	F的
打开	打开	F的	打开

这种组合在计算机内部使用得如此频繁，以至于它被构建为单个单元，并且“x”位不可连接。为了便于理解，它被绘制为一个单元，如下所示：



这张照片和我们原来大门的照片唯一不同的地方是大D后面的小圆圈不见了。

由于不使用'x'，图表也可以简化，如下所示：

a	b	c
F的	F的	F的
F的	打开	F的
打开	F的	F的
打开	打开	打开

这个图表和我们原来的大门图表的唯一区别是，“c”列中的每一项都与原来的图表中的项相反。

想象一下，这种门的组合安装在那个房间里，有两个电灯开关和天花板灯。只有两个开关都亮着，灯才能亮。

所以，如果你走进去看到灯亮着，然后看着开关，你会发现它们都亮着。

不管你决定用哪一个开关来关灯，你把它关掉，灯就会熄灭。

你可能没有注意到，如果你把两个都关掉，然后又想把灯打开，你  
不可能只需要打开一个开关就能完成。

你必须通过同样的实验，翻转两个开关，直到灯亮，你会发现一个开关以及  
这种组合门可以这样描述：要使输出接通，一个输入和

另一个输入必须同时打开。

因此，这种类型的门有一个名字，在传统的非正式术语由计算机人发明，因为它提醒我们什么词和意思，它只是简单地称为“与门”。

现在，为了填补上面有意遗漏的一些细节，我们查看的原始门与AND门类似，只是输出相反，或者AND门为负值。因此，它被称为“与非门”，或者简称为“与非门”。

将两个输入绑定在一起的简单门也有自己的名称。

输出始终与一个输入相反，也就是说，如果输入为ON，则输出不为ON (OFF)。

如果输入为OFF，则输出不为OFF (ON)。

输出总是不是输入，因此，它被称为“非门”。

注意与门和与非门的关系图之间的差异。

它们是相同的，只是在与非门的输出端有一个小圆。

看起来像一个大字母'D'的东西意味着做'AND'函数，这意味着只有当两个输入都打开时才采取行动，而小圆圈意味着切换到相反的方向。

因此，如果两个输入均导通，AND门导通；如果两个输入均导通，NAND门关断。

NOT门以一个三角形开始，

表示接受输入并将其转换为输出。圆圈的意思是转到相反的方向。

AND门在计算机中使用很多，可能是最容易理解的，但我们首先研究NAND门有两个原因。第一个也是次要的原因是NAND门是最容易构建的门。

当您必须构建大量的门时，如果您可以使用最容易构建的门类型，那么它将更便宜和更可靠。

我们首先研究与非门的第二个也是非常重要的原因是：使其成为计算机的计算机中的所有东西都可以由一个或多个与非门构成。

我们已经看到，NOT门和AND门可以由NAND门构成，我们将看到一些更有趣的组合。但每一个都是基于一个叫NAND门的愚蠢的小东西。

本章中的问题是NAND门是计算机的基本构造块，但AND门是第一个有意义的名称的门。所以我们先看了NAND门和NOT门，没有给它们起名字。

然后我们建造了一个“与”门，给它起了个名字，又回去给前两个起了名字。

作为这里语言的注释，“and”一词是普通英语中的连词。

它把两件事联系起来，比如在“我喜欢豌豆和胡萝卜”中。在计算机中，我们以两种新的方式使用这个词。首先，它是一个形容词，一个修饰名词的词。

当我们说“this is an and

gate”时，“gate”这个词是名词，“and”这个词告诉我们它是哪种门。

这就是“and”在本章中的用法。“and”也将用作动词，如“let us and this two bits”。我们将看到并使用。

## 书的后面是这样写的。

回到本书的简单主题，我们已经说过，在计算机中只有一件事，比特。现在我们看到位是用门构造的，所有的门都下到与非门。所以要理解计算机，你需要知道的就是这个非常简单的设备，NAND门。别开玩笑啦！你能理解这件事吗？这样你就能理解整个电脑了。

### 图表

如果你想看看一台机械机器是如何工作的，最好的方法就是看看它的内部，看着零件在工作时移动，拆开它等等。第二个最好的方法是从一本有很多图片的书中学习，这些图片显示了零件以及它们是如何相互作用的。

电脑也是一台机器，但唯一能在里面移动的是看不见的、无声的电。

看着电脑的内部是很无聊的，看起来根本没有发生任何事情。

计算机各个部分的实际结构是一个非常有趣的课题，但我们只想说以下几点：这项技术从一个薄晶圆开始，经过一系列的步骤，它经受各种化学物质、照相处理、加热和蒸发的金属。其结果是一种叫做“芯片”的东西，它的表面有数百万个电子部件。

该过程包括将器件连接到栅极，并连接

把大门分成完整的电脑部分。

然后芯片被封装在一块塑料中，塑料中有引脚从芯片中伸出。

其中几个插在一块板上，你就有了一台电脑。

我们将在本书中“构建”的计算机可以很容易地安装在一个不到四分之一英寸见方的芯片上。

但问题是，与机械机器不同的是，芯片的实际结构非常杂乱，难以理解，而且你根本看不到电。

我们在上一章中看到的图表是展示计算机如何工作的最好方法，所以我们最好能很好地阅读它们。

当电流从一个门的输出出来时，电流就会以最快的速度通过整条电线。

如果一个门的输出接通，那么连接到它的电线上的电就会一直接通。

如果一个门的输出是关闭的，那么整个导线都是关闭的。

我想你可以考虑一下，从大门出来的那部分也包括整个电线。

门的输入不消耗电线中的电，因此一个输出可以连接到一个或多个门的输入。

当电线连接在一起时，图中用一个点来表示，所有连接在一起的电线都像一根电线一样通电。

当线在没有点的图上交叉时，这意味着它们之间没有连接，它们不接触，两个位是分开的。

只要有选择，图表将显示从左向右，或从页面顶部向底部移动的电的路径。

然而，这将有许多例外，特别是在本书的后面。

但是你总是可以通过从一个输出端开始并跟随它到一个输入端来判断电线中的电流是以什么方式移动的。

书中大多数图表都很容易理解。

在少数情况下，还会有一个图表显示每种可能的输入组合的输出。

如果你在看图表时有困难，你可以直接在纸上用铅笔写上ONS和OFF，或者把硬币放在纸上翻过来，使正面代表ONS，反面代表OFF。

不幸的是，下一章的图表可能是整本书中最难理解的，但是一旦你掌握了它，你就会成为一个专业的图表阅读器。

## 记得什么时候

你可能听说过计算机内存，现在我们来看看到底是什么。

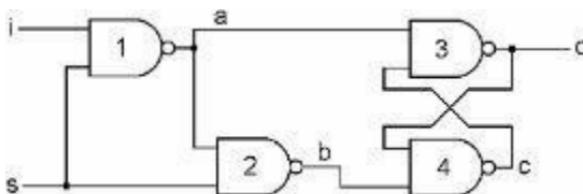
由于计算机内部唯一的东西是位，而位发生的唯一事情是它们打开或关闭，因此，计算机能够“记住”的唯一事情是位是打开还是关闭。

我们现在将看到如何做到这一点。

下图显示了一位计算机内存。这恰好是你能用几扇门做的最巧妙的把戏之一。

我们将详细研究它是如何在这里工作的，在我们理解它之后，我们将用它自己的符号来代替它，并将它用作构建更大更好事物的构件。

它只有四个NAND门，但它的布线有点特别。这是：



这种组合作为一个整体有两个输入和一个输出。

“i”是我们输入要记住的位的位置，“o”是记住的位的输出。

“%s”是一个输入，它告诉这些门何时“设置”内存。

我们还需要查看三条标注为'a'、'b'和'c'的内部线路，以了解这些器件如何协同工作。

试着仔细地遵循这一点，一旦你看到它的工作，你就会明白计算机中最重要和最常用的东西之一

r。

要了解它的工作原理，请从's'on和'i'off开始。

由于“i”和“s”进入门1，一个输入关闭，因此“a”将打开。

由于'a'和's'进入门2，两个输入均开启，因此'b'将关闭。

看看4号门，由于'b'关断，4号门的输出'c'将导通。

由于'c'和'a'均为ON，因此门3的输出'o'将关闭。

“o”返回到门4，提供第二个关断输入，而“c”仍然打开。

这里要注意的重要一点是，with's'on，'o'结束

和“i”一样。

现在，在's'仍然打开的情况下，让我们将'i'更改为'on'。

既然“I”和“S”进入1号门，“A”就会关闭。

'A'位于门2和门3的一侧，因此它们的输出'o'和'b'都必须导通。

'o'和'b'均导通，进入门4并关闭'c'，这又回到门3，为其提供第二个关断输入，而'o'仍导通。这里需要注意的重要一点与我们在上一段中注意到的一样——

在's'on中，'o'的结尾与'i'相同。

到目前为止，我们已经看到当's'打开时，你可以开关地改变'i'，'o'也会随之改变。

'o'会像'i'一样接通和断开。如果's'接通，这个组合并不比'i'和'o'之间的连线更有用。

现在让我们看看当我们关闭's'时会发生什么。

看1号门。

当's'关闭时，'a'将会打开，不管你对'i'做了什么。现在你可以打开和关闭'i'，什么也不会发生。2号门也是一样。'a'可能是开着的，但's'是关着的，所以'b'只能是开着的。'a'和'b'都是开着的，而改变'i'什么也不做。

现在唯一重要的是，最大的问题是，“o”会是什么？

如果'i'和'o'在's'关闭之前打开，则门3的两个输入都关闭，门4的两个输入都打开。

当's'关闭时，'a'接通，这是门3的一个输入。

但另一个输入是关闭的，所以没有任何变化，'o'保持为ON。

如果'i'和'o'在's'关闭之前关闭，则门3的两个输入都打开，门4的两个输入都关闭。

当's'关闭时，'b'接通，这是门4的一个输入。

但是另一个输入是关闭的，所以没有任何变化，'c'保持打开，'o'保持关闭。

因此，当's'关闭时，'o'会发生什么变化的问题的答案是，它保持原来的状态，不再受'i'的影响。

现在我们这里有什么？

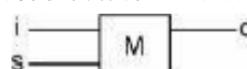
带's'on，'o'做'i'做的事。

在's'关闭之前的最后一刻，'s'关闭，'o'保持原样，'i'保持原样。

现在，“i”可以改变，但“o”仍然是原来的样子。

这种门锁的组合就像“我”在早期的时候一样。

这就是四个与非门的组合可以“记住”的方式。这只是一个内存位，但这是所有计算机内存的基本构建块。所有的计算机内存都是一种保存方法



位在某个时间点设置。

然后将“i”的状态转换为我们的内存位，然后

在“o”出现。那么“i”可能会被关闭，而不会影响任何东西。

所以，如果我们走进来，决定要关灯，我们会先试一下“T”开关，打开和关闭它，什么也不会发生。那我们就试试's'开关。当我们打开它时，灯就会熄灭。

啊哈，我们说，'s'开关控制灯，但它是上下安装的！

所以我们把's'开关关掉，希望灯能重新亮起来，但灯还是关着。 n

现在开关的位置和我们进入房间时一样，它们都关了，但是现在灯也关了  
，孩子，这让人很困惑。

现在，我不想猜测在有人发现之前会发生多少诅咒，但最终他们会发现当's'亮着时，灯会随着'i'亮着或熄灭，而当's'关着时，灯会保持在's'关掉之前的样子。

我们能用一点做什么？

现在我们已经描述了一点，我们已经展示了如何构建一个，如何随着时间的推移记住一个点在较早的时刻处于什么状态，现在呢？我们拿它怎么办？

因为一点只不过是通电或断电，所以我们能做的唯一实际的、真实的事情就是开灯或关灯、烤面包机或诸如此类的东西。

但我们也用一点来代表我们生活中的其他东西。

我们可以取一点，并将其连接到红灯，然后说，当此位打开时，表示停止；当此位关闭时，表示可以继续。或者，如果你点了一份，你想要薯条和你的汉堡一起吃；如果它是关闭的，你只想要汉堡。

这是使用代码的操作。什么是密码？代码是告诉你其他意思的东西。

当某事被认为是有意义的时候，某人必须在某处列出“thing”的所有状态，以及与这些状态中的每一个相关联的含义。

说到比特，因为它只能处于两种不同的状态，所以比特只能表示两种情况中的一种。

位的代码只需要两个含义，其中一个含义与

该位为OFF，而另一个含义将与该位为ON相关联。

这就是你给比特赋予意义的方法。该位本身不包含任何意义；

除了有电或没有电之外，一点空间都没有。M由位外部的某物分配给位。

这里一点也没有交通或炸薯条的问题，我们只是说这里的这一点，连接着十字路口的红灯，当它亮着的时候，你必须停车，当它关着的时候，你可以走了。

再来点，在现金区

ster in a fast food restaurant，意思是把油炸食品放在袋子里，或者不放油炸食品。

这是两个发明一个简单的两项代码的例子。在一种情况下，代码是:bit  
on表示炸薯条，bit off表示没有炸薯条，在另一种情况下，bit off表示go，bit  
on表示stop。

这两个位是相同的，它们只是用于不同的目的，并且有人决定这两个位的含义。

代码写在法律书籍或餐厅经理手册的某处，但代码不在位。

比特的状态只是告诉某人

他们认为当前代码的第h行是正确的。密码就是这样的。

就像间谍通过使用密码传递消息一样，消息可能被其他人看到，但是其他人没有密码，所以他们不知道消息的意思。

也许有个间谍在公寓前窗的窗台上放了一个花盆。

当花盆在窗台的左边时，意思是“1点半在火车站接我”；当花盆在窗台的右边时，意思是“不见面”

“今天，”每天，另一个间谍走在街上，抬头看着窗子，看他今天是否需要去火车站。走在那条街上的每个人都能很容易地看到这条信息，但是他们没有代码，所以这对他们来说毫无意义。然后当两个间谍相遇时，他们可以传递一张用另一个密码写的纸。他们使用见面时不携带的码本对消息进行编码和解码。所以如果他们留言是根本就没有

整封信在电线上任何时候，这些字母都顺着电线往下走，在操作员的脑海中拼凑成点和点，然后拼写成字母，最后再拼写成写在纸上的单词和句子。

因此，电报位通过在可能开启或关闭的情况下具有几个单独的时间来实现两个以上的含义。

如果一台计算机是按照莫尔斯电码的原理建造的，那么它上面就会有一个电灯泡向我们闪烁电码。

因为我们更愿意在屏幕上同时看到完整的字母、单词和句子，所以我们需要的不仅仅是一个位和这个旧代码。

即使在本章中使用的示例中，实际交通灯实际上也有三个位，一个用于红色，一个用于黄色，一个用于绿色。

如果你只有一点，你可以在十字路口开一个红灯，当红灯亮的时候就意味着停车，当红灯熄灭的时候就意味着前进。

但是当它熄灭的时候，你可能想知道它是真的熄灭了，还是灯泡刚刚烧坏了。

因此，在这种情况下，使用三位更有用。

在现实世界中，我们已经看到计算机可以包含字母、单词、句子、整本书，以及数字、图片、声音等等。然而，所有这一切都归结为比特。

如果我们希望我们的计算机内存能够保持一个以上的开或关，或是或否，我们将不得不有一个以上的位。

幸运的是，我们可以做一些更有用的事情，只需使用几个位一起，然后组成一个代码（或可能几个代码），以分配一些有用的意义。

## 别名玫瑰

在我们继续之前，我们将对我们所说的东西进行一个改变。

正如我们所知，计算机中的所有位都是有电或没有电的地方。

我们称这些州为“开”和“关”，这正是它们的本来面目。

尽管这些都是短单词，但在有些地方，用一个符号来描述这些状态要容易得多、更清楚、更简单得多。幸运的是，我们不会发明任何棘手的东西，我们只会用两个符号

我很清楚，数字0和1。从现在起，我们取消0，取消1。

有时我们还会断断续续地使用。

因此， NAND 门的图 表如下 所示：	a	b
c	0	0
1	0	1
1	1	0
1	1	1

0

当然，这很容易理解，但这里需要指出的一点是，计算机部件没有改变，唯一改变的是我们观察机器时所说的。

仅仅因为我们叫一个0或1，这并不意味着数字突然出现并在计算机中运行。

计算机中仍然没有数字（或单词、声音或图片），只有位，完全如前所述。

我们可以让CA

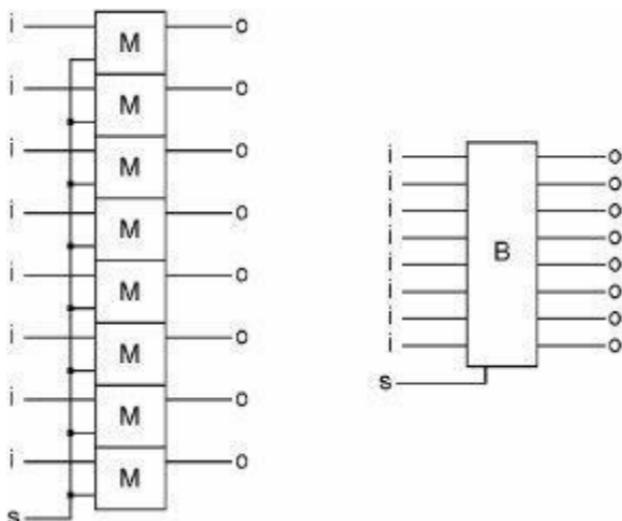
也许这一趋势源于一种利他主义的愿望，即教育公众认识到现代的“事实”，即1与on相同，但它不是事实，而是一个任意的代码。

## 八个就够了

为了能够表示比简单的是/否更重要的东西，我们要做的是将8位堆叠在一个包中，并将它们用作一个单元。这是一个如何完成的图表。

我们占用了8个内存位，每个位都有自己的数据输入'i'和输出'o'，但我们将所有8个设置输入's'连在了一起。

因此，当一个“s”被打开然后再次关闭时，所有这8个“m”将捕获它们的c的状态



或者说“我在同一时间”。

左边的图片显示了所有的8个M，右边的是一样的，只是简单了一点。

此程序集具有名称；它被称为字节，因此在图中称为“B”。

关于这个词的确切来源，有几种相互矛盾的解释，但由于它听起来就像“咬”这个词，你可以把它想象成一个整体，而不是一个更小的单位，一点点。

为了向你展示计算机设计师确实有幽默感，当他们使用四位作为一个单位时，他们称之为蚕食。所以你可以吃一小块樱桃派，或者吃一小口，或者吃一整字节。

当我们有一点的时候，我们只会说它的状态是0或1。

现在我们有了一个字节，我们将像这样写字节的内容:00000000，您可以看到为什么我们从使用OFF/ON切换到0/1。

这显示了八位中的每一位的内容，在本例中，它们都是零。

中间的空格只是为了让它更容易阅读。

左手0或1对应于我们字节中的顶端位，而最右边的0或1则表示最底层位。

正如您现在最好知道的那样，BIT有两种可能的状态，即它可以处于打开或关闭状态。如果有两个位，则这两个位可能处于四种状态。

你还记得我们为与非门的输入绘制的图表吗？

图表上有四条线，每条线对应栅极的两个输入位的可能组合:0-0、0-1、1-0和1-1。

请注意，位的顺序确实很重要——

也就是说，如果您查看两个位，只询问开启了多少位，则只有三种可能性:无位开启、一位开启或两位开启。这将是调用1-0和0-1的组合相同的东西。

为了使用多位来实现一个代码，我们当然关心字节中位的顺序。

当有两个位时，我们想要使用所有四种可能性，所以我们必须保持位的顺序。

当您使用8位时，有多少种不同的可能性？

如果只有一个位，那么它可以处于两种状态中的一种。

如果添加第二个位，则这对位的状态是以前的两倍，因为旧位有两种状态，而新位是一种状态，然后旧位有两种状态，而新位是另一种状态。所以两位有四种状态。

当您添加第三位时，前两个状态总共有四个新位为OFF，四个新位为ON

八个州中的一个。每增加一点，可能的状态数就会增加一倍。  
四位有16种状态，五位有32种，六位有64种，七位有128种，八位有256种，九位有512种状态，依此类推。

我们取8位，称之为字节。

因为位在空间中有一个位置，可以处于两种状态中的一种，所以字节在空间中有八个独立的位置，每个位置都可以是开或关的，它们保持相同的顺序。

字节作为一个整体，是空间中的一个位置，在任何给定的时间都可以处于256个状态中的任何一个，并且可以随时间改变其状态。

代码

一点点只能代表是/否类型的事物，但是现在我们有256种可能性，我们可以在我们的生活中寻找稍微复杂一些的事物。

第一件可能符合要求的事情是书面语言。

如果你看一本书，看到所有不同类型的符号是用来打印这本书，你会看到所有26个字母的字母大写以及小写。

还有数字0到9，还有标点符号，如句点、逗号、引号、问号、括号和其他几个。

还有一些特殊的符号，比如“at”符号(@, )，货币(\$, )等等。

如果你把这些加起来，5

2个字母，10个数字，几十个标点符号，你会得到大约100个不同的符号，这些符号可能会出现在普通书籍的书页上。

从这里开始，我们将使用“字符”一词来表示这类东西，字母、数字或书面语言中使用的其他符号之一。字符可以是字母、数字、标点符号或任何其他类型的符号。

这段代码有几个版本是为不同的目的而设计的，他们今天仍然举行会议，就各种深奥的细节问题达成协议。但对我们不需要关心这些来了解计算机是如何工作的。

它们的基本代码

今天还在使用，我不知道为什么它需要改变。

该代码有一个名称，它是：美国信息交换标准代码。

这通常缩写为ASCII，发音为“aass-

key”。我们不需要在这里打印整个代码，但这里有一个示例。

这是他们想出的20个代码，字母表的前10个字母是大写和小写的：

ASCII代码表的一部分

a	0100 0001	a	0110 0001
b	0100 0010	b	0110 0010
c	0100 0011	c	0110 0011
d	0100 0100	d	0110 0100
E	0100 0101	E	0110 0101
F	0100 0110	F	0110 0110
G	0100 0111	G	0110 0111
H	0100 1000	H	0110 1000
我	0100 1001	我	0110 1001
J	0100 1010	J	0110 1010

每个代码都是唯一的。

值得注意的是，他们是这样安排代码的，使得同一字母的大小写代码除了一位之外都使用相同的代码。左边的第三位对所有大写字母关闭，对所有小写字母打开。

如果你想在你的电脑屏幕上显示一条“你好，乔”的信息，你需要9个字节。

第一个字节将具有大写“H”的代码，第二个字节将具有小写“E”的代码，第三和第四个字节将具有小写“L”的代码，第五个字节将具有小写“O”的代码，第六个字节将具有空格的代码，并且字节7、8和9将包含“J”、“O”和“E”的代码。

注意，甚至有一个空格的代码（顺便说一下，它是0010 0000）。

你可能想知道为什么需要一个空格的代码，但这只是为了向你展示计算机是多么的愚蠢。

它们实际上并不包含句子或单词，只是用ASCII代码表中的代码设置了一些字节，这些代码表示我们在书面语言中使用的单个符号。

其中一个“符号”是缺少任何符号，称为空格，我们用来分隔单词。临屋区

空间告诉我们，读者，这是一个词的结尾，另一个词的开头。

计算机只有字节，每个字节可以处于其256种状态之一。

该状态对计算机没有任何意义。

因此，让我们获取一个内存字节，并将这些位设置为0100 0101。

这意味着我们已经把

字母E输入字节，对吧？ 噢……不是很喜欢。

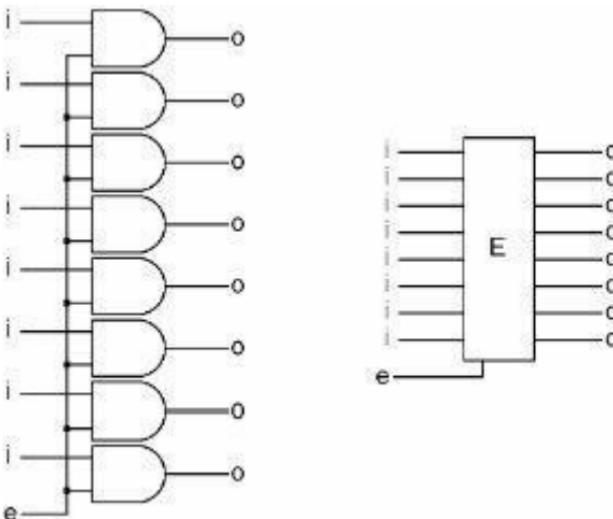
我们已经设置了ASCII代码表中字母E旁边的模式，但是字节中没有任何与“E”相关的固有内容。如果托马斯爱迪生一直在测试他的八个新的实验灯泡，并且让它们排在架子上，第一个、第三个、第四个、第五个和第七个灯泡已经烧毁，那么剩余的灯泡将是与此模式相关的一个字节。但是T上没有一个人他面对地球的时候，会看着那排灯泡，想到字母“E”，因为ASCII还没有发明。字母由代码表示。字节中唯一的内容是代码。

这就是密码的主题。

计算机代码允许您将一个字节中的256种可能模式中的每种模式与其他模式相关联。

这里的另一个语言注释是，有时单词code指的是整个模式列表及其表示的内容，如“*This message was writing with a secret code*”，有时代码只指其中一个模式，如“What code is in that byte? ”，从上下文中可以非常清楚地看出它的使用方式。

返回字节

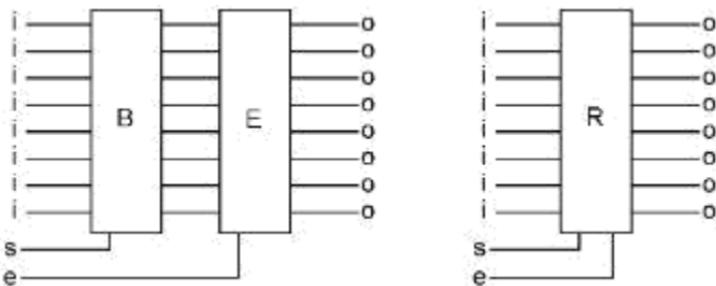


你还记得我们前几章画的记忆字节吗？这是八个内存位和他们的‘线都连接在一起。几乎每次我们需要记住计算机中的一个字节时，我们还需要一个额外的连接到字节输出的部分。这个额外的部分由8个AND门组成。

这八个AND门合在一起被称为“使能器”，左边的图显示了所有的部件，右边的图是一种更简单的绘制方法。

我们将获取字节，并将其连接到使能器，如左手绘图所示。

为了再一次简化，我们可以将其绘制为右侧所示。



现在我们有了一个可以存储8位的组合。

它可以同时捕获它们，并且可以将它们单独保存，也可以将它们释放到其他地方使用。

字节和使能器的这种组合有一个名称，称为寄存器，因此在图中为“R”。

本书中有几个地方的寄存器永远不需要关闭其输出。

在这些情况下，我们将绘制一个寄存器，该寄存器仅有一个“set”位，而没有“enable”

位。我们可能应该将这些器件称为“字节”，但我们仍然将它们称为寄存器。

Register只是指一个记录某种信息的地方，比如一个所有客人都登录的酒店登记簿，或

者一个你写下每一张支票的支票登记簿。对于此计算机部件，记录8个输入位的状态。

然而，该寄存器非常有限，因为它只能保存一组值；

在旅馆登记处，每一位客人都有一条新的电话线路。

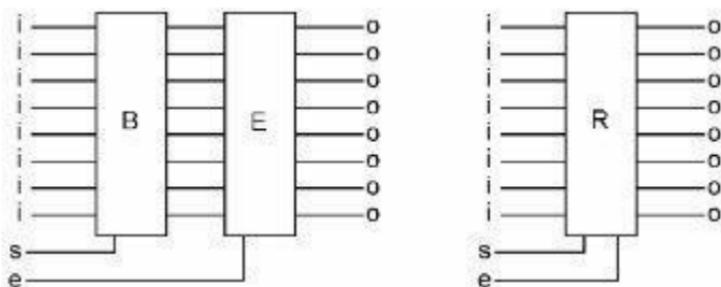
每次在计算机寄存器中存储新状态时，八个内存的前一个状态

Y位丢失。唯一在里面的是最近保存的值。

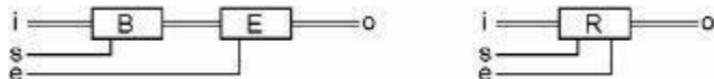
### 神奇巴士

计算机中有许多地方需要八条线将寄存器连接在一起。

例如，我们的寄存器有8个存储器位，每个存储器位都有一个输入和一个输出。  
为了简化我们的图表，我们将用双线替换我们的8条线。



因此我们的寄存器可以如下所示：



或者，我们可以简化它，并用以下其中一个替换它：

这是完全一样的事情，我们只会节省大量的墨水在我们的图纸，他们将更容易理解。



当这些线束中的两个线束之间存在连接时，每个线束中的一根线与另一个线束中的一根线连接，如左图所示。但是我们会简化它，就像右边的图表一样画出来。

现在，这组八条线在计算机中是如此普遍，以至于它有一个名称。它叫公共汽车。

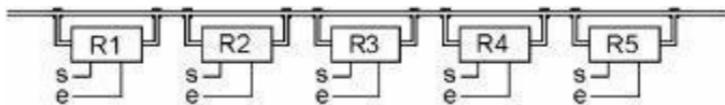
为什么叫公共汽车？

嗯，这可能与旧的电气术语“总线”有关，“总线”指的是在发电厂等地方用作一根非常大的电线的一块金属。但也有一个有趣的相似之处，那就是人们使用的公共汽车的类型。

公共汽车是一种通常沿着预定路线行驶的交通工具，在人们上车或下车的地方会停很多站。他们从某处出发，公共汽车把他们送到他们需要去的其他地方。

在计算机世界中，总线仅仅是一组八条线，它们通向计算机内部的不同位置。

当然，8是承载一个字节信息所需的线数。在计算机内部，字节的内容需要从因此总线连接到所有这些位置，寄存器的设计允许任何选定字节的内容进入总线，并在选定的目的地下车。



在下面的示例中，我们有一个总线，有五个寄存器，每个寄存器的输入和输出都连接到同一总线上。

如果所有的's'位和'e'位都关闭，则每个寄存器将按原样设置，并保持原样。

如果您要将信息从R1复制到R4，请首先打开R1的'e'位。

R1中的数据现在位于总线上，可从所有五个寄存器的输入端获得。

然后，如果短暂地打开和关闭R4的's'位，总线上的数据将被捕获到R4中。

字节已复制。所以电脑公交车有点像运载人的公交车。有一个n

许多站点和字节可以到达它们需要到达的位置。

请注意，我们可以将任何字节复制到任何其他字节中。

您可以将R2复制到R5，或将R4复制到R1。这辆公共汽车朝两个方向开。

使能任何寄存器时，总线上的电会尽可能快地流到总线上其他所有器件的输入端。

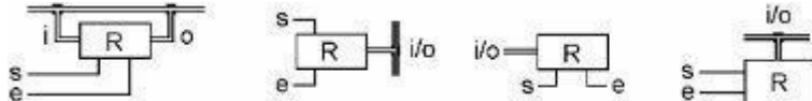
您甚至可以在总线上启用一个寄存器，并同时将其设置为两个或多个其他寄存器。

您不想做的一件事是同时使能总线上两个寄存器的输出。

关于寄存器还有一件事：有许多地方可以将寄存器的输入和输出连接到同一总线，为了进一步简化，我们只需显示一束标有“I/O”的电线，即输入和输出。

就它们的工作方式而言，以下所有内容都是完全等价的。

可以调整图上导线的位置，使其尽可能整洁。



另一种语言注意:字节是一个位置，可以在256个状态之一。

有时我们讨论从这里到那里移动一个字节。根据定义，字节不会在计算机内部移动。字节仅指位置，但有时当有人想要引用字节的当前设置时，他们应该说“Let's copy the contents of R1 into R4”，他们会简化为“move r1 to R4”或“move this byte over there”。他们使用单词byte来指代

字节。同样，上下文通常非常清楚地说明了这一点。

在上面将R1的内容复制到R4的示例中，您可能会听到它被描述为“将字节从R1移动到R4”。从技术上讲，R1和R4是字节，它们不移动，只是内容从一个位置移动到另一个位置。

而且，内容不会离开它们的来源地。

当您完成“移动”一个字节时，“发件人”字节没有更改，它不会丢失它所拥有的内容。

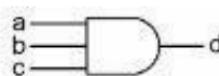
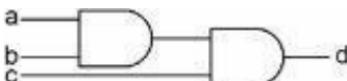
在另一端，最初在“to”字节中的模式现在“gone”了，它没有去任何地方，它只是被新的信息所改写。旧模式就不复存在了。

新信息与仍在第一个字节中的信息完全相同。

字节没有移动，在两个位置仍然有两个字节，但是

第一个字节中的信息已复制到第二个字节中。

更多门组合



现在我们将展示另外两个组合，然后我们将能够把到目前为止我们所知道的组合起来，制造出计算机的前半部分。所以不要灰心，再往前走一点，我们就回家了。

第一个 组合非 常简单 。 它只是 一个AN D门， 有两 个以 上 的 输 入。 如 果您 连 接 两 个 AN D 门， 如 图左 侧 所 示， 您会看 到，要 使'd' 为 ON，所 有三 个 输 入'a' 'b' 和' c'都必 须为ON 。 因 此， 这 个 组 合 可 以	显示其 操作方 式的图 表如下 所示：	a	b
--	---------------------------------	---	---

简单地 绘制为 右图:			
c	d	0	0
0	0	0	0
1	0	0	1
0	0	0	1
1	0	1	0
0	0	1	0
1	0	1	1
0	0	1	1

1

1

设想用另一个AND门替换输入'c'，那么您将有一个四个输入AND门。

然后，您可以用另一个AND门替换四个输入中的任何一个，并使用五个输入AND门。

对于您所做的事情，可以根据需要多次执行此操作。

当您添加输入时，图表将需要越来越多的线条。

每次添加另一个输入时，可以将输入的组合数量增加一倍。

我们看到的原始两个输入与门的图表有四条线，每条线对应一种可能性。

正上方的三个输入端有八条线。

四个输入AND门将有16条线，五个输入将有32条线，等等。但是，在所有情况下，对于AND门，只有一个组合将导致

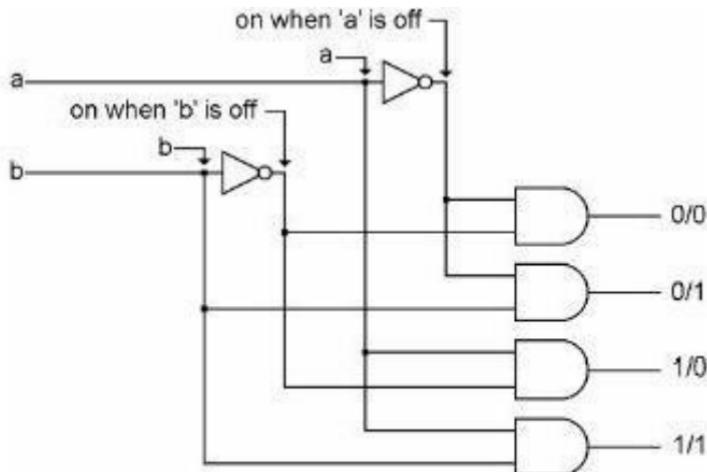
输出接通，即所有输入接通的线路。

这是我们制作计算机前半部分所需的最后一个组合。

这一组合不同于我们迄今所看到的任何情况，因为它的输出多于输入。

我们的第一个示例有两个输入和四个输出。

它不是很复杂，只有两个“非”门和四个“与”门。



在下图中，'a'和'b'是从左侧输入的。两者都连接到NOT门。“非”门产生与其输入相反的输入。  
页面上有四条垂直的线，分别来自'a'和'b'以及'a'和'b'的对立面。因此，对于每个'a'和'b'，页面上都有两条线，其中一条线在其输入接通时接通，另一条线在其输入断开时接通。

现在我们在右边放置四个AND门，并连接每个AND门

一对不同的垂直布线，使得每个AND门将导通四种可能组合'a'和'b'中的不同一种。

标有'0/0'的顶部AND门连接到当'a'断开时导通的布线和当'b'断开时导通的布线，从而当'a'和'b'都为0时导通。

下一个AND门“0/1”连接到当'a'为OFF和'b'时接通的导线，因此当'a'为0、'b'为1时接通，以此类推。

输入可以任意组合开启，两位关闭、一位开启、另一位开启或两位均开启。  
没有，一两个开着。然而，输出将始终有一个且仅有一个输出接通，另三个输出关闭。  
接通的状态由'a'和'b'的当前状态决定。

a

b

1/1

0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

这种组合称为解码器。

名称意味着，如果将两个输入的四种可能状态视为一个代码，则输出将告诉您当前输入的是哪种代码。

也许这不是一个很好的名字，但这是它曾经对某人意味着什么，而且这个名字一直存在。

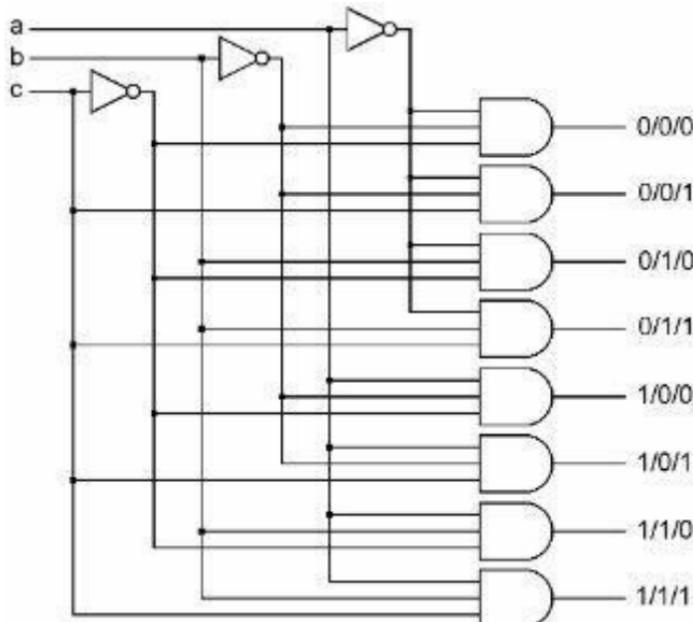
该解码器有两个输入，这意味着可以有四个输入状态的组合，并且有四个输出，一个对应于每个可能的输入组合。

这是可以延长的。

如果我们添加第三个输入，那么将有八种可能的输入组合，如果我们使用八个、三个输入与门，我们可以构建一个三输入、八个输出解码器。

类似地，我们可以构建一个四输入16输出解码器。

解码器以输入数“x”（输出数）命名。如 $2 \times 4$ 、 $3 \times 8$ 、 $4 \times 16$ 、 $5 \times 32$ 、 $6 \times 64$ 等

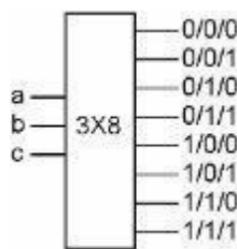


同样，我们将简化我们的图纸，我们不会显示任何内部零件或布线，我们只会有一个盒子的名称，以及我们感兴趣的输入和输出。我们

我见过“与非”门不是“与与”门，“与非”门不是“与与”门“的解码器。

这是一个装满NAND门的盒子，用来做一些有用的事情。

我们知道它的工作原理，其中一个且仅有一个输出始终接通，而哪个输出始终接通取决于三个输入的状态。这就是它所做的一切。



## 计算机的前半部分

让我们用现有的零件做些什么。实际上，我们现在可以构建计算机中一半的内容。

首先，让我们用木头做一些类似的东西（在我们的脑海中），然后我们将回来展示如何构建一个几乎做同样事情的计算机版本。

你知道，在旅馆里，在前台，在店员后面的墙上，有一系列的小木箱孔，每个房间一个。那是他们为客人保留额外房间钥匙和留言或邮件的地方。

或者你可能看过一部老电影，里面一个老邮局的人正在整理邮件。

他坐在一张桌子旁边，桌子后面有一系列的小房间。

他在桌上放了一堆未分类的邮件，一次拿一封，读地址，然后把信放进信封里迟到的小房间。

所以我们要建几个小房间。

我们的是三英寸见方，有十六个立方体洞高，十六个立方体洞宽。

总共四英尺乘四英尺，总共256个小房间。

现在我们来添加一些他们在邮局或酒店里没有的东西。

我们将把一个大木板放在小木箱前面，它是整个木箱宽度的两倍，中间有一个垂直的槽，这个槽刚好足够露出一列16个小木箱。

面板底部有轮子，可以左右滑动，一次露出16个立方体的任何一个立柱，覆盖所有其他立柱。

让我们拿另一块木板，就像第一块木板一样，但是把它侧翻，这样它就比我们的小木箱高一倍，中间的槽是并排的。

第二个面板将被安装在第一个面板的正前面，就像一个窗框，所以它可以上下滑动，一次只暴露一排16个立方体。

现在我们有一系列256个立方体孔，在它们前面有两个开槽的木板，一次只能看到一个立方体。

在每一个立方体中，我们将放置一张纸条，在上面写上八个0和1的可能组合中的一个。

现在，我们将采取的大门，寄存器和解码器，我们已经描述过，并作出一些东西，他们做的几乎一样的事情，我们的立方体设备。

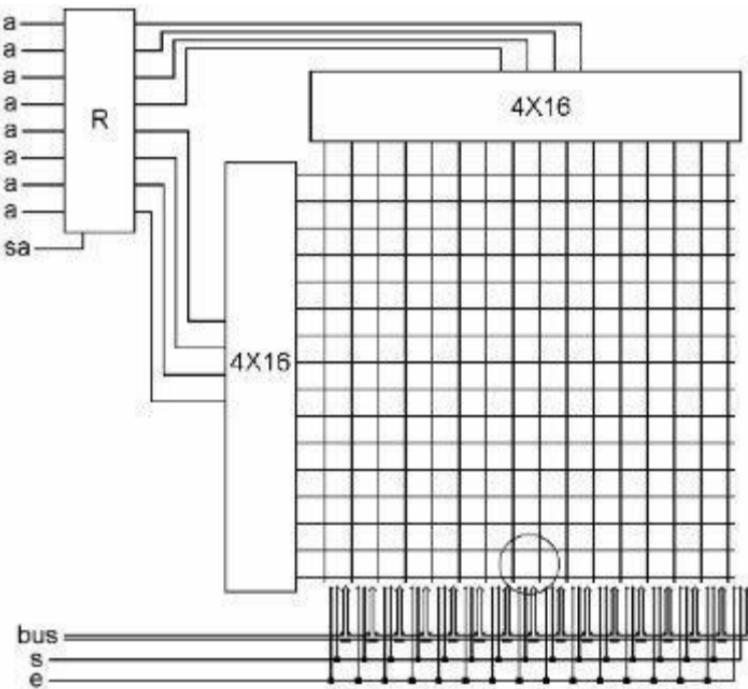
这个东西将有256个地方存放东西，我们将能够选择一个和

任何时候都只有一个这样的地方。

参考下图，我们从单个寄存器开始。它的输入'a'是来自计算机中其他地方的总线。总线上放置一组位，“sa”（设置A）位从1变为0。

该位模式现在存储在该寄存器中，该寄存器是输出始终开启的寄存器之一。

前四个输出位连接到一个 $4 \times 16$ 解码器，其他四个输出位连接到另一个 $4 \times 16$ 解码器。这两个解码器的输出被放置



呈网格状。

这些电线彼此不接触，但这里有16乘16的交叉路口，或者说是256个交叉路口，我们很快就会用到。

如所述，解码器在任何时候都有且仅有一个输出接通，其余输出关闭。

因为这里有两个解码器，所以有一个水平网格线和一个垂直网格线。

因此，在这256个交叉点中，只有一个交叉点的水平和垂直导线都接通。

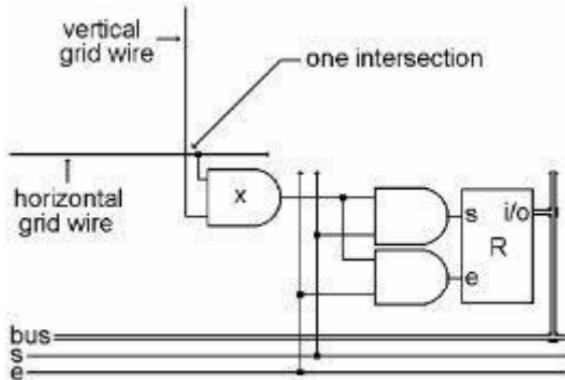
哪个交叉点是WIL

L每次更改R中的值时都会更改，但总会有一条线路同时接通，而另一条255只有一条接通或没有接通。

该图底部是一条总线和一个's'和'e'位，与连接到寄存器的连接相同。

正如你所看到的，它们向上进入网格。

图中没有显示，但是它们在网格下一直向上直到顶端，所以每个



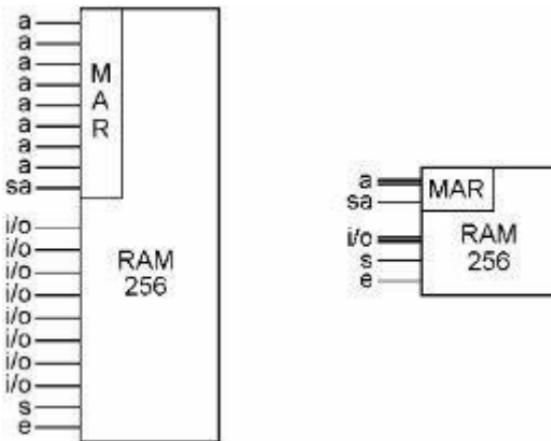
256个十字路口附近有一辆公共汽车和一个's'和'e'位。

上图中有一个圆，围绕着网格的一个交点。

下图放大了这个圆中的内容，显示在256个交叉点的每个交叉点上有三个AND门和一个寄存器。

正如我们所看到的，在这个交叉点有一个与门"x"连接到一个垂直网格线和一个水平网格线。这些“X”门是唯一连接到电网的东西。其余的连接到总线，“s'和'e'位在T

图表的底部。请记住，只有一个交叉路口，两根网格线都是接通的。因此，这些“x”门中有256个，但在任何给定时间，只有一个门的输出开启。该“x”门的输出分别位于两个“与”门的一侧。这两个门控制对该交点处寄存器的设置和使能输入的访问。因此，当“x”门关闭时，无法打开该寄存器的“s”和“e”位。其中255个国家将是这种情况



寄存器，“x”门关闭的寄存器。

但有一个交叉点的“x”门打开，寄存器可以从总线设置，或者其内容可以使能到总线上，然后使用图底部的“s”和“e”位发送到别处。

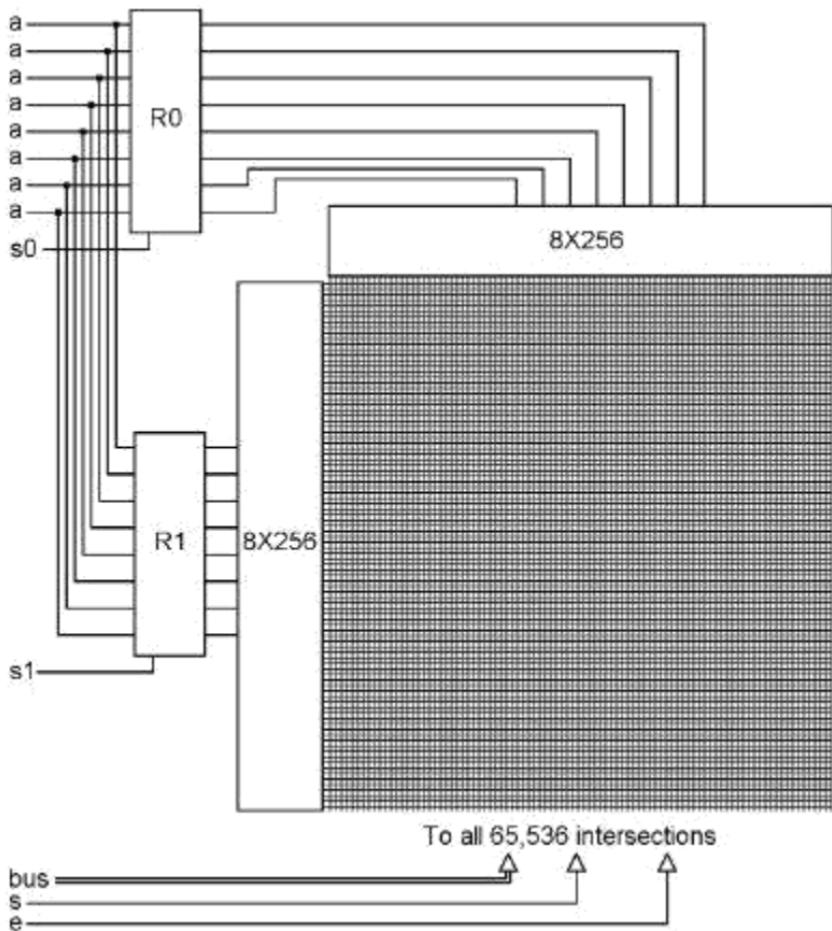
上面是计算机的主存储器。这是建造一台计算机所必需的一半。

它有时被不同的名称调用，但最正确的名称来自这样一个事实，即您可以在一分钟内选择256个字节中的任何一个，然后您可以立即选择256个字节中的任何其他字节，而无论最后一个字节在哪里，或者下一个字节在哪里，选择字节的顺序都没有速度上的优势或劣势。由于这种质量，这是一个

如果您希望能够以随机顺序访问内存的字节，那么可以使用好的内存类型。

所以这种类型的存储器被称为“随机存取存储器”，简称为“RAM”。

这里有一个想法是什么样子的：（不要费心计算网格线，它只可能适合大约一半的打印页。）

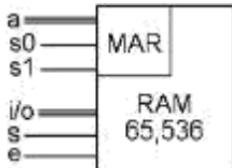


总线一次承载一个字节，因此从该RAM的65,536个存储位置中选择一个将是一个两步过程。

首先，必须在'a'总线上放置一个字节并将其设置为R0，然后必须在'a'总线上放置第二个字节并将其设置为R1。

现在，您可以使用总线和绘图底部的's'和'e'位访问所需的内存位置。

再一次简化，我们有一些看起来非常像我们的256字节RAM，它只是有一个更多的输入位。



对于本书的其余部分，我们将使用256字节RAM，只是为了让事情变得简单。  
如果您想要想象一台具有更大RAM的计算机，那么每次我们向Memory地址寄存器发送一个字节时，您所要做的就是想象发送两个字节。

## 数字

我们将回到代码的主题上片刻。

在此之前，我们研究了一个名为ASCII的代码，它用于表示书面语言。

数字也用在书面语言中，所以数字0到9都有ASCII码。

早些时候，我们看到了部分字母表的20个ASCII代码，这里还有10个，是书面语言中的数字代码：

0	0011 0000
1	0011 0001
2	0011 0010
3。	0011 0011
4。	0011 0100
5	0011 0101
6。	0011 0110
7	0011 0111
8	0011 1000
9	0011 1001

这是一个非常有用的代码，但并不是计算机所做的一切都与书面语言有关。

对于其他任务，还有更适合这些任务的其他代码。

说到数字，如果使用ASCII，一个字节可以是0到9之间的10个数字中的任意一个。

但有时有一个字节总是用来存储一个数字，这个数字将永远不会打印或显示在屏幕上。

在这种情况下，我们可以使用一个不同的代码，它不会在lett上浪费任何可能的状态。

字母表的ER或数字以外的任何东西。

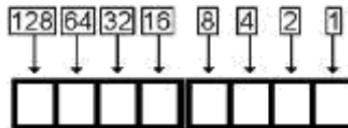
因为一个字节有256种可能的状态，所以您可以让此代码表示256个不同的数字。

因为我们希望包含零，所以这段代码从零开始，一直到255。

由于这是一个很长的章节，下面是这段代码的预览。

它包括为字节中的每个位分配一个数值。

要使用此代码，只需打开与要表示的数字相加的位即可。



要了解此代码如何工作，为什么在计算机中使用，以及如何选择这些位值，我们将研究计算机之外的数字主题。

有三个数字系统，你可能很熟悉，我们可以分析。

在我看来，这三个系统都由两个思想或元素组成——

第一，符号列表，第二，使用这些符号的方法。

也许最古老的数字系统是一个简单的东西叫做理货方舟。

它有两个符号，“”和“/”。使用这些符号的方法是，在计算的前四个事物上分别写下一个“”，然后在第五个标记的前四个事物上写下一个“/”。

只要有必要，你就会一遍又一遍地重复这个步骤，然后当你做完之后，你就会按5-5，10，15，20等分组来计算分数。这个系统非常适合于计算路过的东西，比如说你的

一群羊。当每只动物经过时，你只需再划掉一个记号——你不必划掉“6”而写上“7”。

这个系统还有另一个优点，那就是每点一件东西实际上都有一个标记。

在本章的后面，我们将用可能会混淆的数字来做一些有趣的事情，所以为了保持清晰，我们将使用这个旧系统。

你还记得罗马数字吗？它是一个数字系统，也包括两个元素。

第一个元素是符号，只是从字母表中选择字母，“I”代表1，“V”代表5，“X”代表10，“L”代表50，“C”代表100，“D”代表500，“M”代表1000。

第二个元素是一个方法，它允许您表示没有单个符号的数字。

罗马法说，你写下多个符号，首先是最大的符号，然后把它们加起来，除非较小的符号在较大符号的左边，然后减去它。

所以“ii”是二（加一加一），“iv”是四（从五减去一）。

使这位作者对2000年的到来感到非常高兴的一件事是罗马数字代表的年份变得简单了许多。1999年是

'm cm xcix'，你只需在你的头脑中做三个减法就可以读到那一个。2000年只是“M M”。我们今天使用的普通数字系统也包括两个概念，但这是两个非常不同的概念，是通过阿拉伯而不是罗马传入我们的。

这些想法中的第一个也是关于符号的，在本例中是0、1、2、3、4、5、6、7、8和9。这些数字是表示数量的符号。

第二个想法是一种我们非常习惯的方法，我们本能地使用它。

这种方法说，如果你写下一个数字，它就意味着它所说的。

如果在每个数字旁边写下两位数字

另一个，右边的意思是它所说的，而左边的意思是它所说的十倍。

如果你在旁边写下三位数字

彼此，右边的意思是它所说的，中间的意思是它所说的十倍，左边的意思是它所说的一百倍。

当你想表达一个大于9的数字时，你可以用多个数字来表示，你可以用这个方法来表示，第一个数字左边的位置数告诉你，在你把它们加起来之前，你把它乘了多少次。所以，如果你有246个苹果，那就意味着你有200个苹果加上40个应用程序

莱斯加六个苹果。

那么这是怎么运作的呢？

任何数量的数字都可以用数字0到9来写，但是当你超过9的时候，你必须用两位数。

当你超过九十九岁时，你必须使用三位数。

在999以上，你会转到4位数，等等。如果你往上数，任何一个位置的数字都是“round”和“round-

zero”到9，然后再回到0到9，一遍又一遍地，每当你从9回到0，你就会把数字增加到左边

t乘以1。

因此，您只有十个符号，但您可以根据需要使用其中一个以上的符号，并且它们相对于彼此的位置指定它们的完整值。

这有点奇怪，因为系统是基于十的，但是没有一个符号代表十。

另一方面，这是正确的——符号“0”到“9”确实构成了十个不同的符号。

如果我们也有一个10的符号，那么实际上会有11个不同的符号。

所以想到这件事的人都很聪明。

在这个阿拉伯体系中，一个新的想法是有一个零的符号。

如果你想说你没有苹果，这是有用的，但这也是一个必要的事情，保持数字的位置直。

如果你有50个苹果或107个苹果，你需要数字中的零来知道每个数字的实际位置，所以你可以乘以10的正确次数。

阿拉伯数字系统中的这两种思想（数字和方法）有一个共同点。他们都有数字十。

有十个不同的数字，当你把数字加到一个数字的左边，每个位置的价值是前一个的十倍。

59. 在学校里，当他们第一次教孩子们关于数字的时候，他们会说我们的数字系统是以十为基础的，因为我们有十个手指。

所以这里有一个奇怪的问题：如果这个数字系统是由三趾树懒发明的呢？

他们每只手上只有三根手指，没有拇指。他们会发明一个只有六位数字的数字系统-0, 1, 2, 3, 4和5。这个能用吗？如果你有八个苹果，你会怎么写？

此系统中没有数字“8”。

0-?	0-5	0-9	0-5	0-9
1	2	3	4	5

答案是，由于第一个想法，即数字，被改为只有六位，那么第二个想法，即方法，也必须改变，这样当你向左添加位置时，每个数字都必须乘以六，而不是十。

那这个系统就能运作了。

当你数苹果的时候，你会说“0, 1, 2, 3, 4, 5.....”然后呢？下一个没有“6”

你可以为任意数量的数字发明一个数字系统，比如我们上面看到的10或6，或者3或14，或者你选择的任何数字。但最简单的一个是如果你只有2位数字，0和1。

这一个怎么用？好吧，你会数0, 1...然后你已经没有数字了-

所以回到0，在左边加上1，使下一个数字10然后11，然后你是

又是数字不足，所以100再101，110，111再1000。

这个系统是以两位数为基础的，所以只有两位数，当你把位置加到左边时，每一个都比前一个值高两倍。

正确的位置表示它所说的内容，左边的下一个表示它所说的内容的两倍，下一个表示它所说的内容的四倍，下一个表示它所说的内容的八倍，等等。当你只剩下两个可能的数字时，你不需要做太多的乘法就可以算出总价值

职位的UE。例如，在值为“8”的位置，只能有1，意思是“8”，或者0，意思是“没有8”。

当我们在做的时候，让我们想象一个非常奇怪的动物，每只手上都有八根手指。

那只动物就会发明基于16的数字。

在他们的系统中，他们可以用一个符号写出10到15个字符。

只有当他们16岁到达时，他们才会回到0左右，并且需要在左边的位置放一个1。

要了解它的工作原理，我们需要六个新符号，所以让我们使用字母表的前六个字母。`'a'`将意味着10，`'b'`将意味着Elev

Tally	0-9	0-5	0-1	0-F
I	0	0	0	0
II	1	1	1	1
III	2	2	10	2
IV	3	3	11	3
V	4	4	100	4
VI	5	5	101	5
VII	6	10	110	6
VIII	7	11	111	7
VIII	8	12	1000	8
X	9	13	1001	9
XI	10	14	1010	A
XII	11	15	1011	B
XIII	12	20	1100	C
XIV	13	21	1101	D
XV	14	22	1110	E
XVI	15	23	1111	F
XVII	16	24	10000	10
XVIII	17	25	10001	11
XIX	18	30	10010	12

嗯，“C”的意思是12，“D”的意思是13，“E”的意思是14，“F”的意思是15。

只有在正确的位置使用完所有16个符号后，符号才会用完，下一个数字将是16个，在这个系统中写入“10”。如果你熟悉磅和盎司的重量系统，它有点像这个系统。

一磅有16盎司，所以你知道8盎司等于半磅。加上9盎司和9盎司等于1磅2盎司。

这是一张图表，显示了五种不同的数字系统。第一栏是旧的记分制，以保持其合理。

我们通常把0-9叫做十进制，因为在一些古代语言中，“dec”的意思是10。0-5系统将被称为参议院系统，因为“sen”在其他一些古代语言中的意思是6。

这个只有0和1的新系统被称为二进制，因为“bi”的意思是2，也因为一些古老的语言。

另一个新的系统，0-

F系统，将被称为十六进制系统，因为'hex'是另一个古老的单词，意思是6，'dec'仍然是10，所以它是

六加十制度。

命名不同数字系统的另一种方法是根据它们所基于的数字来调用它们，例如“base 10”或“base

2”等，意思是十进制或二进制等，但请注意，单词“base”后面的数字写在十进制系统中。用二进制写的“2”是“10”，因此如果“10”是用二进制写的，“base 10”就意味着二进制。

事实上，如果'10'写在每个数字系统的数字中，那么每个数字系统都将是'base10'！所以我们可以讨论基数2，基数6，基数10和低音

E16如果我们想要的话，只要我们记住这些基数是用十进制写的。

如果我们谈论二进制，高位，十进制和十六进制，这是一回事，只是可能少一点混乱。同样，在我们通常的十进制数中，最右边的位置是1的个数。

左边的下一个位置是十的个数，等等。每个位置的价值是前一个位置的十倍。

在二进制中，最右边的位置也是1的个数，但第二个是1的个数

实际上，在计算机工业中，人们经常使用十六进制（他们称之为“十六进制”）。

如果查看上面的图表，可以看到四位二进制数可以用一位十六进制数表示。

如果您有一个包含00111100的字节，您可以将其转换为60位十进制，或者直接将其称为“3C十六进制”。现在不要担心，我们不会在本书中使用十六进制，但您可能在某些地方看到过这些类型的数字，现在您知道这些数字是怎么回事了。

## 地址

既然我们有了二进制数代码，我们就可以在计算机中把它用于各种目的了。

我们首先使用它的地方之一是M Emory地址寄存器。

我们放入该寄存器的位模式将使用二进制数代码。

然后选择256个RAM存储位置中的一个。 m

ar中的数字被认为是介于0和255之间的某个数字，因此256个RAM字节中的每一个都可以被认为具有地址。

这很简单，但这里需要指出的一点是，计算机内部地址的确切含义是什么。

在住宅区，每家都有一个地址，就像125米的阿普尔街。

拐角处有一个牌子，上面写着“Maple

St。”，房子上写着数字“125”，这是我们通常对地址的看法。

这里要指出的是，房屋和街道上写着数字或名字。在计算机中，字节没有任何标识

g有关资料或其中所载资料。当您将该数字放在M

Emory地址寄存器中时，选择的只是字节。

选择字节的依据是它所在的位置，而不是该位置包含的任何其他因素。

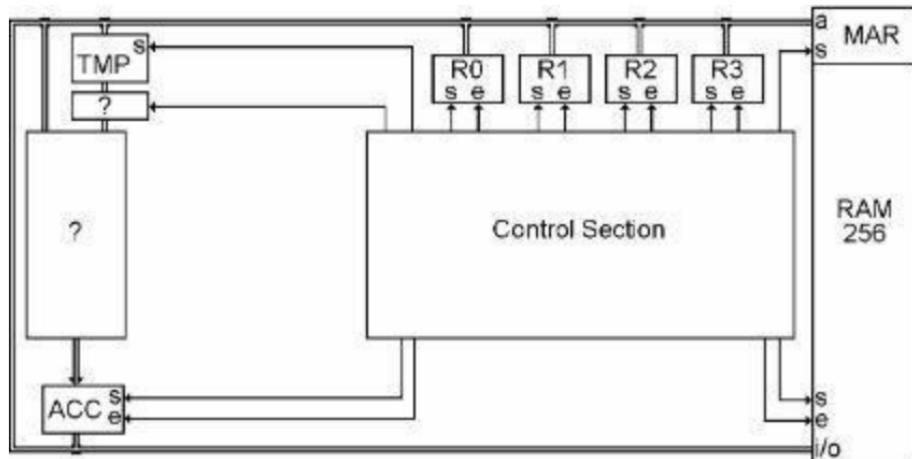
想象一下，一个有16条街道，每条街道有16栋房子的街区。

想象一下，街道上没有标志，房屋上也没有数字。

如果你是的话，你仍然可以找到任何特定的房子

比如，我告诉他去“第七街的第四栋房子”，那栋房子还有地址，也就是一种定位的方法，只是没有任何识别信息。

因此，计算机地址只是一个数字，当该地址被放入内存地址寄存器时，它会导致选择某个字节。



## 电脑的另一半

计算机的另一半最终也是由NAND门构成的，它可能比我们构建的RAM的总部分要少，但它的布局不是那么有规律和重复，所以需要更长的时间来解释。

我们将把这一半的计算机称为“中央处理器”，简称CPU，因为它处理RAM中的字节。它“处理”它们，我们将在接下来的几章中看到这意味着什么。博的共同点是计算机的两边是公共汽车。

以下是CPU的开始。

RAM显示在右侧，总线在RAM上的两个总线连接之间形成一个大环路。

CPU从连接到总线的六个寄存器开始。这六个寄存器都是CPU用来“处理”字节的位置。没那么复杂吧？

图中央标有“Control Section”的大方框将在后面详细检查。

它控制CPU和RAM中的所有“set”和“enable”位。 带问号的方框将在本章后面立即解释。  
现在，我们将研究字节在CPU中的位置。

R0、R1、R2和R3寄存器用作CPU所需字节的短期存储。

它们的输入和输出连接到总线。

它们可以用于许多不同的目的，因此被称为“通用寄存器”。

名为“TM

P”的寄存器表示临时寄存器。

它的输入来自总线，输出向下到一个问题标记框，然后是另一个问题标记框。 TM

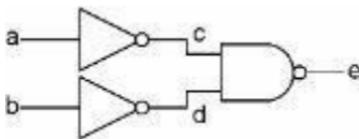
P有一个“set”位，但没有“enable”位，因为我们永远没有理由关闭它的输出。

现在，我们将用问号查看这些框中的内容。

## 更多门

到目前为止，我们使用的是NAND，而不是盖茨。

我们还需要定义另外两个组合门。第一个是这样构建的：



它所做的就是不把两个输入输入到我们的一个好的与非门。

这是它的图表，显示了中间的电线，所以很容易理解。

a	b	c	d	E
0	0	1	1	0
0	1	1	0	1
1	0	0	1	1
1	1	0	0	1

在这种情况下，当两个输入均为OFF时，输出为OFF，但如果'a'或'b'均为ON，或两者均为ON，则输出将为ON。

所以它有另一个非常简单的名字，叫做“或门”，它没有画出所有的零件，而是有自己的图，形状像一个盾牌。图表如下所示：



a	b	c
0	0	0
0	1	1
1	0	1
1	1	1

与AND门一样，可以使用两个以上的输入构建OR门。

只需添加另一个“或”门来代替其中一个输入，就会有三个输入，其中任何一个输入都会打开输出。与AND门一样，每次添加输入时，图表上的行数都会翻一番。

使用或门时，只有所有输入关闭的线路才会关闭输出。

其余所有行都将显示输出为ON。

我们在这里需要的最后一个组合门需要五个门来制造，但它最终要做的非常简单。  
与“或”门类似，当任一输入为ON时，输出为ON，但在此

版本时，如果两个输入均为ON，则输出将返回OFF。

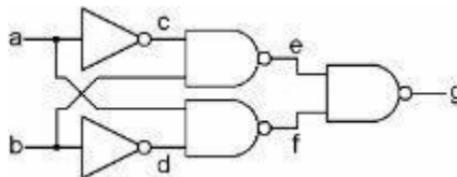
因此它被称为异或门，简称异或门。

如果任一输入或另一输入以独占方式接通，则输出接通。

只有当它是或者，不是如果它是和。

另一种看待它的方式是，如果一个并且只有一个输入为ON，则输出为ON。

从另一个角度来看，如果输入相同，则输出为OFF，如果输入不同，则输出为ON。



a	b	c	d	E	F	G
0	0	1	1	1	1	0
0	1	1	0	0	1	1
1	0	0	1	1	0	1
1	1	0	0	1	1	0

简化图看起来类似于“或”门，但在输入端有一条双曲线。图表如下所示：



a	b	c
0	0	0
0	1	1
1	0	1
1	1	0

我们现在有四种门，它们接受两个输入，产生一个输出。它们是NAND, AND, OR和XOR。下面的图表非常简单：

a	b	NAND	AND	OR	XOR
0	0	1	0	0	0
0	1	1	0	1	1
1	0	1	0	1	1
1	1	0	1	1	0

对于“a”和“b”这四种可能的输入组合，每种类型的门都有自己的输出状态集，并且门的名称可以帮助您记住哪个是哪个。

尽管计算机中的所有内容都是由NAND门构成的，但我们不会在此计算机中单独使用任何NAND门！

现在，我们已经使用它们来构建AND、OR、XOR和NOT门，以及内存位，我们已经完成了NAND门。谢谢你M R。南德门，再见了。

## 扰乱字节

单个门对位操作。两位输入，一位输出。但RAM一次存储和检索一个字节。总线一次移动一个字节。在CPU中，我们希望能够一次处理一个字节。我们需要一些影响整个字节的“门”。

在关于总线的一章中，我们了解了如何将字节的内容从一个寄存器复制到另一个寄存器。这通常称为移动字节。现在我们将看到这方面的一些变化。

首先，我们将看到在字节从一个寄存器移动到另一个寄存器时，有三种方法可以更改字节的内容。

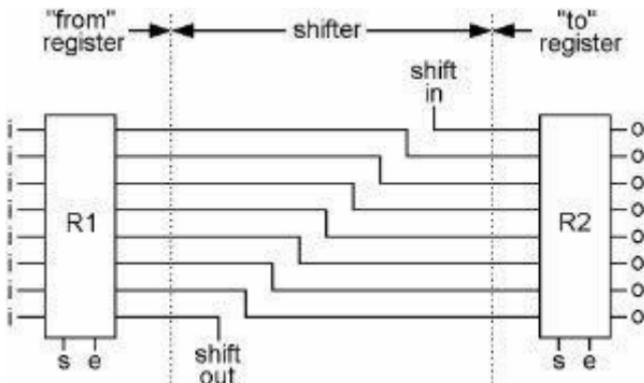
其次，我们将看到四种方法，可以获取两个字节的内容，并使它们相互交互以创建第三个字节的内容。这些都是计算机对字节实际执行的操作。

一切最终归结为这七项行动。

## 左移和右移

变速杆很容易制造。它根本不用任何门，只是把公共汽车的电线接得有点奇怪。

它是这样做的：



这显示两个寄存器通过右移位器连接。移位器只是两个寄存器之间的连线。

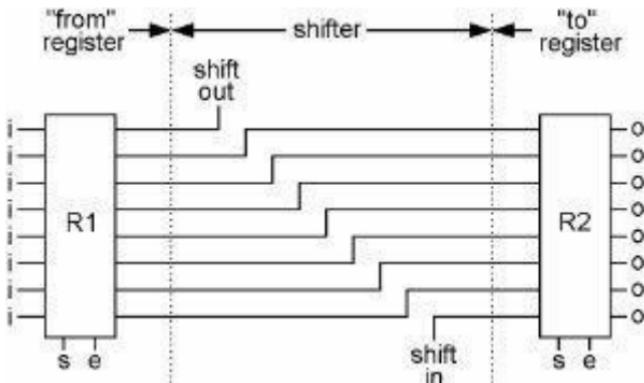
当R1的'E'位被接通，R2的'S'位被接通然后关断时，R1中的所有位都被复制到R2中，但它们被移位到一个位置。

位于底部的位（移出）可以连接到计算机中的其他位，但通常连接回位于顶部的位（移入），并且完成该操作后，最右边的位环绕到最左边的位

在字节的另一端。

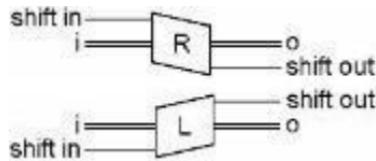
右移器将从0100 0010更改为0010 0001。

如果“移出”连接到“移入”，则右移将从00011001变为10001100



左移器将从0100 0010更改为1000 0100。左换档杆的接线方式如下：

同样，我们有这些图纸的总线版本。他们各有一辆“i”和“o”公共汽车，



还有一个输入和输出位，如下所示：

所以那是移位器，根本没有门。

## 记分员

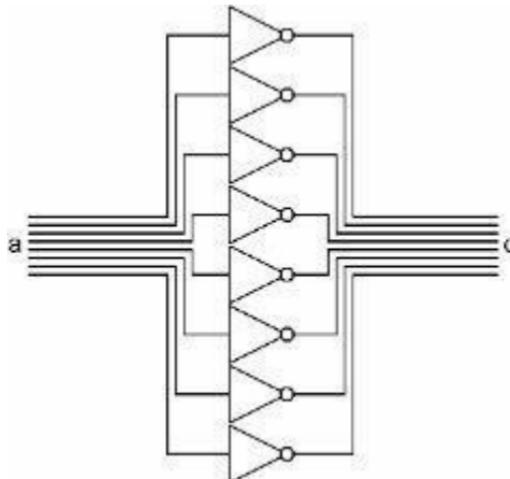
该器件将两个寄存器与八个NOT门相连。每个位都将更改为相反的位。

如果你以0110 1000开始，你将以1001 0111结束。

这个操作有很多用途，第一个是在一些算术函数中。

在我们描述了其他一些事情之后，我们很快就会看到它是如何工作的。

“非”门的另一个名称是“反相器”，因为它与你给它的相反，把它倒过来，或者把它反相。



由于输入和输出都是8条线，我们将使用总线类型图片来简化。



## 安德尔

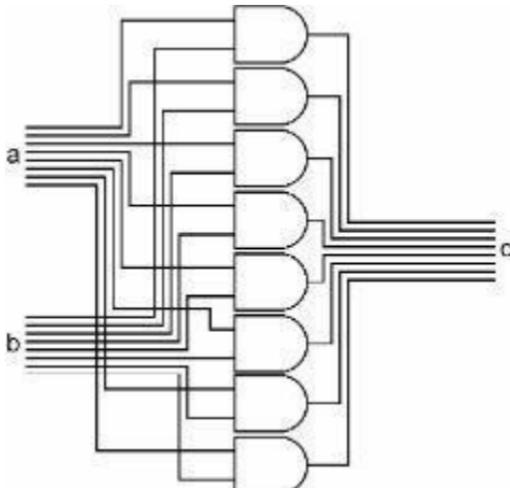
ANDER接受两个输入字节，并将这两个字节中的每一位与为第三个字节。

如您所见，'a'输入总线的8位连接到8个与门的上侧。

“b”输入总线的8位连接到8个与门的下侧。

8个AND门的输出构成此程序集的总线输出'c'。

这样，我们可以和两个字节一起形成第三个字节。



这有很多用途。例如，您可以确保ASCII字母代码是大写的。如果在R0, 0110  
0101中有字母“e”的代码，则可以将位模式1101

1111放入R1，然后再放入R1和R0，并将答案放回R0。

除第三位外，R0中所有开启的位都将复制回R0。

不管第三位以前是开还是关，现在都是关。r0现在将包含0100

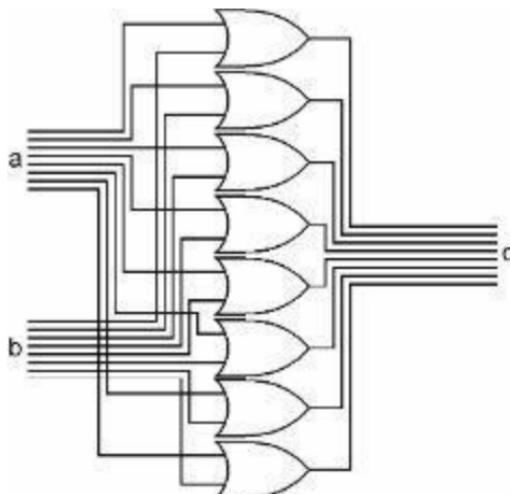
0101，“E”的ASCII代码。这适用于所有ASCII L

因为ASCII的设计方式。



这里是一个更简单的总线类型的图片为ANDER。

奥雷尔



ORER接收两个输入字节，并将这两个字节中的每一个位与第三个字节进行OR运算。  
如您所见，'a'输入总线的8位连接到8个OR门的上侧。

“b”输入总线的8位连接到相同8个“或”门的下侧。

八个或门的输出为该程序集的总线输出'c'。

这样，我们可以或两个字节一起形成第三个字节。

下面是ORER的一个更简单的总线类型图片。



## 独家Orer

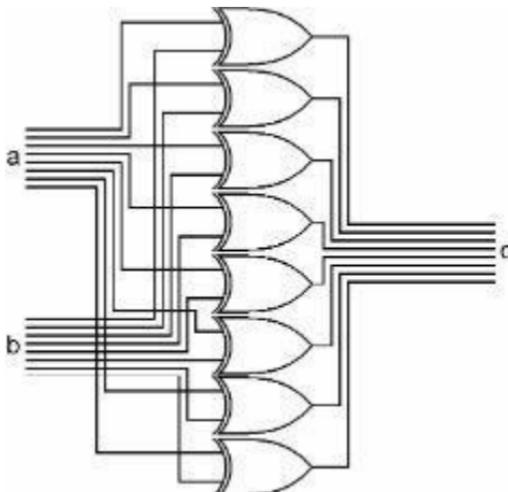
XORER接受两个输入字节，并将这两个字节中的每个位异或为第三个字节。

如您所见，'a'输入总线的8位连接到8个XOR门的上侧。

“b”输入总线的8位连接到8个XOR门的下侧。

八个异或门的输出为该程序集的总线输出'c'。

这样，我们可以将两个字节异或在一起形成第三个字节。



这有什么用？同样，富有想象力的人提出了许多用途。但这是其中之一。

假设您在R1中有一个代码，在R2中有另一个代码。

你想知道这两个密码是不是一样的。将R1和R2异或到R1中。

如果R1和R2包含相同的模式，则R1现在将全部为零。

在R1中，0和1的模式并不重要，如果R2包含相同的模式，则在XOR之后，R1将全部为零。

由于输入和输出都是8条线，因此我们将使用总线类型图片进行简化。



## 加法器

考虑到它的功能，这是一个出人意料的简单组合。

在我们的二进制数字系统中，数字在0到255的范围内以字节表示。

如果您考虑如何对两个常规十进制数进行加法运算，则首先将两个数字相加在右列中。

因为这两个数字可以分别是0到9之间的任意值，所以这两个数字的和将是0到18之间的任意值。如果答案在0到9之间，你就把它写在两个NU下面

2 梅伯斯。

如果答案是从10到18，则写下右数字，并将1加到左边的下一列。

$$\begin{array}{r} 5 \\ 6 \\ + 4 + 7 \end{array}$$

12

在  
二  
进  
制  
数  
系  
统  
中  
,它  
实  
际  
上  
要  
简  
单  
得  
多  
。  
如  
果  
在  
二  
进  
制  
中

执行相同类型的加法，则右列中的两个数字只能分别为 0 或 1。因此，将两个二进制数的

右列相加的唯一可能的答案将是 0、1 或 10（零、一或两）。如果您添加  $0+0$ ，您将得到 0

、  
1+  
0  
或  
0+  
1

，您将得到

1

、  
1+  
1

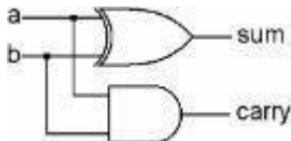
，您将得到右侧列中的  
0

，并将

1 带到左侧列中

。

$$\frac{1}{1} \quad \frac{+0}{+1}$$



我们所描述的大门很容易做到这一点。

两位的**XOR**将告诉我们正确的列答案应该是什么，而两位的**AND**将告诉我们是否需要将1加到左边的列。

如果一位为ON，另一位为OFF，即我们要添加1和0，则右侧列的答案将为1。

如果两个数字都是1，或者两个数字都是0，则右列将为0。

**AND**门仅在两个输入数字均为1时开启。

因此我们很容易地添加了正确的列。

现在我们要将下一列添加到左侧。

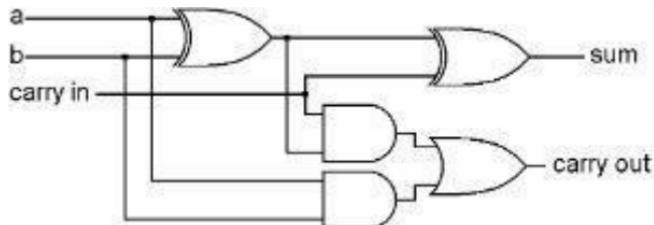
应该是一样的，对吧？

有两个位可以是0或1，但这次我们也有可能从上一列进位。

所以这不一样，这一次我们实际上是在加上三个数字，此列中的两位，加上前一列中可能的进位。

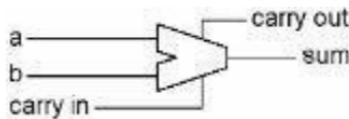
进位 >	0	1	0
1	00	01	10
1	<u>+01</u>	<u>+01</u>	<u>+01</u>
011 年	01	10	11

110



添加三位时，可能的答案是0、1、10或11（零、一、二或三）。

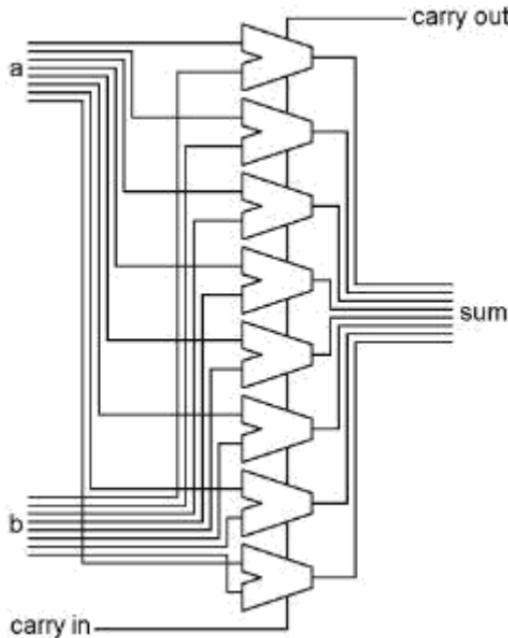
它更复杂，但并非不可能。下面是实现该功能的组合：



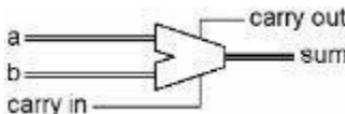
要使一个加法器将两个字节相加，我们需要一个一位加法器用于字节的每个位，每个位的进位输出连接到下一个位的进位输入。

请注意，每个位都有进位，甚至第一位（右列）也是如此。

当我们想要添加大于255的数字时，可以使用此选项。



以及总线输入和输出的简化图:



如果两个数的和大于255，则最左侧（顶部）列的进位输出位将打开，该位将在计算机的其他位置使用。

这就是计算机做加法的方法，每位只有五个门，计算机可以做算术！

## 比较器和零

上面描述的所有内容都将一个或两个字节作为输入，并生成一个字节的输出。

移位器和加法器还会产生一个与输出字节相关的额外输出位。

比较器仅产生两位输出，而不是整个字节。

比较器实际上直接内置到XORER中，因为它可以利用已经存在的门。

XORER产生其输出字节，比较器产生其两位。

这两个函数之间并没有真正的联系，只是碰巧用这种方式构建它很容易。

我们希望比较器所做的是找出输入总线上的两个字节是否完全相等，如果不相等，“a”总线上的字节是否根据二进制数系统更大。

同样的事情很简单。

当输入相同时，异或门关闭，因此如果所有异或门都关闭，则输入相等。

要确定两个二进制数中较大的一个就有点难了。

您必须从两个最高位开始，如果其中一个为ON，另一个为OFF，则ON的数字越大。

如果它们相同，则必须检查下一对较低的位等等，直到找到一对不同的位为止。

但一旦你找到一对不同的，你不想再检查任何位。例如，0010 0000（32）大于0001 1111（31）。前两位是相同的I

n两个字节。

第三位在第一个字节中为ON，在第二个字节中为OFF，因此第一个字节较大。

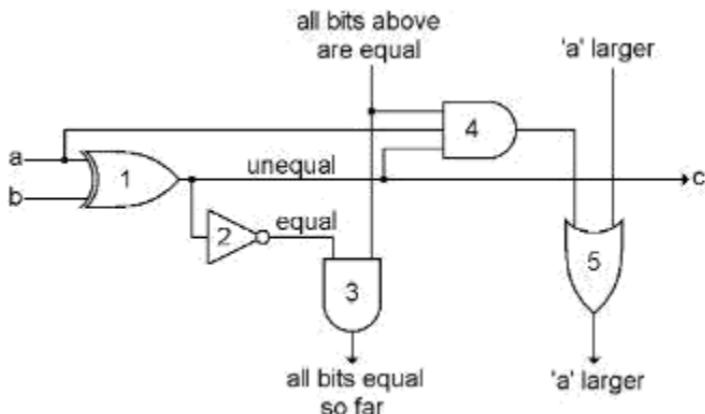
虽然其余位在第二个字节中导通，但它们的总和小于第一个字节中导通的一位。

这是我们希望发生的事情，需要五扇门乘以八个位置才能完成。

由于我们从XORER开始，我们将在每个位置再添加四个门，如图所示。

请记住，在加法器中，进位从最低位位置向上传递到最高位。

在比较器中，有两个位从最高位向下传递到最低位。



这里是比较器的一位。

您可以看到标记为“1”的原始异或门，左侧连接到每个输入总线的最多一位，右侧为输出总线生成一位。

如果门1的输出接通，则表示“a”和“b”不同或不相等。

我们加上2号门，当'a'和'b'相等时，2号门就会打开。

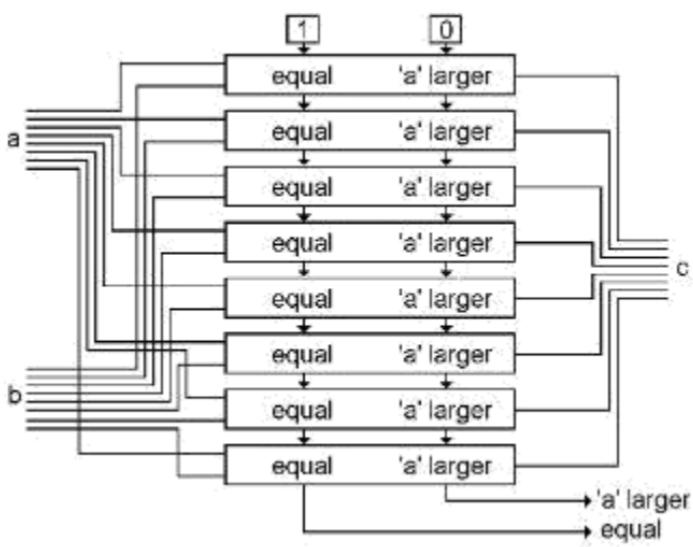
如果门2在每个位置都导通，那么门3也将在每个位置导通，从底部出来的位告诉我们两个输入字节相等。

如果有三件事是真的，4号门就会打开。

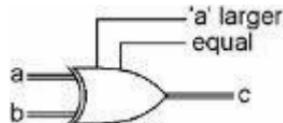
1) 位'a'和'b'不同。2) 位'a'是接通的位。3) 此点以上的所有位都相等。当门4接通时，门5接通，此点以下的每隔一个门5接通，因此比较器的输出'a'较大'。

这些位比较器中有八个堆叠在一起，如下图所示，顶部连接一个“1”和“0”以启动。

XOR函数仍然以'c'输出，现在两个比较器位位于底部。



再次简化，我们将回到总线型XOR图，只需添加比较器的两个新输出位。

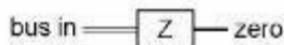


我们的电脑还需要一件东西，它能给我们提供更多的信息。

这是一个简单的门组合，以整个字节为输入，只生成一位作为输出。

当字节中的所有位都关闭时，输出位开启。

换句话说，输出位告诉我们字节的内容何时为零。



它只是一个8输入或门和一个非门。

当或门的任何输入接通时，其输出将接通，而“非”门的输出将关闭。

只有当或的所有8路输入均关断，且输出因此关断时，NOT门的输出才会导通。

右侧显示了更简单的总线表示形式。

## 逻辑

思维主体历来是人们研究和思索的对象。

古希腊有一个叫亚里士多德的人，他在这方面做了很多工作。

他一生中一定遇到过很多不合逻辑的人，因为他发明了一门学科，应该能帮助人们更明智地思考。

他的一个想法是，如果你有两个事实，你可以从前两个事实中得出第三个事实。

在学校，他们有时会给出两个事实的测试，然后给出第三个事实，并问第三个事实是否基于前两个事实是“合乎逻辑的”。您可能还记得以下问题：

如果乔比比尔大，

弗雷德比乔大，

那么弗雷德比比尔大。是真的还是假的？

或

孩子们喜欢糖果。

简是个孩子。

所以简喜欢糖果。是真的还是假的？

亚里士多德称他对这类事物的研究为“逻辑”。

这与我们对计算机的讨论唯一相关的是“逻辑”这个词。亚里士多德的逻辑包含两个事实，构成了第三个事实。

我们的任何计算机部件，如与门，取两位并产生第三位，或八与门取两字节并产生第三字节。因此，这些门所做的事情，被称为逻辑。

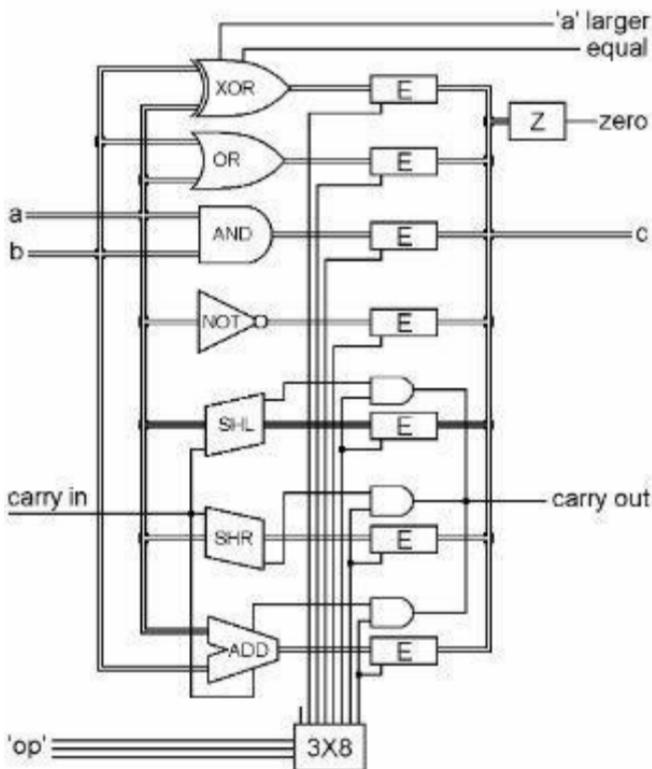
可能有AND逻辑、AND或逻辑和XOR逻辑，但它们的一般术语都是逻辑。

ANDING、ORD和XORD需要两个字节才能构成第三个字节，因此它们非常符合这种逻辑定义。移位和注释也被称为逻辑，尽管它们只需要一个字节的输入来生成输出。加法器虽然有两个输入，而且也很合乎逻辑，但不知何故，它并不属于逻辑范畴，而是属于自己的范畴，即算术。

因此，我们上面描述的对字节执行操作的所有方法都在“算术和逻辑”的标题下。

## 算术逻辑单元

现在，我们已经构建了七种不同的设备，它们可以对字节数据执行算术或逻辑运算。我们将把所有七个器件放在一个单元中，并提供一种方法来选择在任何给定时间使用哪一个器件。这被称为“算术和逻辑单元”，简称为“ALU”。



所有七个器件都连接到输入'a'，具有两个输入的器件也连接到输入'b'。所有七个器件都始终连接到输入，但每个输出都连接到其中一个使能器。

接通使能器的导线连接到解码器的输出，因此在给定时间只能接通一个使能器。

解码器的七个输出使单个器件能够继续到公共输出“c”。解码器的第八个输出在以下情况下使用

这里的一个小麻烦是加法器的进位，以及“shift in”和“shift

从移位器中取出位。

它们的使用方式非常相似，因此从现在开始，我们将把它们统称为进位。

加法器和两个移位器将进位作为输入，并生成进位作为输出。

因此，三个进位输入连接到单个ALU输入，三个输出中的一个与器件的总线输出一起选择。

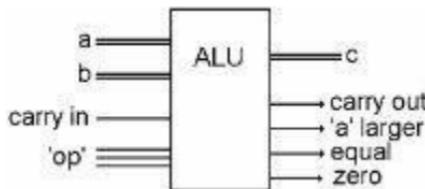
看看上面3x8解码器的最右侧输出，看看它是否使能加法器总线和加法器进位位。

这里有什么？

它是一个具有两个总线输入、一个总线输出以及四个其它位输入和四个其它位输出的盒。三个输入位选择输入总线和输出总线之间的“操作”。

再说一遍，既然我们知道了它的内容和工作原理，我们就不需要展示它的所有部分了。

下面是一种简化的绘制方法：



请注意，上面标有“OP”的三个单个位输入可以有八种不同的组合。

其中七种组合选择前面描述的设备之一。第八种组合不选择任何输出字节，但“a lager”和“equal”位仍然有效，就像它们在任何时候所做的一样，因此如果您只想进行比较，这是要选择的代码。

“op”处的位组合表示某种意义。听起来像是另一个密码。

是的，这是一个三位的代码，我们很快就会用到。

000	添加	添加
001	SHR	右移
010 年	SHL	左移
011 年	不是	不是
100	以及	以及
101	或	或
110	异或	独占或
111	厘米p	比较

算术和逻辑单元是计算机的核心。所有的动作都发生在这里。  
我敢打赌这比你想象的要简单得多。

## 更多处理器

我们还需要一个小装置。

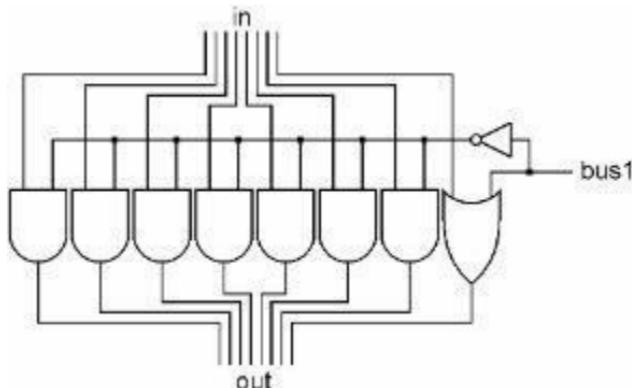
它非常简单，有一个总线输入、一个总线输出和一个其他输入位。

它与使能器非常相似。 其中七位通过AND门，一位通过OR门。

单个位输入决定字节试图通过该器件时发生的情况。 当“bus

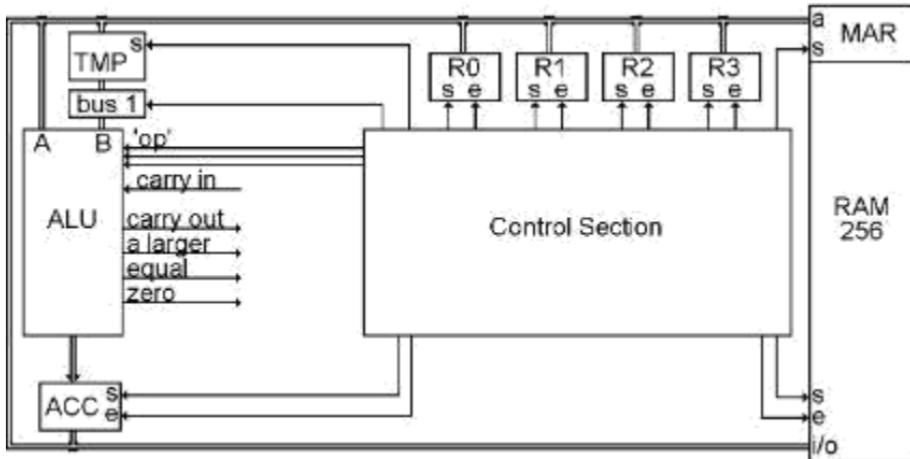
1”位关闭时，输入总线的所有位都不变地传递到输出总线。 当“bus

1”位为ON时，输入字节被忽略，O



输出字节将是00000001，它是二进制中的数字1。

我们将把这个设备称为“总线1”，因为当我们需要它时，它将把数字1放在总线上。



现在我们可以把这个“总线1”和ALU放到CPU中。

我们将更改电线进出ALU的位置，使其更符合我们的图表。

总线输入位于顶部，总线输出位于底部，所有输入和输出位都在右侧。

ALU的输出连接到ACC。ACC接收并临时存储最近ALU操作的结果。

然后，ACC的输出连接到总线，以便根据需要将其内容发送到其他地方。

当我们要执行单输入ALU操作时，必须将ALU的三个“OP”位设置为所需操作，使能总线上所需的寄存器，并将答案设置为ACC。

对于双输入ALU操作，有两个步骤。

首先，我们在总线上启用一个寄存器并将其设置为TM

P。然后，我们在总线上启用第二个寄存器，选择ALU操作，并将答案设置为ACC。

如您所见，如果我们在正确的时间开启或关闭适当的使能和设置位，现在可以将字节数据移入RAM和从RAM移出、移入寄存器和从寄存器移出、通过ALU移至ACC，然后从那里移入寄存器或RAM。这就是计算机内部的情况。没那么复杂吧？

我们需要在适当的时候打开和关闭所有这些控制位，以便我们能够做一些有用的事情。现在是查看标有“Control Section”（控制部分）的框的时候了。

## 时钟

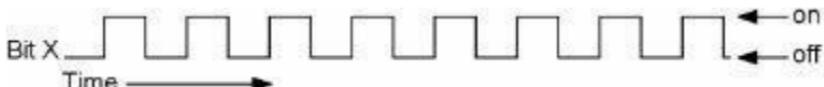
我们需要在适当的时间打开和关闭适当的控制位。

稍后我们将查看适当的位，首先我们将查看适当的时间。

这是一种新的画法，我们称之为图。它显示了一位是如何随时间变化的。

时间从左边开始，向右边前进。

图表上的线有两个可能的位置，向上表示位为ON，向下表示位为OFF。



此图显示位'X'有规律地开和关、开和关。

底部可能有一个时间刻度来显示这种情况发生的速度。

如果页面的整个宽度代表一秒，那么位'X'将每秒开关大约8次。

但是我们不需要这些图中的时间刻度，因为我们只关心两个或更多位之间的相对时序。

在实际的计算机中，速度将非常快，例如比特每秒进行10亿次开关

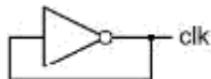
第二。

当某物有规律地重复某一动作时，其中一个动作单独称为循环。

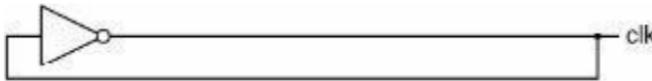
上图显示了大约八个循环。

你可以说，从一次比特接通到下一次比特接通是一个周期，或者你可以说，从关断时间的中间到下一个关断时间的中间是一个周期，只要周期开始于比特处于其活动的某个阶段的某个时间点，并且持续到比特回到动作的同一阶段

又是伊芙蒂。单词“cycle”来自单词“circle”，所以当位变为全圆时，就是一个循环。

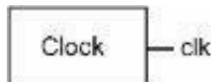


19世纪有一位住在德国的科学家做了一些早期的研究，导致了无线电的出现。他的名字叫海因里希·赫兹，除其他外，他研究的是断断续续的电学。在他去世几十年后，人们决定用他的名字来描述电的断断续续有多快，或者每秒发生多少次循环。因此，一赫兹（简称Hz）意味着电流每秒开关一次。500 Hz表示500每秒0次。



为了更快的速度，我们再一次遇到这些古老的语言，每秒一千次被称为千赫兹，简称千赫兹。

每秒开关一百万次被称为兆赫，简称兆赫，十亿次被称为千兆赫，简称千兆赫。



每台计算机都有一个特殊的位。

计算机中的所有其它位都来自某个地方，它们由其它位或开关来接通或断开。这一个特殊的位完全由自己打开和关闭。

但它没有什么神秘的，它只是非常有规律和非常快地断断续续地进行。

此特殊位的构建非常简单，如下所示：

这样做似乎很愚蠢。只需将“非”门的输出连接回其输入即可？这个能做什么？如果从输出接通开始，电流返回到输入端，进入关断输出的门，关断输出的门又返回到输入端，关断输出。是的，这扇门会尽快打开和关闭。这实际上太快，不能用于任何事情，所以它可以放慢速度，只要延长线，使环路。简化图显示，这是计算机中有输出但没有任何输入的一个特殊位。

此位具有名称。它被称为时钟。

现在我们通常把时钟看作是一个有表盘和指针的东西，或者是屏幕上的一些数字，我们在电脑屏幕的一角看到过这样的时钟。

不幸的是，有人把这类位命名为时钟，而这个名字一直被计算机先驱们所沿用。

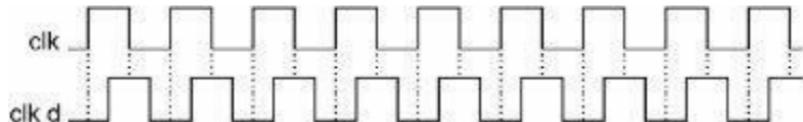
它本可以被称为鼓点、标点符号、心脏或节律部分，但他们称之为时钟。

这就是我们在余下的时间里所说的“时钟”的含义



书。我想这是一个时钟滴答作响，但没有刻度盘。

如果我们想讨论告诉你时间的钟的类型，我们将把它叫做“Time of day clock”，简称“TOD clock”。但“clock”一词将表示这种类型的位。



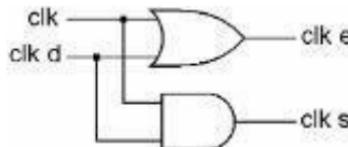
这只钟走得有多快？现在每秒超过10亿次，也就是几千兆赫。

这是电脑公司告诉你的一个主要特征，它将向你展示他们的电脑有多棒。

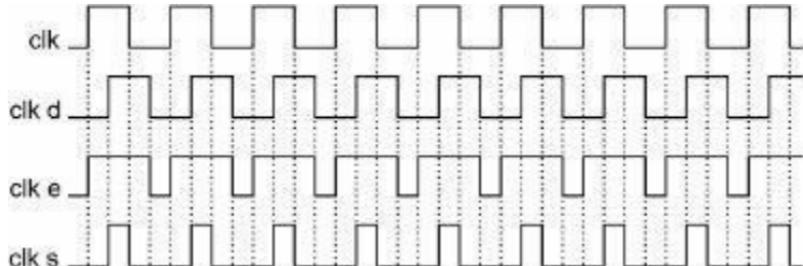
当你看到出售的电脑时，它们所宣传的速度就是时钟的速度。

电脑越快，它就越贵，因为它能在一秒钟内做更多的事情。

这一位的速度决定了节奏的快慢



整个电脑。



要通过总线传输数据，我们首先需要使能一个且仅一个寄存器的输出，以便其电能能够通过总线传输到其他寄存器的输入端。

然后，当数据在总线上时，我们要打开和关闭目标寄存器的SET位。

由于目的寄存器在设置位关闭时捕获总线的状态，因此我们希望在关闭第一个寄存器的使能位之前确保它关闭，以确保

这是这两位的单个开/关周期。



如果您看看这里的时序，就会发现这符合我们的要求：首先使能寄存器的输出，然后，在数据有一点时间沿总线下行后，在第一个寄存器关闭使能位之前，先打开和关闭目标寄存器的SET位。

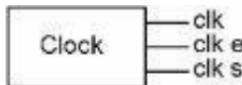
当然，这些时钟位不能直接连接到每个寄存器。

其间必须有其他门，这些门只允许一个寄存器在任何时候使能，并且只允许所需的寄存器接收一个集合。

但所有使能和设置最终都来自这两位，因为它们具有正确的时序。

由于我们将在整个计算机中使用CLK、CLK E和CLK

S，下面是我们将用来显示时钟的图表：



## 做些有用的事

假设我们想做一些有用的事情，比如把一个数字加到另一个数字上。

我们在R0中有一个数字，在R1中还有一个数字要添加到R0中的数字中。

到目前为止，我们已经构建的处理器具有完成此添加的所有连接，但完成此添加需要一个以上的时钟周期。

在第一个时钟周期，我们可以在总线上使能R1，并将其设置为TM P。

在第二个周期中，我们可以使能总线上的R0，将ALU设置为ADD，并将答案设置为ACC。

在第三个周期中，我们可以使能总线上的ACC，并将其设置为R0。

我们现在有旧值R0，加上R0中的R1。

也许这看起来不是很有用，但这是计算机所做的一种小步骤。

任何这样的小步骤都使计算机似乎能做非常复杂的事情。

因此，我们看到处理器要做一些有用的事情，它需要几个步骤。

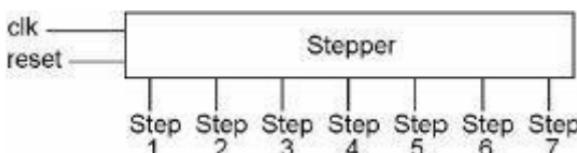
它需要能够按顺序执行操作。我们需要在这个“控制室”里再放一块。

## 一样一样来

本章介绍了一个新的部分称为“步进器”。首先，我们将描述完成的步进器，确切地显示它的功能。在那之后，我们将看到它是如何建造的。

如果你恰好信任你的作者，相信这样的一个步骤可以建立在大门之外，你是如此匆忙，你想跳过这一章的“它是如何建立的”部分，你可能仍然理解计算机。

这是一个完整的踏板。



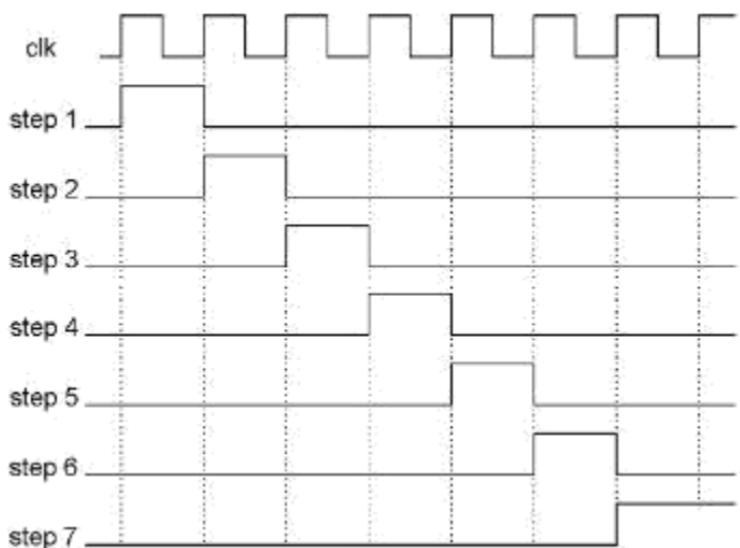
它有两个输入。

其中一个称为'clk'，因为这是我们连接一个正在接通和断开的位的地方，例如我们的原始时钟位。另一个输入称为“reset”，用于将步进器返回到第一步。

对于输出，它有多个位，每个位将导通一个完整的时钟周期，然后逐个关断。

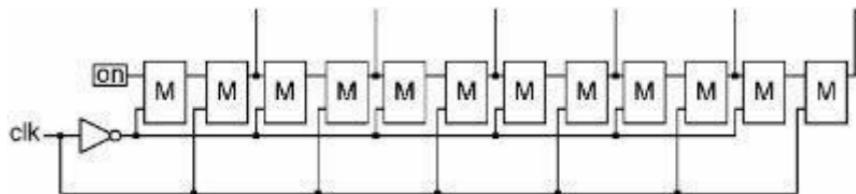
标为“步骤1”的输出开启一个时钟周期，然后标为“步骤2”的输出开启下一个时钟周期，等等。步进器可以构建为mA

以下是输入'clk'位和七步进器输出的曲线图。



这是步进机的制造方法。

它使用与我们用来制作寄存器相同的一些存储器位来完成，但它们的排列方式非常不同。我们不会在这些位中存储任何内容，而是使用它们来创建一系列步骤。步进器由多个以串形式连接的存储器位组成，其中一个位的输出连接到下一个位的输入。下面是一个显示大多数步进器的图表：



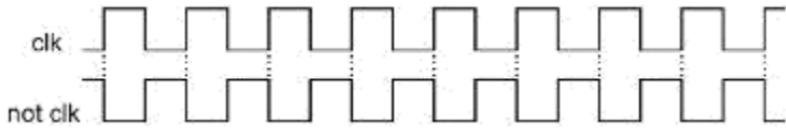
首先看一下“M”内存位系列，就像我们在书的前面使用的一样。

在这幅图中，有12个连接在一起，其中一个是输出连接到下一个的输入，一直到线路的下游。

左边第一位的输入连接到一个始终通电的位置，因此当该“M”的设置位通电时，该“M”将接收该“ON”状态，并将其传递到其输出。

如果查看这些'M'的设置位，您将看到偶数编号'M'的设置位连接到CLK，奇数编号'M'的设置位在通过“非”门后连接到同一时钟。

通过将CLK通过NOT门而产生的新位可以称为'NOT CLK'，我们可以在此图中同时显示这两个位：



那这群门会怎么样？

如果您假设'm'的所有启动都处于关闭状态，然后启动'clk'“ticking”，那么它将执行以下操作。

第一次'clk'接通时，不会发生任何事情，因为第一个'm'的设置位连接到'not clk'，当'clk'接通时，该位关闭。当'clk'关闭时，'not clk'打开，第一个'm'将打开，但第二个'm'不会发生任何情况，因为其'set'位连接到'clk'，后者现在关闭。当"clk"重新打开时，第二个"m"现在将打开。

当时钟滴答作响时，进入第一个存储器位的“on”将沿行下移，每次一位时钟继续，每次时钟关闭一位。因此，每个时钟周期有两位导通。

现在，转到下面的完整步进图，步骤1来自连接到第二个“M”输出的“非”门。由于所有“M”的开始都是关闭的，因此步骤1将一直打开，直到第二个“M”打开，此时步骤1将结束。对于其余步骤，每个步骤将从其左侧'M'接通时持续到其右侧'M'接通时为止。当左'M'接通，右'M'断开时，步骤2-6的AND门的两个输入均为ON。

如果我们连接一个m的输出

而'M'两个空格的输出与门相距较远时，其输出将导通一个完整的时钟周期。

当左输入接通时，每个输入接通，但右输入尚未接通。

这为我们提供了一系列位，每个位开启一个时钟周期，然后关闭。

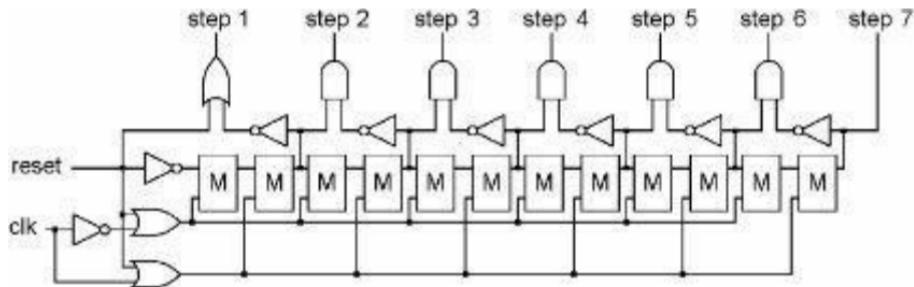
这里唯一缺少的是'M'位继续运行。一旦全部打开，

尽管时钟在不停地滴答作响，但没有更多的行动了。

所以我们需要一种方法来重置它们

我们就可以重新开始了。我们必须找到一种方法来关闭第一个

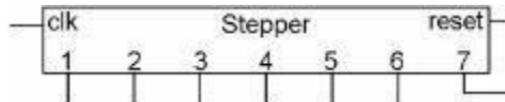
“m”，然后同时打开所有设置的位。当这种情况发生时，



第一个'M'的输入将尽可能快地通过所有'M'。我们将添加一个

名为“重置”的新输入将完成这些工作。

当我们打开“reset”时，它使第一个“m”位的输入变为0，并打开所有



我们已将复位位重新定位到关系图的右侧，并将其连接到最后一个步骤（7，），以便步进器将自动复位。

然而，第7步不会打开很长时间，因为一旦0可以通过'm'字符串，它就会关闭自己。这意味着步骤7不会持续足够长的时间，无法用于总线上的一次数据传输。

我们想要完成的所有事情都将在步骤1到6中完成。

## 一切都在掌控之中

用我们的钟，我们有一个鼓点，使事情进行。

它有一个基本输出，另外两个输出旨在促进寄存器内容从一个寄存器移动到另一个寄存器。有了步进器，我们有一系列的位相继上传，每一位对应一个时钟周期。

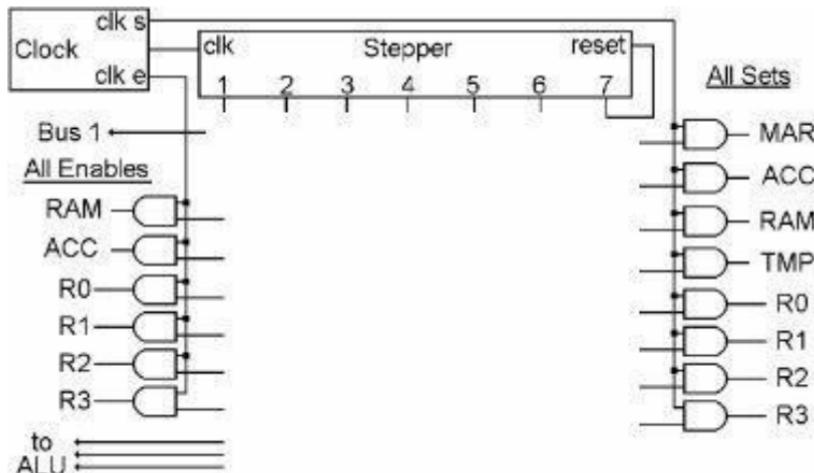
还记得前几章我们看到的CPU图吗？

它显示总线、ALU、6个寄存器，甚至计算机的另一半（RAM）都连接得非常整齐。至少所有的公共汽车线路都在那里。

但所有寄存器、RAM、总线1和ALU都由来自标有“Control

Section”（控制部分）的神秘盒子的电线控制，我们对此一无所知。

现在是时候看看盒子里的东西了。



此绘图是计算机控制部分的开头。最上面是钟和踏板。

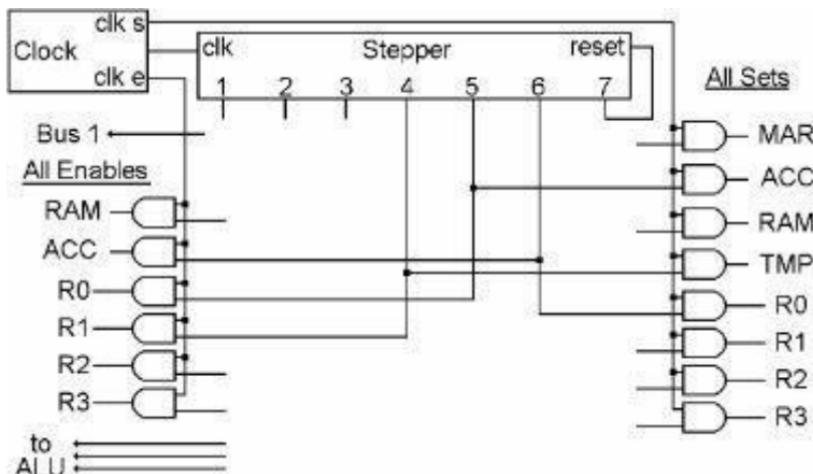
然后，寄存器和RAM中的所有控制位都集中在一个位置，左侧为所有“使能”位，右侧为所有“设置”位。然后，我们将AND门的输出连接到每个“使能”位和每个“设置”位。每个AND门的一个输入连接到左侧的“enables”的“clk e”或“sets”的“clk s”右边。

因此，如果使用AND门的其他输入来选择任何寄存器，则左侧所有项目的“使能”位将永远不会开启，除非在“CLK E”时间内。

类似地，在右侧，这些寄存器中的任何一个的“set”位仅在“clk s”时间内开启。

这是个总机。我们需要让电脑做些什么的所有东西都在这里的一个地方。我们所需要做的就是以智能的方式将一些控制位连接到一些步骤，然后就会发生一些有用的事情。

做一些有用的事，重新审视



现在我们已经有了控制部分的开头，我们只需添加几条线，就可以完成前面假设的简单添加，即将R1添加到R0。

要“做一些有用的事情”，比如将R1添加到R0，我们所要做的就是连接中间的几根电线，如图中的步骤4、5和6所示。

每一步都会导致CPU图中所示的某些部分发生一些变化。

每个步骤都连接到左侧的一个“enable”和右侧的一个“set”，因此会导致一个器件将其输出连接到总线，另一个器件保存其输入端现在显示的内容。

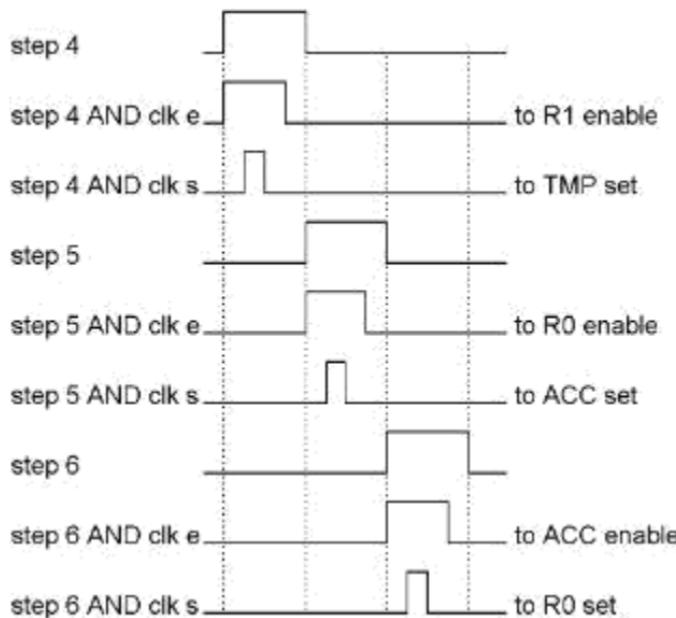
步骤4连接到R1'enable'和TM P'

设置。“步骤5连线到R0”enable“和ACC”set。“ALU”op“位不需要任何连接，因为A DD的”op“代码是000。 步骤六连接到acc'enable'和r0'set'。

在步骤4中，R1使能，TM P置1。 R1的内容穿过总线（在CPU图中），并被TM P捕获。

在步骤5中，R0使能，ACC置1。

如果我们想做除加法以外的事情，这一步我们将打开适当的alu'op'代码位。



在步骤6中，ACC使能，R0置1。

下面是步骤图，显示了何时使能和设置每个寄存器。

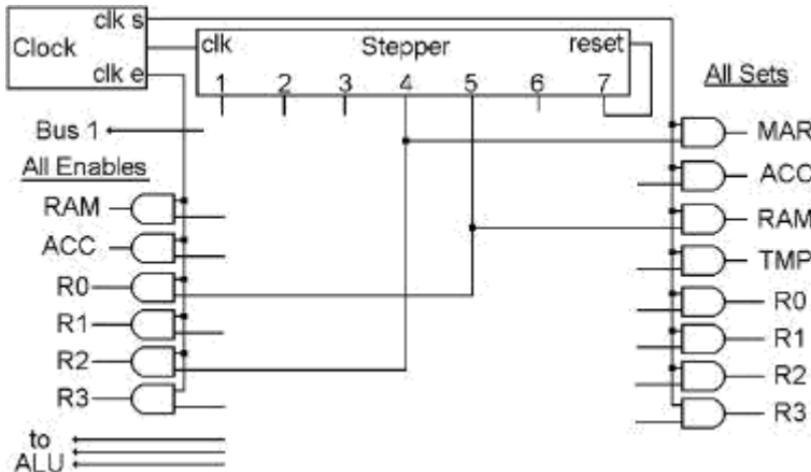
R0现在包含R0加上R1的原始内容之和。

这就是计算机如何使事情发生在一个严格控制的芭蕾舞步和字节移动在机器内。

在步骤7中，将步进器重置到步骤1，在步骤1中重复该过程。

当然，一遍又一遍地做这个加法并不是很有用，即使您从R0和R1中的数字1开始，R0也会非常快地达到255。

此操作的布线比另一个操作简单，仅有两个使能和两个设置。



我们可以利用RAM、6个寄存器和ALU进行多种组合。

我们可以从RAM中获取一个字节，并将其移至四个寄存器中的任意一个，我们可以通过ALU移动任意一个或两个寄存器，然后将它们相加，或者将它们相减，或者将它们相乘，或者将它们相除，等等。

我们需要一种方法，让CPU一次做一件事，下一次做另一件事。

控制部分需要一些东西来告诉它在每个序列中做什么。

## 接下来是什么？

现在有个可怕的主意。想象一下，一个员工在快餐店所做的工作被分解为各个要素。走到柜台前，说：“我可以点菜了吗？”听答案，接收银台上的“芝士汉堡”按钮，等等。现在让我们假设，在这样一个机构工作涉及到256个或更少的个人行为。然后，您可以创建一个代码，将一个字节的一种状态与每个单独的activ相关联。雇员的身份。然后可以将员工的操作序列表示为字节序列。

首先，我们组成一个代码表。我们在这页的左边写了一些代码。

然后我们决定这些代码的含义，并在代码旁边写下这些含义。

现在，我们有了一个员工可能采取的所有可能操作的列表，以及一个表示每个操作的代码：

0000 0000=步行到柜台

00000001=说：“我可以点菜了吗？”

000010=听答案

0000 0011=Press芝士汉堡按钮

0000 0100=按下炸薯条按钮。

0000 0101=按下牛奶按钮

0000 0110=按下“总计”按钮

0000 0111=收钱

0000 1000=给客户零钱

0000 1001=打开空袋子

0000 1010=P将芝士汉堡系在袋子里

0000 1011=袋装花边薯条

0000 1100=将牛奶容器系在袋子里

0000 1101=将袋子递给顾客

10000000=转到右6位中的步长号。

0100 0000=如果“是”，则转到右侧6位中的步骤号。

0001 0000=回家。

1. 现在，如果我们想描述员工应该如何行动，我们需要编写一系列事件，他应该遵循这些事件：
2. 0000 0000=步行到柜台。
3. 00000001=说：“我可以点菜了吗？”
4. 0100 0010=如果客户没有应答，请转到步骤2。
5. 000010=听答案。
6. 0100 0111=如果顾客不说芝士汉堡，请转到步骤7。
7. 000011=按下芝士汉堡按钮。
8. 0100 1001=如果客户不说薯条，请转到步骤9。
9. 0000 0100=按下炸薯条按钮。
10. 0100 1011=如果客户不说牛奶，请转到步骤11。

11. 0000 0101=按下牛奶按钮。
12. 0100 1101=如果客户说仅此而已,请转到步骤13。
13. 1000 0100=返回步骤4。
14. 0000 0110=按下“总计”按钮。
15. 0000 0111=收钱。
16. 0000 1000=进行更改并将其交给客户。
17. 0000 1001=打开空袋子。
18. 0101 0011=如果订单不包括芝士汉堡,请转到步骤19。
19. 0000 1010=把芝士汉堡系在袋子里。
21. 0101 0110=如果订单不包括薯条,请转到步骤22。
22. 0000 1011=袋子里有P条花边薯条。
23. 0101 1000=如果订单不包括牛奶,请转到步骤24。
24. 0000 1100=将牛奶容器系在袋子里。
25. 00001101=将袋子递给顾客。
26. 0101 1011=如果是退出时间,请转到步骤27。
27. 10000001=返回到步骤1。

0001 0000=回家。

我们的电脑也许能“理解”这样的代码。

## 第一个伟大的发明

我们需要的是某种方法来执行从一个步进器序列到下一个的不同操作。

我们怎么能用一种方式连接一个序列，然后用另一种方式连接下一个序列呢？

当然，答案是使用更多的门。

用于一个操作的布线可以与AND门连接或断开，而用于不同操作的布线可以与一些更多AND门连接或断开。可能还有第三和第四种或更多的可能性。只要其中只有一个这些操作同时连接，这将很好地工作。

现在我们有几种不同的操作可以完成，但是您如何选择要完成的操作呢？

这一章的题目是“第一个伟大的发明”，那么这个发明是什么呢？

本发明是我们将有一系列的指令在RAM中，这将告诉CPU做什么。

我们需要三件事才能使这件事成功。

本发明的第一部分是，我们将向CPU添加另一个寄存器。

该寄存器将被称为“指令寄存器”，简称“IR”。来自该寄存器的位将“指示”CPU做什么。IR从总线获取输入，其输出进入CPU的控制部分，在那里位选择几种可能的操作之一。

本发明的第二部分是CPU中的另一寄存器，简称为“指令地址寄存器”或“IAR”。

该寄存器的输入和输出与总线连接，就像通用寄存器一样，但该寄存器只有一个用途，即存储下一条指令的RAM地址，我们希望将其移入IR。

如果IAR包含00001010（十进制10），则下一条将被移至IR的指令是驻留在RAM地址10的字节。

本发明的第三部分是控制部分中的一些布线，其使用步进器将期望的“指令”从RAM移动到IR，将1添加到IAR中的地址，并执行已被放入IR中的指令所要求的动作。

当该指令完成时，步进器重新开始，但现在IAR已经添加了1，因此当它从RAM中获得该指令时，它将是位于以下RAM地址的另一条指令。

这三部分的结果是一个伟大的发明。

这就是为什么我们可以让计算机做许多不同的事情。

我们的总线、ALU、RAM和寄存器使许多组合成为可能。

IR的内容将决定将哪些寄存器发送到何处，以及对其执行何种算术或逻辑运算。

我们所要做的就是在RAM中放置一系列字节，这些字节代表一系列我们想要做的事情，一个接一个。

CPU将要使用的这一系列驻留在RAM中的字节称为

“程序。”

这里发生的基本情况是CPU从RAM“获取”一条指令，然后“执行”该指令。

然后它获取下一个并执行它。这种情况一次又一次地发生，每秒数百万或数十亿次。这就是计算机所做的事情的简单性。

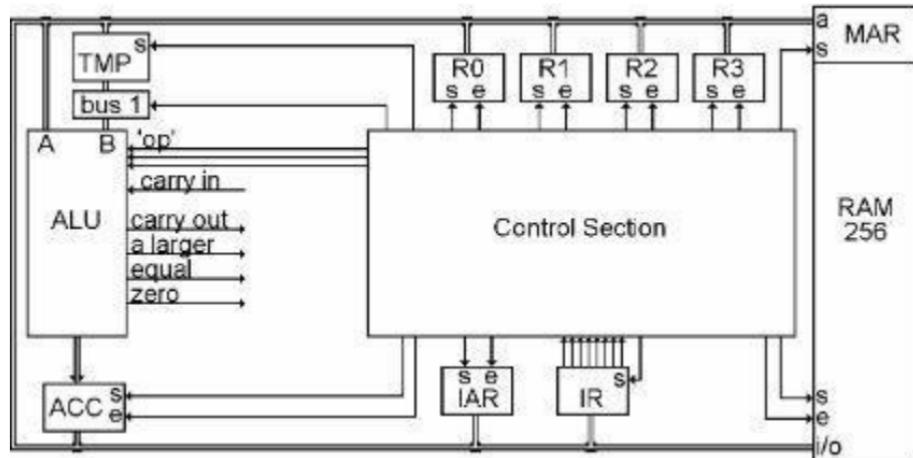
有人把一个程序放到RAM中，这个程序如果设计巧妙，就会使计算机做一些人们认为有用的事情。

这台计算机的步进器有七个步骤。步骤7的目的只是将步进器重置回步骤1。

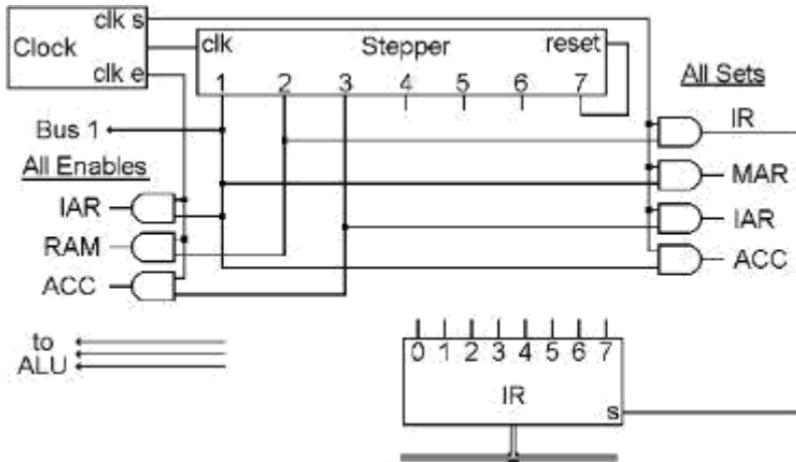
所以有六个步骤，CPU在这六个步骤中做一些小事情。每个步骤持续一个时钟周期。

作为一个整体所采取的六个步骤被称为“指令周期”，CPU需要六个步骤来完成获取和执行一条指令所需的所有操作。

如果我们假设时钟在1千兆赫处滴答作响，那么我们的计算机将能够执行166,666,666条指令



每一秒。



这是CPU的图片，其中添加了两个新寄存器。

在那里，它们在控制部分之下，连接到总线。

IAR有一个“set”和“enable”，IR只有一个“set”，就像TM P和M

AR一样，因为它们的输出没有连接到总线，所以我们永远不需要关闭它们。

下面是执行指令周期的“获取”部分的控制部分内的布线。

它使用步进器的前三个步骤，并且对于所有类型的指令都是相同的。

这里显示了步进器的前三个步骤，并导致从RAM“获取”下一个“指令”。

然后，其余的步骤“执行”“指令”。确切地说，在步骤4、5和6中要做什么，是由所获取的指令的内容决定的。然后，步进器重新开始，获取下一条指令，并执行它。

ALU将是二进制值为1的字节。

如果在'clks'期间打开ACC的'set'，我们将捕获ACC中IAR加1的和。

这恰好是我们在处理完当前的一条指令后想要获取的指令的地址！

步骤2在总线上启用RAM中的当前选定字节，并将其设置为IR。

这是我们将在该指令周期的步骤4、5和6中“执行”的指令。

在图中，您可以看到来自步骤2的线路连接到RAM的“enable”和IR的“set”。

在步骤3中，我们需要完成IAR的更新。

我们在步骤1中向其添加了1，但答案仍然在ACC中。

它需要在下一个指令周期开始之前移至IAR。

因此，您可以看到来自步骤3的线路连接到ACC的“enable”和IAR的“set”。

到步骤4时，指令已经从RAM移到IR，现在步骤4、5和6可以执行IR内容所要求的任何操作。

当该操作完成且步进器复位时，序列将重新开始，但现在IAR已经添加了1，因此下一个RAM地址处的指令将被提取并执行。

这种在RAM中放置一系列指令并让CPU执行它们的想法是一个伟大的发明。

## 说明书

我们现在有了一个新的寄存器，称为指令寄存器，它包含一个字节，用来告诉控制部分该做什么。放入此寄存器的模式具有意义。

听起来像是另一个密码，事实上，是的。此代码将称为“指令代码”。

因为我们是从头开始构建这台计算机的，所以我们可以发明自己的指令代码。

我们将采取256个不同的代码，可以放在指令寄存器，并决定他们将意味着什么。

然后，我们必须设计控制单元内部的布线，使这些指令按照我们说过的那样执行。

你还记得二进制编码吗？

我们说，这是最接近“自然”的计算机代码，因为它是基于相同的方法，我们使用我们的正常数字系统。然后是ASCII码，这是一群人在一次会议上发明的。

ASCII根本不是自然的，它只是那些人的决定。

现在我们有了指令代码，它也将是一个完全被发明的代码——它并不自然。

m为许多不同类型的计算机发明了任何不同的指令代码。

我们不会在这里学习它们中的任何一个，也不需要在以后学习它们中的任何一个，除非你在必要的时候继续从事高技术的职业。

但是所有的指令代码都是相似的，因为它们是使计算机工作的东西。

这本书中唯一的指令代码是一个

我们为我们的简单电脑发明的。

在发明我们的指令代码时，最重要的事情将是我们能够使代码工作的布线多么简单。

可能有多少种不同的指令？

由于指令寄存器是字节，因此可能有多达256条不同的指令。

幸运的是，我们将只有9种类型的指令，所有256种组合都将属于这些类别之一。

它们很容易描述。

所有指令都涉及在总线上移动字节。

这些指令将导致字节进出RAM，进出寄存器，有时通过ALU。

在接下来的章节中，对于每种类型的指令，我们将研究该指令的位、使其工作所需的门和布线，以及另一种我们可以用来简化程序编写的方便代码。

### 算术或逻辑指令

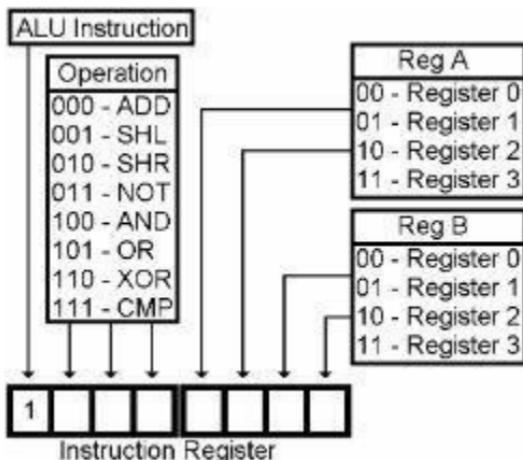
第一种类型的指令是使用ALU的类型，就像前面的ADD操作一样。

正如您所记得的，ALU有八项功能，其中一些功能使用两个字节的输入，另一些功能只使用一个字节的输入。在其中的7种情况下，它有一个字节的输出。

这类指令将选择一个ALU操作和两个寄存器。这是计算机能做的最多的指令。

它实际上有128个变体，因为有8个操作和4个寄存器，您可以从4个寄存器中选择两个。即8乘以4乘以4，或者128种可能的使用该指令的方式。

因此，这不仅仅是一条指令，而是一整类指令，它们都使用相同的接线来获得工作



搞定了。

如果选择一个输入操作，如SHL、SHR或NOT，则字节将来自

**REG A**, 通过ALU, 答案将放在**REG B**

**B**中。您可以选择从一个寄存器转到另一个寄存器, 例如从**R1**转到**R3**, 或者选择从一个寄存器转回到同一个寄存器, 例如从**R2**转到**R2**。

当您执行后一操作时, 注册表的原始内容将被替换。

对于两个输入操作, **REG A**和**REG B**将被发送到ALU, 而答案将被发送到**REG B**。因此, **REG B**中的任何内容 (它是操作的输入之一) 将被答案替换。

还可以为两个输入指定相同的寄存器。

例如, 如果要将所有0放在寄存器1中, 只需将**R1**与**R1**异或即可。

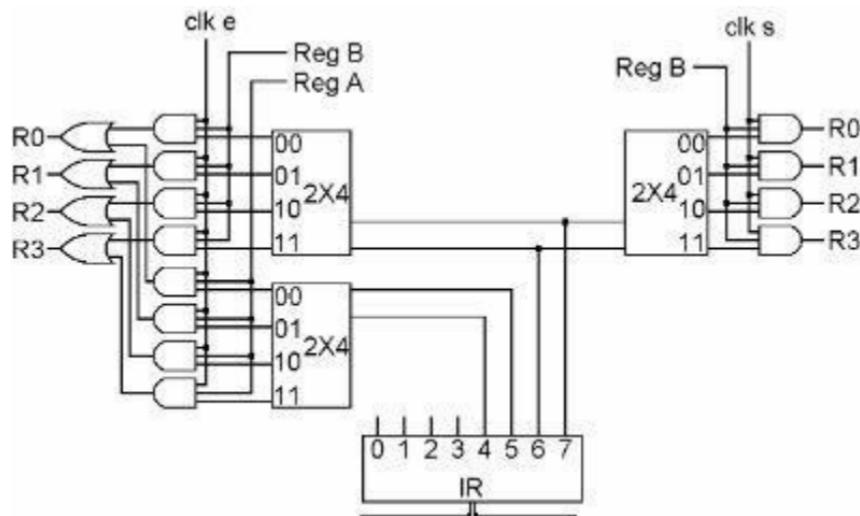
无论**R1**的开头是什么, 所有位比较都是相同的, 这使得所有位的输出为零, 然后放置

返回**R1**。

#### CM

**P**操作采用两个输入, 并对它们进行比较, 看它们是否相等, 如果不相等, 则看第一个输入是否较大。但**CM P**操作不存储其输出字节。

它不会替换任何一个输入字节的内容。



ALU指令控制单元中的布线非常简单，但有一个额外的东西将被许多类型的指令使用，我们需要首先查看。这与寄存器有关。在“Do some business revisioned”中，我们使用了两个寄存器。

要使用它们，我们只需将每个寄存器的AND门连接到步进器的所需步进。

这很好，但在ALU指令和许多其它指令中，指令寄存器中的位指定了吉斯特用。

因此，我们不想直接连接到任何一个寄存器，我们需要能够连接到任何一个寄存器，但让指令中的位准确选择哪个寄存器。以下是执行此操作的控制部分布线：先看右边。

当我们想要设置一个通用寄存器时，我们将正确的步骤连接到这条我们称为“REG B”的线路上。正如您所看到的，“CLK S”连接到所有四个AND门。“REG B”也连接到所有四个AND门。但这四个AND门各有三个输入。

每个AND门的第三个输入来自 $2 \times 4$ 解码器。

请记住，在任何给定的时间，解码器的一个且仅有一个输出接通，因此实际只选择一个寄存器来使其

“设置”位打开。

解码器的输入来自IR的最后两位，因此它们决定哪个寄存器将由标记为“REG B”的线设置。如果您回顾ALU指令代码的位图，它显示指令的最后两位是决定要用 $\text{REG B}$ 的寄存器。

图片的左边和右边非常相似，除了每件事都有两个。

请记住，在ADD等ALU指令中，我们需要使能两个寄存器，一次一个寄存器，用于ALU的输入。指令的最后两位也用于左侧的“REG B”，您可以看到，“CLK E”、“REG B”和解码器用于在一个寄存器的正确步骤中使能该寄存器。

IR的位4和位5用于在其正确步骤期间使用SEPAR使能“REG A

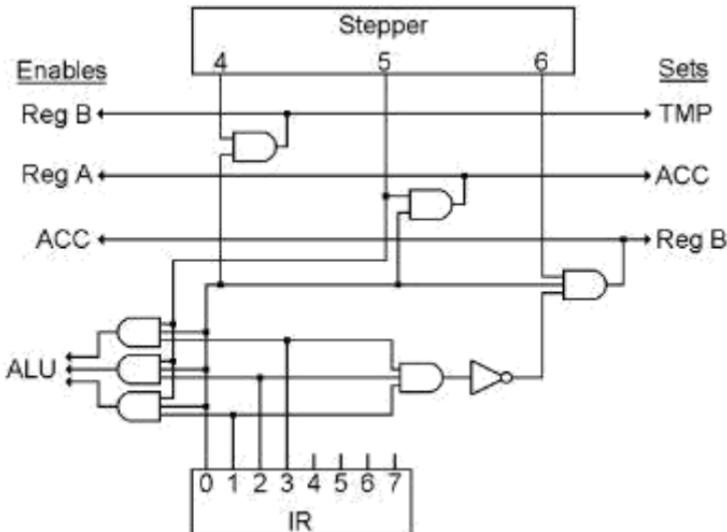
ATE解码器和名为“REG

A”的线这两种结构的输出在转到实际寄存器使能位之前进行“或”运算。

我们永远不会同时选择“Reg A”和“Reg B”。

当已获取的指令以1开头时会发生什么？

这意味着这是一个ALU指令，我们需要做三件事。首先，我们要将“REG B”移至TM P。然后，我们要告诉ALU要执行哪些操作，将“REG A”放在总线上，并将ALU的输出设置为ACC。然后我们要将ACC移动到“Reg B。”



IR的位0决定这是否是ALU指令。

当位0为ON时，位0被连线以使ALU指令的所有步骤发生。

下图显示了添加到控制部分的八个门和导线，这些门和导线使ALU指令的步骤4、5和6执行我们需要它们执行的操作。

在下图中，在IR的正上方和左侧，有三个AND门。

这些门的输出到ALU上的三条“op”线，这三条“op”线告诉ALU该执行哪种操作。

这三个AND门中的每一个都有三个输入。每个门的一个输入连接到IR的位0。

每个门的第二输入从步进器连线到步骤5。每个门的其余输入连接到IR的位1、2和3。

因此，除了步骤5中IR位0恰好为1之外，到ALU的三条线始终为000。

此时，通向ALU的线路将与IR的位1、2和3相同。

IR位0沿图上继续向上，右转并连接到另外三个AND门的一侧。

这些门的另一侧连接到步骤4、5和6。

ALU指令现已完成。

步骤7重置步进器，然后该步进器再次执行其步骤、获取下一条指令等。

我们将在这里再发明一个东西，那就是在一张纸上写CPU指令的速记法。  
在指令代码中，10001011的意思是“将R2添加到R3中”，但要查看10001011并立即想到添加和寄存器，需要大量的实践。  
反过来想想也需要大量的记忆，也就是说，如果你想异或两个寄存器，异或的指令代码是什么？编写add之类的东西会更容易一些。

R2、R3或XOR R1、R1。

这种使用速记的想法有一个名字，它被称为计算机语言。  因此，除了发明指令代码外，我们还将发明一种表示指令代码的计算机语言。  ALU指令产生了我们新语言的前八个单词。		语言
添加RA和RB并将答案放入RB中	添加	我正在学习

Ra、Rb	SHR	Ra、Rb
右移RA并将答案放入RB	SHL	Ra、Rb
左移Ra并将答案放入Rb中	不是	Ra、Rb
而不是RA，并将答案放在RB中	以及	Ra、Rb
RA和RB，并将答案放入RB中	或	Ra、Rb
或RA和RB，然后将答案放入RB中	异或	Ra、Rb
异或RA和RB进入RB	厘米p	Ra、Rb

比较RA和RB

当一个人想写一个计算机程序时，他可以直接在指令代码中写，或者使用计算机语言。

当然，如果你用一种计算机语言编写一个程序，它必须被翻译成实际的指令代码，然后才能放入RAM并执行。

#### 加载和存储指令

加载和存储指令非常简单。它们在RAM和寄存器之间移动一个字节。

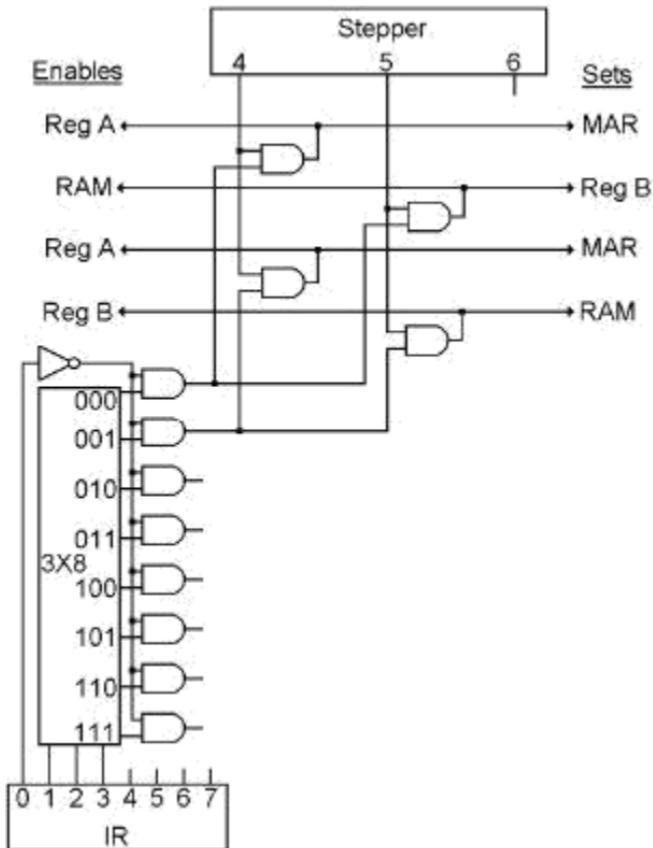
它们彼此非常相似，所以我们将在一章中讨论它们。

稍后我们将讨论这些指令的详细信息，但首先我们需要知道什么时候在指令寄存器中加载或存储指令。有了ALU指令，我们只需要知道位0是开的。

每种其它类型指令的代码以位0关闭开始，因此，如果我们将NOT门连接到位0，当该NOT门打开时，这将告诉我们还有其它类型的指令。

在这台计算机中，有八种类型的指令

这不是ALU指令，所以当位0关闭时，我们将使用IR的下三位来告诉我们确切的指令类型。



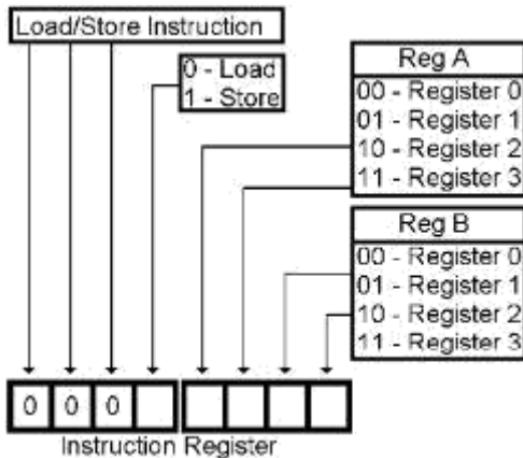
在ALU指令中进入ALU的三位也进入控制部分的3x8解码器。

如您所知，解码器的一个且仅有一个输出始终导通，因此我们将在输出上设置与门，以防止任何输出在ALU指令期间流向任何地方。

但当它不是ALU指令时，打开的解码器的一个输出将通过其AND门，并依次连接到更多的门，以生成适当的指令

第一条指令将一个字节从RAM移到寄存器，这称为“加载”指令。

另一个则相反，它将一个字节从寄存器移到RAM，称为“存储”指令。



LOAD指令的指令代码是0000， STORE指令的指令代码是0001。

这两种情况下的其余4位都指定了两个寄存器，就像ALU指令一样，但在这种情况下，一个寄存器将用于选择RAM中的一个位置，另一个寄存器将从该RAM位置加载或存储到该RAM位置。

步骤4对于这两个指令是相同的。其中一个寄存器由IR位4和5选择，并在总线上使能。

然后将总线设置为M AR，从而选择RAM中的一个地址。

在步骤5中，IR位6和7选择另一个CPU寄存器。

对于LOAD指令，RAM使能到总线上，总线设置到所选寄存器中。

对于STORE指令，所选寄存器在总线上使能，总线设置为RAM。

这些指令中的每一个只需要两个步骤来完成，步骤6将什么也不做。

以下是我们的计算机语言的两个新词：

语言		我正在学习
LD	Ra、Rb	从RA中的RAM地址加载RB
ST	Ra、Rb	将RB存储到RA中的RAM地址

## 数据指令

下面是一个有趣的说明。

它所做的只是将一个字节从RAM加载到寄存器中，就像上面的LOAD指令一样。

不过，它的不同之处在于，它将从RAM中的何处获取该字节。

在数据指令中，数据来自下一条指令应该位于的位置。

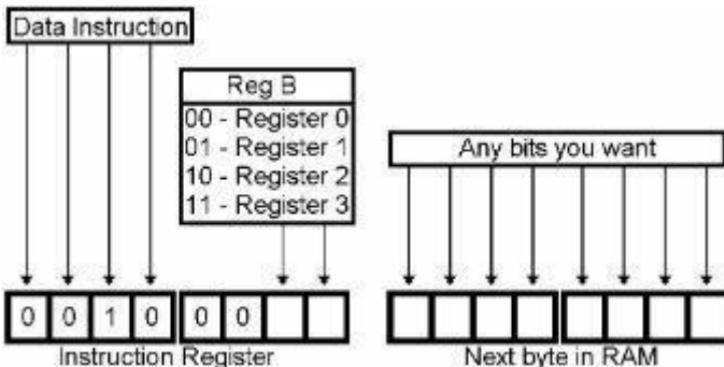
所以您可以考虑这个指令实际上有两个字节长！

第一个字节是指令，下一个字节是一些将被放入寄存器的数据。

这个数据很容易找到，因为当我们在IR中得到指令时，IAR已经被更新了，所以它指向这个字节。

这是数据指令的指令代码。位0至3为0010。未使用位4和位5。

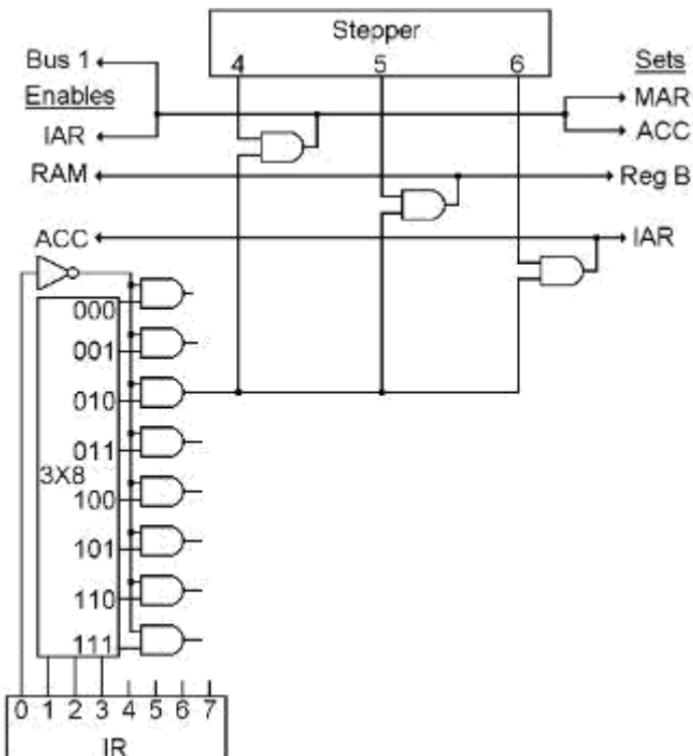
位6和位7选择将载入第二字节数据的寄存器。



该指令所需做的全部工作是，在步骤4中，将IAR发送到M

AR，在步骤5中，将RAM发送到所需的CPU寄存器。然而，还有一件事需要去做。

因为指令的第二个字节只是可能是任何东西的数据，所以我们不想将第二个字节作为指令执行。我们需要第二次将1添加到IAR，以便它跳过这个字节并指向下一条指令。我们将以与步骤1和3中相同的方式执行此操作。在步骤4中，当



我们将IAR发送到MAR，我们将利用ALU同时计算IAR和一些东西的事实，我们将打开“总线1”，并将答案设置为ACC。

步骤5仍然将数据移至寄存器，在步骤6中，我们可以将ACC移至IAR。

下面是我们的计算机语言的另一个新词：	语言
--------------------	----

我正在学习	RB, xxxx xxxx 将这8位加载到RB
-------	-------------------------

## 第二大发明

第一个伟大的发明是在RAM中有一串由CPU逐个执行的指令。

但我们的时钟非常快，RAM的数量有限。

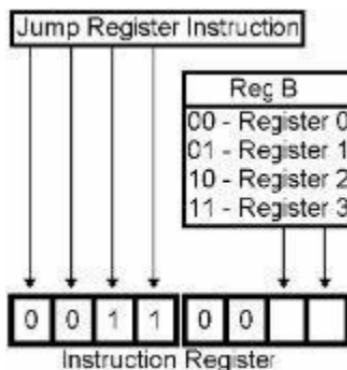
在不到一秒钟的时间内，当我们执行完RAM中的所有指令时，会发生什么呢？

幸运的是，我们不需要回答这个问题，因为有人提出了另一种非常重要的指令，它是允许计算机做它所做的事情所必需的第二个伟大发明。

由于我们的CPU及其控制部分的多功能安排，使其工作是一件非常简单的事情，但它的重要性不应因为这种简单性而丧失。

这种新型指令称为跳转指令，它所做的就是更改IAR的内容，从而更改RAM中下一条指令的位置以及后续指令的来源。

本章中描述的跳转指令的确切类型称为“跳转寄存器”指令。 它只需将Reg B的内容移入IAR。 下面是它的说明代码：



计算机在RAM中一个接一个地执行一系列指令，其中一个指令突然改变了IAR的内容。那会发生什么事呢？将获取的下一条指令将不是最后一条指令之后的指令。

它将位于加载到IAR中的任何RAM地址。

并且它将从该点继续执行下一个跳转指令等，直到它执行另一个跳转指令。

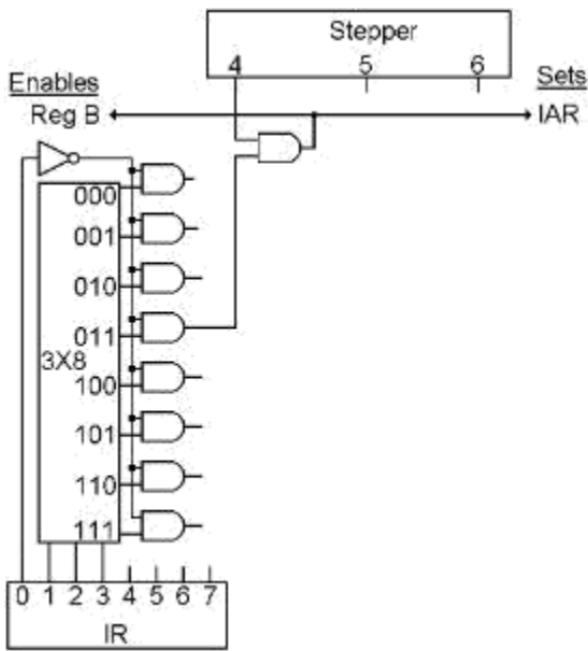
跳变寄存器指令的接线只需一步。

在步骤4中，所选寄存器在总线上使能，并设置为IAR，仅此而已。

如果我们想要加速我们的CPU，我们可以使用步骤5来重置步进器。

但是为了使我们的图表保持简单，我们不会为此而费心。

第5步和第6步不会起任何作用。



下面是我们的计算机语言的另一个新词:

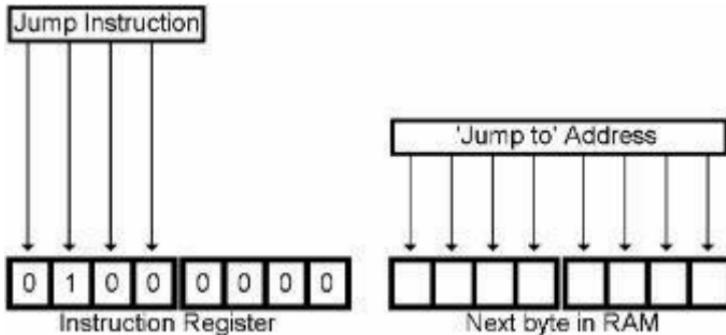
语言学习	
------	--

JM公关	RB跳转到RB中的地址
------	-------------

## 跳跃的另一种方式

这是另一种类型的跳转指令。它类似于数据指令，因为它使用两个字节。

它将IAR替换为紧接在指令字节之后的RAM中的字节，从而改变下一个和后续指令在RAM中的来源。这是它的指令代码。本指令中不使用位4、5、6和7：



这种类型的跳转指令叫做“跳转”，当你知道你想要跳转到的地址，当你在编写程序时，它是很有用的。

当您要跳转到的地址在运行时被计算为程序时，跳转寄存器指令更有用，并且可能并不总是相同的。

使用跳转指令可以做的一件事是创建一个反复执行的指令循环。

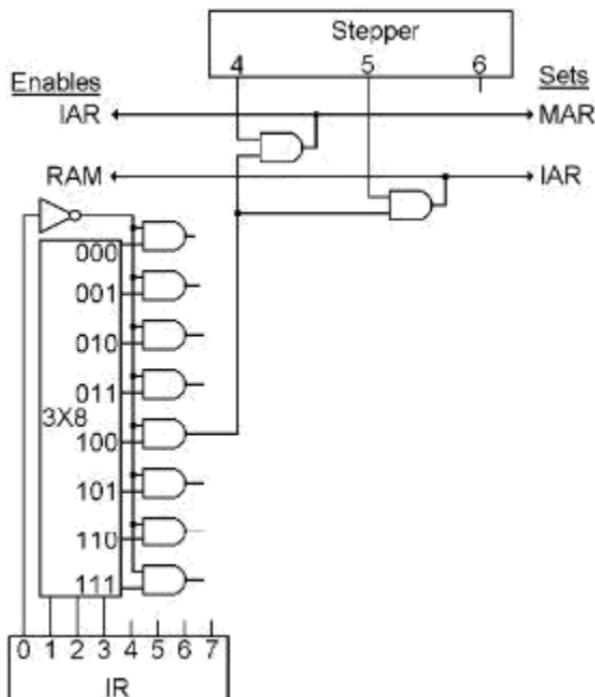
RAM中可以有一系列50条指令，最后一条指令“跳转”回第一条指令。

与数据指令一样，IAR已经指向我们需要的字节。

与数据指令不同，我们不需要第二次将1添加到IAR中，因为无论如何我们都要替换它。所以我们只需要两步。在步骤4中，我们将IAR发送到MAR。

在步骤5中，我们将所选RAM字节移至IAR。第六步什么也做不了。

以下是使其工作的布线：



下面是我们的计算机语言的另一个新词:

语言学习	
------	--

JM P	地址
------	----

跳转到下一个字节中的地址

## 第三大发明

这是第三个，也是最后一个，使计算机成为计算机的发明。

这就像跳转指令，但有时跳转，有时不跳转。

当然，跳跃或不跳跃只是两种可能性，所以只需一位就能确定会发生什么。

最后，我们将在本章中介绍的是这一点的来源。

你还记得从ALU出来，然后再回到ALU的“携带”部分吗？

该位来自加法器或移位器之一。

如果将两个数相加，结果得到大于255的值，则进位位将接通。

如果您左移一个具有左位ON的字节，或者右移一个具有右位ON的字节，这些情况也会打开ALU的CORREOT位。

还有一个位告诉我们ALU的两个输入是否相等，另一个位告诉我们A输入是否更大，

还有一个位告诉我们ALU的输出是否全为零。

这些位是唯一我们还没有在CPU中找到归宿的东西。

这四个位将被称为“标志”位，它们将用于决定是否执行RAM中的下一条指令或跳转到其他地址。

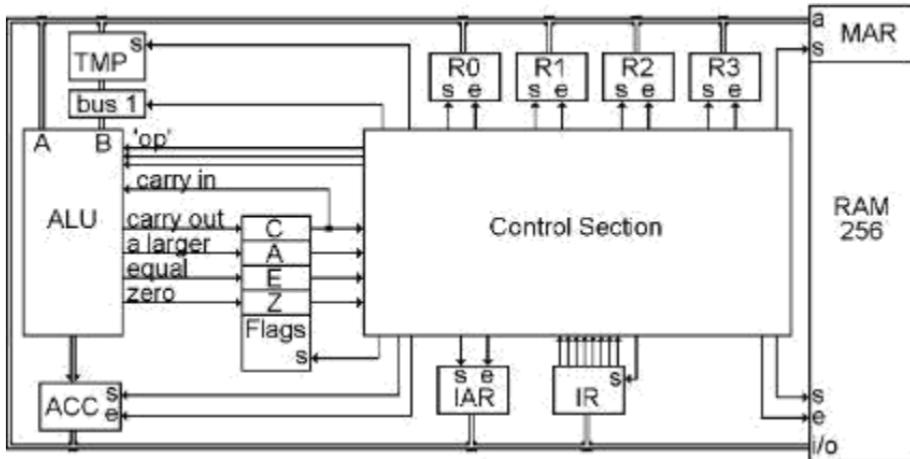
我们试图让计算机能够完成的是，它首先执行一个ALU指令，然后让一个或多个“跳转如果”指令跟随它。“Jump If”将根据ALU指令期间发生的情况跳转或不跳转。

当然，当“Jump If”执行时，ALU指令的结果早就消失了。

如果您返回并查看ALU指令的详细信息，那么只有在步骤5中，所有正确的输入才会进入ALU，所需的答案才会出来。此时，答案被设置为ACC。

所有四个标志位的时序相同，仅在ALU指令的第5步期间有效。

因此，我们需要一种方法将标志位的状态保存为



它们是在ALU指令的第5步中。

这是我们要添加到CPU的最后一个寄存器。

这将称为标志寄存器，我们只使用其中的四位，每个标志一位。

来自ALU的标志位连接到该寄存器的输入，它将在ALU指令的第5步期间设置，就像ACC一样，并且它将保持这样设置，直到下次执行ALU指令。

因此，如果您有一个ALU指令后跟一个“跳转如果”指令，“标志”位可以用来“决定”是否跳转。

每个指令周期都使用步骤1中的ALU将1添加到下一条指令的地址，但只有ALU指令的步骤5具有设置标志的连接。

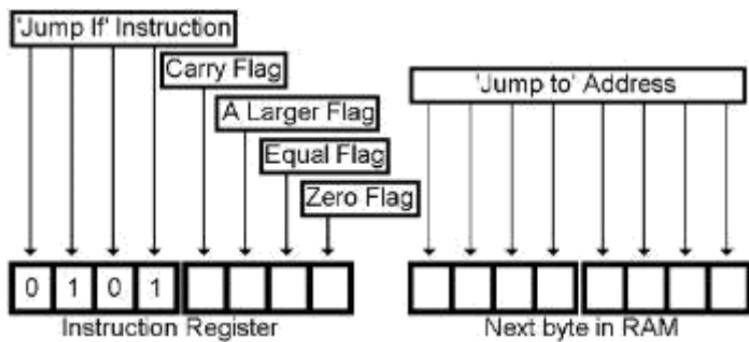
（我们没有在ALU指令的接线中显示此连接，因为我们还没有引入标志REG，但它将出现在已完成的控制部分图中。）

下面是“如果”指令的指令代码。指令的后四位告诉CPU应该检查哪个或哪些标志。在与要测试的标志相对应的指令位中放置“%1”。

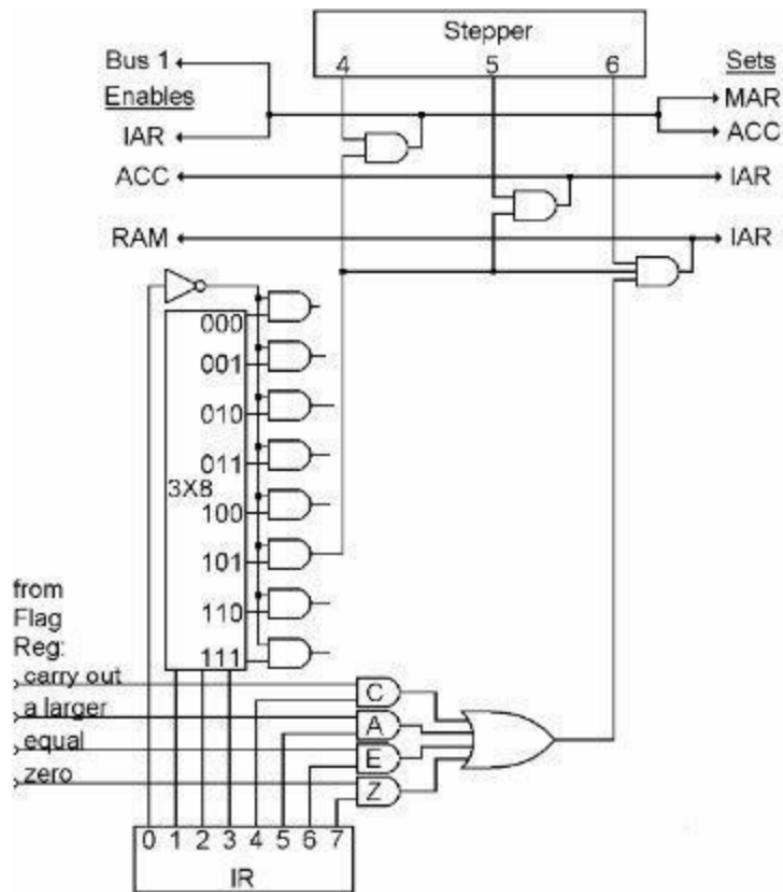
如果您测试的任何一个标志处于打开状态，则跳转将发生。

这种安排给了我们许多方法来决定是否跳跃。

如果执行跳转，则有第二个字节包含要跳转到的地址。



以下是控制部分中的接线，用于执行跳转IF指令。



步骤4将IAR移动到MAR，这样我们就可以准备好获得“跳转到地址”，如果我们跳转，就会使用该“跳转到地址”。

但是因为我们可能不会跳转，所以我们还需要计算RAM中下一条指令的地址。因此步骤4还打开总线1并在ACC中设置答案。

在步骤5中，我们将ACC移至IAR，以便在不跳转的情况下获取下一条指令。

第6步是作出“决定”的地方。

如果AND门的第三个输入为ON，则将指令的第二个字节从RAM移至IAR。

第三个输入来自具有四个输入的“或”门。

这四个输入在与IR中的跳变IF指令的最后四位进行AND运算后，来自四个标志位。

例如，如果指令的“equal”位中有“1”，并且“equal”标志位为ON，则会发生跳变。

这里有更多关于我们的计算机语言的单词。

'J表示跳跃，'C表示进位，'A'表示A更大，'E'表示A等于B，'Z'表示答案都是零。

下面是测试单个标志的语言单词：

语言	我正在学习	
JC	地址	进位为ON时跳转
JA	地址	如果A大于B，则跳转
JE	地址	如果A等于B，则跳转
JZ	地址	如果答案为零，则跳转

您还可以通过在四个位中的一个以上位置1，同时测试多个标志位。

实际上，由于有4位，所以有16种可能的组合，但所有4位都关闭的组合是无效的，因为它永远不会跳转。为了完整起见，下面是其余的可能性：

语言		我正在学习
JCA	地址	进位跳跃或更大
JCE	地址	进位跳跃或A等于B
JCZ	地址	如果进位或答案为零则跳转
宰	地址	如果A大于或等于B，则跳转
贾斯	地址	如果A较大或答案为零，则跳转
耶兹	地址	如果A等于B或答案为零，则跳转
JCAE	地址	如果进位或A大于或等于B，则跳转
JCAZ	地址	进位跳跃、大进位跳跃、零跳跃
JCEZ	地址	如果进位或A等于B或0，则跳转
亚埃兹	地址	如果A大于或等于B或0，则跳转



JCAEZ

|地址

|进位跳跃，一个较大的、相等的或零的

---

## 清除标志指令

有一个恼人的细节，我们需要在这里。

当您进行加法或移位时，您有可能通过该操作打开进位标志。

这是必要的，我们将其用于跳转IF指令，如上一章所述。

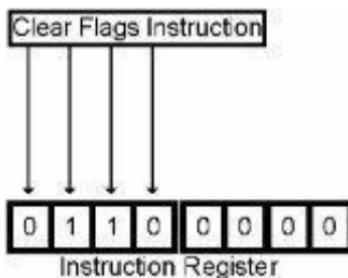
进位标志也用作加法和移位操作的输入。

这样做的目的是为了添加大于255的数字，并将位从一个寄存器移位到另一个寄存器。

出现的问题是，如果您只是添加两个单字节数字，您并不关心任何先前的进位，但是进位标志仍然可以从先前的操作中设置。在这种情况下，您可以添加 $2+2$ 并得到 $5!$

更大的计算机有几种方法来实现这一点，但对于我们来说，我们只需要一个明确的标志指令，您需要使用之前，任何添加或移位，意外的进位将是一个问题。

这是该指令的指令代码。位4、5、6和7不使用。



这方面的布线非常简单，有点棘手。

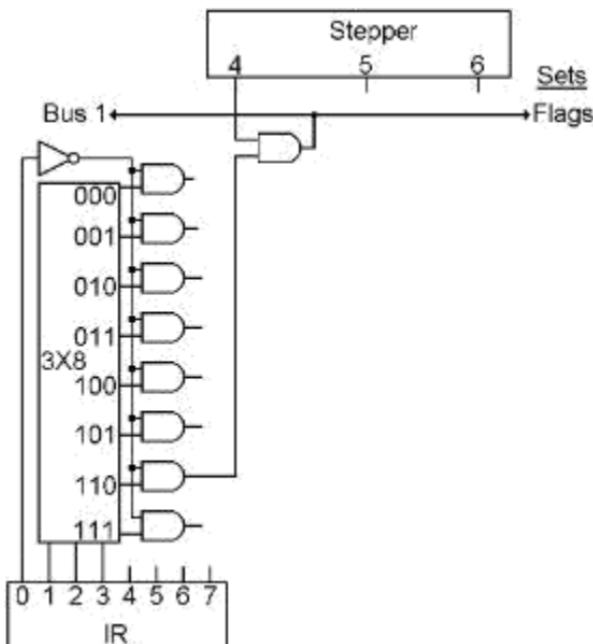
我们不会在总线上启用任何东西，因此它和'A'ALU输入都将为零。

我们将打开“总线1”，因此“B”输入为00000001。

我们不会向ALU发送操作，因此它将处于添加模式。

因此，ALU将相加0和1，并且可能有进位输入。那么答案将是00000001或000010。

但不会有进位输出，答案不是零，B大于A，所以“等于”和“大于



两个都会关掉。此时，当所有四个标志位都关闭时，我们“设置”标志REG。  
这是我们语言的另一个词。

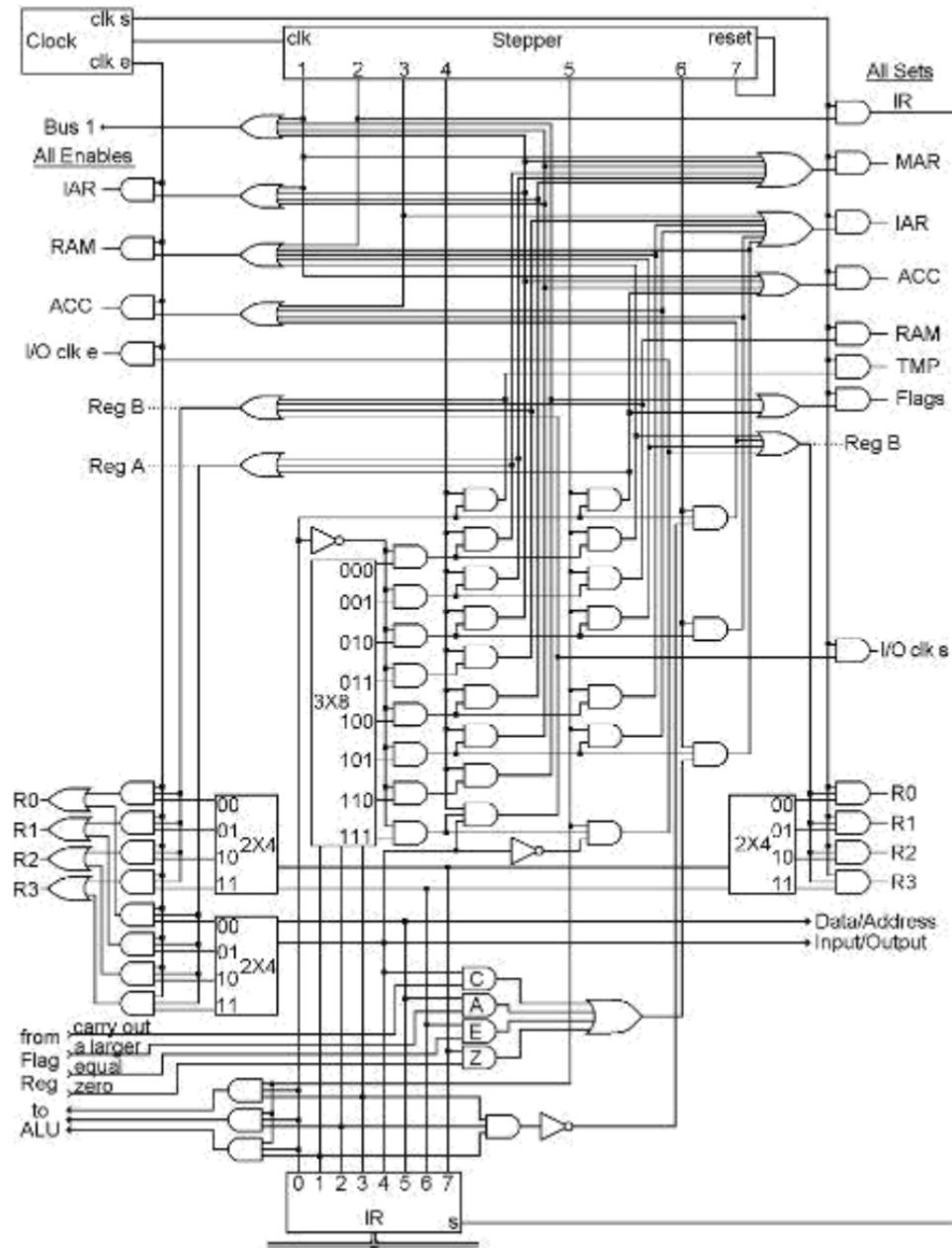
语言学习

CLF

## 清除所有标志

我们现在已经连上了CPU的控制部分。

因此，我们可以在RAM中放置一系列指令，时钟、步进器、指令寄存器和布线将获取并执行这些指令。以下是整个控制部分：



是的，这看起来很复杂，但我们已经看过了它的每一部分。唯一的

我们必须添加一些“或”门，因为大多数“enables”和“sets”需要多个连接。

这实际上比RAM的器件少得多，但重复性要好得多。

这里最糟糕的就是把电线从一个地方接到另一个地方。

放置在指令寄存器中的字节会导致某种活动的发生。

每个可能的模式都会导致不同的活动。

因此，我们有一个代码，其中256个可能的代码中的每一个表示不同的特定活动。

如前所述，这称为指令代码。

它的另一个名称是“机器语言”，因为这是机器（计算机）“能理解的唯一语言（代码）。”你“告诉”机器该做什么，给它一个你想让它执行的命令列表。

但是你必须说它“能理解”的唯一语言，如果你给它输入正确字节大小的ONS和OFF模式，你就可以让它做一些有用的事情。

这里是所有的指令代码和我们的速记语言汇集在一个地方。

指令代码	语言	意义
1000	拉尔布	添加 Ra、Rb 添加
1001	拉尔布	SHR Ra、Rb 右移
1010	拉尔布	SHL Ra、Rb 左移
1011	拉尔布	不是 Ra、Rb 不是
1100	拉尔布	以及 Ra、Rb 以及
1101	拉尔布	或 Ra、Rb 或
1110	拉尔布	异或 Ra、Rb 独占或
1111	拉尔布 《议定书》/ 《公约》缔 约方会议	Ra、Rb 比较
0000	拉尔布	LD Ra、Rb 从RA中的RAM地址加载RB
0001	拉尔布	ST Ra、Rb 将RB存储到RA中的RAM地址
0010	00Rb xxxxxxxx	数据 RB, 数据 将这8位加载到RB中
0011	00RB	JMP R 经常预算 跳转到RB中的地址
0100	0000xxxxxxxx	JMP 地址 跳转到下一个字节中的地址
0101	CAEZ XXXXXXXX	JCAEZ 地址 如果任何测试标志为ON，则跳转
0110	0000	CLF 清除所有标志

信不信由你，你所见过的计算机所做的一切，都是CPU执行一长串指令的结果，比如上面的指令。

## 再谈算术

我们不想在这个问题上花太多的时间，但是到目前为止，我们所看到的唯一看起来像算术的东西是加法器，所以我们将看看稍微复杂一些的算术的简单例子。

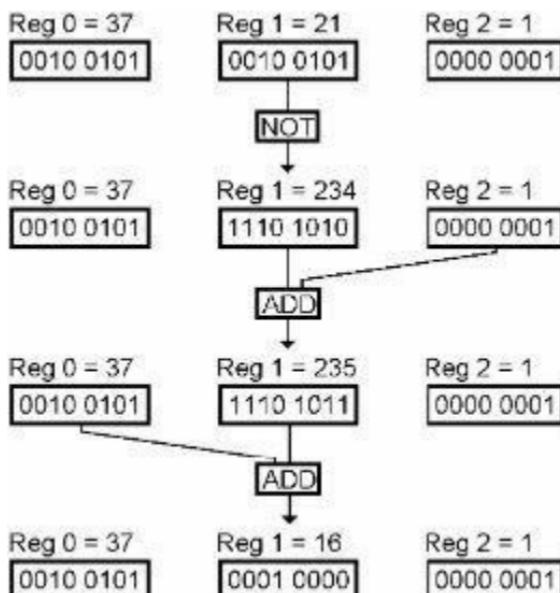
不是教你如何像计算机一样工作，而是向你证明它是可行的。

下面是减法的方法。这是用加法器和非门完成的。

如果要从R0中减去R1，首先不要将R1重新放回自身。

然后将1添加到R1，再将R0添加到R1。

这显示了从37:



最后一步是添加 $37+235$ ，其答案应该是272。但单个寄存器不能容纳大于255的数字。

因此，加法器打开进位，剩下的8位答案是0001 0000，即16，37减21的正确答案。

为什么不加和加会导致减法呢？你为什么要在零之后加上1？为什么忽略进位位？

我们不打算试图在这本书中回答这些问题中的任何一个。

这些细节使极少数工程师无法睡个好觉。

这些勇敢的人研究这些问题，设计出普通人不必理解的方法。

下面是乘法的方法。

当我们用铅笔和纸在十进制系统中进行乘法运算时，你必须记住乘法表，你知道，3乘8等于24，6乘9等于54，等等。

在二进制中，乘法实际上比十进制容易得多。

1乘以1等于1，对于其他组合，答案是0！再简单不过了！

这里有一个用铅笔和纸乘以5的例子。

$$\begin{array}{r} 00000101 \\ \times 00000101 \\ \hline 00000101 \\ 00000000 \\ 00000101 \\ 00000000 \\ 00000000 \\ 00000000 \\ 00000000 \\ 00000000 \\ \hline 00000000011001 \end{array}$$

如果你看看这里发生了什么，如果底部数字的正确数字是1，你把顶部的数字放在答案中。

然后，对于左边的每个数字，将顶端的数字向左移动，如果底部的数字是1，将移动的顶端的数字加到答案上。当你通过底部数字的8位时，你就完成了。

因此乘法由加法器和移位器完成。就这么简单。您可以编写这样一个简单的程序：

R0包含底部数字，R1包含顶部数字，R2包含答案。

R3用于在经过8次循环后跳出循环。

RAM地址	指令	评论	
50	数据	R3, 00 00 0001	*将“%1”放入R3
52	异或	R2、R2	*将“0”放入R2
53	CLF		*清除标志
54	SHR	r0	*一位进位标志
55	JC	59	*添加
57	JM P	61	*跳过添加
59	CLF		*清除标志
60	添加	R1、R2	*添加此行
61	CLF		*清除标志



62	SHL	r1	*M ULT Top By 2
63	SHL	R3	*移位计数器
64	JC	68	*完成后退出
66	JM P	53	*执行下一步
68	(程序中下一条指令)		

请查看当此程序前三次循环时，寄存器会发生什么变化。

r0

r1

R2

R3

开始时（52岁以后）：

0000 0101 0000 0101 0000 0000 0000 0001

第一次（63岁以后）：

0000 0010 0000 1010 0000 0101 0000 0010

第二次（63岁以后）：

0000 0001 0001 0100 0000 0101 0000 0100

第三次（63岁以后）：

0000 0000 0010 1000 0001 1001 0000 1000

这里发生的重要事情是，R1已经两次添加到R2中。

它发生在第一次通过时，R1包含00000101，第三次通过时，R1已经左移两次，因此包含00010100。R2现在包含0001

1001二进制数，即 $16+8+1$ ，或25十进制数，这是5乘以5的正确答案。

循环将再重复5次，直到R3中的位移出到进位标志，但总数不会增加，因为有N r0中没有更多的1。

这个项目将进行八次。我们从R3中的00000001开始。在程序快结束时，R3向左移动。前七次通过时，将没有进位，因此程序将到达'JM P 53'并返回到程序的第三条指令。

第八次R3左移时，开启的一位从R3移出并进入进位标志。因此，“JC 68”将跳过“JM P 53”，并继续执行此后的任何指令。

R0中的字节右移，以测试哪些位为ON。R1中的字节向左移位，将其乘以2。

当R0中有一个位时，将R1添加到R2中。这就是所有的一切。

在本例中，我们没有讨论的一件事是，如果乘法的答案大于255，会发生什么情况。  
如果乘法程序将两个一字节相乘  
数字，它应该能够处理一个两字节的答案。  
这将照顾到任何两个数字，你可能会开始。  
这将通过进位标志和一些更多的跳转如果指令来实现。 我们不会用细节来折磨读者。

阅读上面的程序与阅读我们在书中看到的图表是完全不同的技巧。  
我希望你能理解，但没有人会因为这本书而成为阅读程序的专家。  
计算机也有处理负数和小数点的方法。  
这些细节非常枯燥，研究它们并不能提高我们对计算机工作原理的理解。  
归根结底还是南德·盖茨。 我们的简单计算机可以用程序做所有这些事情。

## 外面的世界

到目前为止，我们所描述的是整个计算机。 它由RAM和CPU两部分组成。 就这些了。 这些简单的操作是计算机能做的最复杂的事情。

执行指令、用ALU修改字节、从程序的一个部分跳转到另一个部分的能力，最重要的是，根据计算结果跳转或不跳转的能力。 这是计算机能做的。

这些都是简单的事情，但是由于它的操作方式是q

令人欣慰的是，它可以做大量的这些操作，可以导致一些看起来令人印象深刻的东西。

这两个部分使它成为一台计算机，但如果计算机所能做的就是运行一个程序并重新排列RAM中的字节，那么没有人会知道它在做什么。

因此，还有一件事，计算机需要是有用的，这是一种与外部世界沟通的方式。

处理计算机之外的任何事情都被称为“输入/输出”或简称为“I/O”。

输出是指从计算机输出的数据； 输入是指输入计算机的数据。

有些东西只输入，比如键盘，有些东西只输出，比如显示屏，有些东西既输入又输出，比如磁盘。

I/O所需的只是几条线和一条新指令。

对于这些线，我们要做的就是将CPU总线扩展到计算机外部，并添加另外四条线。

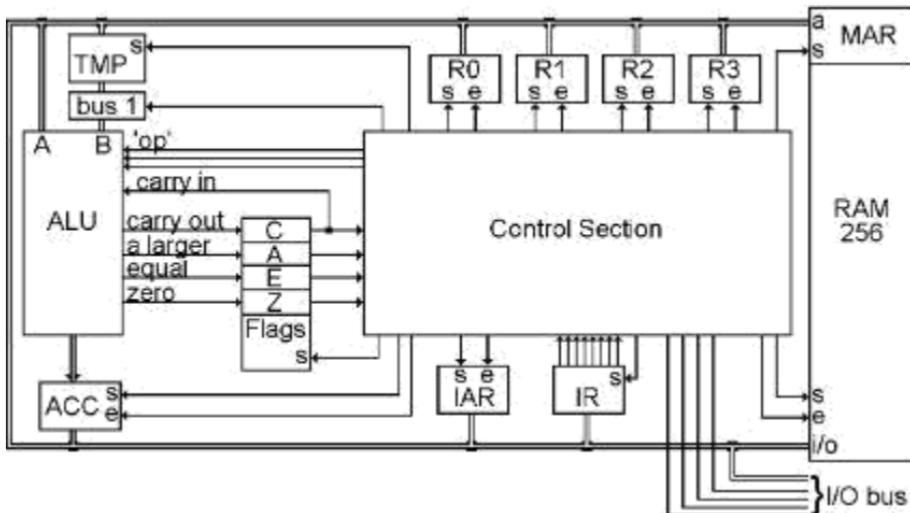
这12条线的组合称为I/O总线。 连接到计算机的所有东西都连接到这个I/O总线上。

连接到I/O总线的设备被称为“外围设备”，因为它们不在计算机内部，而是在计算机外部，在计算机外围设备上（计算机周围的区域）。

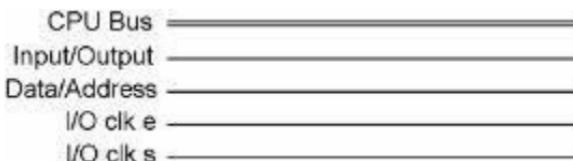
很多东西都可以连接到I/O总线上，但是计算机控制这个过程，一次只有一个东西是活动的。

连接到I/O总线的每件东西都必须有自己独特的I/O地址。

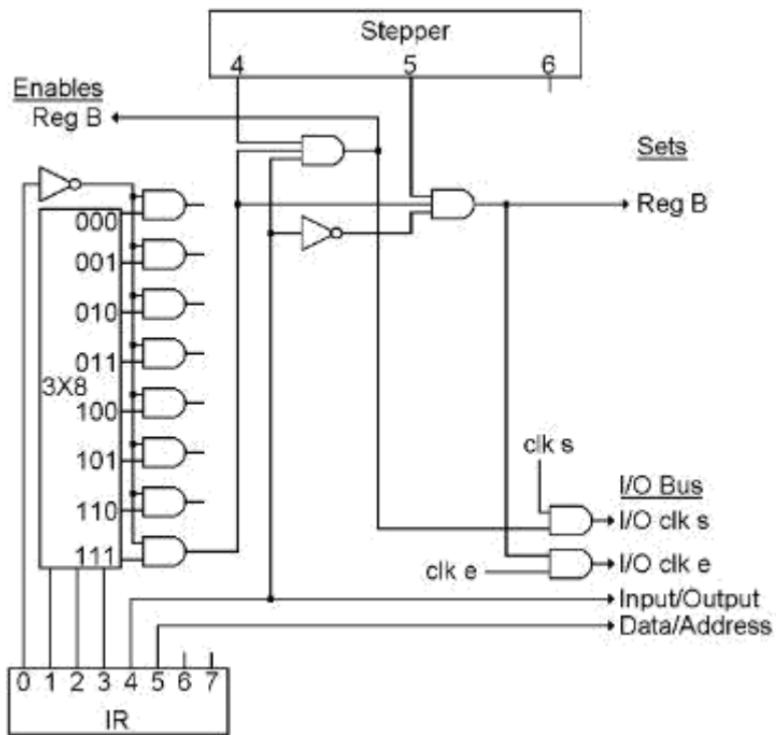
这与RAM中字节的地址不同，外围设备放在总线上时只会识别一些“数字”。



这是I/O总线在CPU中的样子，在图的右下角。

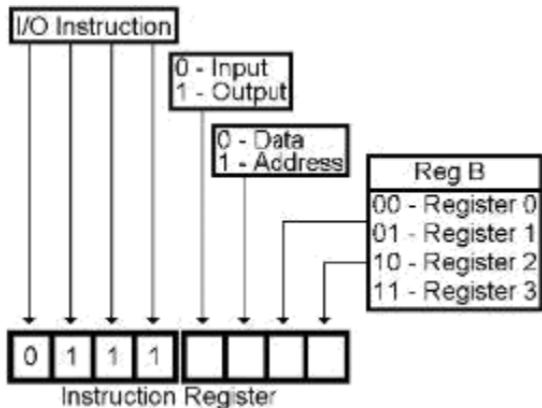


下图中是I/O总线的连线。CPU总线与其他任何地方的CPU总线都是相同的八线束。“输入/输出”线确定CPU总线上的数据将朝哪个方向移动（输入或输出）。“data/address”线告诉我们是要传输一个字节的数据，还是要传输一个I/O地址，该地址从许多可以连接到I/O总线的设备中选择一个。“I/O CLK E”和“I/O CLK S”用于使能和设置寄存器，以便可以将字节移回，



四号。

以下是I/O指令的指令代码：



这一条指令可以根据IR位4和5以四种不同的方式使用，因此我们的语言有四个新词。

语言		我正在学习
在	数据， R B	将I/O数据输入RB
在	地址， 经常预算	输入I/O地址到RB
出局	数据， R B	将RB作为数据输出到I/O
出局	地址， 经常预算	将RB输出到I/O作为地址

每个I/O设备都有其独特的特性，因此需要独特的器件和布线将其连接到I/O总线。将设备连接到总线的部件集合称为“设备适配器”。每种类型的适配器都有一个特定的名称，例如“键盘适配器”或“磁盘适配器”。

适配器不执行任何操作，除非其地址出现在总线上。

当它这样做时，适配器将响应计算机发送给它的命令。

使用“out

addr”指令，计算机打开地址线，并将要与之通话的设备的地址放在CPU总线上。

外围设备识别其地址并激活。

每隔一个外围设备都有其他地址，所以它们不会响应。

我们不打算描述I/O系统中的每个门。

此时，您应该相信可以通过总线通过几根控制线传输字节的信息。

本章的信息仅仅是I/O系统的简单性。CPU和RAM是计算机。

所有其他的东西，磁盘，打印机，键盘，鼠标，显示屏，发声的东西，连接到互联网的东西，所有这些东西都是外围设备，他们所能做的就是交流

从计算机接收数据字节，或

向计算机发送字节数据。

不同设备的适配器具有不同的功能、不同数量的寄存器，并且对于CPU中运行的程序必须执行哪些操作才能正确操作设备有不同的要求。

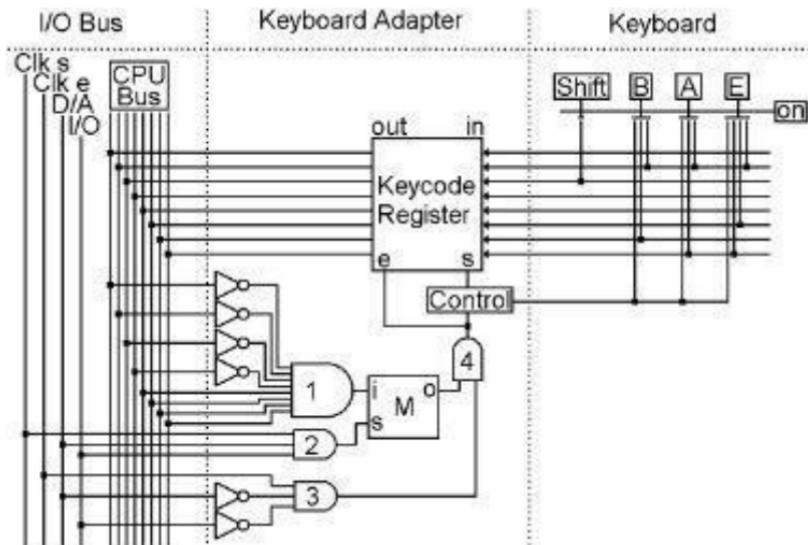
但他们不会做比这更奇特的事。

计算机用CPU执行的非常少的简单I/O命令来控制进程。

键盘

键盘是连接到I/O总线的最简单外设之一。

它是一个只提供输入的设备，一次只向CPU显示一个字节。



键盘里面有八根电线，右边是它自己的小总线。

当您按下一个键时，它只需将电连接到创建与按下的键相对应的ASCII代码所需的电线。

当按下一个键时，这个写着“Control”的小框也会得到通知，并将ASCII代码设置到键码寄存器中。

按下键后，在键码寄存器中会有一个ASCII码等待。

下面是CPU如何将代码放入其中一个寄存器中。

AND门#1有8个输入。它们连接到CPU总线，其中四个通过NOT门。

因此，当总线包含00001111时，AND门将打开。这是此键盘适配器的I/O地址。

AND门#3在IN数据指令的“CLK E”时间开启。如果M  
Emory位为ON，则门#4将接通，并在总线上使能键码寄存器，该寄存器将设置为CPU  
中的寄存器B。

每个连接到I/O总线的适配器都需要具有我们在1号和2号门中看到的电路类型以及上面的存储器位。每个适配器将具有打开门#1的不同组合；  
这就是允许CPU单独选择每个适配器的原因。

下面是一个小程序，它将当前的按键移动到CPU的reg3中。

指令		评论
数据	R2, 0000 1111	*将键盘地址放入注册表2
出局	地址, R2	*选择键盘
在	数据, R3	*按下ASCII键
异或	R2、R2	*清除注册表2中的地址
出局	地址, R2	*取消选择键盘

这个小“控制”框在键码寄存器被发送到CPU后清除它。

运行在CPU中的程序将定期检查键盘适配器，如果接收到的字节都是零，则没有按下任何键。

如果字节上有一个或多个位，则程序将执行程序当时设计为与击键有关的任何操作。  
同样，我们不会通过键盘适配器中的每个门。

所有设备适配器都具有相同种类的电路，以便能够在寻址时做出响应，并根据需要发送或接收字节信息。但这并不比这更复杂。这就是I/O设备和适配器所做的全部工作。

## 显示屏

电视和电脑显示屏的工作原理是一样的，它们之间的主要区别只是显示什么。这实际上不是计算机技术，因为你不需要显示屏就能拥有一台计算机，但大多数计算机都有一个屏幕，而且计算机花了很多时间使屏幕看起来像某种东西，所以我们需要了解一点它是如何工作的。

电视似乎给了你有声音的动态图像。

图片和声音是分开进行的，在这一章中，我们只关注图片是如何工作的。

首先要知道的是，尽管图片看起来在移动，但实际上是一系列呈现得如此之快以至于眼睛都没有注意到的静止图片。你可能已经知道了，但接下来的事情是。

你看过电影。这是一系列的图片。

要看电影，你把胶卷放进放映机里，放映机通过一张图片照射光，然后把胶卷移到下一张图片，通过它照射光，等等。它通常每秒运行24张图片，其中CH的速度足以给人一种不断移动的画面的错觉。

电视的速度要快一些，大约每秒30张图片，但是电影和电视之间还有一个更大的区别。有了电影胶卷，每张静止的照片都同时放映。

每幅画都是完整的，当你透过它照射光时，画面的每一部分都会同时出现在屏幕上。电视不能做到这一点。它没有一个完整的图片放在屏幕上所有在一次。

电视在一个瞬间所能做的就是点亮屏幕上的一个点。

它点亮一个点，然后点亮另一个点，然后再点亮另一个点，非常快，直到一整幅画中的点都被点亮。

整个屏幕上的点组成了一幅静止的图片，因此它必须在三十分之一秒内点亮所有的点，然后用下一幅图片重新点亮所有的点，等等，直到它在一秒内将30幅图片上的点放到屏幕上。所以电视我

It'我们非常忙，每秒点亮单个的点，是屏幕上点数的30倍。

通常，左上角的点首先被点亮，然后右上角的点被点亮，依此类推穿过屏幕的顶部到达右上角。然后从第二行点开始，再次穿过屏幕、第三行等，直到扫描完整个屏幕。每个点的亮度是高的或低的，因此屏幕上的每个部分都被点亮到适当的亮度，以使屏幕看起来像预期的图像。在任何一个时刻，电视只处理屏幕上的一个单独的点。因此，电视有两种错觉——

一种是来自一系列静止图像的运动错觉，另一种是实际绘制的完整静止图像的错觉——一次一个点。

这第二种错觉是由屏幕的组成，每一个点只会点亮一小部分秒，然后它就会立即消失。

幸运的是，无论屏幕是由什么组成的，在点亮点到1/30之间，它都会继续发光

第次

一秒钟后同样的点又亮了。

对眼睛来说，你只是看到了一幅动人的图画，但是有很多事情使它看起来是那样的。

在计算机中，屏幕上的一个点被称为“像素”，简称“像素”。

电脑屏幕就像电视一样工作。

它们还必须每秒扫描整个屏幕30次，以照亮每个单独的像素，从而使图像出现。

即使屏幕上的内容没有改变，计算机中的某些东西也必须每秒扫描30次屏幕上未改变的图像。不扫描，不拍照-这就是它的基本原理。

我们不打算在这里详细讨论CPU和RAM的工作原理，这两种技术使它成为一台计算机，但如果想知道计算机如何将我们可以阅读的内容放到屏幕上，我们需要对它的工作原理有一个基本的了解。

在本章中，我们将研究最简单的屏幕类型，即黑白屏幕，其像素只能完全打开或完全关闭。这种类型的屏幕可以显示字符和由线条画构成的图片类型。

在本书的后面，我们将看到几个简单的变化，使屏幕显示的东西，如彩色照片。

主要有三个部分。首先是电脑，我们已经看到它是如何工作的。

它有一个I/O总线，可以将字节移入和移出计算机外部的东西。第二个是屏幕。

屏幕只是一个很大的像素网格，每个像素都可以被选择，一次一个，当被选择时，可以打开，也可以不打开。

第三项是“显示适配器”，显示适配器一端连接到I/O总线，另一端连接到屏幕。

例如，让我们使用一个旧类型的屏幕。

它是一个黑白屏幕，在屏幕上显示320个像素，在屏幕下显示200个像素。

总共64,000个像素

屏幕。

屏幕上的每个像素都有一个由两个数字组成的唯一地址，第一个数字是左右或水平位置，另一个是上下或垂直位置。左上像素的地址是0, 0, 右下像素的地址是319,199  
64,000像素乘以每秒30幅图片意味着此显示适配器的时钟需要每秒滴答1,920,000次。  
由于一个字节中有8位，我们将需要8,000字节的显示RAM来告诉64,000个屏幕像素中的每一个是开启还是关闭。

显示适配器具有设置当前像素的水平位置的寄存器。

显示适配器在时钟的每一个滴答声中将1添加到该寄存器。

它从零开始，当其中的数字达到319时，下一步将其重置为零。

所以它一遍又一遍地从零到319。还有一个寄存器，用于设置当前像素的垂直位置。

每次水平寄存器复位到零，显示适配器都会将1添加到垂直寄存器。什么时候

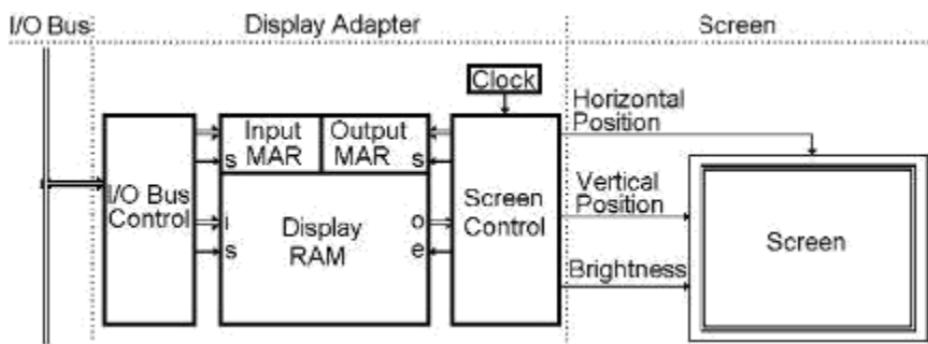
垂直寄存器达到199，下一步将其复位为0。因此，当水平寄存器从0变为319

200次时，垂直寄存器从0变为199次。

当前选定的屏幕像素由这些寄存器控制，因此当水平寄存器从0变为319时，当前像素穿过屏幕一次。

然后垂直寄存器中添加一个，当前像素向下移动到下一行的第一个像素。

因此，时钟以及水平和垂直寄存器一次一个地选择屏幕上的每个像素，从左到右一行，然后向下选择下一行中的每个像素，然后选择下一行等，直到屏幕上的每个像素都被一次选择为止。然后一切又重新开始。



同时，还有一个寄存器包含一个显示RAM地址。

虽然每8个像素只需要一个新字节，但该寄存器也会逐步通过。

每个字节的比特一次一个，以八个连续的像素发送到屏幕，以打开或关闭它们。

每隔8个像素后，RAM地址寄存器就会添加1。

当所有像素都已被逐个通过时，整个RAM也已被逐个通过，并且一幅图像已被显示。当水平寄存器和垂直寄存器都达到最大值并复位为0时，RAM地址也复位为0。显示适配器将大部分时间花在绘制屏幕上。

它还必须做的唯一一件事是接受来自I/O总线的命令，这些命令将更改显示适配器RAM的内容。当CPU中运行的程序需要更改屏幕上的内容时，它将使用I/O OUT命令选择显示适配器，然后发送一个显示适配器RAM地址，然后发送一个字节的数据存储在该地址。然后，当适配器继续重新绘制屏幕时，新数据将在适当的位置将梨放在屏幕上。

显示适配器RAM的构建方式与计算机中的RAM不同。它将输入和输出函数分开。所有存储位置的输入连接到输入总线，所有存储位置的输出连接到输出总线，但输入总线和输出总线保持分离。

然后有两个独立的存储器地址寄存器，一个用于输入，另一个用于输出。输入mar有一个网格，该网格仅选择将“设置”哪个字节，而输出mar

有一个单独的网格，它只选择要“启用”的字节。

有一个解决方案，它涉及到...

## 另一个代码

当您想要打印或显示书面语言时，您需要将ASCII代码翻译成活人可读的内容。我们有一个代码01000101，它出现在字母“E”旁边的ASCII代码表中，但是计算机如何将01000101转换为可读的“E”呢？

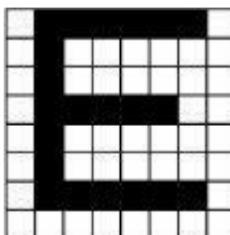
我们有一个显示屏，但是显示屏只是一个像素网格，到目前为止我们所描述的任何东西都没有人类可读的E。

为了在屏幕上显示一个“e”，必须有某种东西使这个形状被我们识别为字母表中的一个字母。

因此，我们需要另一个代码。这段代码实际上是关于由点组成的小图片。

对于我们希望能够在屏幕上绘制的每个字符，我们需要一张该字符的小图片。

如果您采用8像素宽、8像素高的网格，您可以决定哪些像素必须在屏幕上才能生成与要在屏幕上绘制的字符类似的小图片，如下所示：



如果您将此图片转换为开和关，则可以将其存储在8个字节中。

如果你想要在屏幕上显示100个不同的字符，那么你需要100个不同的像这样的小图片，它需要800字节的RAM来存储它。

我们的小电脑只有一个256字节的RAM，所以这将是一个很好的时间去想象我们前面描述的那个更大的版本。

这800字节是一种称为“字体”的代码类型。

如果要使字符出现在屏幕上的某个位置，则需要从字体中选择正确的小图片，然后使用I/O指令将图片的八个字节复制到显示适配器RAM中的适当字节。

如果我们字体中的图片按照与ASCII代码表相同的顺序排列，那么我们可以使用ASCII代码的数值来查找字体中相应的图片。“E”的ASCII代码是0100 0101。

如果将二进制数代码应用于相同的1和0模式，则得到十进制数69。

那么，“E”是ASCII中的第69个代码，而“E”的图片将是字体中的第69个图片。

由于每张图片中有8个字节，所以将69b相乘。

Y8，这就告诉你“e”的图片将是  
从地址552开始的八个字节。

RAM Address	Byte Contents	Display RAM Address	Screen Pixels	Screen
552	01111110	000		
553	01000000	040		
554	01000000	080		
555	01111100	120		
556	01000000	160		
557	01000000	200		
558	01111110	240		
559	00000000	280		

现在，我们需要知道将这些字节复制到显示RAM中的何处。

假设我们希望在屏幕的最左上角显示一个“e”。打开我们感兴趣的像素的位在哪里？第一行很简单，它是显示RAM的前八位，地址0。

因此，我们使用一系列OUT指令来复制RAM地址552以显示RAM地址0。

现在，显示RAM中的第二行在哪里？在我之前，显示器将绘制顶部行的所有320位T向下移动到第二行。这意味着它在每行使用40个字节，所以最上面的行使用0-39个字节。

这意味着在RAM地址553处的“e”的图片的第二字节需要在显示RAM中的地址40处写入。类似地，第三到第八字节在字节80、120、160、200、240和280处被写入。完成所有这些操作后，您将在屏幕上看到一个完整的“E”。

如果你想在屏幕上写一个“X”就在“E”旁边，你可以找到

E“X”字体中的8个字节，并将它们复制到显示RAM字节1、41、81、121、161、201、241和281中。

如果屏幕上需要27'E'，只需将字体中的'E'复制到显示RAM中的27个不同位置即可。

当然，仅仅为了让一个字母出现在屏幕上，这似乎需要做很多工作。

执行此操作的程序将需要一个指令循环，该指令循环计算第一个“from”和“to”地址，然后发出相应的out指令将第一个字节复制到显示RAM。

然后循环将重复，每次更新两个地址，直到所有八个字节都被复制到适当的位置。

我们不打算写这个程序，但它可以很容易地是一个50指令

在完成之前必须循环8次的程序。

这意味着，仅仅在屏幕上显示一个字符就需要400个指令周期！

如果在屏幕上绘制1000个字符，则可能需要400,000个指令周期。

另一方面，这仍然只是这台计算机在一秒钟内所能做的大约四分之一。

这只是向你展示了为什么计算机需要这么快。

他们所做的每一件事都很小，要完成任何事情都需要大量的步骤。

关于代码的最后一句话

我们在计算机中看到了几个使用的代码。每一个都是为一个特定的目的而设计的。

单独的编码消息以字节为单位，移动并用于完成任务。

字节不“知道”使用了哪些代码来选择它们包含的模式。

字节本身中没有任何内容告诉您它应该是哪种代码。

计算机的某些部分是用各种代码构建的。

在ALU中，加法器和比较器被构建为将字节视为包含用二进制数编码的值。

存储器地址寄存器和指令地址寄存器也是如此。

指令寄存器被构建为将其内容视为包含用指令代码编码的值。

字节是哑的。它们只包含开关模式。如果一个字节包含0100

0101，并且您将它发送到打印机，它将打印字母“E”。如果您将它发送到指令寄存器，计算机将执行跳转指令。如果将其发送到M

Emory地址寄存器，它将选择RAM的字节数69。

如果你把它送到加法器的一边，它会把69加到加法器的另一边。如果您将其发送到

显示屏上，它将设置三个像素开和五个像素关。

计算机的每一个部件都是用一个代码设计的，但是一旦它被构建，头脑就消失了，甚至代码也消失了。它只是做了它本来要做的事。

可以在计算机中发明和使用的代码没有限制。 程序员总是发明新的代码。

就像前面提到的快餐店的收银机一样，那台机器里的某个地方有一点意思是“包括炸薯条”。

## 磁盘

大多数计算机都有磁盘。这只是连接到I/O总线的另一个外设。

磁盘的任务非常简单；它可以做两件事。

您可以发送它将要存储的字节，也可以告诉它发送回以前存储的一些字节。

大多数计算机都有磁盘有两个原因。

首先，它们能够存储大量字节，比计算机的RAM大很多倍。

CPU只能执行RAM中的程序，只能操作RAM中的字节。

但是永远没有足够的内存来存储您可能想要用计算机做的所有事情。

因此，磁盘将存储所有内容，当您想要执行一项操作时，磁盘上用于该操作的字节将被复制到RAM AN中

d已使用。

然后，当您想要执行不同的操作时，新活动的字节将从磁盘复制到用于第一个活动的相同RAM区域中。

计算机有磁盘的第二个原因是，当您关闭电源时，存储在磁盘上的字节不会消失。

当您关闭计算机时，RAM会丢失其设置，当您重新打开计算机时，所有字节都是0000 0000，但磁盘会保留在其上已写入的所有内容。

计算机位被定义为有电或没有电的地方。

但在此之前，我们把它定义为一个可以处于两种不同状态之一的地方。

在磁盘上，电位被转换成磁盘表面上以某种方式磁化的位置。

由于磁体有南北两极，圆盘上的点可以南北磁化或南北磁化。

一个方向表示0，另一个方向表示1。

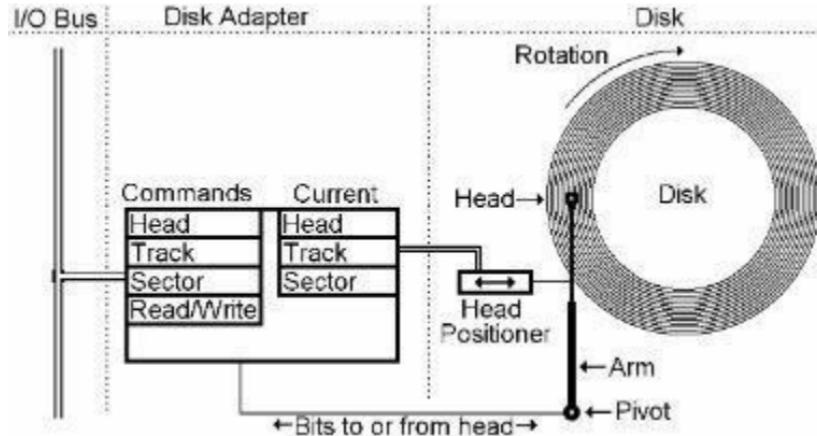
一旦一个光斑被磁化，它就会保持这种状态，除非同一个光斑被磁化。

关闭电源对磁化点没有影响。

顾名思义，磁盘是一个圆形的东西，它旋转得很快。 它涂有一种容易磁化的材料。  
你还记得电报吗？ 在接收端，有一块金属，金属周围缠绕着一根电线。  
当电流通过电线时，那块金属变成了磁铁。

磁盘有一个很小的版本，叫做“头”安装在一个手臂上。

该臂将头部保持得非常靠近旋转盘的表面，并且该臂可以向后摆动并且



第四，使磁头可以到达磁盘表面上的任何一点。

如果你把电流通过磁头，它可以磁化磁盘表面。而且，它的工作原理恰恰相反；  
当磁头经过一个磁化区域时，它会使缠绕在磁头周围的电线中出现电流。

因此，磁头可以在磁盘上写入或读取先前在磁盘上写入的内容。

字节的位被一个接一个地写入磁盘表面。

圆盘的表面被分成一系列的圆环，称为磁道，彼此非常接近。

头部可以在表面上移动，并停留在任何一条轨道上。

每个圆形轨道通常被分成称为扇形的短段。

由于磁盘有两面，通常两面都涂有磁性材料，每面都有一个磁头。

在RAM中，每个字节都有自己的地址。

在磁盘上，也有一种方法来定位字节，但它是非常不同的。

您必须指定字节块所在的头、跟踪和扇区。

这是磁盘上的数据具有的“地址”类型，如“头0，磁道57，扇区15”。在该地址，不仅有一个字节，而且有一个字节块，通常有数千个字节。

对于我们书中的示例，由于RAM太小，我们将讨论一个存储

选择磁头的命令可以立即完成，但是当它获得选择磁道的命令时，它必须将磁头移动到该磁道，这在指令周期方面需要很长的时间。

当它获得选择扇区的命令时，它必须等待该扇区

旋转到头部所在的位置，这也需要很长的教学周期。

当CPU已经确定磁头已经到达期望的磁道和扇区时，可以执行用于读取或写入的I/O命令，并且一次一个字节将通过I/O总线传输。

读取或写入字节块的程序必须继续该过程，直到整个字节块完成为止。

在我们简单的I/O系统中，单个字节在磁盘和

CPU寄存器。

正在运行的程序必须将这些字节移入RAM或从RAM移出，通常是在连续的位置。

这就是磁盘的全部功能。

你可能用过一台有磁盘的电脑，不需要知道任何关于磁头、磁道和扇区的信息。

这是一件好事，因为在这种详细程度上处理磁盘是很烦人的。

我们将在本书后面介绍磁盘的一般用法。

另一种语言注释：有几个词的意思几乎是一样的，但出于某种原因，某些词与某些技术搭配使用。

如果你想给某人寄一封信，首先你把它写在一张纸上，然后当收信人收到信时，他会读它。

在磁带录音机的时代，你会从一盘空白磁带开始。

然后你会在录音带上录一些音乐。

当你想再听一遍音乐的时候，你就会播放录音带。

当涉及到计算机磁盘时，把东西放在磁盘上被称为写。从磁盘上取东西叫做读取。

将某物放入RAM中称为写入或存储。从RAM中获取内容称为读取或检索。

将某物放入CPU寄存器通常称为加载。

把音乐放到磁盘上有时叫做录音，有时叫做刻录。

收听磁盘通常仍称为播放，但如果正在将其复制到计算机上，则称为翻录。

**写、录、存、装、烧都是一样的意思。  
阅读、检索、播放和撕裂也非常相似。  
它们的意思是一样的，只是文字上的不同。**

对不起，夫人

还有一件事，大多数计算机有作为其输入/输出系统的一部分。

一台计算机不需要其中的一台被称为计算机，所以我们将不会通过构建它所需的每一扇门。但这是一件很常见的事情，所以我们将描述它是如何工作的。

你知道，如果毛姆在厨房里搅拌一壶汤，小乔伊跑进来说：“我想要一杯牛奶，”毛姆会放下勺子，走到橱柜前，拿起杯子，到冰箱前，倒牛奶，递给乔伊，然后她会回到炉子边，拿起勺子，继续搅拌汤。

汤的搅动被一杯牛奶打断了，然后汤的搅动又恢复了。

大多数计算机都有一个“中断”，它的工作原理与Mom和Joey的情况非常相似。

中断从多加一条线到I/O总线开始。 某些设备适配器使用这条线让CPU知道现在是CPU执行I/O操作的好时机，就像有人按键盘上的键之后一样。 当设备适配器打开中断位时，下一次步进器返回步骤1时，下一个指令周期将不执行通常的读取和执行操作，而是执行以下操作：	1。  2。  3。  4。  5。  6。
将二进制0移动到mar	2。
将IAR移至RAM	3。
将二进制1移动到mar	4。
将标志移动到RAM	5。
将二进制2移动到mar	6。

将RAM移至IAR	7。
将二进制3移动到m ar	8。

## 将RAM移至标志

此序列的结果是，当前IAR和标志保存到RAM地址0和1，并用RAM字节地址2和3的内容替换。然后CPU返回到正常的获取和执行操作。但是IAR已经被替换了！因此下一条指令将从RAM字节2中的任何地址获取。

换句话说，CPU一直在做的事情被保存，CPU被发送去做其他事情。

如果在此新活动结束时，程序将RAM字节0和1放回IAR和标志中，CPU将从中断前中断的位置恢复。

这个系统对于处理I/O操作非常有用。

在没有中断的情况下，CPU中运行的程序必须确保检查I/O总线上的所有设备定期的。

有了中断，程序可以做它设计要做的任何事情，处理键盘输入之类的事情的程序将根据中断系统的需要自动调用。

这是一个中断系统。

就该语言而言，计算机设计者采用了一个现有的动词'interrupt'，并以三种方式使用它：它是“键盘中断了程序”中的动词，它是“这是中断系统”中的形容词，它是“CPU执行了中断”中的名词。

## 这就是所有的人

是的，这是我们对计算机的描述的结尾。只有这些了。

您所看到的计算机所做的一切都是这些非常简单的操作的长串接，这些操作包括字节的添加、记数、移位、ANDING、ORD、XORD、存储、加载、跳转和I/O操作，都是通过执行RAM中的指令代码完成的。这就是电脑变成电脑的原因。

这是计算机中智能的总和。这就是计算机所能做的全部工作。这是一台机器  
OE正是它的设计用途，仅此而已。

就像锤子一样，它是人类设计的工具，用来完成人类定义的任务。

它完全按照设计完成任务。

也像锤子一样，如果不加区别地扔出去，它会做出一些不可预知和破坏性的事情。

计算机能做的事情的多样性仅限于那些为计算机运行而创建程序的人的想象力和聪明才智。制造计算机的人一直在使它们更快、更小、更便宜和更可靠。

当我们想到一台电脑时，我们可能会想到那个放在桌子上的盒子，盒子上有一个键盘、鼠标、屏幕和打印机。但是计算机在很多地方都使用。

你的汽车里有一台控制发动机的电脑。你的手机里有一台电脑。

大多数有线电视或卫星电视盒里都有一台电脑。它们的共同点是都有CPU和RAM。

这些差异都存在于外围设备中。手机有一个小键盘

和扬声器，以及用于外围设备的双向无线电。

您的汽车发动机上有各种传感器和控制装置，以及仪表板上用于外围设备的拨号盘。

快餐店的收银机有一个有趣的键盘，一个小显示屏和一个小收据打印机。一些红绿灯中的计算机根据一天中的时间和穿过嵌入在道路中的传感器的交通量来改变红绿灯。但是CPU和RAM使它

一台电脑，外围设备可以是非常不同的。

在本书的其余部分，我们将研究与理解计算机如何使用有关的各种主题，一些与计算机相关的有趣词汇，它们的一些弱点和一些其他未解决的问题。

### 硬件和软件

你听说过硬件。这个词已经流传很久了。五金店已经有一个多世纪了。

我认为五金店最初卖的是硬的东西，比如锅、锅、螺丝刀、铲子、锤子、钉子、犁等。也许“五金”是指用金属制成的东西。

今天，一些五金店不再出售锅和平底锅，但他们出售各种各样的硬物，比如螺栓和割草机，还有木材和许多软的东西，比如

地毯、壁纸、油漆等，但这些软的东西不叫软件。

“software”这个词是在计算机工业的早期发明的，用来区别计算机本身和它内部的比特状态。软件是指在计算机中设置或关闭位的方式，而不是计算机本身。

请记住，位可以是ON或OFF。

该位在空间中有一个位置，它是由某物构成的，它存在于空间中，它可以被看到。

比特是硬件。位是开还是关是重要的，但它不是单独的部分

插进电脑里，电脑里的东西是可以改变的，可以塑造的，它是“软的”，可以改变，但你不能把它全部拿在手里。这东西叫软件。

想想一盘空白录像带。那就在上面录部电影。

空白录像带和放电影的录像带有什么区别？

它看起来一样，它的重量一样，你看不到任何不同的表面上的磁带。

表面覆盖着可以磁化的非常细微的颗粒。

在空白带中，带的整个表面在随机方向上被磁化。

在磁带上录制电影后，磁带上的一些小地方在一个方向上被磁化

TION和其他一些小地方在另一个方向被磁化。

没有任何东西被添加到磁带中或从磁带中取出，这只是磁性粒子被磁化的方式。

当你把录像带放进录像机时，它会播放一部电影。

磁带是硬件，磁带上磁化方向的模式是软件。

机器。你不能称它，不能测量它，也不能用钳子把它拿起来。

所以它是“ware”的一部分，但不是硬件。剩下的唯一称呼它为“软件”。

## 程序

如前所述，RAM中的一系列指令称为程序。

程序有很多种大小。

一般来说，一个程序是一个软件，它拥有完成特定任务所需的一切。

一个系统将是更大的东西，由几个程序组成。

一个程序可能由几个称为“例程”的较小部分组成，而例程又可能由子例程组成。

系统、程序、例程和子例程之间没有硬定义和快速定义。

程序是它们的总称，唯一的区别是它们的大小和使用方式。

两种类型的程序之间还有另一个区别，这与它们的大小无关。

大多数家用和商用计算机上都安装了许多程序。

大多数这些程序都用来做所有者想做的事情。

这些程序之所以称为应用程序，是因为它们是为了将计算机应用于需要解决的问题而编写的。大多数计算机上有一个程序不是应用程序。

它的工作是处理计算机本身，并帮助计算机

**e应用程序。这一个不是应用程序的程序称为操作系统。**

## 操作系统

一个“操作系统”，简称“OS”，是一个有很多部分和多个目标的大型程序。

它的第一个任务是在您第一次打开计算机时启动并运行计算机。

它的另一个任务是启动和结束应用程序，并给每个应用程序一个运行时间。

它是那台计算机上每一个其他程序的“老板”。

当RAM中有多个程序时，在它们之间切换的是操作系统。

它允许一个程序运行一小段时间，然后是另一个程序，然后是另一个程序。

如果RAM中有十个程序，每个程序每次运行百分之一秒，那么每个程序将能够执行数百万个

指令，每秒几次。

看起来所有的十个程序都同时运行，因为每个程序都要做一些事情，速度比眼睛所能看到的还要快。

操作系统还向应用程序提供服务。

当应用程序需要读取、写入磁盘或在屏幕上绘制字母时，它不必执行完成任务所需的所有复杂I/O指令。操作系统有一些小例程，它一直保存在RAM中，用于这些目的。

应用程序使用这些例程之一所需做的就是加载寄存器中的一些信息，然后跳转到正确的OS例程的地址。下面是一个例子，说明如何执行该操作。

假设您想在屏幕上绘制一个字符。首先，将所需字符的ASCII代码放入R0。

然后将希望显示在屏幕上的行号和列号放入R1和R2中。

这里有一个棘手的问题：你把下一份申请表的地址

TION程序，进入R3。现在跳到OS例程。

例程将处理在屏幕上绘制字符的所有细节，然后它的最后一条指令将是JM PR R3。

因此，这些例程可以从任何应用程序中“调用”，完成后，例程将跳回到调用它的应用程序中的下一条指令。

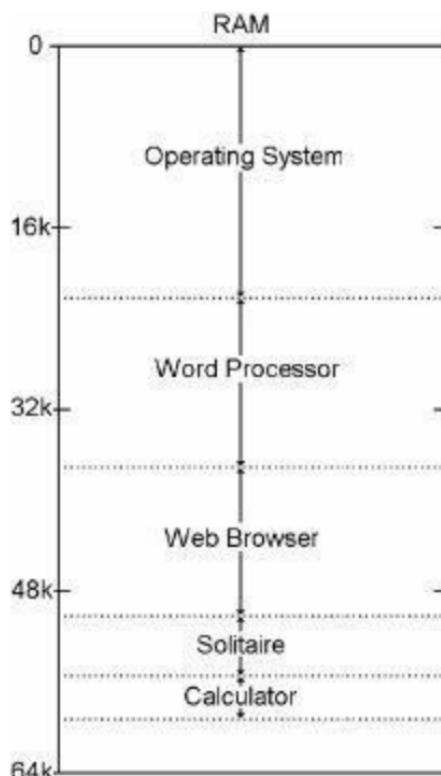
让OS执行所有I/O功能有几个原因。

其一，它使编写应用程序变得更容易，程序员甚至不需要知道外围设备的实际工作原理。

另一个原因是，如果每个应用程序都有自己的所有I/O例程副本，那么就会浪费大量的RAM。

其中一个最重要的原因是操作系统可以检查程序是否应该被允许做它要求做的事情。

这是操作系统其他工作的一部分



F是老板。

它从磁盘读取的数据，以及

还有很多其他的目的。

这就是你的普通电脑里发生的事情。RAM中有许多不同的程序和数据区域。

操作系统跳转到一个程序，程序跳转到一个例程，例程跳转到一个子例程。

每个程序都对其数据进行处理、计算或执行I/O操作。

当每一个完成，它跳回到它来自的地方。

CPU一次从一个程序执行一条指令，如果这些指令写得很智能，每个程序就会一个一个地完成它的工作

，而不干扰其他部分。

如果我们的计算机包括一个“中断系统”，就像我们前面几章所描述的那样，每当有人按键盘上的一个键或移动鼠标时，就会出现一个中断，该中断会调用操作系统的一部分来确定是哪个I/O设备导致了中断，然后调用适当的例程来处理它是什么。

完成后，CPU将继续执行中断发生时正在运行的任何程序的下一条指令。

这一切看起来都很复杂，如此多的指令在眨眼之间就被执行了。

有一些组织程序的方法和良好的编程实践可以使它更容易理解。

对这些问题的研究将简化软件，就像我希望这本书简化了硬件一样。

但这将是另一整本书的主题。

## 语言

当你只写1和0的时候，写程序是非常困难的，但这是CPU“理解”的唯一代码。

什么是语言？ 口语，如英语，是用声音来表示物体、动作和思想的一种方式。

书面语是用文字符号来表达口语声音的一种方式。

听起来像是另一个代码，一个表示代码的代码。我们就是摆脱不了这些东西！

你还记得我们在查看CPU指令代码和控制部分的布线时使用的速记吗？		指令		语言
添加	100 0	拉尔布	添加	代码
Ra、Rb	100 1	拉尔布	SHR	Ra、Rb
右移	101 0	拉尔布	SHL	Ra、Rb
左移	101 1	拉尔布	不是	Ra、Rb

不是

计算机语言是表示指令代码的一种方式。它的目的是使编写计算机程序更容易。

为了使用这种语言，您需要用ASCII字符编写所需的程序，并将其保存到文件中。然后，将一个名为“编译器”的特殊程序加载到RAM中，并跳转到它的第一条指令。编译器将读取ASCII文件，将每一行转换为它所表示的指令代码，并将所有指令代码字节写入第二个文件。

然后，第二个文件可能被加载到RAM中，当CPU跳转到它的第一条指令时，您用ASCII编写的程序

希望能达到你的目的。

当然，当计算机第一次被发明时，所有的程序都必须直接用1和0来写。

然后有人厌倦了这种乏味的编程方式，决定编写第一个编译器。

从那以后，程序用这种简单的语言编写，然后由编译器翻译成指令代码。

使用原始编译器，您甚至可以编写更好的编译器。

因此，为了使计算机语言存在，需要两件事，一组组成语言的单词（另一个代码）和一个编译器，编译器将书面语言编译成计算机指令代码。

我们在这本书中看到的语言只有大约20个单词。每个字与此计算机能够执行的指令之一直接相关。

你写的每一行都会产生一条计算机指令。

用这种语言编写87行程序时，编译器生成的指令代码文件将包含87条指令。

然后有人发明了一种“更高级”的语言，这种语言的一行可以产生多条计算机指令。例如，我们的计算机没有进行减法运算的指令。

但是编译器可以被设计成它可以识别语言中的一个新词，比如“subra，rb”，然后生成任意多的机器指令来进行减法运算。

如果你能用47条指令来做一些奇妙的事情，你可以说几句话在你的语言里，这意味着那些花哨的东西。

然后有人发明了一种更高级的语言，其中组成语言的单词甚至不像CPU的实际指令。

编译器有很多工作要做，但仍然生成指令代码来执行该语言中的单词所表示的内容。高级语言中的几行代码可能如下所示：

余额=2 000

利率=.034

打印“Hello Joe，您今年的兴趣是:\$”

打印余额x利率

有许多计算机语言。

有些语言是为了做科学工作而设计的，有些语言是为了商业目的而设计的，有些语言是为了更通用的目的而设计的。低级语言对于某些目的来说仍然是最好的。

## 文件系统

正如我们前面看到的，磁盘的实际工作方式对于大多数使用计算机的人来说是非常陌生的。

为了让事情变得更容易，有人发明了一种叫做“文件”的想法，文件应该和人们在办公室使用的纸质文件相似。纸质档案是一张纸板，折成两半，放在文件柜里。

此文件夹上有一个选项卡，您可以在其中为文件夹写入某种名称，然后可以在文件夹中放置一张或多张纸。

计算机文件是一串字节，可以是任何长度，从一个字节到磁盘上可用的所有字节。

文件也有名称。磁盘上可能有许多文件，每个文件都有自己的名称。

当然，这些文件只是一个想法。

为了使文件系统工作，操作系统提供了一组软件，使磁盘看起来像一个文件柜，而不是头、磁道、扇区和字节块。

该文件系统为应用程序提供了一种简单的磁盘使用方式。

应用程序可以要求操作系统创建、读取、写入或擦除称为文件的内容。

应用程序只需要知道文件的名称。

打开它，请求字节，发送字节，使其更大或更小，关闭文件。

OS使用磁盘的一部分来维护文件名列表，以及每个文件的长度和数据的第一扇区的磁盘地址（头、磁道、扇区）。

如果文件小于磁盘扇区，这就是所需的全部，但如果文件大于一个扇区，则还有一个列表，其中包含保存文件所需的磁盘类型地址。

应用程序说创建一个名为“给Jane的信”的文件，然后用户键入给Jane的信并保存它。

该程序告诉OS字母在RAM中的位置和长度，OS将其写入适当扇区中的磁盘，并更新文件长度和任何必要的磁盘类型地址列表。

要使用文件系统，应用程序需要遵循某种规则。

如果要将一些字节写入磁盘，则需要告诉OS文件的名称、要写入的字节的RAM地址以及要写入的字节数。

通常，您会将所有这些信息放在RAM中的一系列字节中的某个位置，然后将这些信息的第一个字节的RAM地址放在其中一个寄存器中，然后执行跳转指令T

HAT跳转到将文件写入磁盘的操作系统内的例程。

所有的细节都由这个例程来处理，这个例程是操作系统的一部分。

如果您要求操作系统查看您的磁盘，它将显示所有文件名的列表，并且通常是它们的大小以及上次写入它们的日期和时间。

你可以把各种各样的东西存储在文件里。

文件的名称通常由两部分组成，中间用句点分隔，比如“xxxx.yyy”。句点前的部分是某种名称，比如“给Jane的信”，点后的部分是某种类型，比如“doc”，它是“document”的缩写。句点前的部分告诉您文件中的内容。

点后面的部分告诉您文件中包含的数据类型，换句话说，它使用的代码。

文件的类型告诉您和操作系统文件中的数据使用什么代码。

在一个流行的操作系统中，“。txt”表示文本，这意味着文件包含ASCII。

“。bmp”是指位图，它是一幅图片。

“。exe”表示可执行文件，这意味着它是一个程序，因此包含指令代码。

如果您询问OS可以执行哪些程序，它将显示以“。exe”结尾的文件列表。

如果您要求提供可以查看的图片列表，它将显示以“。bmp”结尾的文件列表。

**有许多可能的文件类型，任何程序都可以发明自己的类型，并使用任何代码或代码的组合。**

## 错误

计算机是一台相当复杂的机器，它一个接一个地非常快地完成一系列简单的事情。

这里会出什么差错？

在计算的早期，当计算机中的每个门相对昂贵的时候，有时有些部件实际上有可移动的部件来进行电气连接。两块金属必须接触才能把电送到建筑工人想要的地方。

有时当机器停止正常工作时，修理工人会往里面看，看看出了什么问题，他会发现一只蜘蛛爬进机器里，自己被卡在里面了。

在这两块本来应该互相接触的金属之间。

然后，当一块金属移动到另一块时，蜘蛛挡住了路，他们不愿碰。

这样，电力就不能到达需要的地方，机器也就不能正常工作了。

修复人员会删除bug，清理联系人，并报告“计算机中有一个bug”，他的字面意思是一个bug。

随着时间的推移，每当一台计算机出现错误操作时，人们就会说这台计算机有一个bug。有两类主要的计算机bug:硬件和软件。

硬件错误实际上意味着计算机坏了。

这可能和你打开电脑一样严重，它会着火，因为RAM中有一个字节，其中的一个位总是关闭的。

软件错误可以有多种形式，但它们最终都是程序员的错误。

错误编写程序的方法可能比正确编写程序的方法多得多。有些错误只是

创建某种不正确的结果，其他错误会导致计算机“崩溃”。

我最喜欢的一个愚蠢的程序员故事是这样的：有人赊购了一辆汽车。

他借到了一本优惠券，每次付款都要寄一张优惠券。

但当他第一次付款时，他不小心用了书中的最后一张优惠券而不是第一张。

几周后，他收到了贷款公司的一封电脑生成的信，信中说：“谢谢你全额还清贷款，下次你需要贷款时请再次使用我们的服务。”显然，程序只是检查了C

票面金额，如果等于书中最高的票面金额，跳到例程全额还款。

在决定偿还贷款之前，它至少应该检查一下贷款的余额。

这是一个微妙的错误，贷款公司可能要等到几个月后审计账目时才会发现。

计算机完全按照命令去做，而且大部分时间都是足够的，但是编写程序并不是为了预测某些情况下的所有情况。

时代发生在现实世界中。

最糟糕的软件缺陷之一就是陷入了一个循环。

程序执行一系列指令，然后跳回到系列的开头并一遍又一遍地执行。

当然，循环在编程中一直都被使用，但是它们被用来做一些有有限数量相似步骤的事情。

它可能重复，直到50字节已经被移动到某处，或者继续检查用户是否按键盘上的键。但是计算机将在某个点退出循环，并且C

继续执行下一个任务。

但是，如果有某种编程错误，其中有一个循环没有出路，计算机将似乎完全卡住。这有时被称为“挂起”，整个计算机可能需要关闭并重新启动，以脱离循环并返回到有用的操作中。

有各种各样的错误，最终导致CPU试图执行指令代码以外的内容。

假设您的程序驻留在地址10到150，并且您有一些ASCII数据，例如地址151到210处的名称和电话号码。

如果程序写得不正确，以致在某些条件下它将跳转到地址180，则它将继续从地址180开始提取和执行字节。如果180-189填充了“Jane SMI”的ASCII

“程序”现在将执行完全的垃圾，一系列的字节，不是设计成指令代码。

它可能会将自己放入一个循环，或者跳回程序中的某个位置，或者发出命令擦除磁盘驱动器。它将以通常的高速处理垃圾。

如果您查看字节中的模式，您可以看到它会做什么，但它可能是任何事情。

如果地址180的名字是“Bill Jones”，那就完全不同了。硅

NCE它并不是为了有用而设计的，很可能它只是不断地把内存中的东西弄得一团糟，直到计算机必须关机才能停止运行。

如果程序意外地将“John Smith”写入存储字体的位置，则可能发生另一种类型的错误。在这种情况下，屏幕上显示的每个字母“e

之后会是这样的：“。”

计算机每秒执行数亿条指令，只需要一条错误的指令就能使整个过程停止。

因此，以完全“无bug”的方式对计算机进行编程是一个备受关注的话题。

几乎所有的编程都是用语言来完成的，这些语言的编译器被设计成生成避免最严重错误类型的指令代码，并警告程序员如果

违反了某些良好的编程实践。

不过，编译器可能会有错误，而且他们永远也无法发现上面汽车贷款中的错误。

正如你所看到的，计算机及其软件是相当脆弱的东西。

每扇门每次都必须工作，每条执行的指令都必须正确。

当你考虑到所有可能出错的事情时，正常情况下正确的事情所占的高比例实际上是相当令人印象深刻的。

电脑疾病？

这里可能发生了两件不同的事。

如果您的朋友给您发送了一个名为“funning.mov”的文件，并且您的操作系统包含一个播放“.mov”文件的程序，那么操作系统将把该程序加载到RAM中，该程序将读取“funning.mov”中的图片。

文件并将其显示在屏幕上。这很好，运行的程序已经在您的计算机上了。

“funning.mov”文件只是提供了一系列显示在屏幕上的图片。

但是如果你的朋友给你发送了一个名为“funning.exe”的文件，那么当你要求操作系统播放电影时，它会将“funning.exe”加载到RAM中并跳转到它的第一条指令。

现在，您的计算机中正在运行一个来自其他地方的程序。

如果这是一个病毒程序，它可能会为您播放电影，以便您不会怀疑任何事情，但它可以做任何其他事情，它想要的，到您的磁盘上的文件，而您正在观看电影。

它很可能会自行安装并进入%d

奥曼状态持续数天或数周，你甚至都不知道你的电脑被“感染”了，但迟早它会活过来，做任何它被设计来做的破坏。

这种恶意程序被称为病毒，因为它的工作方式与真正的病毒感染生物的方式相似。

真正的病毒是比一个细胞动物还小的东西。

因为病毒本身不能繁殖，所以它不太符合活着的条件。

然而，它们确实是通过侵入活着的细胞来繁殖的。

一旦进入细胞，病毒就利用该细胞的机制复制自身，然后再继续感染其他细胞。

**计算机病毒本身也不能复制或做任何其他事情。**

**它需要进入计算机，并以某种方式由该CPU执行一次。**

**当它第一次运行时，它会将自己插入到操作系统中的某个位置，以便以后定期执行。**

**这些指令将对运行它们的计算机造成任何损害，而且它们通常还会做一些旨在将病毒传播到其他计算机上的事情。**

计算机。

固件

当然，RAM是任何计算机的重要部分。

将字节写入RAM并再次将其读出的能力是机器工作方式的一个组成部分。

但在某些计算机中，RAM的某些部分仅在计算机启动时才被写入，此后这些部分在计算机运行时保持不变。这在任何总是运行相同程序的计算机上都可能是正确的。

RAM的一半可能用于包含程序，而RAM的另一半则用于包含程序正在处理的数据。程序的一半必须在某个点加载，但在此之后，CPU只需读取b。

以获取并执行它们。

当你遇到这种情况时，你可以用正常的方式构建一半的计算机内存，而用另一半，你可以跳过与非门，直接将每一位连接到程序模式中的开或关。

当然，您不能写入预布线RAM，但您可以很好地读取它。

这种类型的RAM称为只读内存，简称ROM。

您使用它的方式与使用RAM的方式相同，但您只读取它。

ROM有两个优点。在计算机的早期，当RAM非常昂贵时，ROM比RAM便宜得多。

另一个优点是，当您第一次打开计算机时，不必再将程序加载到RAM中。

它已经在ROM中，准备由CPU执行。

这里的重点是一个新词。

由于软件之所以被命名为“soft”是因为它是可变的，当涉及到ROM时，您仍然有一个位模式，但它们不再那么软了。不能写入ROM，不能更改位。

因此这种类型的内存被称为“固件”，它是永久写入硬件的软件。

但这还不是故事的结尾。上述ROM必须在工厂中以这种方式构建。

多年来，这一想法得到了改进，更易于使用。

下一个进步是，有人有一个聪明的想法，就是在工厂里把每一位都放在ROM上，但有一种方法可以用大量的功率写入ROM，这可能烧毁单个连接，将单个位变为OFF。因此，该ROM可以在出厂后编程。这被称为“可编程ROM”或简称为“PROM”。

然后有人想出了如何制作一个舞会，如果它暴露在紫外光下半小时，就可以修复所有断开的连接。这被称为“可擦除的舞会”，简称“EPROM”。

顺便说一下，ROM一词也被用来指任何类型的永久设置的存储器，比如预录盘，比如在“CD ROM”中，但它最初的定义只适用于与RAM一样工作的东西。

## 靴子

靴子和电脑有什么关系？

嗯，有一句老话是这样说的：“用你自己的鞋带把自己拉起来。”这是一个笑话，字面意思是把鞋带缝进许多靴子里，用来帮助你把靴子拉到脚上。

开玩笑的是，如果你穿着这样一双靴子，想从地上爬起来，而不是爬梯子或绳子，你只需用力拉一下靴子就能从地上爬起来。当然可以

这只适用于卡通，但这个短语的意思是在没有明显的方法的情况下做某事，或者在没有通常使用的工具的情况下做某事，或者在没有其他人帮助的情况下独自完成某事。

在计算机中，有一个类似于需要离开地面并且没有可用的工具来完成它的问题。

当计算机运行时，内存中充满了正在执行某项操作的程序，当计算机操作员输入命令以启动另一个程序时，操作系统将程序定位在磁盘上，将其加载到内存中，然后跳转到程序的第一条指令。现在程序正在运行。

但是，当您第一次打开计算机时，如何将操作系统放入内存中呢？

在内存中运行的程序需要告诉磁盘驱动器发送一些指令代码，程序需要在适当的位置将该代码写入内存，然后跳转到第一条指令以使新程序运行。

但是当你打开电脑时，内存中的每一个字节都是零。内存中根本没有任何指令。

这是不可能的情况，你需要一个程序

在内存中获取一个程序，但那里什么都没有。

因此，为了让计算机首先运行，计算机必须做一些不可能的事情。

它必须靠自己的力量站起来！

很久以前，在计算机的早期，机器的前面板上有开关和按钮，允许操作员将字节数据直接输入寄存器，然后从寄存器输入RAM。

您可以用这种方式手动输入一个短程序，然后启动它运行。

这个名为“引导加载程序”的程序将是您所能编写的最小的程序，它将指示计算机从外围设备读取字节，将其存储在RAM中，然后跳转到第一个I

建筑。

当引导加载程序执行时，它将一个大得多的程序加载到内存中，例如操作系统的开始，然后计算机将变得可用。

现在，有更简单的方法将第一个程序加载到计算机中，事实上，它会在计算机打开后立即自动发生。

但是这个过程仍然会发生，第一步被称为“引导”或“引导”，它只意味着将第一个程序放入内存并开始执行它。

这个问题最常见的解决办法有三个部分。首先，IAR设计为当电源首次接通时，其所有位不是0，而是最后一位为0，但其馀位为1。因此，对于我们的小计算机来说，第一个要获取的指令将位于地址11111110。第二，类似于RAM的最后32字节（235-256）的东西将被ROM代替，用一个简单的程序硬连线，该程序访问磁盘驱动器，选择磁头0、磁道0、扇区0、将该扇区读入RAM，然后跳转到它的第一字节。第三部分那么，最好在磁盘的第一个扇区上写一个程序。

顺便说一下，这个扇区被称为“引导记录”。

这个词'boot'已成为计算机语言中的动词。

它意味着在没有程序的情况下将程序加载到RAM中。

有时人们用它来表示将任何程序加载到RAM中，但它的原始含义只适用于将第一个程序加载到其他空白RAM中。

数字与模拟

你肯定听说过这些术语。

似乎任何与计算机相关的东西都是数字的，而其他的东西都不是。

但这还不够接近事实。

如果你有一个三英尺高的平台，你可以建造楼梯让人们爬上去，或者建一个坡道。

在坡道上，你可以爬到地板和平台之间的任何水平；

在楼梯上，你只能选择有多少台阶。坡道是模拟的，楼梯是数字的。

假设你想在你的花园里建一条人行道。你可以选择用混凝土或砖头建造人行道。如果砖块是三英寸宽，那么你可以做一个30英寸宽，33英寸宽，但不是31或32英寸宽的砖块步行。如果你用混凝土做人行道，你可以把它倒到你想要的任何宽度。砖块是数字的，混凝土是模拟的。

如果你有一本旧书和一幅旧油画，而且你想把它们都复印一份，你将会有更轻松的时间复印这本书。

即使这本书的书页发黄，角上有狗耳朵，里面有污迹和虫洞，只要你能读懂书中的每一个字母，你就可以完全按照作者的意图重新键入整篇文章。

随着油画，原来的颜色可能已经褪色，并被泥土遮蔽。EA的确切位置

CH刷毛的每一笔，油漆的每一个点的厚度，相邻的颜色混合的方式，都可以复制得非常详细，但不可避免地会有一些细微的差别。

书中的每一个字母都来自于一个特定数量的可能性列表；

油漆颜色的变化及其在画布上的位置是无限的。这本书是数字的，这幅画是模拟的。这就是模拟和数字的区别。我们周围的世界大多是模拟的。

大多数旧技术都是模拟的，如电话、留声机、收音机、电视、录音机和录像带。奇怪的是，最古老的设备之一，电报，是数字化的。

随着数字技术的高度发展和价格的低廉，ADI公司正一个接一个地被实现同样功能的数字版本所取代。

声音是一个类比的东西。

老式电话是一种模拟机器，它把模拟声音转换成一种电模式，模拟声音，然后通过电线传到另一部电话。一种新的数字电话接收模拟声音，并将其转换成数字代码。然后数字代码传送到另一个数字电话，在那里数字代码被转换回模拟声音。

当模拟电话工作正常时，为什么会有人费心去发明数字电话呢？

答案当然是，虽然模拟电话工作，但它并不完美。

当一个模拟的电模式经过长距离时，沿途可能会发生很多事情。

随着它的移动，它变得越来越小，所以它必须被放大，这就引入了噪声，并且当它接近其他电气设备时，来自其他设备的一些模式可以获得MI

加入了谈话。声音越远，噪声和失真就越大。

对声音模拟的每一次改变都成为声音的一部分，从另一端传出。

进入数字技术的救援。

当您通过长距离发送数字代码时，各个位会受到相同类型的失真和噪声，并且它们确实会略有变化。然而，如果一个位仅为97%，而不是100%，这并不重要。

门的输入

只需要“知道”位是开还是关，它只需要在这两个选项之间“决定”。

只要一个比特还在半路上，它所进入的门就会以完全相同的方式工作，就像比特已经完全打开一样。

因此，结束时的数字模式与开始时一样好，当转换回模拟模式时，根本没有噪声或失真，听起来就像是隔壁的人。

每种方法都有优缺点，但总的来说，数字技术的好处远远大于缺点。

数字的最大优势可能与复制有关。

当你复制一张像乙烯基唱片一样的东西时，你可以把它录到录音机上，或者我猜你甚至可以用所有的设备来制作一张新的乙烯基唱片。

但原件和复印件之间会有一定程度的差异。首先，所有机器都有精度限制。

任何物理对象的副本都可能与原始对象非常接近，但永远不会非常精确。

第二，如果说的话

原件上的任何划痕或尘埃颗粒，复制品就会有这些缺陷的复制品。

第三，每次播放唱片时，唱片和针头之间的摩擦实际上会磨损少量的乙

烯基。如果你使用录音机，声音中总会有一个很低的“嘶嘶声”。

如果您复制一个副本，以及一个副本，等等，更改将变得越来越大，在每一个阶段。

当涉及到数字的东西时，只要原件中的每一位都在拷贝中，我们每次都会得到一个精确的拷贝。

你可以复制一份副本，还有一份副本，等等，它们中的每一份都将与原件完全相同。

如果你想制作无限数量的拷贝并永久保存某样东西，数字无疑是一条出路。

到目前为止，我们制造的计算机和外围设备都是完全数字化的。

如果我们想要做的只是数字方面的事情，比如算术和书面语言，我们可以把它留在那里。

然而，如果我们想让我们的电脑播放音乐和处理彩色照片，还有一件事我们需要看。

我撒了谎-有点

计算机中有一块硬件并非完全由与非门构成。

这件事并不是真的有必要让一台计算机成为一台计算机，但大多数计算机都有一些。

它们用于从模拟转换为数字，或从数字转换为模拟。

人的眼睛和耳朵对类似的东西有反应。

我们听到的东西可以是响亮的或柔和的，我们看到的东西可以是明亮的或黑暗的，可以是多种颜色中的任何一种。

我们上面描述的计算机显示屏具有320x200或64,000像素。

但是每个像素只有一个位来告诉它该做什么，是开还是关。

这对于在屏幕上显示书面语言是很好的，或者可以用来绘制线条图，任何只有两个亮度级别的东西。但我们都在电脑屏幕上看到过照片。

首先，需要有一种方法把不同的颜色在屏幕上。

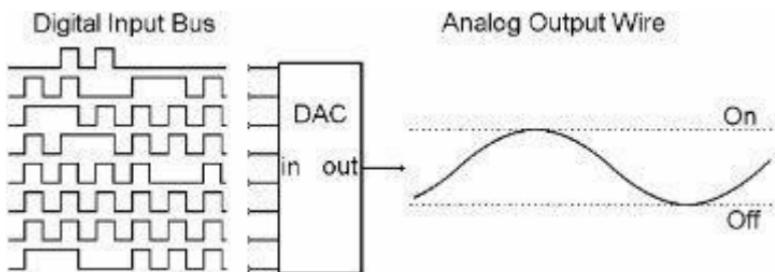
如果你拿出放大镜，看看彩色电脑或电视屏幕，你会发现屏幕实际上是由蓝、红、绿三种不同颜色的小点组成的。 每个像素有三个部分，每种颜色一个。

当显示适配器扫描屏幕时，它同时选择每个像素的所有三种颜色。

为了实现这一功能，您需要一个称为“数模转换器”或简称为“DAC”的器件，该数字指定256个不同级别的亮度。 DAC有8路数字输入和1路模拟输出。

它的工作方式是将输入作为二进制数进行连接，输出在OFF和ON之间有256个电平。  
输出在OFF和ON之间有256个等级，并达到输入编号指定的级别。如果

输入为128，输出为半导通。对于A64，输出将为ON的四分之一。  
对于0，输出将完全关闭。



为了使该彩色屏幕正常工作，显示适配器需要一次访问三个字节，将它们连接到三个DAC，并将DAC的输出连接到正在绘制的当前像素中的三种颜色。  
彩色屏幕就是这样工作的。

当我们在上一章中定义“模拟”时，我们说它是一个从完全关闭到完全打开的连续变量。  
但我们的DAC的“模拟”输出实际上只有256个不同的电平。

它更接近模拟，而不是一点点，但它仍然有步骤。

计算机所做的是以足够小的步长来近似模拟对象，以欺骗预期的受众。

说到眼睛，256种不同的亮度水平就足够了。

如果需要较小的步骤来愚弄目标受众，则可以制作一个数字端为16位的DAC。

因此，您可以用0到65535之间的任意数字表示数字输入。

模拟端仍然只能在完全关断和完全开启之间变化，但阶跃的大小将小得多，因为在有65536个阶跃。

当涉及到耳朵时，它可以听到非常小的差异，因此需要一个16位DAC才能获得高质量的声音。

从音乐到演讲再到雷鸣般的轰鸣，所有的声音都是空气的震动。

它们的变化在于空气的振动速度，以及空气的确切振动方式。

人的耳朵可以听到从低端的大约20赫兹到高端的20,000赫兹（20 kHz）的振动，所以这是计算机设计用来处理的振动范围。

任何电子机器发出声音，都有一种叫做扬声器的装置。

扬声器所做的就是在空气中来回移动，使空气振动。如果它使空气振动的方式与原来的录音完全一样，听起来就像原来的一样。

为了在计算机中存储声音，扬声器的位置被划分为65536个可能的位置。

然后将第二部分分为44,100个部分。

在第二个部分中的每个部分，扬声器的期望位置被存储为两个字节的数字。

这是足够的信息，以非常高的质量再现声音。

要播放高质量的立体声音乐，计算机需要一个“声音外设”，其中有两个16位DAC，其模拟输出连接到扬声器。它还有自己的时钟，时钟频率为44,100 Hz。

每次勾选后，它将获取下两个两字节的数字，并将它们连接到DAC的数字端。

就速度而言，这将是每秒176,400字节。

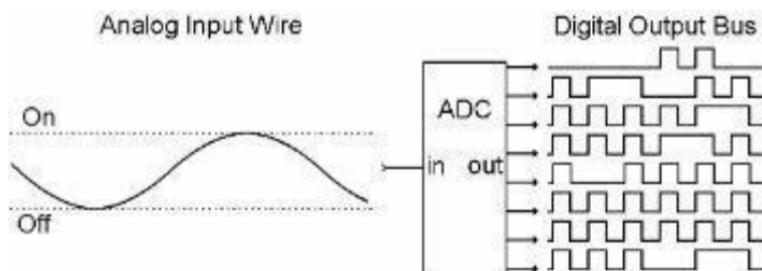
当然，这是快速的，但请记住，我们的计算机时钟滴答十亿次每秒。

这意味着计算机可以向声音外围设备发送四个字节，在需要发送下四个字节之前，关闭并在其他任务上执行大约4000条指令。

相反，有一个“模数转换器”，简称“ADC”。

这用于将麦克风中的声音转换为一系列字节，或者用于照相机将图片转换为一系列字节。 输入端有一条线，可以是从一路断开到一路接通的任意位置。

ADC将其输出转换为0至255之间的数字，



8位ADC或0-65,535（对于16位ADC）。此数字表示输入打开或关闭的程度。

半导通为128或32,768，四分之一导通为64或16,384，等等。此过程与DAC的功能恰好相反。

**DAC和ADC不是由与非门构成的，它们具有收音机所具有的电子器件。**

**他们如何做他们所做的不是这本书的主题。**

**所以我说电脑里的所有东西都是NAND GATES做的可能是我撒谎了？**  
**其实不是，因为DAC和ADC仅用于某些类型的外设，而不用于计算机本身。**

## 全面披露

我们在这里建了一台非常小的计算机。

它是可以发明的最小的计算机，它可以做任何必要的事情来配得上计算机的名字。

我认为自从1952年以来没有人制造过这么小的计算机，在现实世界中也没有人制造过这样的计算机。

如果一个真正的计算机设计师读过这本书，我相信他一定会对这里错过的制造更好的机器的所有机会大发雷霆。但是，我们的目标仍然是尽可能简单地说明计算机原理。  
这是一台八位计算机。

这意味着处理器中的寄存器为8位，总线为8位，在此机器中，甚至M  
Emory地址寄存器也为8位。

随着大多数计算机的实际建成，而RAM中的单个字节仍然是8位，其他所有的都扩展到16位、32位或64位，或者在机器的不同部分将它们组合起来。

我们的RAM只有256个字节，小得离谱，但使用8位M  
Emory地址寄存器就可以做到这一点。

如果使用16位，您可以拥有65,536字节的RAM（即64KB）；如果使用24位，您可以拥有16MB；如果使用32位，您可以拥有4GB的RAM。

真正的计算机有一些这台计算机没有的东西，但是它们不能做这台计算机不能做的事情。

在我们的计算机中，如果你想向左移动一个字节三位，你可以在你的程序中放置三个向左移动的指令。

在大多数真实的计算机中，它们都有移位器，可以在一条指令中移位任意数量的位。但是结果是一样的，不管是哪种情况，您的字节看起来都一样，真正的计算机只会更快地完成任务。

在我们的计算机中，加法器可以加两个8位数字。

如果你想增加16位数字，你必须使用一些软件来做。

在大多数计算机中，加法器可以在一条指令中增加16或32位数字。

同样，结果是一样的，一个比另一个快。

所以，是的，我们的计算机是一台简单的、小的、相对慢的计算机，但是它可以做任何更复杂的机器可以做的事情。

使机器更大的东西，是为了更快地完成任务，在更少的时钟周期内完成任务，用更少的时间完成同样的任务

指令，同时对多个字节进行操作。但机器的工作性质是完全一样的。  
他们能做的每一件事都可以归结为移位、ANDING、ORD、XORD、ADDING和NOT  
E字节。本书中没有遗漏任何其他花哨的操作类型。

在一台更大的机器上，你可以在一条指令中完成加法、减法、乘法和除法。  
这是因为它们有大量的门被布置成“硬件乘法器”之类的东西，没有理由向你展示如  
何构建其中一个门的细节，对于少数需要构建一个门的人来说，这是一项非常复杂  
的工作。这是可以理解的，这一切最终归因于与非门，就像其他一切一样。

但是我们已经看到了如何做所有的数学运算

只有一个加法器，移位器，而不是门和一些软件。

硬件乘法器速度更快，但结果完全相同。

**更大的机器有更多的寄存器，寄存器都是多字节的，它们有加法器，可以同时加三个数字，但指令仍然可以归结为相同的简单操作。**

**你对电脑的了解并不小，因为我们看过一台小电脑。**

## 哲学

为什么我们在一本关于计算机的书里有一章叫做“哲学”？

本书中唯一接近哲学问题的是它的题目，“但它怎么知道？”我们稍后将尝试回答这个问题。

这本书是关于我们今天拥有的计算机的。但是未来呢？

随着计算机和软件的不断进步，有一天会出现行走、说话、行为和人一样的计算机化机器人吗？

有一天我们必须决定是否给予这些机器人与人类同等的法律权利，这一天会到来吗？计算机最终会取代人，占领世界吗？

为了回答这些问题，人们经常提到一个在哲学领域多年来一直悬而未决的重大问题。

问题是，人是完全由我们可以看到和解剖的结构体构成的，还是每个人都有一个内在的精神成分，它解释了意识、爱、荣誉、幸福、痛苦等品质。

这一问题远远超出了本书的范围，尽管许多书对每一种观点都进行了论证，但仍然没有令人信服的答案。科学界有人说，我们正在建造有意识的计算机，这将会发生。

人文科学中有人说，这是不可能的，因为你不能制造一种精神。

双方都无法左右对方。

如果我们把大脑定义为被头骨包围的那块看起来很可笑的灰色肉，把头脑定义为对意识、记忆、创造力、思维以及我们头脑中所注意到的其他一切负责的东西，那么我们就可以把这个大的哲学问题重述为：“大脑和头脑是一回事吗？”

那么当涉及到制造一个令人信服的人类机器人的问题时，有两种可能性。

如果大脑和头脑是一回事，你今天可能无法建立一个合成的人，但随着时间的推移，最终你可以理解大脑中的每一个结构和功能，建立一个同样复杂的东西，从而产生真正的意识，而这实际上应该像任何其他人一样。

如果大脑和头脑不是一回事，那么建立一个机器人伙伴将永远是关于模拟人类，而不是建立同等质量和价值的东西。

重述这个问题并不使回答变得更容易，但是这种将我们对头脑的了解与我们对大脑的了解分开的想法可能是有用的。

早些时候，我们曾说过，我们将展示计算机是如何工作的，这样我们就可以看到它们能够做什么，也可以看到它们不能做什么。我们要拿走我们所知道的

关于大脑和我们对头脑的了解，并将它们与我们对计算机的新知识逐一进行比较。在这样做的时候，我们可以找出异同点，或许可以回答一些争议较小的问题。

计算机可以很容易地完成某些事情，例如将数列相加。

一台计算机可以在一秒种内完成数百万次的加法运算。

头脑几乎不能同时记住两个数字，更不用说在没有纸和笔的情况下把它们相加了。

大脑似乎有能力同时查看和考虑相对大量的数据。

当我想起我最喜欢的猫时，我可以重新体验它的样子，听到它咕噜咕噜的叫声，感觉到它毛皮的柔软和它被捡起来时的重量。这些是我认识我的宠物的一些方式。

对我们的电脑来说，想想一只猫意味着什么？

它可以将猫的图片和猫的声音编码在旋转的磁盘或RAM中的文件中。这是在思考吗？

如果你通过ALU逐个运行这些文件的字节，你会这样想吗？

如果你把这幅画放到屏幕上，那会不会是在想什么呢？

如果你把声音播放给扬声器，那是在思考吗？

计算机中编码的声音和图片只是它们所在位置的字节模式。

它们看起来不像任何东西，听起来也不像任何东西，除非它们被发送到设计它们的外围设备。如果它们被发送到屏幕和扬声器，计算机就看不到或听不见它们。

当然，您的计算机可以有一个摄像机指向屏幕，麦克风听声音，但计算机仍然看不到图片或听到声音，它将只是C

收集更多的字节串，这些字节串与发送到屏幕和扬声器的字节串非常相似。

，它是什么，它在哪里？什么看着

整合整体？

我们刚刚看到了电脑里的所有东西。计算机在总线上一次移动一个字节。

它所做的最奇特的事情是将两个字节添加到一个字节中。

它“做”的所有其他事情都只不过是简单的字节存储。

存储的字节除了维护自己的当前设置之外，什么也不做。

一台计算机根本没有任何设备可以将图片的元素集成到其他任何东西中，没有任何地方可以存储其他东西，也没有任何东西可以用来查看。

它。

我并不是说不能制造出能够执行这些功能的东西，我只是说我们今天所知道的计算机目前不包括任何这样的设备。

还有一个问题。

如果一个大脑像一台计算机一样工作，那么它需要有一个CPU运行的程序。

这个程序从何而来？

尽管大脑有数万亿个细胞，但整个人体都是从一个受精卵细胞开始的。

所以大脑的任何程序，都必须存在于这个细胞中，大概存在于DNA中。

科学家们现在已经破译了人类的全部DNA序列。

DNA的有趣之处在于它是一长串只有四种类型的东西。是数码的！

这一串中的许多片段被用来进行化学反应，产生蛋白质等，但大部分被称为“垃圾DNA”，因为没有人知道它的目的是什么。

但是，即使你考虑到DNA的全部都用于计算机软件，那么这个程序中可能有大约10亿条指令。

这是一个很大的软件，但一般的家庭电脑可能有那么多的软件加载到它的硬盘上，这将远远不够运行一个人。

有人说，人类的计算机程序本身。作为一名程序员，我无法想象这将如何工作。虽然程序确实可以根据收集到的数据积累数据并修改其工作方式，但这与编写新程序不同。

如果有人写了这个程序，可以写任何新的需要的程序，将有大量的计算机程序员永远失去工作。

然后是计算机犯的错误和人们犯的错误。

如果一台计算机陷入了一个循环，它似乎已经完全停止了。

你见过一个人在街上突然停止工作吗？所有的功能都停止了。

这个人会摔倒，直到他的电脑重新启动。

人确实会时不时地垮下来，但这通常是因为其他部位骨折，比如心脏病发作，你可以看到这个人把疼痛看成是

它把他们打倒了。

但是如果人类的计算机陷入一个循环，就会立即失去知觉，身体就会倒下

一瘸一拐没有挣扎。

我从来没有见过，但如果大脑就像一台电脑一样运作，你就会期望它在一个相当有规律的基础上运行。

还有速度的问题。正如我们所看到的，一台简单的计算机可以在一秒内完成10亿件事情。

说到大脑，它的神经与计算机中的神经有一些相似之处。

神经也可以把电流从一个地方传到另一个地方。

在计算机中，电线从大门出来，进入其他大门。

在大脑中，神经通过“突触”连接在一起。这些突触是神经之间的空间，其中一条神经中的电流产生化学反应，然后导致

ES是下一个制造自己电力的神经。这些化学反应非常缓慢。

没有人能证明这些神经与计算机中的电线有任何相似之处，但由于它们缺乏速度，即使连接相似，也不太可能起到多大作用。

当电流迅速通过神经细胞后，它到达突触，在那里化学反应大约需要百分之一秒才能完成。

这意味着我们用与非门构建的简单计算机可以在同一时间完成200万件事情，而只有一件事情可以完成。

ULD由神经和突触构成的计算机来完成。

另一个大脑和计算机之间的差异非常明显的领域是人脸识别领域。头脑对此很在行。如果你走进一个有五十人参加的聚会，你会在几秒钟内知道你是在一群朋友中还是在一群陌生人中。

人们对如何实现这一壮举做了大量的研究，发现了许多非常有趣的信息。

也有很多猜测，有很多迷人的理论

关于基本原则和机制的信息。但其完整、准确的结构和功能尚未被揭示。

如果给计算机一个人的图片文件，然后再给它同一个文件，它可以逐个字节比较两个文件，并看到一个文件中的每个字节与另一个文件中的相应字节完全相等。

但是，如果给计算机两张同一人在不同时间、不同角度、不同照明或不同年龄拍摄的照片，则这两个文件的字节将不逐个字节匹配。以便计算机确定

这两个文件代表同一个人是一项艰巨的任务。

它必须运行非常复杂的程序，对文件执行高级数学函数，以查找其中的模式，然后从不同的角度计算出这些模式可能是什么样子，然后将这些内容与存储在磁盘上的每一个其他面进行比较，选择最接近的匹配项，然后确定它是否足够接近看起来相似的人或仅仅是某人。

关键是计算机有一种基于这些原理处理图片的方法

电脑在上面工作。

仅仅使用这些原理还没有产生出能够识别人脸的计算机或软件，其识别速度和准确率与任何普通人都相差无几。

如果你想造一台像人一样工作的机器，那当然是最好的方法了

就是找出人是如何工作的，然后建造一台机器，它的工作原理相同，部件的工作原理相同，连接方式也与人相同。

当托马斯·爱迪生发明留声机时，他正在处理声音的问题。声音是空气的振动。

于是他发明了一种装置，可以捕捉空气中的振动，并将其转化为蜡筒表面的振动槽。

然后，通过将凹槽中的振动传递回空气中，可以再现声音。

关键是，为了重现声音，他找到了声音的工作原理，然后制造了一台同样原理的机器。

声音是一种振动，留声机的凹槽是一种振动。

关于什么使人滴答作响的问题已经做了很多研究。

关于如何使计算机做人们做的事情，人们做了很多研究。

很多东西已经被发现，很多东西已经被发明。

我不想贬低在这些领域所做的任何工作或取得的成果。

但是有许多东西还没有被发现或发明。

任何死脑筋都已解剖，并对其部位进行了研究和分类。

大脑中确实含有神经细胞，可以将电流从一个地方传输到另一个地方。

这是大脑和计算机之间的相似之处。但对活体人脑实际运作的研究必然是有限的。

大多数观察是在意外或疾病导致的手术中进行的。

我对受伤或生病后的行为变化有过任何观察

脑部某些部位出血。

通过这项研究，人们有可能将某些功能与大脑的某些区域联系起来。

但是还没有人发现总线、时钟、寄存器、ALU或RAM。

大脑记忆的确切机制仍是个谜。

研究表明，随着时间的推移，神经会产生新的连接，人们认为这是学习的机制，但没有人能够说这一特定的神经起到了确切的作用，就像我们对计算机中的单个电线所起的作用一样。

任何进入计算机的东西都会被转换成一种或另一种代码。

键盘每击键产生一字节的ASCII码，麦克风每秒产生44,100个二进制数，彩色照相机每像素产生三个二进制数，每秒30次，依此类推。

没有人在大脑中孤立地使用任何代码，如ASCII、二进制数、字体或指令代码。

他们可能在那里，但并没有被孤立。

没有人追踪到一个想法，也没有人在同一个W中找到一个内存

是的，我们可以跟踪计算机中程序的运行。

人们普遍认为，大脑的工作方式比一台计算机分散得多，有数千或数十亿个计算机元件相互协作并分担工作。但尚未找到这类分子。在计算领域，这个想法被称为“并行处理”，拥有数十或数百个CPU的计算机已经建成。

但这些电脑还没有产生人类的替代品。

把这一切想象成一个谜。 人们如何工作是难题的一个方面。

让电脑做人们做的事情是另一个难题。

拼图的两边都在拼凑。

问题是，随着双方都在取得进展，看起来越来越像这是两个不同的难题，他们不是走到一起的中间。它们并没有汇聚成一幅图画。

研究人员非常了解这些发展。

但谈到流行文化，人们总是听到新的发明，看到科幻电影描绘的未来，合乎逻辑的结论似乎是，研究将继续一个接一个地解决问题，直到10年、20年或30年后我们将有机电朋友。

在过去的一个世纪里，我们征服了电、飞行、太空旅行、化学、核能等等，那么为什么不征服大脑和/或思维呢？ T

然而，他的研究仍处于这样一个阶段：每找到一个新的答案，就会产生一个以上的新问题。

因此，无论我们从哪种角度看，大脑和头脑的工作原理都与我们所知的计算机不同。

我说“我们知道它们”是因为将来可能会发明其他类型的计算机。

但是，我们今天所有的计算机都属于“存储程序数字计算机”的定义，它们的所有操作原理都已在本书中介绍过。

然而，这些都不能“证明”一个合成的人类永远不可能被制造出来，这仅仅意味着本书中介绍的计算机原理不足以完成这项工作。

一些完全不同类型的设备，在一些完全不同的原则集上运行，可能能够做到这一点。但我们不能评论这样的设备，直到有人发明了一个。

回到一个简单的问题，你还记得乔和保温瓶吗？

他认为保温瓶里有某种温度传感器，里面有加热器和冷却器。

但即使它有所有的机器，它仍然不知道该做什么，它将只是一个机械设备，打开加热器或冷却器，取决于温度的饮料放在它。

现在我们知道了电脑里的东西，以及它是如何工作的，我认为很明显，“但它怎么知道？”这个问题的答案只是“它什么都不知道！”