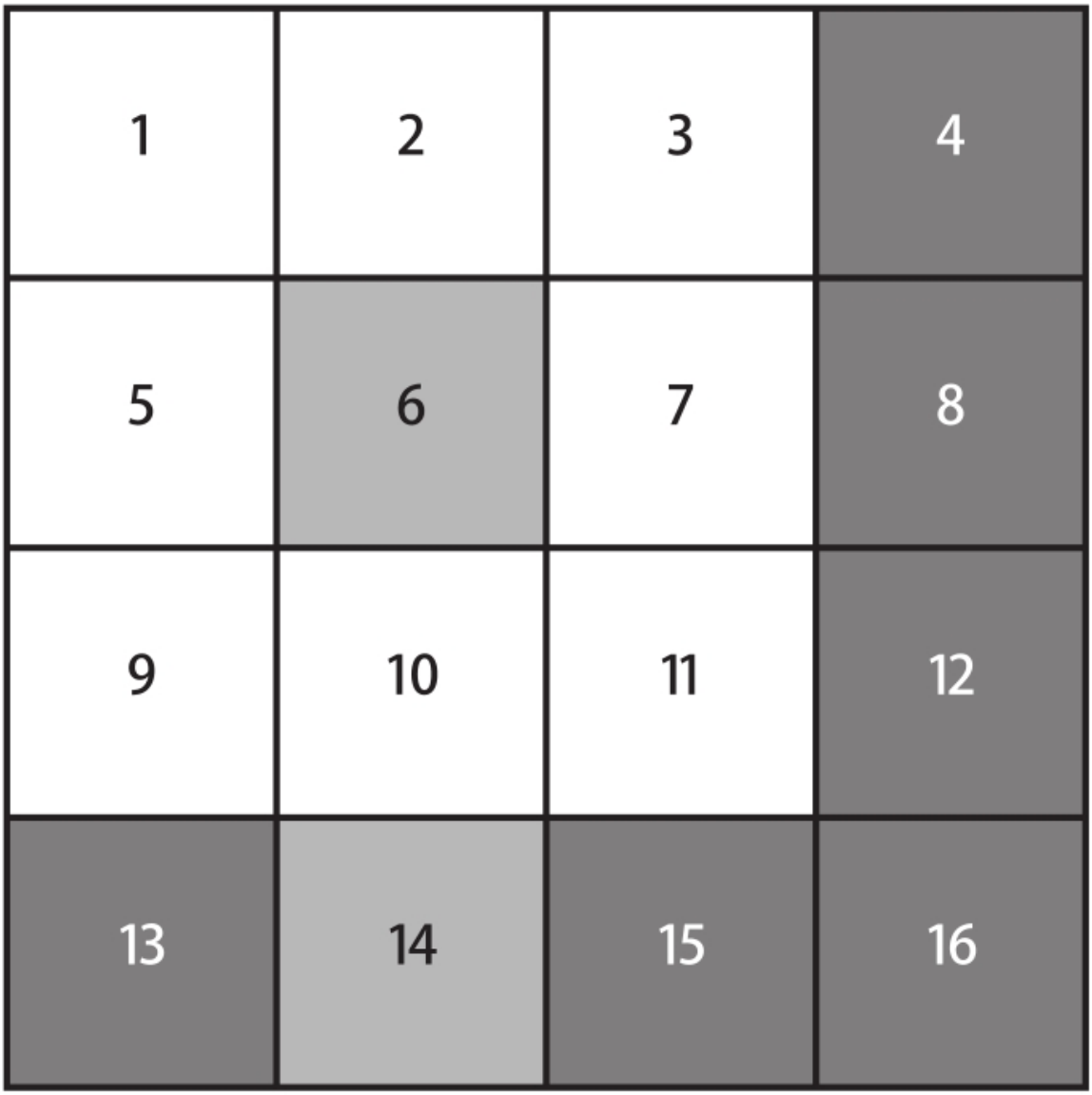
▲图A-1 示例图像格式中的像素顺序



▲图A-2 将4×4的图像网格覆盖在花朵图像上

注意，我编写了一个简单的网页来模拟该16像素和2位灰度值的特殊系统，详见https://www.howcomputersreallywork.com/grayscale/。

（3）答案：略

练习2-5：

答案：

（*A* OR *B*）AND *C*的真值表见表A-3

表A-3 （*A* OR *B*）AND *C*真值表解决方案

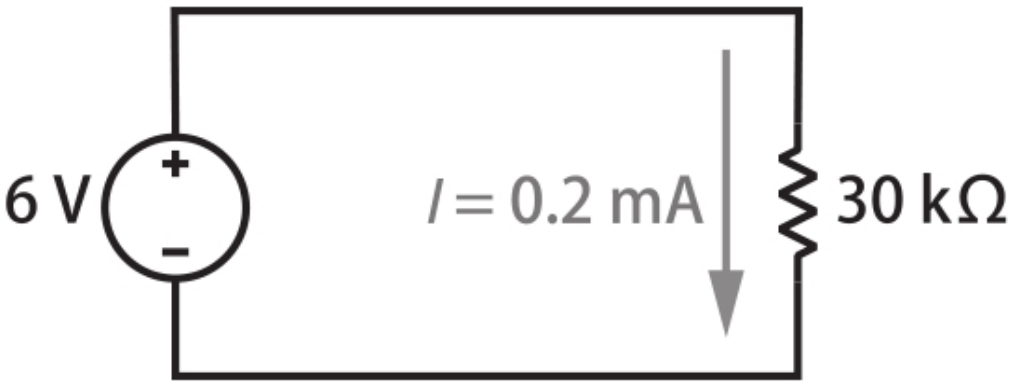
A	*B*	*C*	输出
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

练习3-1：

答案：

欧姆定律告诉我们：电流等于电压除以电阻。因此计算过程如下，电路图标注见图A-3。

$$I = \frac{6\ \text{V}}{30\ 000\ \Omega} = 0.0002\ \text{A} = 0.2\ \text{mA}$$



▲图A-3 用欧姆定律计算电流

练习3-2：

答案：

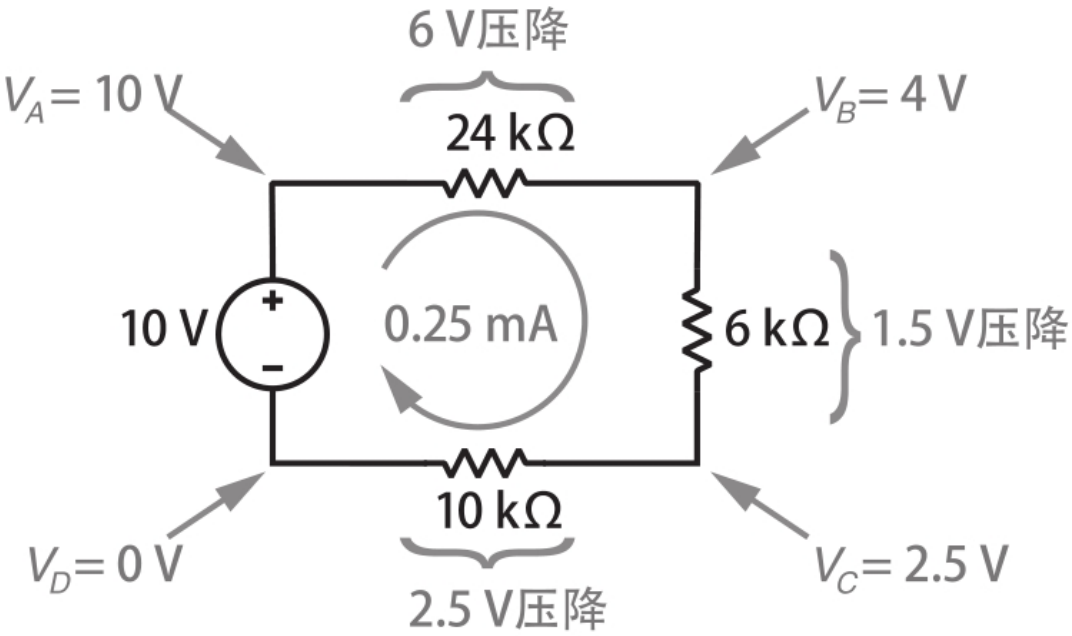
电路总电阻为24kΩ+6kΩ+10kΩ=40kΩ。我们可以用欧姆定律计算出电流：10V/40kΩ=0.25mA，如图A-4所示。

现在用欧姆定律计算24kΩ电阻的压降：V=0.25mA×24kΩ=6V。这表示V_B比V_A低6V，所以V_B=10V-6V=4V。6kΩ电阻的压降为0.25mA×6kΩ=1.5V，所以V_C=V_B-1.5V=2.5V。这样，留给10kΩ电阻的压降就是2.5V，我们可以从基尔霍夫电压定律推导出来，也可以用欧姆定律计算出来。

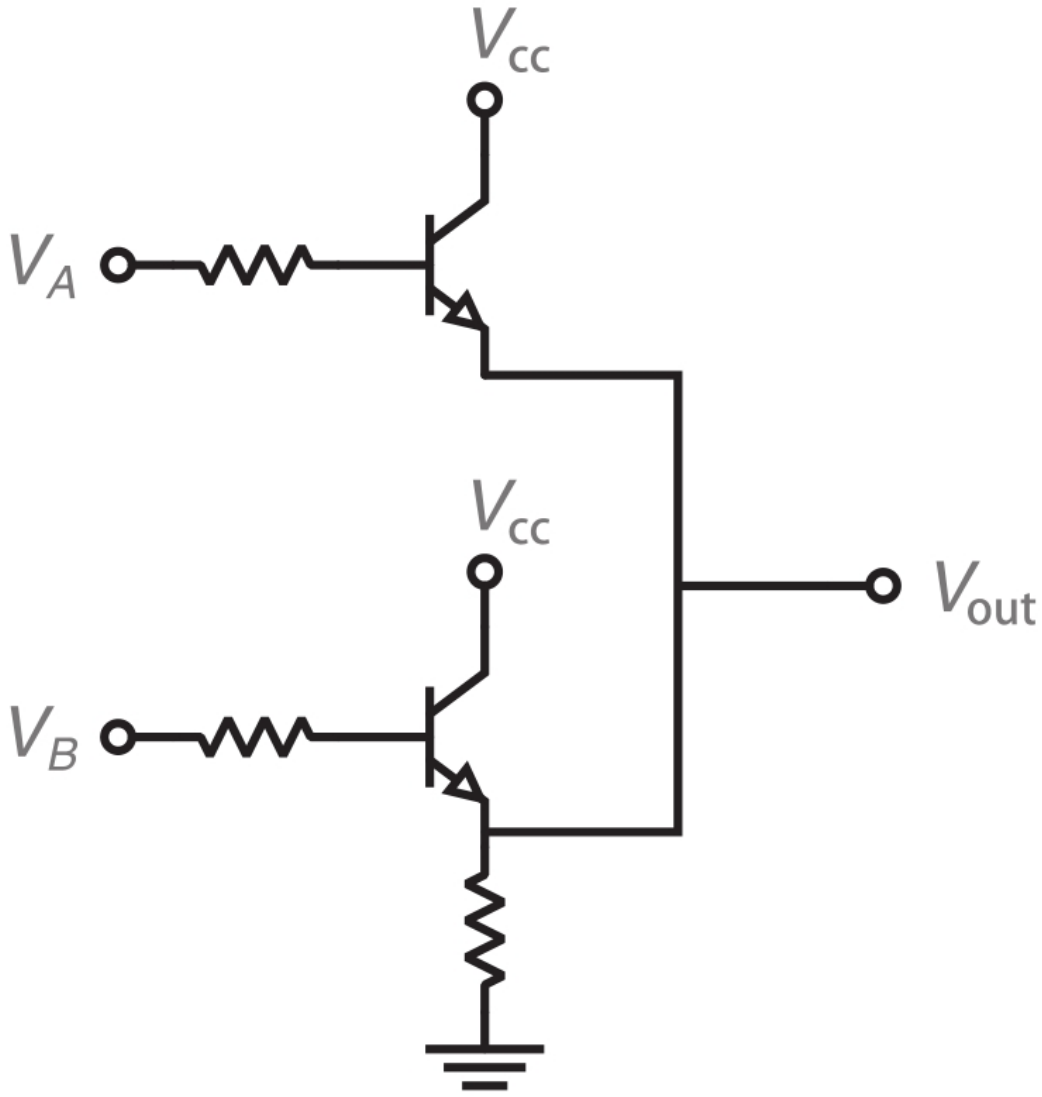
练习4-1：

答案：

图A-5给出了用NPN晶体管实现逻辑OR的解决方案。



▲图A-4 电路上的压降

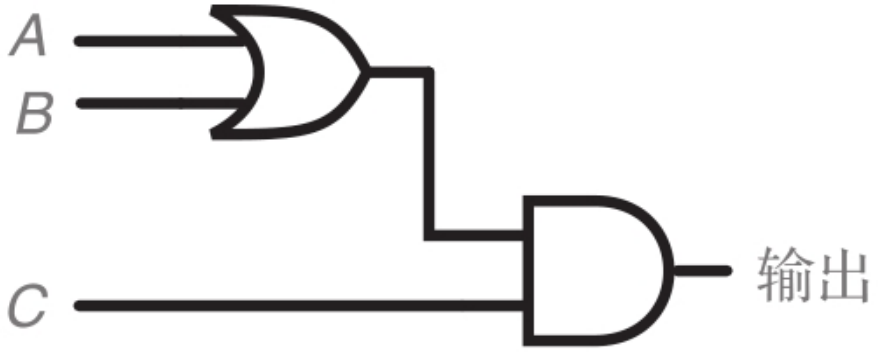


▲图A-5 用NPN晶体管实现逻辑OR

练习4-2：

答案：

图A-6给出了实现（*A* OR *B*）AND *C*的解决方案。



图A-6 （*A* OR *B*） AND *C*的逻辑门图

练习5-1：

答案：

答案中的前导0是可选的。

0001+0010=0011

0011+0001=0100

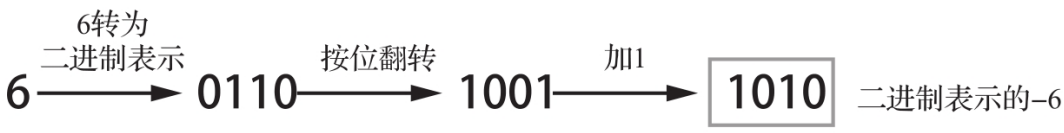
0101+0011=1000

0111+0011=1010

练习5-2：

答案：

参见图A-7。

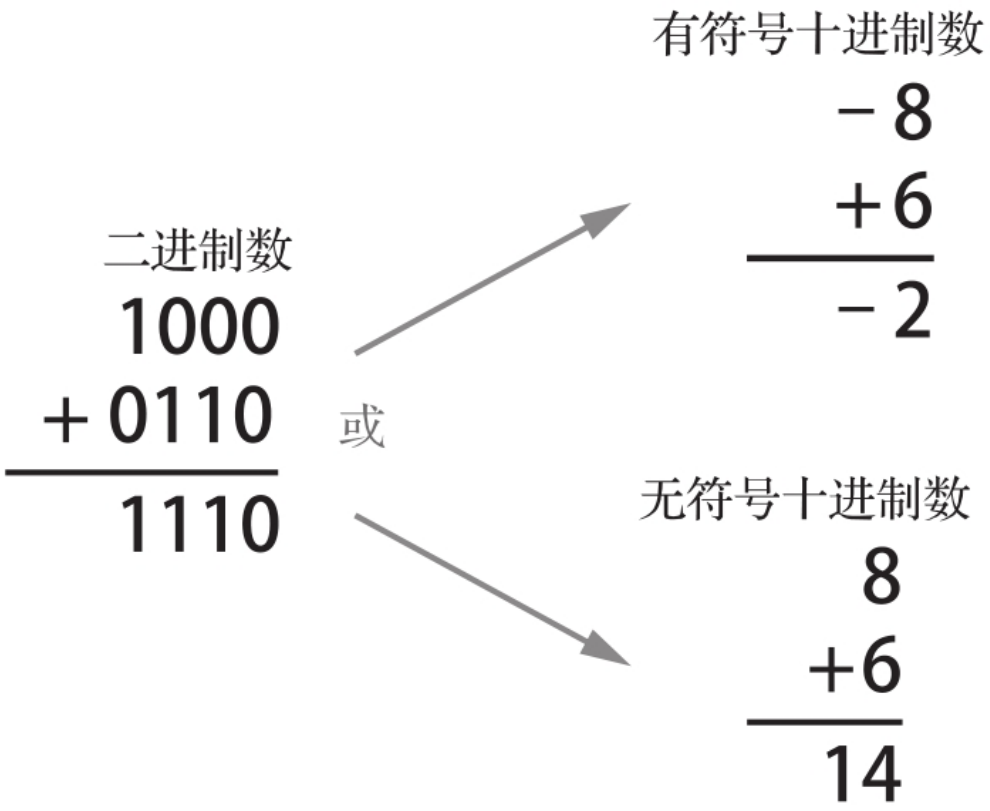


图A-7 确定6的补码

练习5-3：

答案：

参见图A-8。



图A-8 1000加上0110

练习7-1：

答案：

回顾第1章中的SI前缀，可知1GB的内存有2^{30}或1073741824个字节，所以4GB就是这个数的4倍，即4294967296个字节。如果取$\log_2$（4294967296）就得到32，所以用32位就可以表示4GB内存中每一个字节的唯一地址。

如果你的计算器或应用程序不提供$\log_2 x$函数，那么请注意：

$$\log_2(n) = \frac{\log(n)}{\log(2)}$$

有了这些信息，你可以用log（4294967296）除以log（2）得到$\log_2$（4294967296）。结果应该是32。

我们还可以用另一种方法得到这个答案。由于内存地址是从0开始，而不是从1开始分配的，4GB内存的地址范围就是从0到4294967295（比字节数少1）。在十六进制中，4294967295是0xFFFFFFFF。这是个8位的十六进制数，由于每个十六进制符号代表二进制的4位，因此很容易得到：需要4×8=32位。

练习7-2：略

练习8-1：

答案：

完成这个练习后，请查看表A-4，看看运行该汇编代码的每一个步骤。表中的每一行表示一条指令的执行。对每条指令来说，我们会跟踪r0和r3的值。箭头（→）表示寄存器值从左边的变成右边的。在“说明”列中，我用等号表示“被设置为”，它在这里不作为等式的数学检验。例如，r0=r3×r0表示“r0被设置为r3与r0的乘积”。

表A-4 阶乘运算的汇编代码（逐步执行）

地址	指令	r0	r3	说明
		4	?	你想计算 4 的阶乘，所以在代码运行前设置 r0 = 4。r3 初始未知
0001007c	subs r3, r0, #1	4	? → 3	r3 = r0 - 1 = 4 - 1 = 3
00010080	ble 0x10090	4	3	r3 > 0，所以不分支；相反，继续执行 10084
00010084	mul r0, r3, r0	4 → 12	3	r0 = r3 × r0 = 3 × 4 = 12
00010088	subs r3, r3, #1	12	3 → 2	r3 减 1
0001008c	bne 0x10084	12	2	r3 不为 0，所以跳转到分支 10084
00010084	mul r0, r3, r0	12 → 24	2	r0 = r3 × r0 = 2 × 12 = 24
00010088	subs r3, r3, #1	24	2 → 1	r3 减 1

(续)

地址	指令	r0	r3	说明
0001008c	bne 0x10084	24	1	r3 不为 0，所以跳转到分支 10084
00010084	mul r0, r3, r0	24 → 24	1	r0 = r3 × r0 = 1 × 24 = 24
00010088	subs r3, r3, #1	24	1 → 0	r3 减 1
0001008c	bne 0x10084	24	0	r3 为 0，所以不分支；相反，继续执行 10090
00010090		24	0	我们完成了该算法，结果可以在 r0 中找到，现在它等于 24，与预期一致

希望你自己尝试时所看到的结果与这个表是一样的。现在，我们已经演练了一遍*n*=4时的代码，请思考如下问题：

（1）如果我们通过初始设置r0=1来计算1的阶乘，会发生什么？

（2）根据阶乘的数学定义，0的阶乘为1。我们的算法适用于这种情况吗？如果我们初始设置r0=0，会得到什么结果？

（3）你可能已经注意到了，在代码的下一次迭代中，预期结果24被保存到r0中。也就是说，这个程序多循环了一次，但这和r0没有关系。你觉得为什么要这样编写代码？

（4）假设我们使用的是32位寄存器，那么*n*是否有实际的上限？也就是说，是否能提供一个*n*值使得结果太大以致32位寄存器无法容纳？

下面是对上述问题的回答：

（1）第一条sub指令设置r3=0，其后的ble指令跳转到0x10090，因为r3=0。此时，r0中的结果仍然为1，这就是预期的输出。

（2）不适用，我们的算法行不通。第一条sub指令设置r3=-1，其后的ble指令跳转到0x10090，因为r3为负。此时，r0中的结果仍然为0，这不是预期的输出。

（3）*n*的阶乘是小于或等于*n*的正整数的乘积。保持这个定义的正确性意味着用1乘以r0，即使这样做不会改变最终结果。这意味着当r3等于1时，代码会有一次额外的循环。我们可以通过跳过这个与1的乘法运算来提高代码的效率，但我保留了它，以保持阶乘数学定义的正确性。

（4）32位整数的最大值为2^{32}-1=4294967295。如果也需要表示负数，那么最大值为2147483647。因此，如果我们尝试计算的阶乘结果大于40亿（或20亿），我们就会得到不准确的结果。由此可得，*n*=12是我们能使用的最大*n*值。13的阶乘超过60亿，这对32位整数来说太大了。

练习9-1：

答案：

图A-9展示了x和y取值为11和5时，AND、OR和XOR是如何进行按位运算的。

```
x = 11 = 1011          x = 11 = 1011          x = 11 = 1011
y = 5  = 0101          y = 5  = 0101          y = 5  = 0101
---------- AND         ---------- OR          ---------- XOR
     0001                   1111                   1110
```

图A-9 两个值的按位运算

因此，a的值为1，b的值为15（二进制的1111），c的值为14（二进制的1110）。

练习9-2：

答案：

在你继续阅读之前，我强烈建议你尝试完成这个练习！如果亲自做一下，你将学到更多。完成这个练习后，请参阅表A-5，查看运行示例C代码的每一个步骤。箭头（→）意味着变量值从左边的值变成右边的值。

```
// Calculate the factorial of n.
int factorial(int n)
{
  int result = n;

  while(--n > 0)
  {
    result = result * n;
  }

  return result;
}
```

▼表A-5 阶乘运算的C代码（逐步执行）

语句	结果	n	说明
`int factorial(int n)`	?	4	我们想计算 4 的阶乘，所以设置 n = 4 作为函数的输入
`int result = n;`	? → 4	4	设置 result 的初始值为 n 的值
`while(--n > 0)`	4	4 → 3	n 减 1　n > 0，所以运行 while 循环的循环体
`result = result * n;`	4 → 12	3	result = 4 × 3
`while(--n > 0)`	12	3 → 2	n 减 1　n > 0，所以再次运行 while 循环的循环体
`result = result * n;`	12 → 24	2	result = 12 × 2
`while(--n > 0)`	24	2 → 1	n 减 1　n > 0，所以再次运行 while 循环的循环体
`result = result * n;`	24 → 24	1	result = 24 × 1
`while(--n > 0)`	24	1 → 0	n 减 1　n=0，所以退出 while 循环
`return result;`	24	0	我们完成了该算法，计算结果可以在 result 中找到，现在它等于 24，与预期一致

希望你自己尝试时所看到的结果与这个表是一样的。

练习10-1：略

练习11-1：

答案：

如同我们在第11章看到的，计算机子网的网络ID是192.168.0.128。假设两个设备在同一个子网上，它们将共享一个子网掩码和网络ID。把另一台计算机的IP地址与子网掩码进行逻辑AND会给我们提供一个网络ID。

```
  IP = 192.168.0.200   = 11000000.10101000.00000000.11001000
MASK = 255.255.255.224 = 11111111.11111111.11111111.11100000
 AND = 192.168.0.192   = 11000000.10101000.00000000.11000000 = The network id
```

另一台计算机的网络ID（192.168.0.192）与你的计算机网络ID（192.168.0.128）不匹配，所以它们在不同的子网上。这意味着，这些主机之间的通信需要通过路由器。

练习11-2：

答案：

❑DNS:53

❑SSH:22

❑SMTP:25

练习12-1：

答案：

对于https://example.com/photos?subject=cat&color=black，有：

❑协议：https

❑主机：example.com

❑路径：photos

❑查询：subject=cat&color=black

对于http://192.168.1.20:8080/docs/set5/two-trees.pdf，有：

❑协议：http

❑主机：192.168.1.20

❑端口：8080

❑路径：docs/set5/two-trees.pdf

对于mailto:someone@example.com，有：

❑协议：mailto

❑用户名：someone

❑主机：example.com

附录B 相关资源

本附录包含的信息能帮助你开始本书的设计任务。我们将介绍需要的电子元件、如何为数字电路供电，以及如何设置Raspberry Pi。

为设计任务购买电子元件

以动手方式使用电子元件和编程有助于你学习本书的概念，但尝试获得各种组件可能会令人望而生畏。本部分将给出设计所需的电子元件。

下面是设计要用到的全部组件（你可以一次性购买好）：

❑面包板（至少是830孔的面包板。如果你打算在每个练习之后都拆除电路，那么可以只购买一个面包板。如果想保持电路的完整性，则需要一个以上的面包板）；

❑电阻（各种各样的电阻，要用到的具体值为：47kΩ、10kΩ、4.7kΩ、1kΩ、470Ω、330Ω、220Ω）；

❑数字万用表；

❑9V电池；

❑9V电池夹连接器；

❑一组5mm或3mm的红色LED（发光二极管）；

❑两个NPN BJT晶体管，型号2N2222，TO-92封装（也被称为PN2222）；

❑设计用于面包板的跨接线（公对公和公对母）；

❑适合面包板的按钮或滑动开关；

❑7402集成电路；

❑7408集成电路；

❑7432集成电路；

❑两个7473集成电路；

❑7486集成电路；

❑220μF电解电容器；

❑10μF电解电容器；

❑5V电源；

❑Raspberry Pi以及相关内容；

❑推荐用接线夹，它们使你能轻松地把电池连接到面包板或把万用表连接到电路；

❑可选剥线钳。

虽然这个列表给出了某些组件的特定数量，但是为防止损坏或是出于实验的目的，对于有些元件，你可以购买更多数量。建议晶体管多备几

个，集成电路可以每种多备一个。

7400产品编号

寻找一个合适的7400系列集成电路（IC）可能是一项挑战，因为用于标识这些芯片的产品编号包含了更多的细节而不仅仅是74*xx*标识符。7400系列有许多子系列，每一个都有自己的产品编号方案。此外，制造商还在产品编号上添加自己的前缀和后缀。一开始这可能会令人困惑，所以我们先来看个例子。我最近想买一个7408 IC，但我实际订购的型号是SN74LS08N。如图B-1所示，我们把这个型号分解开来。

图B-1 解读7400系列产品编号

SN74LS08N是由德州仪器公司生产的7408 AND门，属于低功耗肖特基子系列，通孔DIP封装。不用担心"低功耗肖特基"的详细信息，只需要知道它是元件的常用子系列，该元件是为我们的目的服务的。

对于本书中的设计任务，你要确保使用的元件可以相互兼容。设计任务假设你使用的元件与原始7400逻辑电平（5V）兼容。考虑到现有的元件，建议购买LS或HCT系列元件。如果想要7408，则可以买SN74LS08N或SN74HCT08N。一般来说，你应该可以在一个电路里混用LS和HCT系列元件。前缀字母（本例中为SN）对兼容性来说不重要，你不需要从特定制造

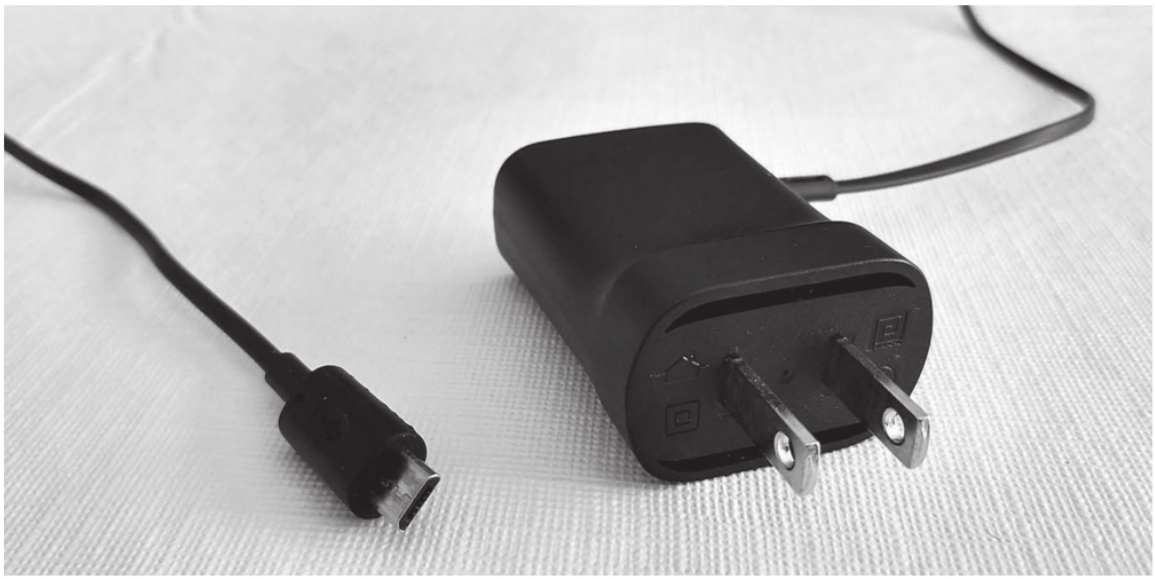
商那里购买元件。后缀很重要，因为它表示封装的类型。一定要让元件适合面包板——N型元件工作良好。

为数字电路供电

7400系列逻辑门需要5V电压，所以9V电池不能为这些集成电路供电。让我们看看哪些元件可以为7400电路供电。

USB充电器

自2010年以来，许多智能手机充电器都有一个微型USB连接器。大约在2016年，USB Type-C（简称USB-C）连接器开始变得更加普遍。幸运的是，每一种USB都提供5V直流电，所以USB充电器是为7400系列集成电路供电的绝佳选择，图B-2展示了一个USB充电器。如果你和我一样，你家里可能已经有一堆旧的微型USB手机充电器了。



图B-2 微型USB手机充电器

但是，这里有一个挑战：微型USB连接器不能插入面包板，至少在没有其他帮助的情况不行！要在面包板上使用USB充电器，一个好的选择是买一个微型USB接线板，如图B-3所示。把USB充电器插入接线板，然后把接线板插入面包板。Adafruit、SparkFun和Amazon都有这些产品。这里可能需