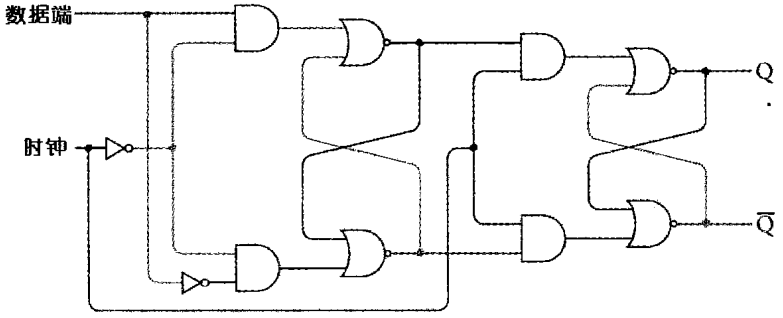
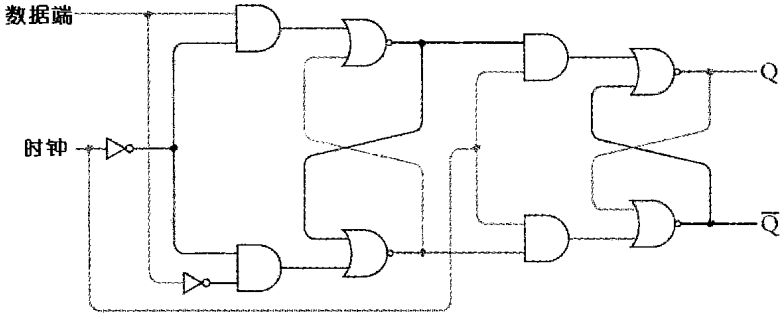
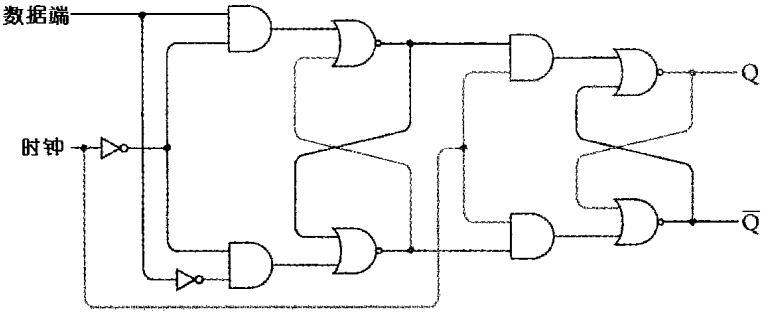


这改变了第一级触发器的状态，因为时钟输入取反变为 1。但第二级触发器状态保持不变，因为时钟输入仍然为 0。现在把时钟输入变为 1。



这就引起了第二级触发器输出的改变，使 Q 输出变为 1。不同点在于，无论数据端输入发生何种变化（比方说变为 0）都不会影响 Q 的输出。

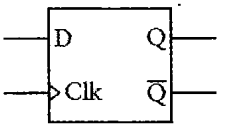


只有在时钟输入从 0 变化到 1 的瞬间 Q 和 $\overline{Q}$ 输出才发生变化。

边沿触发的 D 型触发器的功能表需要一个新的符号来表示从 0 到 1 的瞬时变化，即用一个向上的箭头（↑）表示，如下表所示。

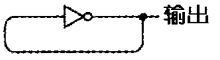
输入		输出	
D	Clk	Q	$\overline{Q}$
0	↑	0	1
1	↑	1	0
X	0	Q	$\overline{Q}$

表中箭头表示当时钟端由 0 变为 1 时（称为时钟信号的“正跳变”，“负跳变”是指从 1 变为 0），Q 端输出与数据端输入是相同的。触发器的符号如下图所示。

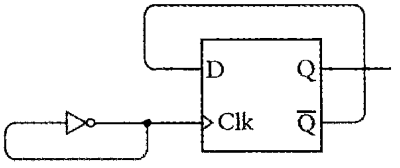


图中的小三角符号表示触发器是边沿触发的。

下面展示的是一个使用边沿 D 型触发器的电路，这个电路是不能用电平触发形式复制出来的。先回忆一下本章开始构造的振荡器，其输出在 0 和 1 之间变化。



把振荡器的输出与边沿触发的 D 型触发器的时钟端输入连接，同时把 $\overline{Q}$ 端输出连接到本身的 D 输入端。



这个触发器的输出同时又是它自己的输入。反馈紧接着反馈！（实际上，这种构造可能是有问题的，振荡器是由状态来回迅速改变的继电器构成的，其输出与构成触发器的继电器相连，而这些其他的继电器不一定能跟得上振荡器的速度。为了避免这些问题，这里假设振荡器中的继电器比电路中其他地方的继电器速度要慢得多）

仔细看一看下面的功能表就可以明白在电路中发生的情况了，电路启动时，假设时钟输入为 0 且 Q 输出也为 0，则 $\overline{Q}$ 端输出为 1，而 $\overline{Q}$ 是和 D 端输入相连的。

输入		输出	
D	Clk	Q	$\overline{Q}$
1	0	0	1

当时钟输入从 0 变为 1 时，Q 输出与 D 输入相同。

输入		输出	
D	Clk	Q	$\overline{Q}$
1	0	0	1
1	↑	1	0

但是由于 $\overline{Q}$ 的输出变为 0，因此 D 输入也变为 0。现在时钟输入为 1，如下表所示。

输入		输出	
D	Clk	Q	$\overline{Q}$
1	0	0	1
1	↑	1	0
0	1	1	0

当时钟输入变回 0 时，不会影响到输出，如下表所示。

输入		输出	
D	Clk	Q	$\overline{Q}$
1	0	0	1
1	↑	1	0
0	1	1	0
0	0	1	0

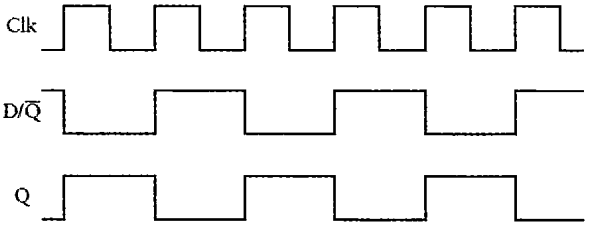
现在时钟端输入又变为 1。由于 D 输入为 0，那么 Q 输出为 0 且 $\overline{Q}$ 输出为 1。

输入		输出	
D	Clk	Q	$\overline{Q}$
1	0	0	1
1	↑	1	0
0	1	1	0
0	0	1	0
0	↑	0	1

所以 D 输入也变为 1，如下表所示。

输入		输出	
D	Clk	Q	$\overline{Q}$
1	0	0	1
1	↑	1	0
0	1	1	0
0	0	1	0
0	↑	0	1
1	1	0	1

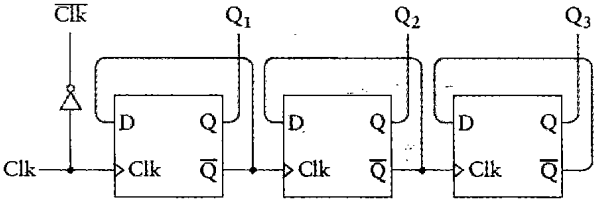
以上发生的现象可以简单总结为：每当时钟输入由 0 变为 1 时，Q 端输出就发生变化，或者从 0 到 1，或者由 1 到 0。下面的时序图可以更加清楚地说明这个问题。



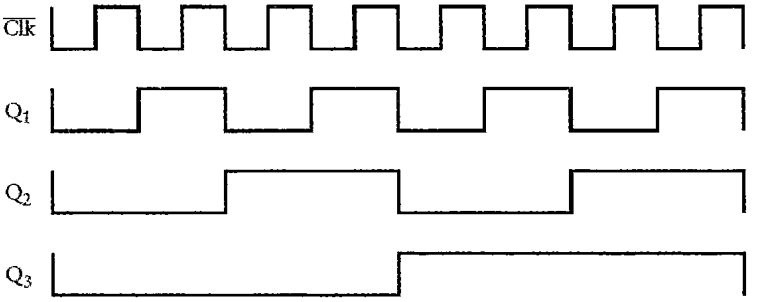
当时钟端 Clk 输入由 0 变为 1 时，D 的值（与 $\overline{Q}$ 的值相同）被输出到 Q 端。当下一次 Clk 信号由 0 变为 1 时，D 和 $\overline{Q}$ 的值同样会改变。

如果这个振荡器的频率是 20Hz（即 20 个周期的时间为 1s），那么 Q 的输出频率是它的一半，即 10Hz，由于这个原因，这种电路称为分频器（frequency divider），它的 $\overline{Q}$ 输出反馈到触发器的数据端输入 D。

当然，分频器的输出可以作为另一个分频器的 Clk 输入，并再一次进行分频。下面是三个分频器连接在一起的示意图。

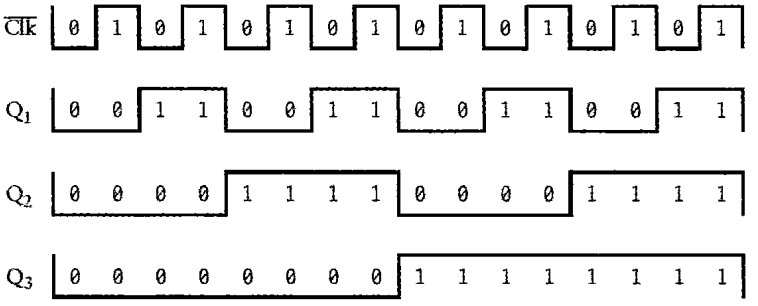


上图顶部的 4 个信号变化规律如下图所示。



这里只给出了这幅图的一部分，因为这个电路会重复上述过程周而复始地变化下去。在这幅图中，你有没有发现眼熟的东西呢？

提示一下，把这些信号标上 0 和 1。



现在看出来了么？试着把这个图顺时针旋转 90° ，然后读一读每一行的 4 位数字，它们分别对应了十进制中的 0～15 中的一个数。

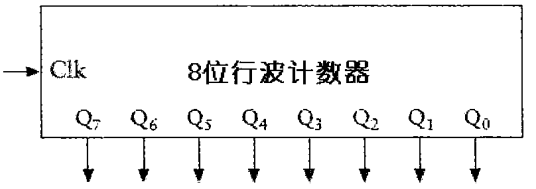
二进制	十进制
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9

续表

1010	10
1011	11
1100	12
1101	13
1110	14
1111	15

可以看出，这个电路不仅仅具备了一个计数功能。当然，如果在这个电路中添加更多的触发器，其计数范围就会更大。在第 8 章中提到一个顺序递增的二进制序列，每一列数字在 0 和 1 之间的变化频率是其右边那一列数字变化频率的一半，这个计数器就是模仿了这一点。在每一次时钟信号的正跳变时，计数器的输出是增加的，即递增 1。

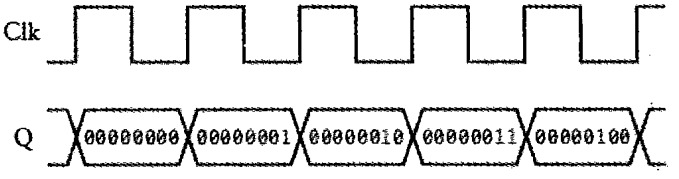
把 8 个触发器连接在一起，然后放入一个盒子中，构成了一个 8 位计数器。



这个计数器称为“8 位行波计数器”，因为每一个触发器的输出都是下一个触发器的时钟输入。变化是在触发器中一级一级地顺序传递的，最后一级触发器的变化必定会有一些延迟，更先进的计数器是“并行（同步）计数器”，这种计数器的所有输出是在同一时刻改变的。

在计数器中输出端用 $Q_0 \sim Q_7$ 标记，在最右边的 Q_0 是第一个触发器的输出。如果将灯泡连到这些输出端上，就可以将 8 位数字读出来。

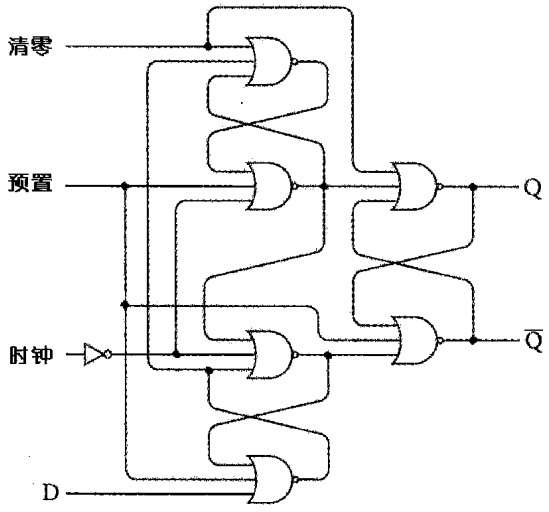
这样一个计数器的时序图可以将 8 个输出分别表示出来，也可以将它们作为整体一起表示出来，如下图所示。



时钟信号的每一个正跳变发生时，一些 Q 输出可能会改变，而另外一些可能不变，但总体上来说它们所表示的二进制编码递增了 1。

本章前面提到过可以找到某种方法来确定振荡器频率，现在已经找到这种方法了。如果把一个振荡器连接到 8 位计数器的时钟输入端上，那么这个计数器会显示出振荡器经过的循环次数。当计数器总数达到 11111111（十进制的 255），它又返回为 00000000。使用计数器确定振荡器频率的最简单的方法就是把计数器的 8 个输出端分别接到 8 只灯泡上。当所有的输出都是 0 时（即所有灯泡都是熄灭的），启动一个秒表计时；当所有灯泡都点亮时，停止秒表计时。这就是振荡器循环 256 次所需要的时间。假设这个时间为 10s，则振荡器的频率是 256 ÷ 10，即 25.6 Hz。

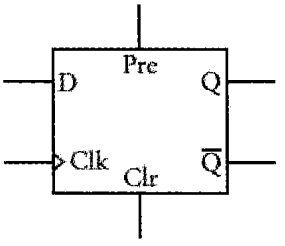
随着触发器功能的增加，它的结构也变得更加复杂，下面给出了一个带预置和清零功能的边沿型 D 触发器。



通常情况下，预置和清零信号输入会覆盖时钟和数据端输入，且两个输入都为 0，当预置信号为 1 时，Q 变为 1，$\overline{Q}$ 变为 0。当清零信号为 1，Q 变为 0，$\overline{Q}$ 变为 1（同 R-S 触发器中的 S 和 R 输入端一样，预置和清零信号不能同时为 1）。除此之外，该触发器工作原理是和普通边沿 D 触发器是一样的。

输入				输出	
Pre	Clr	D	Clk	Q	$\overline{Q}$
1	0	X	X	1	0
0	1	X	X	0	1
0	0	0	↑	0	1
0	0	1	↑	1	0
0	0	X	0	Q	$\overline{Q}$

电路图符号可以简单地用下图来代替。



现在，我们已经懂得如何使用继电器来做加法、减法和计数了，这是一件很有成就感的事情，因为我们使用的硬件是 100 多年前就存在的东西。我们还有更多未知领域要去探索，请稍事休息，停止思考那些构造方面的问题，回过头再来看看数字方面的问题吧。

15 字节与十六进制

通过对上一章中两类改进加法机的剖析，我们学习了数据路径（Data Path）这个概念。纵观整个电路的脉络，每 8 个比特流为一组，如潺潺溪流般在器件与器件之间流动。其实，这 8 位比特流就是加法器、锁存器以及数据选择器的输入形式，同时它也是这些器件单元的输出形式。这 8 位的比特流可以用开关的不同状态组合所定义，而且可以用灯泡的亮灭来显示。这样一来，这些电路中数据路径的位宽（bits wide）就是 8。为什么我们要把它定义为“8”位呢？为什么没有定义为 6 位、7 位、9 位或 10 位呢？

要想用偷懒的方式回答上面这个问题也很简单，答案就是这些加法机都是对第 12 章中原始加法机的改进，而它所“流传”下来的位宽恰好就是 8 位。但是追根究底，原始加法机偏偏是 8 位其实没有什么特别的原因。只不过在每次使用 8 位位宽时，一切工作都显得非常方便——一种优雅的比特化（biteful）的比特流。或许大家已经感觉到我在极力隐瞒一些东西，无法否认，其实我心中一直都很清楚（或许你也知道）：8 比特代表一个字节（byte）。

字节这个词最早起源于 1956 年前后，由 IBM 公司提出。最早的拼写方式是 bite，但为了避免与 bit 混淆用 y 代替了 i。曾几何时，字节仅表示某一数据路径上的位数，直到 20 世纪 60 年代中叶，在 IBM 的 360 系统的发展下（一种大规模复杂的商用计算机），字节这个词逐渐开始用来表示一组 8 比特数据。

由于有 8 位，一个字节的取值范围为 00000000 到 11111111。相应地，它还可以表示成为 0～255 之间的正整数，如果将一个数的补码作为其相对应的负数，那么一个字节可以表示在–128～127 范围内的正、负整数。一个给定的字节可以代表 2^8，即 256 种不同事物中的一个。

让我们再来看看“8”这个数字，当 8 作为比特流的一种尺度，它的确表现出了非常完美的特质。字节在很多方面都比单独的比特更胜一筹。IBM 采用字节有一个很重要的原因，就是这样一来数字就可以按照 BCD 形式（第 23 章中将会讲述）方便地保存。在本章随后的讲述中，我们会发现凑巧的是：全世界大部分书面语言（除了中文、日文以及韩文中使用的象形文字体系）的基本字符数都少于 256，所以字节是一种理想的保存文本的手段。字节同样适合表示黑白图像中的灰度值，这是由于肉眼能区分的灰度约为 256 种。当一个字节无法表示所有信息（如刚提到的中文、日文以及韩文中使用的象形文字体系等），我们只需采用两个字节——就可以表示 2^{16} 也就是 65536 个不同的物体——这也是一种很好的解决方案。

字节的一半——即 4 比特——我们称之为半字节（nibble，也可拼写成 nybble），在计算机这个领域，它并不像字节那样经常使用。

由于字节在计算机内部出现的频率较高，如果可以使用一种简洁的方式将它的内在含义准确表达出来，将会为我们带来很多方便。假如一个 8 位二进制数表示为 10110110，这种表达方式自然而又直观，但它还不够简洁。

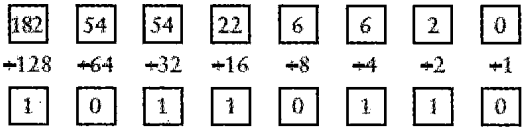
我们完全可以采用十进制表示法来表示字节，但从二进制转换到十进制需要进行一系列计算——计算方法并不复杂，但是比较麻烦。我曾在第 8 章曾介绍过一种直观的计算方法。因为每一位二进制数对应着 2 的不同幂，因此我们可以把二进制数写到第一行，在底下写出每一位二进制数对应的 2 的乘方数，然后采用“列相乘、行相加”就可以得到对应的十进制数。下图表示了 10110110 的转换过程。

1	0	1	1	0	1	1	0
×128	×64	×32	×16	×8	×4	×2	×1

128 + 0 + 32 + 16 + 0 + 4 + 2 + 0 = 182

把十进制数转换为二进制数就需要一点点技巧了。我们可以用这个十进制数不断除以递减排列的 2 的幂，每除一次得到的商就是所要求的二进制数中的一位，而余数成为

下一次运算的除数，它的作用是除以下一个较小一点的 2 的幂。下图表示了十进制数 182 转换成二进制形式的过程。



第 8 章对这种计算方法有更详细的描述。不过，在十进制数和二进制数之间进行转换还需用到一些工具，例如要借助笔和纸进行一系列演算。

第 8 章中我们还学到了八进制数，也称为八进制数字系统。这种系统仅使用到了数字 0、1、2、3、4、5、6 还有 7。八进制数和二进制数之间的转换简洁方便，只要记住 0～7 这 8 个数字所对应的 3 位二进制数即可。下面这张表就表示了这种对应关系。

二进制数	八进制数
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

假如要把一个二进制数（如 10110110）转化为 8 进制，可以从最右端的数字开始。每 3 比特看做一组，这样每组便对应着一个八进制数：

$$\underbrace{10}_{2}\underbrace{110}_{6}\underbrace{110}_{6}$$

10110110 这个字节很容易就表示为八进制数 266。这种方法简洁明了，八进制用来表示字节不失为一个好方法。但还是有那么一点美中不足。

字节可以表示的二进制数的范围为 00000000～11111111，如果采用八进制表示，那么相对应的范围也随之变成了 000～377。仔细分析下先前的例子，我们从右到左把 3 位二进制数对应于中间以及最靠右的八进制数，而最靠左的八进制数却是由 2 位二进制数对

应的。如果我们将 16 位二进制数直接表示为八进制会得到如下结果：

$$\underbrace{10}_{1}\underbrace{110}_{3}\underbrace{011}_{1}\underbrace{110}_{7}\underbrace{001}_{0}\underbrace{101}_{5}$$

如果我们把这个 16 位二进制数平分为两个字节并将其分别表示为八进制数会得到如下所示的不同结果：

$$\underbrace{10110011}_{2\ 6\ 3} \qquad \underbrace{11000101}_{3\ 0\ 5}$$

为了使多字节值能和分开表示的单字节值取得一致，我们需要一种可以等分单个字节的系统，按照这种思想，我们可以把每个字节等分成 4 组，每组 2 比特（基于 4 的计数系统）；还可以等分为 2 组，每组 4 比特（基于 16 的计数系统）。

基于 16 的计数系统（Base 16），对于我们而言是一种全新的系统。基于 16 的计数系统也被称为十六进制（hexadecimal），这个单词很容易让我们产生混淆。这是因为很多以 hexa-为前缀的单词（比如 Hexagon， Hexapod 以及 Hexameter）都会与 6 或多或少有些联系。而 hexadecimal 这个词却偏偏代表了 16（Sixteen）这个含义。虽然在《微软出版物风格与技术手册》（*The Microsoft Manual of Style for Technical Publications*）中明确说明“请勿将十六进制缩写为 hex”，但包括我在内的绝大多数人还总是在不经意间使用这种缩写。

十六进制还有其他一些特别之处。比如在十进制中，我们通常可以用下面这种方式计数：

0 1 2 3 4 5 6 7 8 9 10 11 12...

我们仔细回忆一下在八进制中，8 和 9 这两个数字是不需要的，就像下面这样：

0 1 2 3 4 5 6 7 10 11 12...

同理，四进制计数中我们也不需要 4、5、6、7 这些数字，就像下面这样：

0 1 2 3 10 11 12...

在二进制中，这一切变得更加简单，我们所需要的只有 0 和 1：

0 1 10 11 100...

十六进制计数和上面所讲的这一系列情况是完全不同的，它需要比十进制更多的数

字来计数。十六进制的计数过程将会是下面这种形式：

0 1 2 3 4 5 6 7 8 9 ? ? ? ? ? ? 10 11 12...

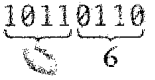
上图中 10（准确地来说是一个 1 紧挨着一个 0）这个数字代表的准确含义实际上应该是十进制中的 16。图中的问号表明我们还需要 6 个符号来完整表示十六进制系统。这些符号究竟是什么？它们出于何处？仔细想想，原有的计数系统中的符号都有自己的唯一身份，所以应该而且必须引入新的符号。寻找 6 个新符号对我们来说是小菜一碟，例如下面这几个符号：

不像我们所使用的大多数数字符号，上面列出的每个符号都很容易识记，而且其背后都隐含着实际数字意义，这种形象的符号使得它们便于记忆。比如这个 10 加仑的牛仔帽、一个橄榄球（11 个人组成一支橄榄球队）、一打（12 个）面包圈、一只黑猫（使人们想起不吉利的 13）、一轮满月（一般出现在后弦月 14 天之后的夜晚），一把匕首让人们联想到凯撒大帝（Julius Caesar）在三月的月中（第 15 日）被刺杀。

两个十六进制数可以完整地代表一个字节。这也意味着一个十六进制数恰好由 4 位二进制数组成，即半字节。下面这张表描述了如何在二进制、十六进制、十进制数之间进行转换。

二进制	十六进制	十进制	二进制	十六进制	十进制
0000	0	0	1000	8	8
0001	1	1	1001	9	9
0010	2	2	1010		10
0011	3	3	1011		11
0100	4	4	1100		12
0101	5	5	1101		13
0110	6	6	1110		14
0111	7	7	1111		15

让我们再来看看如何把字节 10110110 转换成十六进制。



无论是多字节还是单字节，我们都可以进行转换。

一个字节能且只能由一对十六进制数来表示。

只可惜（或许你也松了口气）我们不打算用橄榄球或一打面包圈来表示十六进制数。虽然这种方案完全可行，但它不是很方便也不够正规，有时还会让人感到迷惑。事实上十六进制中缺少的 6 个符号由 6 个拉丁字母来表示，就像下面这样：

0 1 2 3 4 5 6 7 8 9 A B C D E F 10 11 12...

下面这张表描述了正规的二进制、十六进制、十进制转换的过程。

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

这样一来字节10110110就可以表示为十六进制的B6，而不用画一个橄榄球那么麻烦。我们再回忆回忆前面的章节，下面举一个完整的进制转换的例子（用下标代表进制），比如：

10110110_{TWO}（10110110_2）

它的四进制形式表示为：

2312_{FOUR}（2312_4）

它的八进制形式表示为：

266_{EITHT}（266_8）

它的十进制形式表示为：

182_{TEN}（182_{10}）

与此类似，它的十六进制形式可以表示为：

$B6_{SIXTEEN}$（$B6_{16}$）

但是这种写法有点累赘，幸好还有几种表达十六进制数通用方法。你还可以用如下方法表示：

$B6_{HEX}$

在本书中将使用一种更加简洁实用的方法，那就是用一个小写的 h 紧跟在数字后边表示这个数是以十六进制表示的，就像这样：

B6h

通过分析可以得到十六进制数的每一位代表 16 的不同整数幂的倍数，如下图所示。