二进制	十六进制	字符	二进制	十六进制	字符	二进制	十六进制	字符
00100000	20	[Space]	01000000	40	@	01100000	60	`
00100001	21	!	01000001	41	A	01100001	61	a
00100010	22	"	01000010	42	B	01100010	62	b
00100011	23	#	01000011	43	C	01100011	63	c
00100100	24	$	01000100	44	D	01100100	64	d
00100101	25	%	01000101	45	E	01100101	65	e
00100110	26	&	01000110	46	F	01100110	66	f
00100111	27	'	01000111	47	G	01100111	67	g
00101000	28	(	01001000	48	H	01101000	68	h
00101001	29	)	01001001	49	I	01101001	69	i
00101010	2A	*	01001010	4A	J	01101010	6A	j
00101011	2B	+	01001011	4B	K	01101011	6B	k
00101100	2C	,	01001100	4C	L	01101100	6C	l
00101101	2D	-	01001101	4D	M	01101101	6D	m
00101110	2E	.	01001110	4E	N	01101110	6E	n
00101111	2F	/	01001111	4F	O	01101111	6F	o
00110000	30	0	01010000	50	P	01110000	70	p
00110001	31	1	01010001	51	Q	01110001	71	q
00110010	32	2	01010010	52	R	01110010	72	r
00110011	33	3	01010011	53	S	01110011	73	s
00110100	34	4	01010100	54	T	01110100	74	t
00110101	35	5	01010101	55	U	01110101	75	u
00110110	36	6	01010110	56	V	01110110	76	v
00110111	37	7	01010111	57	W	01110111	77	w
00111000	38	8	01011000	58	X	01111000	78	x
00111001	39	9	01011001	59	Y	01111001	79	y
00111010	3A	:	01011010	5A	Z	01111010	7A	z
00111011	3B	;	01011011	5B	[	01111011	7B	{
00111100	3C	<	01011100	5C	\	01111100	7C	|
00111101	3D	=	01011101	5D	]	01111101	7D	}
00111110	3E	>	01011110	5E	^	01111110	7E	~
00111111	3F	?	01011111	5F	_	01111111	7F	[Delete]

用数字格式表示文本相当简单，像ASCII这样，系统把每个字符或符号映射成一个唯一的位序列，然后，计算设备解释这个位序列并把合适的符号显示给用户。

2.1.3 数字颜色和图像

前面讲解了如何用二进制表示数字和文本，现在探索另一种类型的数据：颜色。任何具有彩色图像显示器的计算设备都要具备一些描述颜色的系统。如你所想，和文本一样，我们已经有了一些存储颜色数据的标准方式。我们稍后将介绍这些系统，但首先我们先来自己设计一下用数字描述颜色的系统。

我们把颜色局限在黑色、白色和灰色。这种有限的颜色集称为灰度。如同处理文本一样，我们先确定想要表示的不同灰色的数量。为简单起见，我们选择黑色、白色、深灰色和浅灰色，一共4种灰度。表示4种颜色需要多少位呢？只需要2位。2位数可以表示4种不同的值，因为2^2等于4。

练习2-3：创建自己的系统来表示灰度

定义一种方式来数字化表示黑色、白色、深灰色和浅灰色。这里的正确答案不止一个，示例答案请参阅附录A。

设计好用二进制表示灰度的系统后，便可以在这个基础之上创建自己的系统来描述简单的灰度图像。图像本质上是二维平面上的颜色排列。这些颜色通常排列在一个网格上，网格由单一颜色的小方块构成，这种小方块被称为像素。图2-1就是一幅简单的图像。

图2-1中的图像长度为4个像素，高度为4个像素，一共是16个像素。如果你眯起眼睛并发挥想象力，你可能会看见一朵白色的花和一片黑色的天空。这个图像只包含三种颜色：白色、浅灰色和深灰色。

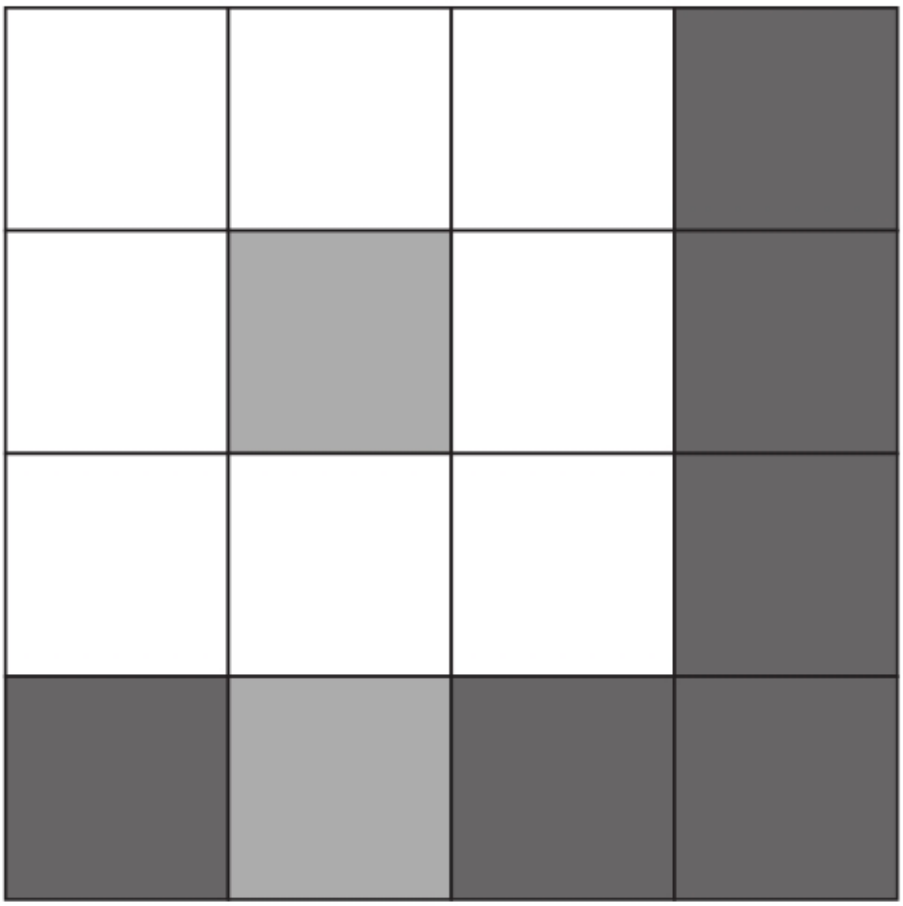

图2-1 一幅简单的图像

注意

图2-1由一些非常大的像素构成，以显示一个点。现代电视、计算机显示器和智能手机屏幕也可以被视为像素网格，但是每个像素非常小。例如，高分辨率显示器一般都是1920×1080像素（长度×高度），总共大约有200万像素！再比如，数码照片常常包含超过1000万像素。

练习2-4：创建自己的方法来表示简单图像

（1）在练习2-3表示灰度颜色的系统基础上，设计一种方法来表示由这些颜色构成的图像。如果想要更简单，你可以假设图像始终是4×4像素的，就像图2-1一样。

（2）使用第1步得到的方法，写出图2-1中花朵图像的二进制表示。

（3）向朋友解释你表示图像的方法，然后把你的二进制数据给你的朋友看，看看他们是否能在不看原图的情况下画出图像。

这里不存在唯一正确的答案，示例答案请参阅附录A。

在练习2-4的第2步中，你就像计算机程序，负责把图像编码成二进制数据。在第3步中，你朋友就像负责反向操作的计算机程序，把二进制数据解码为图像。希望他能破译你的二进制数据并画出一朵花！如果他成功了，那么太棒了，你们一起演示了软件是如何编码和解码数据的！如果进展得没有这么顺利，最后画出来的像腌菜而不像花朵，那也行。此时，你们就演示了有时软件会有缺陷，从而导致意想不到的结果的情况。

2.1.4 表示颜色和图像的方法

正如前面提到的，业内已经定义了用数字方式表示颜色和图像的标准方法。对于灰度图像，一个常用方法是每个像素使用8位表示，总共允许表示256种灰度。每个像素的值通常代表光的强度，0表示没有光（全黑色），255表示完全光（全白色），0～256之间的值就代表从全黑到全白的各种灰度。图2-2显示了使用8位编码方案的各种灰度。

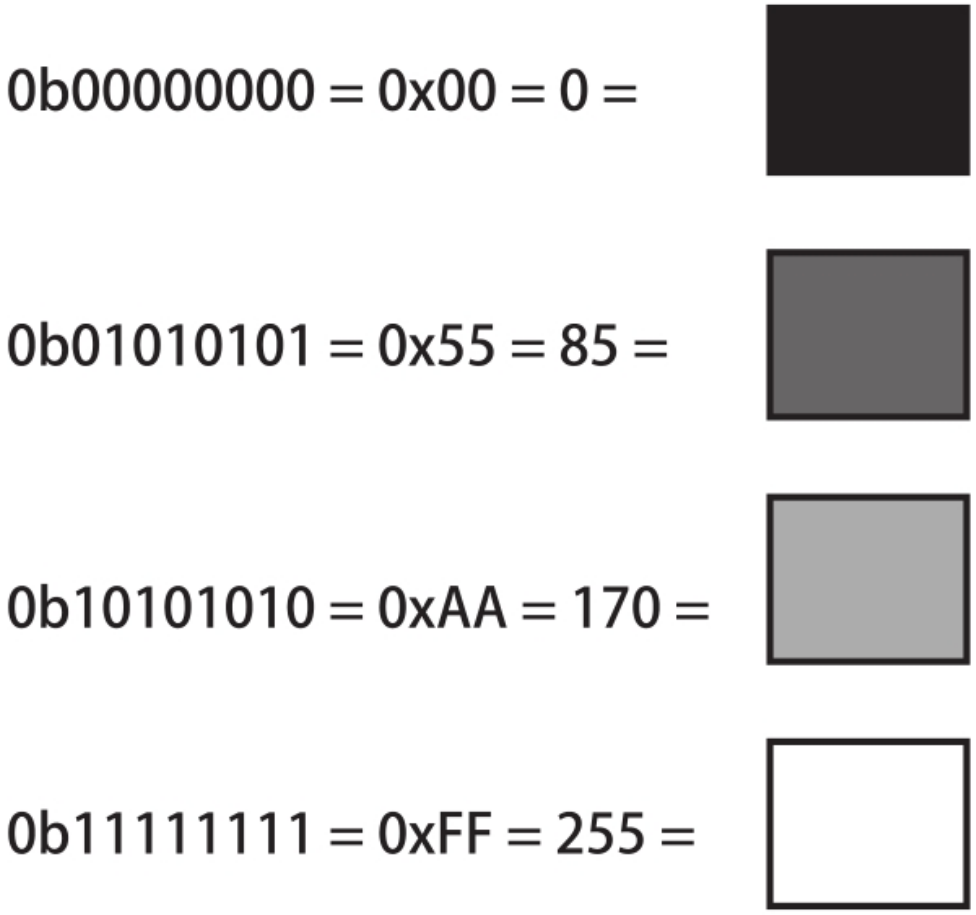

图2-2 用8位表示的灰度（显示为二进制、十六进制和十进制）

表示除灰度以外的颜色的方法与之类似。虽然可以使用8位数来表示灰度，但是RGB方法使用3个8位数来表示红色、绿色和蓝色的强度，这3个数合起来表示一种颜色。每个颜色分量都需要分配8位，这就意味着需要24位来表示整体颜色。

注意

RGB以加色模型为基础，颜色是红光、绿光和蓝光混合而成的。这与绘画中的减色模型相反，后者的颜色分量是红色、黄色和蓝色。

比如，RGB中表示红色是把8个红色位全部置1，表示其他两种颜色的16位全部置0。如果你想表示黄色（黄色是红色和绿色的混合色，没有蓝

色），那么可以把红色和绿色位全部置1，把蓝色位全部置0，如图2-3所示。

在图2-3的两个例子中，被包含的颜色都设为1，但RGB系统也允许红、蓝和绿组成色为部分强度。每个组成色可以从00000000（十进制的0或十六进制的0）变化到11111111（十进制的255或十六进制的FF）。值越小表示对应颜色的光越暗，值越大表示光越亮。有了这种灵活性，我们就可以表示几乎所有可以想象到的颜色。

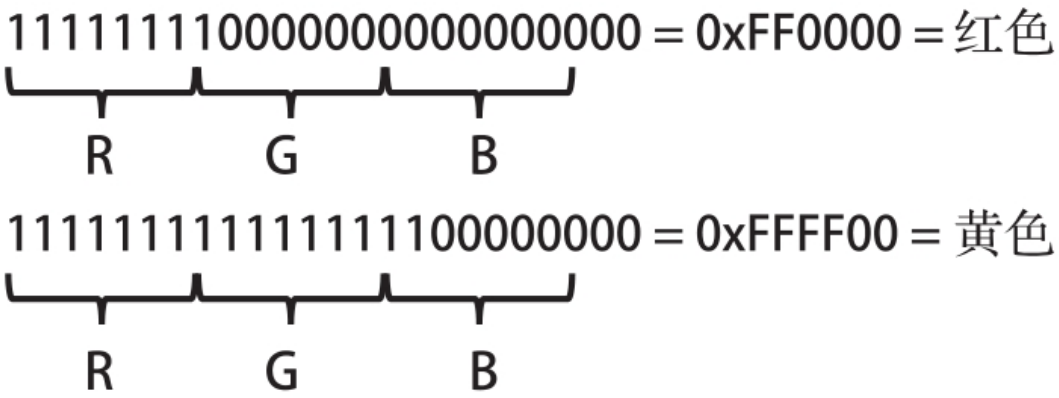

图2-3 用RGB表示红色和黄色

除了有表示颜色的标准方法之外，还有多种常用方法来表示整幅图像。如图2-1所示，我们可以用像素网格来构造图像，其中的每个像素都被设置为特定的颜色。现有多种图像格式，位图（bitmap）是其中一种简单的图像表示方法，位图中保存了每个像素的RGB颜色数据。与位图相比，其他图像格式（比如JPEG和PNG）使用了压缩技术来减少存储图像所需的字节数。

2.1.5 解释二进制数据

现在来看另一个二进制值：01100001011000100110011。你觉得它表示的是什么？如果我们假设它是一个ASCII文本字符串，它表示的就是“abc”。它也可能表示一个24位的RGB颜色，这使得它成为一个灰度值。它

还可能表示一个正整数，即十进制数6382179。图2-4给出了该值的各种解释。

那么，到底指哪个呢？它可以是其中的任何一种意思，也可能完全是其他的意思。这完全取决于解释这个数据的上下文。文本编辑程序会假设这个数据是文本，而图像查看器则会假设它是图像中某个像素的颜色，计算器会假设它是一个数字。每个程序都被编写成期待特定格式的数据，所以单个二进制值在不同的上下文中会有不同的含义。

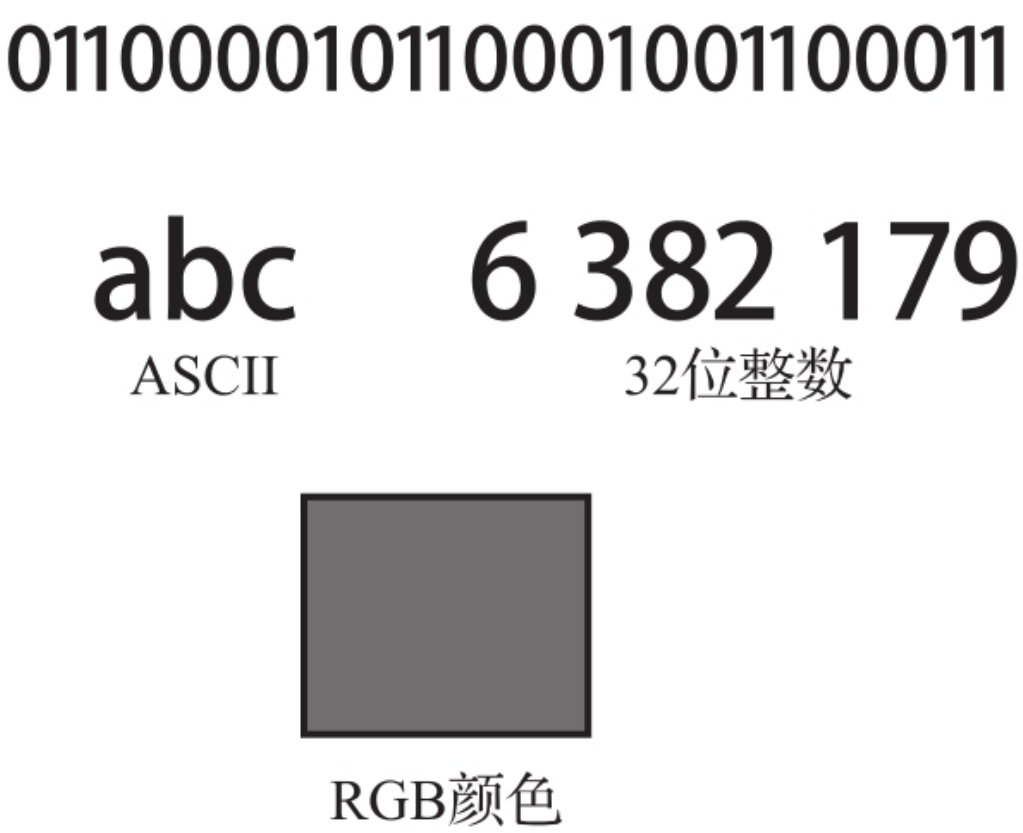

图2-4 01100001011000100110011的解释

前面演示了如何用二进制数据表示数字、文本、颜色和图像。由此，你可以对存储其他类型数据（比如音频和视频）的方法做出一些有根据的猜想。对于可数字化的数据类型，并没有具体限制。而数字表示也并不总是原始数据的完美复制，但在许多情况下，这并不是问题。能把任意事物

解释成0/1序列是非常有用的，因为一旦我们建造了一个处理二进制数据的设备，我们就可以通过软件让它处理任何类型的数据！

2.2 二进制逻辑

我们已经了解了用二进制表示数据是有其实用性的，但计算机所做的不只是存储数据，它也允许我们处理数据。有了计算机的帮助，我们可以阅读、编辑、创建、转换、共享并用其他方式操作数据。计算机使我们能利用硬件以多种方法来处理数据，我们可以通过编程来执行一系列简单指令——比如"两个数相加"或"检查两个值是否相等"。实现这些指令的计算机处理器基本上都是以二进制逻辑为基础的，这是一种描述逻辑语句的系统，其中的变量取值只能二选一：真或假。现在，我们来研究一下二进制逻辑，在此过程中，我们将再次看到计算机中的所有事物是如何归结到1和0的。

让我们考虑一下二进制为什么天然就适合进行逻辑运算。通常，当人们说到逻辑时，他们的意思是推理，即对已知条件进行思考以得出有效的结论。当一组事实被呈现时，逻辑推理可以让我们确定另一个相关陈述是否是事实。逻辑是关于真理的——什么是真，什么是假。同样，一个位的值也只能二选一：1或0。因此，一个位可以被用来表示一个逻辑状态：真（1）或假（0）。

我们来看一个逻辑语句的例子：

```
GIVEN a shape has four straight sides,
AND GIVEN the shape has four right angles,
I CONCLUDE that the shape is a rectangle.
```

这个例子有两个条件（4条边、4个直角），这两个条件都为真，结论才为真。对于这种情况，我们用逻辑运算符AND（"与"）把两个语句连接在一起。如果任意一个条件为假，那么结论也为假。表2-2给出了相应的逻辑结论。

表2-2 判断是否为矩形的逻辑语句

4 条边	4 个直角	是否为矩形
假	假	假
假	真	假
真	假	假
真	真	真

对于表2-2，我们可以对每一行做如下解释：

❑如果图形没有4条边，也没有4个直角，那么它不是矩形。

❑如果图形没有4条边，但有4个直角，那么它不是矩形。

❑如果图形有4条边，但没有4个直角，那么它不是矩形。

❑如果图形有4条边，也有4个直角，那么它是矩形！

这种类型的表被称为真值表：展示所有可能的条件组合（输入）及其逻辑结论（输出）的表格。表2-2是专门为关于矩形的语句编写的，但实际上，同样的表适用于任何用AND连接的逻辑语句。

表2-3更加通用，它使用*A*和*B*来表示两个输入条件，输出表示逻辑结果。具体说来，输出是*A* AND *B*的结果。

表2-4做了一点修改。因为本书是关于计算机的，所以把“假”表示为0，“真”表示为1，就像计算机一样。

▼表2-3 AND真值表

A	*B*	输出
假	假	假
假	真	假
真	假	假
真	真	真

▼表2-4 AND真值表（用0和1表示）

A	*B*	输出
0	0	0
0	1	0
1	0	0
1	1	1

当你处理使用0和1的数字系统时，表2-4是AND真值表的标准形式。计算机工程师用这样的表来表示组件在面对一组特定输入时的行为。现在，让我们来看一下它是如何处理其他逻辑运算符和更复杂的逻辑语句的。

假设你现在工作的商店只给两类顾客（儿童和戴太阳镜的人）打折。其他人都没有资格享受折扣。如果你想把商店的折扣规则声明为一个逻辑表达式，你可以这样说：

```
GIVEN the customer is a child,
OR GIVEN the customer is wearing sunglasses,
I CONCLUDE that the customer is eligible for a discount.
```

这里有两个条件（儿童、戴太阳镜），至少有一个条件必须为真，结论才为真。在这种情况下，我们用逻辑运算符OR（“或”）把两个语句连接在一起。只要任意一个条件为真，那么结论就为真。我们可以将其表示为一个真值表，如表2-5所示。

表2-5 OR真值表

A	*B*	输出
0	0	0
0	1	1
1	0	1
1	1	1

观察表2-5中的输入和输出，我们可以很快地看出：当顾客是儿童（*A*=1），或者当顾客戴太阳镜（*B*=1）时，顾客能获得折扣（输出=1）。注意，表2-4和表2-5的*A*、*B*输入列的值是完全相同的。这是有意义的，因为这两个表都有两个输入，所以可能有一样的输入组合。不同的是输出列。

下面看用AND和OR组成的更复杂的逻辑语句。在这个例子中，假设我在阳光灿烂且温暖的每一天都要去海滩，并且假设我还在每年生日的时候去海滩。实际上，我总是只在这些特定条件下才会去海滩——我的妻子说我在这方面非常固执。结合这些条件，我们得到如下逻辑语句：

```
GIVEN it is sunny AND GIVEN it is warm,
OR GIVEN it is my birthday,
I CONCLUDE that I am going to the beach.
```

我们来标记输入条件，并为这个表达式编写一个真值表。这个例子的条件有：

条件*A* 阳光灿烂

条件*B* 天气温暖

条件*C* 今天是我的生日

逻辑表达式如下：

```
(A AND B) OR C
```

如同在代数表达式中一样，给*A* AND *B*加上括号意味着表达式的这部分应该先求值。表2-6给出了这个逻辑表达式的真值表。

表2-6比简单的AND真值表要复杂一点，但它仍然是可以理解的。表格的形式使其便于查找特定条件并查看结论。例如，第三行告诉我们如果*A*=0（阳光不灿烂），*B*=1（天气温暖），*C*=0（今天不是我的生日），那么输出为0（今天我不会去海滩）。

表2-6 （*A* AND *B*）OR *C*真值表

A	*B*	*C*	输出
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

这种逻辑是计算机经常要处理的。实际上，如前所述，计算机的基本功能可以浓缩为一组逻辑运算。虽然简单的AND运算符看上去与智能手机或笔记本计算机的功能相去甚远，但所有数字计算机都是基于逻辑运算的。

练习2-5：为逻辑表达式编写真值表

表2-7给出了一个逻辑表达式的3个输入。对于表达式（*A* OR *B*) AND *C*，给出该真值表的输出。

表2-7 （*A* OR *B*）AND *C*真值表

A	*B*	*C*	输出
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

除了AND和OR，数字系统设计中还有其他几种常用的逻辑运算符。下面将介绍每一种运算符并给出其真值表。第4章会再次用到这些知识。

逻辑运算符NOT（“非”）的输出与输入条件相反。也就是说，如果*A*是真，那么输出就不为真，反之亦然。如表2-8所示，NOT只有一个输入。

表2-8 NOT真值表

A	输出
0	1
1	0

运算符NAND（“与非”）的意思NOT AND，因此其输出是对AND结果取反的结果。如果两个输入都为真，则结果为假，否则，结果为真，如表2-9所示。

NOR（“或非”）运算符的意思是NOT OR，因此其输出是对OR结果取反的结果。如果两个输入都为假，则结果为真，否则，结果为假，如表2-10所示。

▼表2-9 NAND真值表

A	*B*	输出
0	0	1
0	1	1
1	0	1
1	1	0

▼表2-10 NOR真值表

A	*B*	输出
0	0	1
0	1	0
1	0	0
1	1	0

XOR的意思是异或，意味着只有单个（互斥）输入为真，结果才为真。也就是说，仅当*A*为真或仅当*B*为真时，输出为真，如果两个输入都为真，则结果为假，如表2-11所示。

表2-11 XOR真值表

A	*B*	输出
0	0	0
0	1	1
1	0	1
1	1	0

对二值变量（取真或假）逻辑功能的研究被称为布尔代数或布尔逻辑。George Boole在19世纪初描述了这种逻辑方法，那时还没有数字计算机。他的成果被证明是数字电子学（包括计算机）的发展基础。