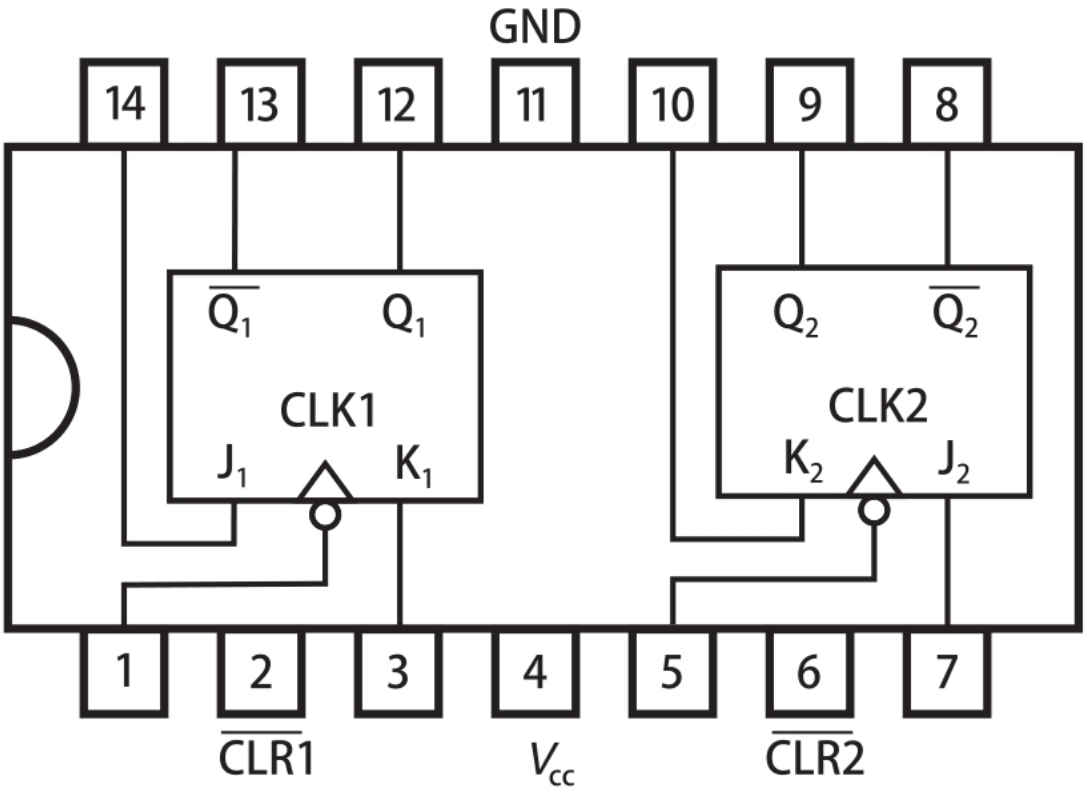
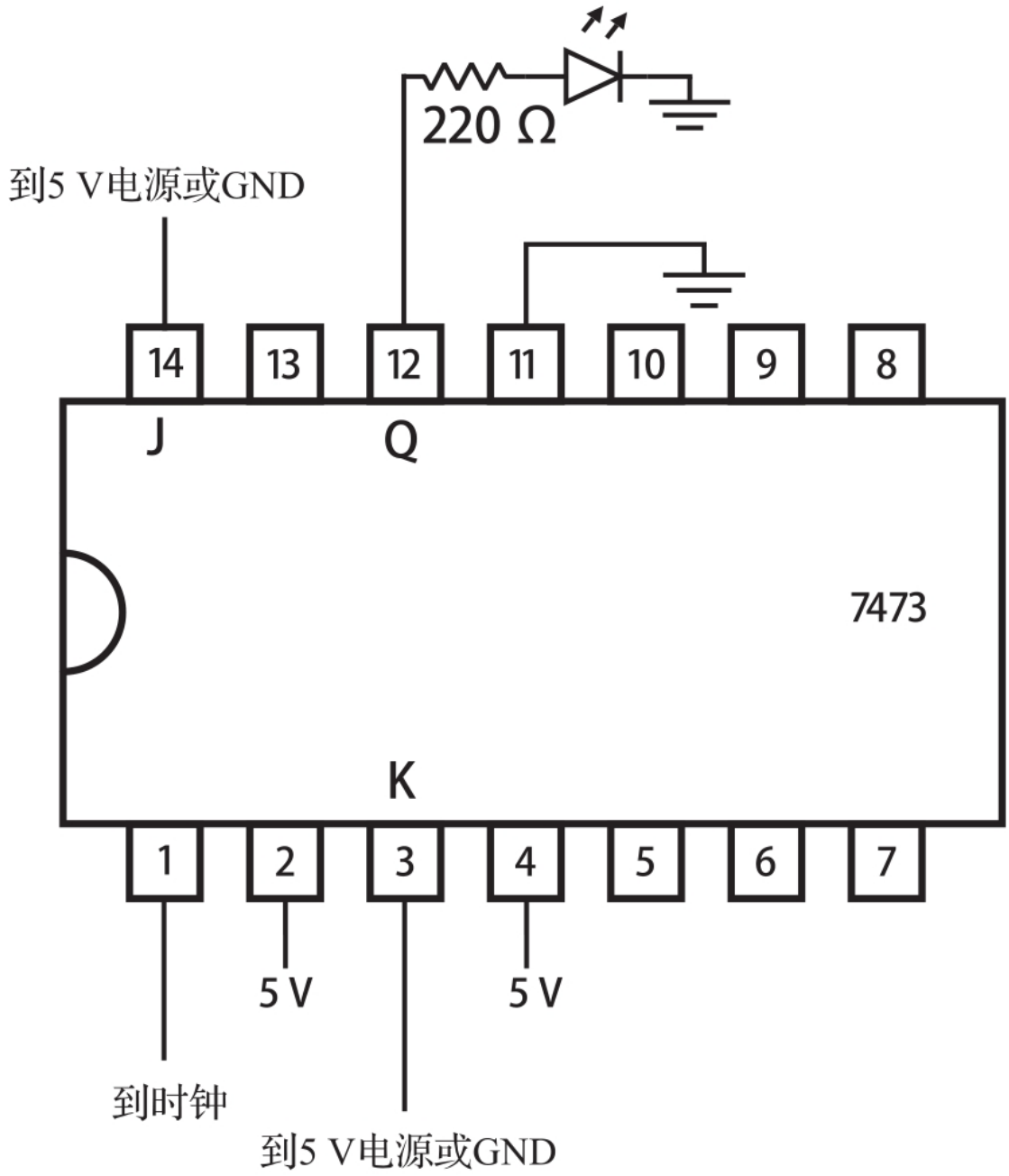
时，保存的位被清零。有时也被称为复位信号，不要与SR锁存器的*R*输入搞混。把JK触发器的输入（引脚2）连接到5V电源，以防止触发器复位。要测试单个触发器，你可以如图6-28所示连接芯片。



▲图6-27 7473 IC的引脚图



▲图6-28 一个简单的JK触发器测试电路

把之前搭建的SR锁存器作为时钟，将SR锁存器的输出*Q*（7402的引脚3和引脚4）连接到7473的时钟输入（引脚1）。现在，尝试把7473的输入*J*（引脚14）和*K*（引脚3）设置为5V或接地。你应该看到，这样做对JK触发器的输出LED没有影响，直到时钟信号从高电平变为低电平。提醒：先按S

再按R把时钟信号先设置为高电平，然后设置为低电平，以产生SR锁存器时钟脉冲。回到表6-3，查看JK触发器的预期行为，保证电路按预期工作。

保持本电路完整，将它用在设计11中。

设计11：搭建3位计数器

在本设计中，你将搭建本章之前所描述的3位计数器。把*Q*输出连接到LED，以便观察输出。

本设计需要如下组件：

❑在设计10中搭建的电路（包括设计9的手动时钟）；

❑额外的7473 IC；

❑7408 IC（含4个AND门）；

❑47kΩ电阻；

❑10μF电解电容器；

❑额外的按钮或开关；

❑跨接线；

❑两个额外的LED；

❑与LED一起使用的两个限流电阻（每个约为220Ω）。

按图6-29所示连接所有组件。IC的引脚编号显示在方框中。

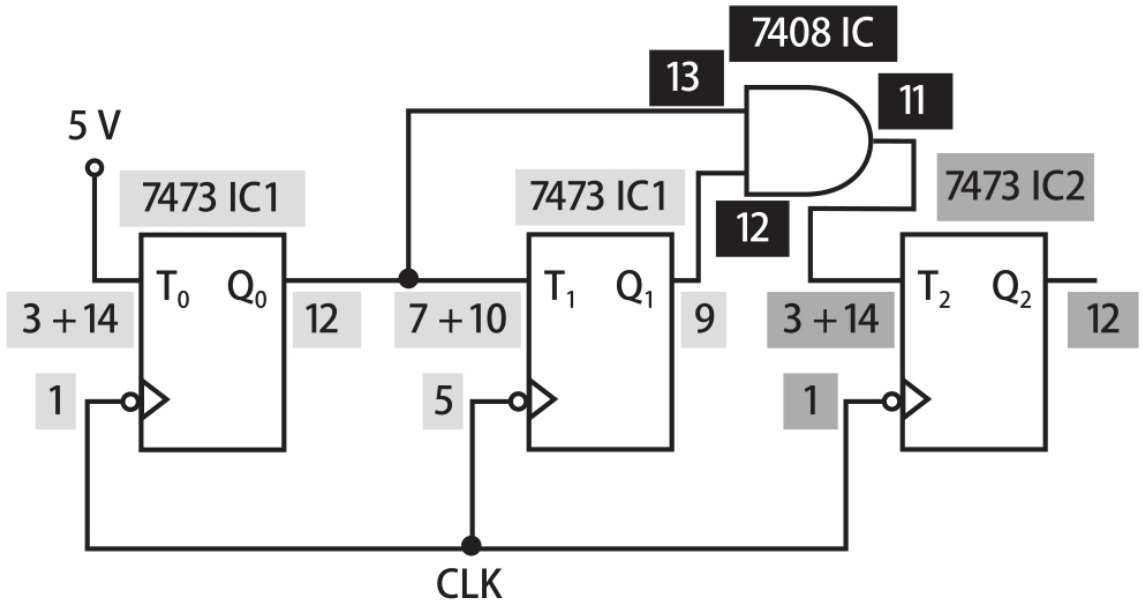


图6-29 用T触发器构建的3位计数器，引脚编号已显示

除了图6-29所示的引脚连接外，还要注意下列连接：

❑两个7473 IC都需要把引脚4和引脚11分别连接到5V电源和接地端。

❑7408应把引脚7连接到接地端，引脚14连接到5V电源。

❑*Q*、*Q*和*Q*应通过220Ω电阻连接到LED，这样就能看到位的变化。

012

❑手动时钟输出（7402上的引脚3和引脚4）应连接到3个触发器的CLK端（第一个7473上的引脚1和引脚5，第二个7473上的引脚1）。

$\overline{\text{CLR}}$ $\overline{\text{CLR}}$

这个电路在不可预测的状态下启动。你可以通过复位3个触发器来手动纠正这个错误，但这样做很乏味。相反，可以增加一个上电复位电路来保证触发器启动时输出都为0。7473封装中的每个触发器都有输入，当其保持低电平时，不论时钟是何状态，都会复位触发器。你想要使在启动时暂时为低电平，然后变为高电平并保持不变。这可以确保接通电源时计数器从

零开始。为了更好地测量，你还可以添加COUNTER RESET按钮，按下它就可以手动复位计数器。这种复位功能如图6-30所示。

$\overline{\text{CLR}}$ $\overline{\text{CLR}}$ $\overline{\text{CLR}}$ $\overline{\text{CLR}}$

当第一次对图6-30所示的电路通电时，电容器就像短路一样，保持低电平，电路处于初始状态。一旦电容器充电，它就像开路一样，变为高电平，电路准备就绪。按下COUNTER RESET按钮或开关也会使变为低电平并复位电路。这个电路需要连接到输入引脚：第一个7473芯片的引脚2和引脚6，第二个7473芯片的引脚2。在设计10中，第一个7473上的引脚2连接到5V电源，在连接上电复位电路之前，请务必断开这个连接。请记住正确定位电解电容器端子——负极应该接地。

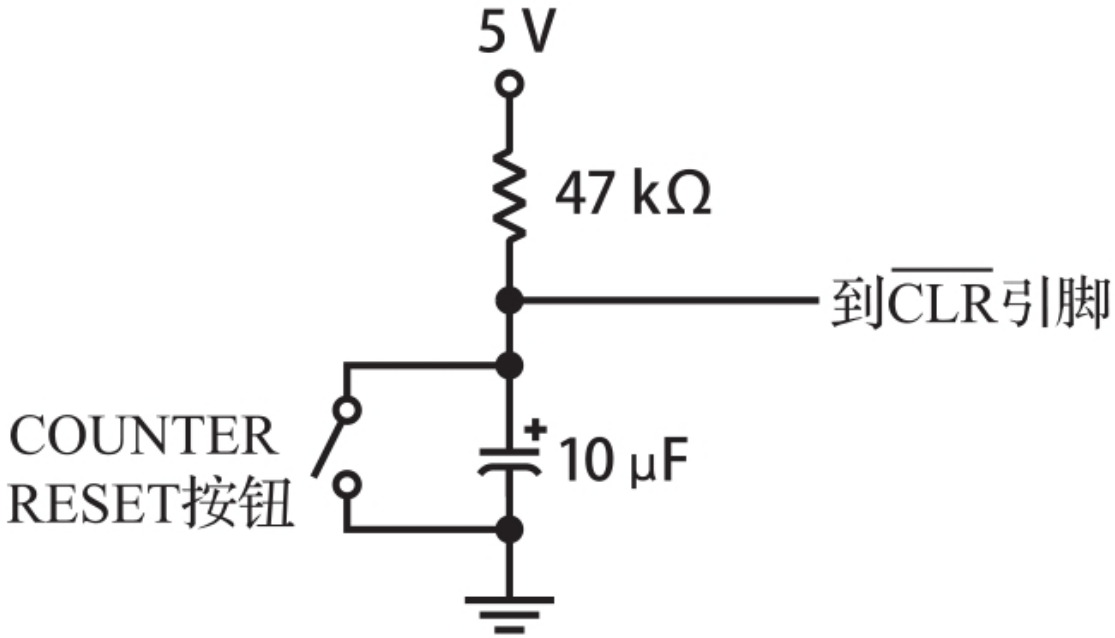


图6-30 3位计数器的上电复位电路

上电复位后，电路应该以计数器为000开始运行。向电路发送一个时钟脉冲，会让计数器在时钟下降沿加1。提示：先按S把时钟信号设置为高电平，再按R把时钟信号设置为低电平，以使SR锁存器时钟产生脉冲。测试从000计数到111并务必使计数器按预期工作。

第7章

计算机硬件

前面的章节介绍了计算机的基本要素——二进制、数字电路和存储器。现在，我们来看看这些要素是如何在计算机中组合在一起的，计算机不仅仅是其组件之和。本章首先概括地介绍一下计算机硬件，然后深入研究计算机的三种组件：主存（主存储器）、处理器和输入/输出。

7.1 计算机硬件概述

我们先概括地介绍一下计算机与其他电子设备的不同之处。之前，我们已经了解了如何使用逻辑电路和存储设备来搭建电路，使之执行有用的任务。我们用逻辑门搭建的电路的功能在设计上是硬连线的。如果想增加或修改功能，就必须改变电路的物理设计。这在面包板上是可以实现的，但是对于已经制造出来并交给用户的设备而言，通常不会选择改变硬件。仅在硬件中定义设备的功能限制了我们快速创新和改进设计的能力。

到目前为止，我们构造的电路使我们得以一窥计算机的工作原理，但我们忽略了计算机的一个关键要素：可编程性。也就是说，计算机必须能够在不改变硬件的前提下执行新任务。为了实现这一非凡的功能，计算机必须能接受一组指令（程序）并执行由它们指定的操作。因此，它必须有能按照程序指定顺序执行各种操作的硬件。可编程性是区分计算机设备与非计算机设备的关键。本章将讨论计算机硬件，即计算机的物理组成部分。这与软件相反，软件是告诉计算机做什么的指令，我们将在第8章讨论。

运行软件的能力把计算机与固定用途的设备区别开来。也就是说，软件仍然需要硬件，那么，在实现一台通用计算机时需要怎样的硬件呢？首先，我们需要存储器。我们已经介绍了1位存储设备，比如锁存器和触发器，计算机中的存储器类型是这些简单存储设备在概念上的扩展。计算机

中使用的主存储器被称为主存，不过，我们一般把它叫作内存或随机存取存储器（Random Access Memory，RAM）。它是易失性的，这意味着它只在通电的时候才能保持数据。“随机存取”表明访问存储器内任意位置需要的时间大致相同。

然后，我们需要中央处理器（Central Processing Unit，CPU），这是第二个关键组件，通常简称为“处理器”，这个组件执行软件中指定的指令。CPU可以直接访问主存。现在，大多数处理器都是微处理器，即单个集成电路上的多个CPU。单集成电路上的CPU具有降低成本、提高可靠性和增强性能的优点。CPU是我们之前介绍的数字逻辑电路在概念上的扩展。

尽管主存和CPU是计算机的最低硬件需求，但实际上大多数计算机设备都需要与外部世界进行交互，而这就需要输入/输出（I/O）设备。本章将详细地介绍主存、CPU和I/O设备。图7-1展示了这三个组件。

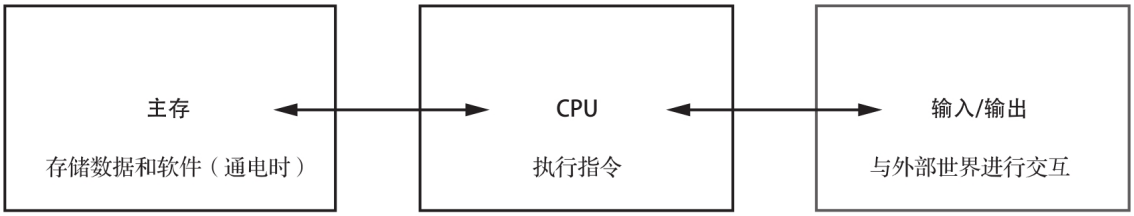


图7-1 计算机的硬件组件

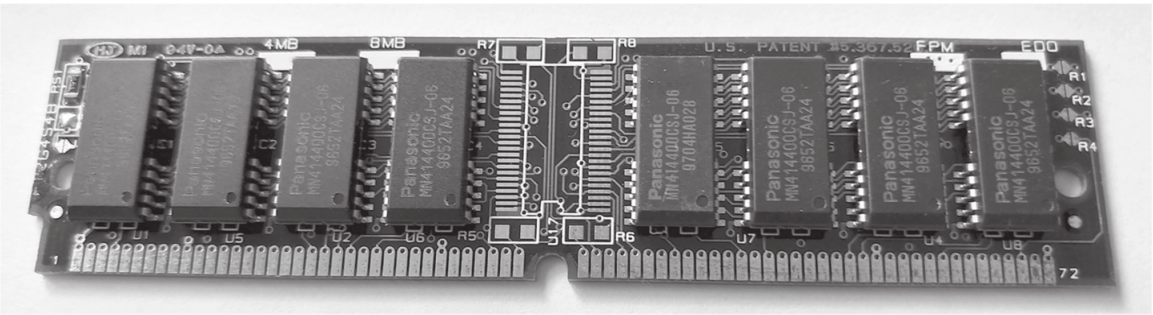
7.2 主存

在执行程序的时候，计算机需要一个地方来存放程序的指令和相关数据。例如，当计算机运行文字处理器来编辑文档时，计算机需要一个地方来保存程序、文档内容和编辑状态——文档的哪些部分是可见的、光标的位置等。所有这些数据最终都会变成CPU可以访问的一系列位。主存（即内存）处理保存这些0/1数据的任务。

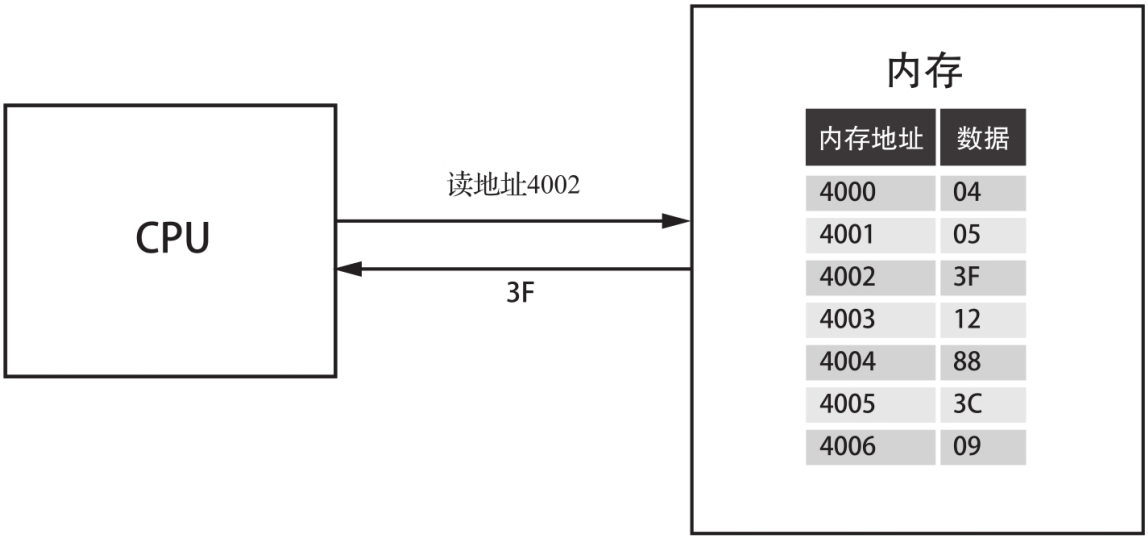
让我们探讨一下内存是如何在计算机中工作的。计算机内存有两种常用类型：静态随机存取存储器（SRAM）和动态随机存取存储器（DRAM）。在这两种类型中，基本单位都是存储单元（memory cell），

即可以存储一个位的电路。在SRAM中，存储单元是触发器。SRAM是静态的，因为它的触发器存储单元在通电时能保持其位值。DRAM存储单元是利用晶体管和电容器实现的。随着时间的推移，电容器的电荷会泄漏，所以必须定期把数据重写入存储单元。这种存储单元的刷新使得DRAM是动态的。现在，由于其相对较低的价格，DRAM通常用作内存。SRAM更快，但是更贵，所以用于速度至关重要的场景，比如用作高速缓存，我们将在后面介绍它。图7-2展示了一个RAM“条”的例子。

总的来说，你可以把RAM的内部想象成一个由存储单元组成的网格。网格中每个1位单元都可以用二维坐标——其在网格中的位置——来标识。一次访问一位的效率很低，所以RAM并行访问多个1位存储单元的网格，允许同时读或写多个位——比如整个字节。内存中一组位的位置被称为内存地址，它是一个标识内存位置的数值。内存通常是按字节编址的，即一个内存地址指的是一个8位的数据。内存布局的内部细节或内存的实现对于CPU（或程序员）而言不是必须了解的知识！主要应理解的是计算机把数值地址分配给内存字节，而CPU可以读写这些地址，如图7-3所示。



▲图7-2 随机存取存储器



▲图7-3 CPU从内存地址读取一个字节

让我们考虑一个虚拟计算机系统，它的寻址范围可以达到64KB。按照如今的标准，这对计算机来说是一个很小的内存，但作为示例足够了。我们假设这个虚拟计算机的内存是按字节寻址的，每个内存地址都代表一个字节。这就意味着我们要为每个内存字节提供一个唯一的地址，由于64KB等于64×1024=65536字节，因此我们需要65536个地址。每个地址都是一个数字，内存地址一般从0开始，所以我们的地址范围是0～65535（0xFFFF）。

由于这个虚构的64KB计算机是数字设备，因此内存地址最终用二进制来表示。在这个系统中，我们需要多少位来表示内存地址呢？*n*位二进制数可以表示的值的数量为2^n。因此，我们需要知道$2^n=65536$时的*n*值。2的某次方的逆运算就是以2为底的对数。因此，$\log_2 2^n=n$，而$\log_2(65536)=16$，即$2^{16}=65536$。需要16位的内存地址来寻址65536个字节。

或者再简单一点，由于我们已经知道最高编号的内存地址为0xFFFF，且我们还知道每个十六进制符号代表4个位，因此我们可以看出需要16位。同样，虚构的计算机可以寻址65536个字节，每个字节分配一个16位的内存地址。表7-1给出了16位内存地址布局的一些示例数据。

表7-1 带示例数据的16位内存地址布局（跳过中间地址）

内存地址（二进制）	内存地址（十六进制）	示例数据
0000000000000000	0000	23
0000000000000001	0001	51
0000000000000010	0002	4A
⋮	⋮	⋮
1111111111111101	FFFD	03
1111111111111110	FFFE	94
1111111111111111	FFFF	82

为什么位数很重要？表示内存地址的位数通常是计算机设计中关键的一环。它限制了计算机可以访问的内存容量，并影响着程序在底层对内存的处理。

我们假设虚拟计算机从内存地址0x0002开始存放了一个ASCII字符串“Hello”。由于每个ASCII字符都需要1个字节，因此保存“Hello”就要5个字节。在查看内存时，一般用十六进制来表示内存地址和这些内存地址的内容。表7-2提供了从地址0x0002开始存放的“Hello”的直观视图。

表7-2 存储在内存中的“Hello”

内存地址	数据字节	ASCII 数据
0000	00	
0001	00	
0002	48	H
0003	65	e
0004	6C	l
0005	6C	l
0006	6F	o
0007	00	
⋮	⋮	
FFFF	00	

使用这种格式可以清楚地表明一个地址只保存一个字节，所以存放5个ASCII字符所需的地址就是从0x0002到0x0006。请注意，表7-2把其他内存地址中的值表示为00，但实际上，假设随机地址保存0是不安全的，它可能是任何值。也就是说，某些编程语言中的标准做法是用空终止符（一个等

于0的字节）结束文本字符串，在这种情况下，我们实际上希望看到地址0x0007中的值为00。

允许检查计算机内存的应用程序一般以类似于图7-4所示的格式来表示内存的内容。

图7-4中最左边的一列是用十六进制表示的内存地址，随后的16个值表示该地址及其后15个地址中的字节值。这种表示方法比表7-2更为紧凑，但它意味着每个地址并不是唯一被表示出来的。在图7-4中，我们再次看到ASCII字符串"Hello"从地址0x0002开始存放。

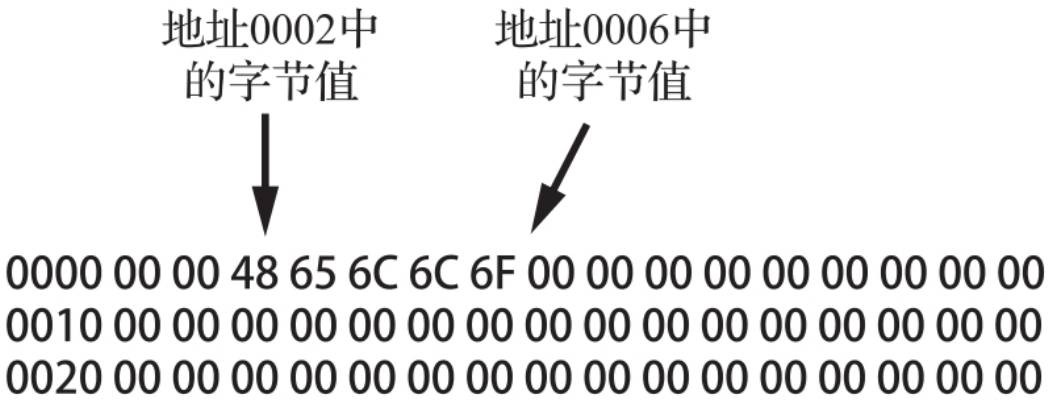


图7-4 内存字节的典型视图

我们假设的64KB RAM计算机作为例子足够了，但是现代计算设备往往具有大得多的内存。

练习7-1：计算所需的位数

利用刚刚描述的方法确定寻址4GB内存所需的位数。你需要回顾表1-3中关于SI前缀的参考信息。请记住，每个字节都分配了一个唯一的地址，这个地址是一个数字。

7.3 中央处理器

内存为计算机提供了一个存放数据和程序的地方，而执行这些指令的是CPU或处理器。正是处理器使得计算机能灵活地运行程序，这些程序在设计处理器时甚至都没有被想到。处理器实现一组指令，然后程序员可以用这些指令来构建有意义的软件。尽管每条指令都很简单，但是这些基本指令是构建所有软件的基石。

下面是CPU支持的一些指令类型：

❑内存访问：读、写指令（对内存）。

❑算术运算：加、减、乘、除、递增。

❑逻辑运算：AND、OR、NOT。

❑程序流：跳转（到程序特定的部分）、调用（某个子程序）。

我们将在第8章讨论具体的CPU指令，但现在的重点是了解CPU指令仅仅是处理器可以执行的操作。它们相当简单（如两数相加、从内存地址读取数据、执行逻辑AND运算等）。程序由这些指令的有序集合组成。用烹饪来类比的话，那么，CPU是厨师，程序是菜谱，程序中的每条指令就是菜谱上的一个步骤，厨师知道如何执行这些步骤。

程序指令驻留在内存中。CPU读取这些指令，这样它就能运行程序。图7-5展示了一个由CPU从内存中读取的简单程序。

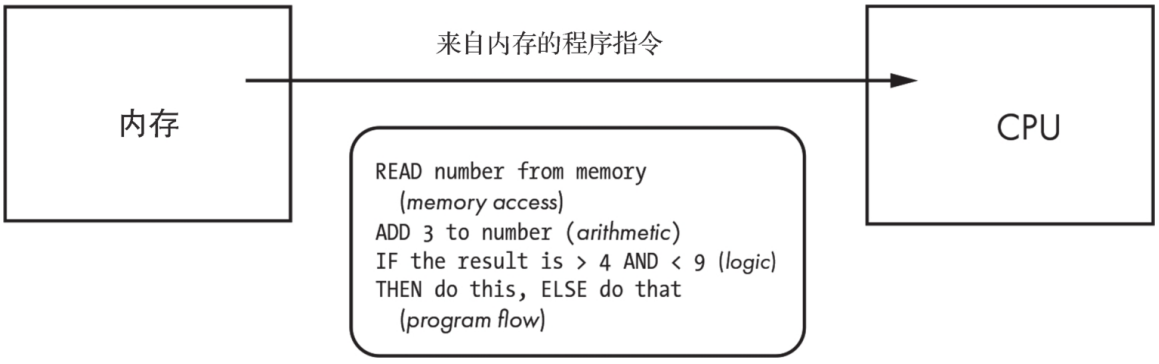


图7-5 从内存读取示例程序并在CPU上运行

图7-5中的示例程序是用伪代码编写的，它不是用真正的编程语言编写的，而是一种人类可读的程序描述。该程序中的几条指令分别属于前述的类型（内存访问、算术运算、逻辑运算和程序流）。程序首先读取保存在内存特定地址中的数，然后在这个数上加3，之后执行两个条件的逻辑与运算。如果逻辑运算结果为真，那么程序将执行this；否则，程序将执行that。不管你是否相信，所有的程序本质上都只是这些基本类型的各种组合。

7.3.1 指令集架构

尽管所有的CPU都实现了这些指令类型，但是不同处理器可用的指令还是有所不同。在一种CPU上存在的一些指令在另一种CPU上根本就不存在。甚至几乎所有CPU上都有的指令其实现方式也不尽相同。例如，用于表示“两数相加”的特定二进制位序列在不同类型的处理器上是不同的。使用相同指令的一系列CPU共享一个指令集架构（Instruction Set Architecture，ISA）——简称架构，它是表示CPU工作原理的模型。为特定ISA构建的软件可以在实现该ISA的任何CPU上运行。对于多种处理器模型，甚至不同制造商的处理器模型而言，都可以实现相同的架构。这样的处理器的内部工作方式可能差异很大，但通过遵循相同的ISA，它们可以运行相同的软件。当前有两种主流的指令集架构：x86和ARM。

大多数台式计算机、笔记本计算机和服务器都使用x86 CPU。该名称来源于Intel公司对其处理器的命名约定（每个处理器名称都以86结尾），如从1978年发布的8086到后来的80186、80286、80386和80486。在80486（简称486）之后，Intel开始用诸如Pentium和Celeron等名称来命名其处理器，虽然名称变了，但这些处理器仍然是x86 CPU。Intel之外的其他公司也生产x86处理器，特别是Advanced Micro Devices（AMD）公司。

术语x86指的是一组相关的架构。随着时间的推移，新的指令被添加到x86架构中，但每一代架构都试图保持向后兼容性。这通常意味着在较早x86 CPU上开发的软件可以在较新的x86 CPU上运行，但是利用了新x86 CPU指令，在较新的x86 CPU上开发的软件则不能在较早的x86 CPU上运行，它们不能理解这些新的指令。

x86架构主要包括三代处理器：16位、32位和64位处理器。让我们暂停一下，看看CPU是16位、32位或64位处理器是什么意思。与处理器相关的位数——也称为它的位数或字长，是指处理器一次可以处理的位数。因此，32位CPU可以操作长度为32位的数值。再具体一点，这表示计算机架构具有32位寄存器、32位地址总线或32位数据总线，或者这三者都是32位的。我们将在后面详细讨论寄存器、数据总线和地址总线。

回到x86架构及其几代处理器，最初在1978年发布的8086处理器是16位处理器。受到8086的鼓舞，Intel继续生产了兼容的处理器。Intel后续的x86处理器也是16位的，直到1985发布了80386处理器，它采用了新的32位版x86架构。32位版的x86有时也被称为IA-32。多亏了向后兼容性，现代x86处理器仍然支持IA-32。图7-6展示了x86处理器的一个例子。

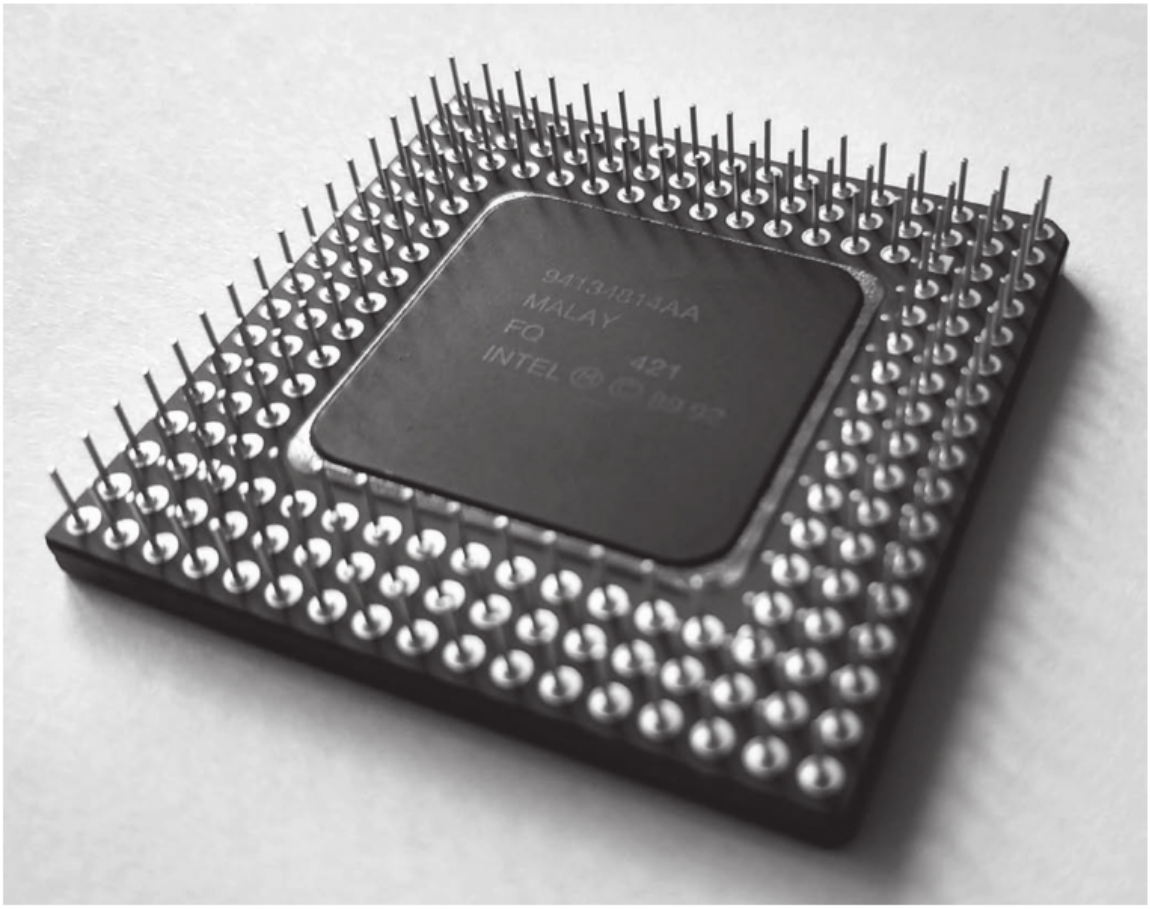


图7-6 Intel 486 SX（一个32位x86处理器）

有意思的是，是AMD而不是Intel把x86带入64位时代。在20世纪90年代末，Intel的64位研究重点放在一种新的CPU架构上，这种架构被称为IA-64或Itanium，但它不是x86 ISA，最终它成了服务器的定制产品。在Intel关注Itanium时，AMD抓住机会扩展了x86架构。2003年，AMD发布了第一款64位的x86 CPU——Opteron处理器。AMD的架构最初被称为AMD64，后来Intel采用了这个架构并把它称为Intel 64。这两种实现在功能上基本相同，现在64位的x86通常称为x64或x86-64。

尽管x86统治了个人计算机和服务器世界，但ARM处理器主宰了智能手机和平板电脑等移动设备领域。很多公司都生产ARM处理器。一家名为ARM Holding的公司开发了ARM架构，并将其设计授权给其他公司来实现。ARM CPU通常用于片上系统（System-on-Chip，SoC）设计，这里，单个集成电路不仅包含CPU，还包含内存和其他硬件。ARM架构作为32位ISA起源于20世纪80年代。ARM架构的64位版本于2011年推出。相比于x86处理器，由于降低了功耗和成本，ARM处理器在移动设备中很受欢迎。ARM处理器也可以用于PC，但为了保持与现有x86 PC软件的向后兼容性，市场主要还是集中在x86上。不过在2020年，Apple宣布它们打算把macOS计算机从x86迁移到ARM CPU上。

7.3.2 内部结构

在内部，CPU由多个组件构成，这些组件协同工作以执行指令。我们将关注三个基本组件：处理器寄存器、算术逻辑单元和控制单元。处理器寄存器位于CPU里面，在处理过程中保存数据。算术逻辑单元（Arithmetic Logic Unit，ALU）执行逻辑和算术运算。处理器控制单元指挥CPU，并和处理器寄存器、ALU和内存通信。图7-7显示了CPU架构的简化示意图。

现在来看处理器寄存器。内存保存正在执行的程序的数据。但是，当程序要对一段数据进行操作时，CPU需要在处理器硬件中设置一个临时的位置来存放数据。为了达到这个目的，CPU有一些小型的内部存储位置，它们被称为处理器寄存器（简称“寄存器”）。与访问内存相比，CPU访问寄存器是非常快的操作，但是寄存器只能存放少量的数据。由于寄存器太小了，