因此衡量每个寄存器大小的单位是位而不是字节。例如，32位CPU通常有32位“宽”的寄存器，这表示每个寄存器可以保存32位的数据。寄存器是在被称为寄存器文件（不要和文档或图片等数据文件搞混）的组件中实现的。寄存器文件中使用的存储单元一般是SRAM。

ALU在CPU内部处理逻辑和算术运算。我们之前介绍过组合逻辑电路，其输出是输入的函数。处理器的ALU就是一个完整的组合逻辑电路。ALU的输入是被称为操作数的数值，以及指示要对这些操作数执行哪种操作的编码。ALU输出该操作的结果和状态信息，这个状态信息提供了关于操作执行的更多细节。

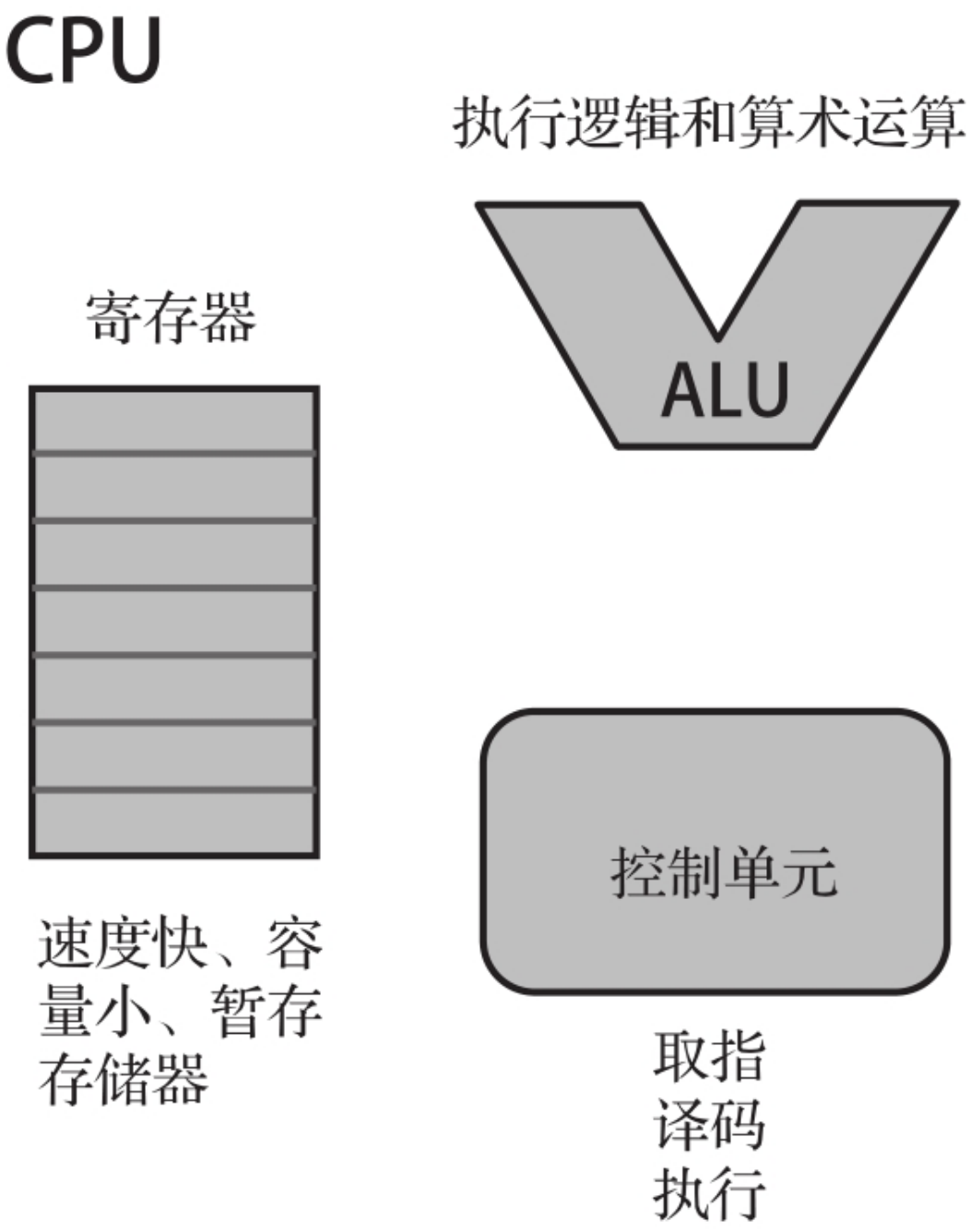

图7-7 CPU架构的简化示意图

控制单元是CPU的协调器。它以循环方式工作：从内存取一条指令，对该指令进行译码，然后执行它。由于正在运行的程序保存在内存中，因此控制单元需要知道读取哪个内存地址来取出下一条指令。控制单元通过查

看一个被称为程序计数器（Program Counter，PC）的寄存器来确定这个地址，该寄存器在x86中也被称为指令指针。程序计数器存放下一条要执行的指令的地址。控制单元从指定的内存地址读取指令，把该指令存入被称为指令寄存器的寄存器中，并更新程序计数器使其指向下一条指令。然后，控制单元对当前指令进行译码，理解表示指令的1和0的含义。译码后，控制单元执行指令，这可能需要CPU中其他组件的协作。例如，执行加法运算就需要控制单元指示ALU执行所需的算术运算。一旦指令完成，控制单元就重复这个循环：取指、译码、执行。

7.3.3 时钟、内核和高速缓存

由于CPU执行有序的指令集，你可能想知道是什么使得CPU从一条指令前进到下一条指令。我们之前演示了如何利用时钟信号让电路从一个状态进入另一个状态，比如在计数器电路中。同样的原理在这里也适用。CPU接收输入的时钟信号，如图7-8所示，时钟脉冲作为提供给CPU的信号，使其在状态间转换。

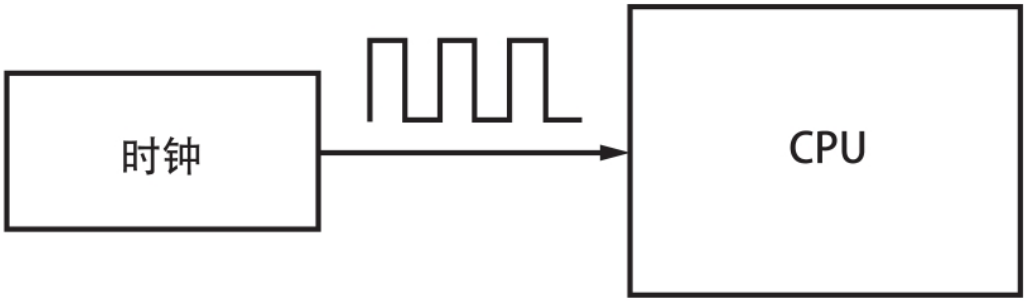

图7-8 时钟为CPU提供振荡信号

认为CPU每个时钟周期只执行一条指令是过于简单的想法。有些指令需要多个周期才能完成。此外，现代CPU使用一种被称为流水线的方法，把指令分为更小的步骤，这样多条指令的步骤就可以由一个处理器并行运行。例如，处理器可以在取一条指令的同时对另一条指令进行译码并执行第三条指令。不过，把时钟的每个脉冲都看作使CPU在执行程序时前进的信号还

是有帮助的。现代CPU的时钟频率以GHz为单位。例如，2GHz的CPU的时钟每秒振荡20亿次！

提高时钟频率会让CPU在单位时间内执行更多的指令。可惜的是，我们不能以任意高的时钟频率来运行CPU。CPU的输入时钟频率有上限，当超过这个上限时，CPU会产生过多的热量。而且，CPU的逻辑门也可能跟不上，导致意外的错误和崩溃。多年来，计算机行业中的CPU时钟频率上限在稳步提高。这种时钟频率的提高很大程度上是由于制造工艺的定期改进导致了晶体管密度的增加，这就使得CPU能具备更高的时钟频率，同时其功耗又大致保持不变。1978年，Intel 8086的时钟频率为5MHz，到了1999年，Intel PentiumIII的时钟频率为500MHz，仅用了大约20年时钟频率就提高了100倍！之后，CPU时钟频率继续快速提高，直到2000年初超过了3GHz的阈值。从那时起，尽管晶体管的数量还在持续增加，但与晶体管尺寸有关的物理上限使得显著提升时钟频率变得不现实。

随着时钟频率上限趋于稳定，处理器设计者开始寻求一种新方法来让CPU完成更多的工作。CPU的设计不再关注如何提高时钟频率，而是开始关注多条指令的并行执行。人们提出了多核CPU的概念，这种CPU具有多个处理单元，各处理单元被称为内核。如图7-9所示，CPU内核实际上是一个独立的处理器，它与其他独立处理器一起存在于一个CPU封装中。

请注意，并行运行的多个内核与流水线是不一样的。多核的并行性意味着每个内核处理不同的任务，即处理一组不同的指令。相反，流水线是发生在每个内核中的，它允许一个内核并行执行多条指令的小步骤。

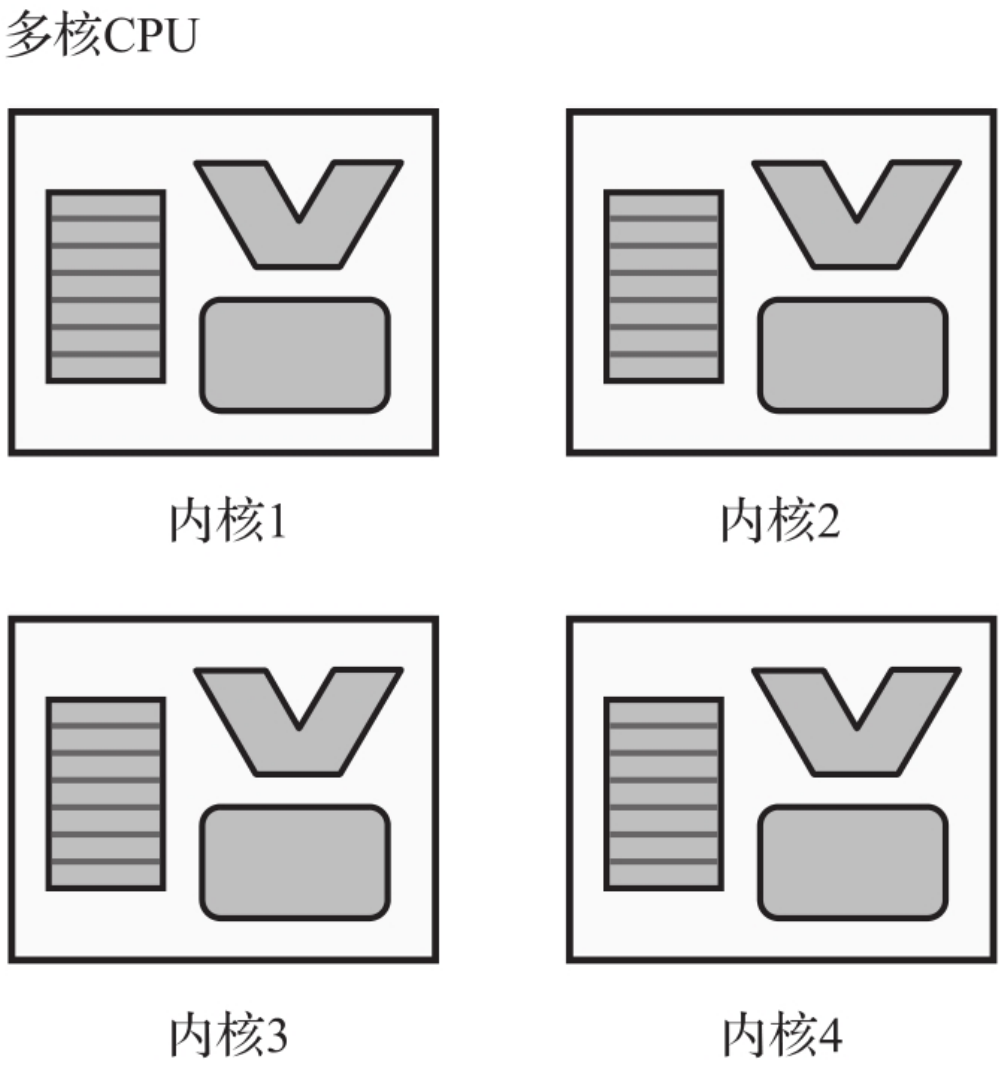

图7-9 四核CPU——每个内核都有自己的寄存器、ALU和控制单元

添加到CPU处理器的每个内核都打开了一扇通向计算机并行执行其他指令的门。也就是说，向计算机的CPU添加多个内核并不意味着所有的应用程序都会立即或同等受益。必须编写软件来利用并行处理指令的优势，以获得多核硬件的最大收益。但是，即使单个程序的设计没有考虑并行性，整个计算机系统也可以受益，因为现代操作系统一次会运行多个程序。

我之前描述过CPU是如何把数据从内存加载到寄存器进行处理，然后再把数据从寄存器存回到内存以备后续使用的。事实证明，程序往往会一次又一次地访问相同的内存位置。正如你所预料的，多次返回内存来访问相

同的数据是低效的！为了避免这种低效，CPU中驻留了小容量的存储器以保存经常被访问的内存数据的副本。这种存储器被称为CPU的高速缓存。

处理器会检查高速缓存，以查看其想要访问的数据是否在其中。如果在，处理器就可以通过读写高速缓存而不是内存来加快访问速度。当需要的数据不在高速缓存中时，处理器可以在从内存中读取该数据后把它放到高速缓存中。常见的处理器有多级高速缓存，一般是三级。我们把这些高速缓存划分为L1高速缓存、L2高速缓存和L3高速缓存。如图7-10所示，CPU首先查找L1是否有所需数据，然后查找L2，再然后查找L3，最后才查找内存。L1高速缓存的访问速度是最快的，但它的容量也是最小的。L2的速度较之慢一些，但容量要大一些，L3则速度更慢，容量更大。请记住，即使是这些逐渐变慢的高速缓存，它们的访问速度也比内存访问速度要快。

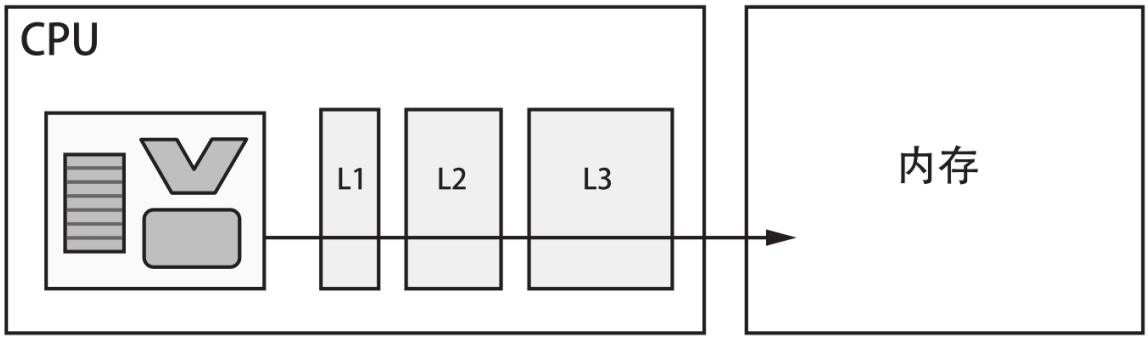

图7-10 具有三级高速缓存的单核CPU

在多核CPU中，有些高速缓存是专属每个内核的，有些则在内核之间共享。例如，每个内核都有自己的L1高速缓存，而L2和L3高速缓存则是共享的，如图7-11所示。

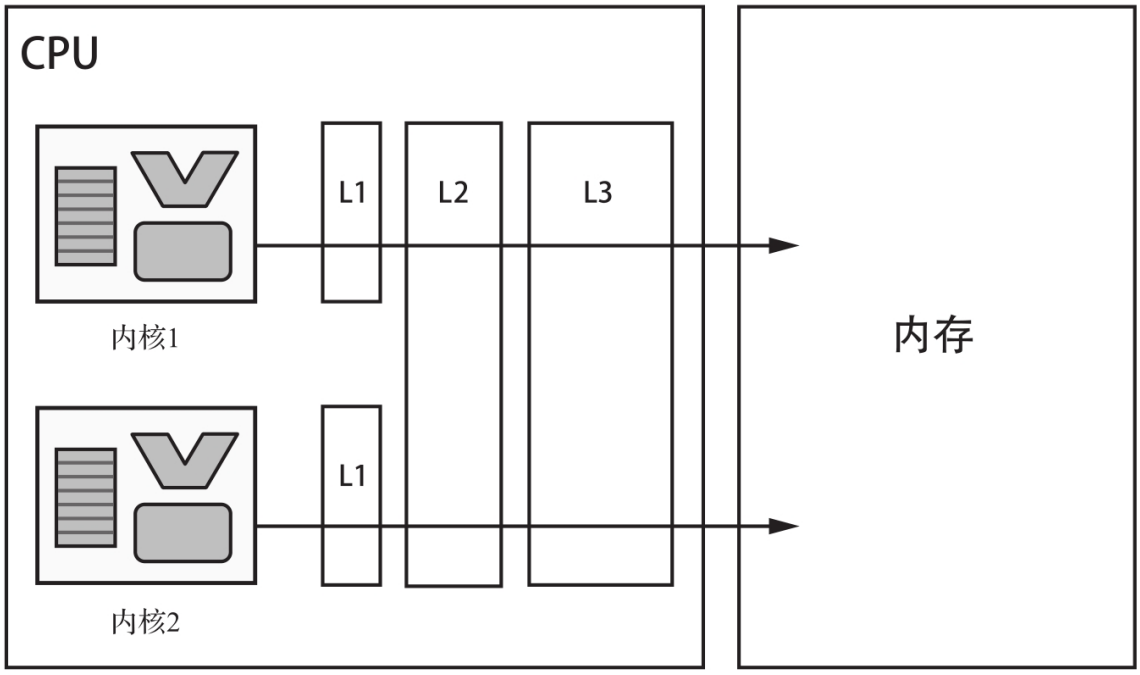

图7-11 带高速缓存的双核CPU。每个内核都有自己的L1高速缓存，而L2和L3高速缓存则是共享的

7.4 其他组件

我已经概括地叙述了计算机所需的两个基本组件：内存和处理器。但是，只有内存和处理器的设备与我们想要的有用设备之间还有些差距。第一个差距是，内存和CPU都是易失性的，它们在断电后就会丢失状态。第二个差距是，只有内存和处理器的计算机是无法与外界进行交互的。现在让我们看看辅存和I/O设备是如何填补这些差距的。

7.4.1 辅存

如果计算机只包含内存和处理器，那么每次设备断电时，它就会丢失所有的数据！为了强调这一点，这里的“数据”不仅是指用户的文件和设置，还包括已安装的应用程序，甚至是操作系统本身。这种相当不方便的计算机在每次开机时都需要人加载OS和应用程序。这可能会阻止用户关闭机

器。不管你是否相信，前几代计算机就是这么工作的，但幸运的是，现在的工作方式不是这样的。

为了解决这个问题，计算机使用了辅存。辅存是非易失性的，所以即使在系统断电时，它也能保存数据。与RAM不同，辅存不是直接由CPU寻址的。相比于RAM，通常这种存储器单位容量的价格更加便宜；与内存相比，它能有更大的存储容量。但是，辅存比RAM慢得多，它不适合替代内存。

在现代计算机设备中，硬盘驱动器和固态驱动器是最常见的辅存设备。硬盘驱动器（Hard Disk Drive，HDD）用高速旋转磁盘上的磁性来存储数据，固态驱动器（Solid-State Drive，SSD）用非易失性存储单元中的电荷来存储数据。与HDD相比，SSD更快，更安静，更能容忍机械故障，因为SSD没有移动部件。图7-12是两个辅存设备的照片。

图7-12 一个1997年的4 GB硬盘驱动器和一个现代32 GB MicroSD卡（一种固态存储器）

有了辅存，计算机就可以按需加载数据。当一台计算机上电时，操作系统将数据从辅存加载到内存，任何被设置为启动时运行的应用程序也会被加载。启动后，当应用程序运行时，程序代码会从辅存加载到内存。这也适用于本地存储的任何用户数据（文档、音乐、设置等），这些数据必须从辅存加载到内存，然后才能被使用。在通常的用法中，辅存一般简称为存储器，而内存（内存储器）/主存（主存储器）则简称为内存或RAM。

7.4.2 输入/输出

即使使用了辅存，目前假设的计算机还存在一个问题。一个由处理器、内存和辅存构成的计算机无法与外界交互！这就需要输入/输出（I/O）设备了。I/O设备是一种组件，它允许计算机从外界接收输入（键盘、鼠标），向外界发送数据（显示器、打印机），或两者皆可（触摸屏）。人与计算机的交互需要通过I/O设备实现。计算机与计算机的交互也需要通过I/O设备实现，一般是以计算机网络（比如互联网）的形式实现。辅存设备实际上是一种I/O设备。你可能不会把访问内部存储器看作输入/输出，但从CPU的角度来看，读写存储器只是另一种I/O操作。从存储设备读取数据是输入，向存储设备写入数据是输出。图7-13给出了一些输入和输出的例子。

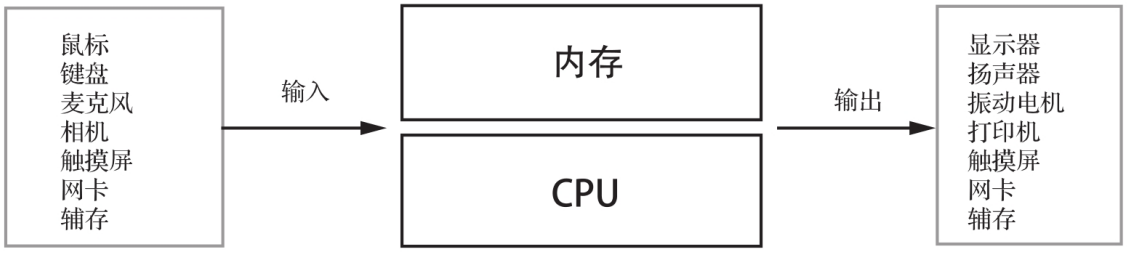

图7-13 输入和输出的常见类型

那么CPU是如何与I/O设备进行通信的呢？计算机可以连接多种多样的I/O设备，它需要一种标准方式来与这些设备进行通信。为了理解这一点，

我们需要先讨论一下物理地址空间，即计算机可用的硬件存储器地址范围。在7.2节中，我们已经介绍了内存字节是如何分配地址的。给定计算机系统上全部内存地址都会用一定数量的位来表示。位的数量不仅决定了每个内存地址的大小，而且决定了计算机硬件可用地址的范围——物理地址空间。地址空间通常大于计算机内部安装的RAM容量，从而留下一些未使用的物理内存地址。

例如，若一台计算机有32位的物理地址空间，则其物理地址范围为从0x00000000到0xFFFFFFFF（32位数能表示的最大地址）。这大约有40亿个地址或4GB的地址空间，其中的每个地址都代表一个字节。假设这台计算机有3GB的RAM，那么75%的可用物理地址被分配给了RAM字节。

现在让我们回到计算机是如何与I/O设备进行通信的这个问题上。物理地址空间中的地址并不总是指向内存字节，它们也可以指向I/O设备。当物理地址空间被映射到I/O设备时，CPU只需对分配给其的内存地址进行读写就可以与该设备通信，这被称为内存映射I/O（Memory-Mapped I/O，MMIO），如图7-14所示。当计算机把I/O设备的存储空间看作内存时，它的CPU就不需要任何专用的I/O操作指令。

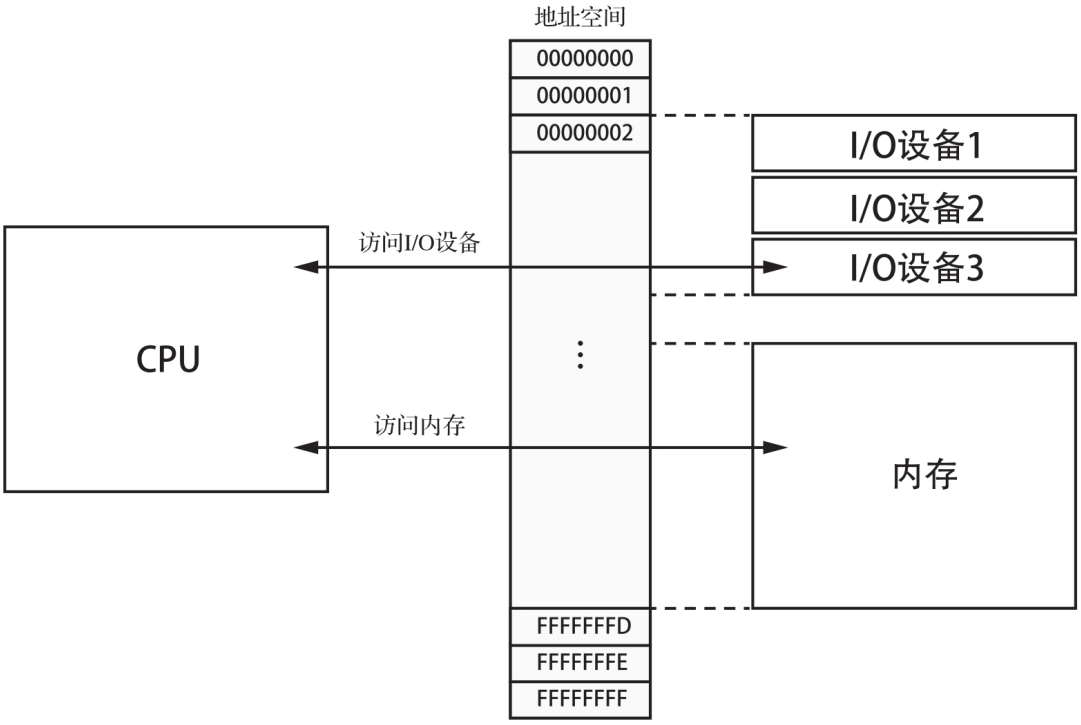

图7-14 内存映射I/O

但是，有些CPU系列，特别是x86，确实包含了访问I/O设备的专用指令。当计算机使用这种方式时，它不是把I/O设备映射到物理内存地址，而是给设备分配一个I/O端口。端口就像是一个内存地址，但它不是指向内存中的位置，而是指向I/O设备。你可以把I/O端口集合看作另一个地址空间，它和内存地址不在一个空间。这意味着端口0x378与物理内存地址0x378指的不是同一个东西。通过独立的端口地址空间访问I/O设备被称为端口映射I/O（Port-Mapped I/O，PMIO）。当前，x86 CPU同时支持端口映射I/O和内存映射I/O。

I/O端口地址和内存映射I/O地址通常指向设备控制器，而不是直接指向存储在设备上的数据。例如，对于硬盘驱动器来说，磁盘的字节不会直接映射到地址空间。相反，硬盘驱动器的控制器提供一个通过I/O端口或内存映射I/O地址可以访问的接口，从而允许CPU请求对磁盘位置进行读写操作。

练习7-2：了解生活中的硬件设备

选择一些计算机设备——比如笔记本计算机、智能手机或游戏机。就每个设备回答以下问题。你可能需要查看设备自身的设置才能找到答案，也可能需要通过网络进行一些研究。

❑这个设备有什么样的CPU?

❑该CPU是32位的还是64位（或者其他）的?

❑该CPU的时钟频率是多少?

❑该CPU有L1、L2或L3高速缓存吗？如果有的话，多少钱?

❑该CPU使用哪种指令集架构?

❑该CPU有多少个内核?

❑该设备的内存容量有多大？是哪种类型的内存?

❑该设备的辅存容量有多大？是哪种类型的辅存？

❑该设备有哪些I/O设备？

7.5 总线通信

到目前为止，我们已经介绍了计算机中的内存、CPU和I/O设备。我们还探讨了CPU通过内存地址空间与内存和I/O设备之间的通信。现在，让我们进一步讨论一下CPU是如何与内存和I/O设备通信的。

总线是计算机组件使用的一种硬件通信系统。总线的实现方式有多种，但在计算机发展的早期，总线只是一组并列的导线，每根导线携带一个电信号。每根导线上的电压表示一个位，这就使得多个数据位能并行传输。现在的总线设计并不总是这么简单，但其意图与之类似。

CPU、内存和I/O设备之间的通信经常使用三种常见的总线类型。地址总线充当选择器，用于选择CPU想要访问的内存地址。例如，如果程序希望向地址0x2FE写入数据，那么CPU就把0x2FE写到地址总线上。数据总线传送从内存读出的值以及向内存写入的值，所以如果CPU想要把值25写入内存，那么25就会被写到数据总线上，如果CPU正在从内存读取数据，那么CPU就从数据总线上读取这个值。控制总线管理发生在其他两种总线上的操作。例如，CPU用控制总线来指示一个写操作即将发生，控制总线可以携带一个信号来指示某个操作的状态。图7-15展示了CPU是如何使用地址总线、数据总线和控制总线来读取内存的。

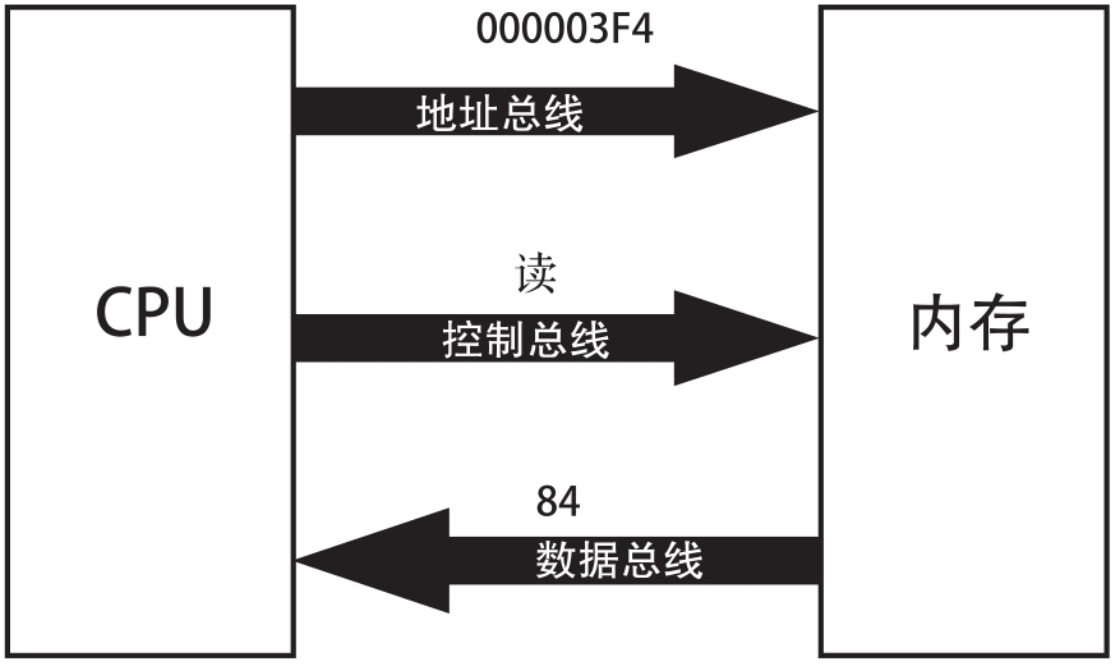

图7-15 CPU请求读取地址000003F4中的值，得到返回值84

在图7-15展示的例子中，CPU需要读取内存地址000003F4中存储的值。为此，CPU把000003F4写到地址总线。CPU还要在控制总线上设置一个特定的值，指示它希望执行一个读操作。这些总线的更新值作为内存控制器（管理与内存的交互的电路）的输入，告诉控制器CPU想要读取存储在内存地址000003F4中的值。作为回应，内存控制器检索存储在地址000003F4中的值（本例中为84），并把它写到数据总线上，然后CPU可以从总线上读取该值。

7.6 总结

本章介绍了计算机硬件：执行指令的中央处理器（CPU）、在上电时保存指令和数据的随机存取存储器（RAM），以及与外界交互的输入/输出（I/O）设备。你了解了内存由单个位存储单元组成，SRAM中用触发器实现存储单元，DRAM中用晶体管和电容器实现存储单元。我们还介绍了内存地址是如何工作的，其中的每个地址都指向一个内存字节。你了解了CPU架构，包括x86和ARM。我们探讨了CPU内部是如何工作的，了解寄存器、

ALU和控制单元。我们介绍了辅存和其他类型的I/O设备。最后，我们还讨论了总线通信。

第8章将从硬件转到使计算机独一无二的元素——软件。我们将讨论处理器执行的底层指令，以及这些指令是如何组合在一起来执行有用的操作的。你将有机会用汇编语言编写软件，并用调试工具来探究机器码。

第8章

机器码与汇编语言

我们已经介绍了计算机的物理部分：内存、CPU和I/O设备。理解计算机的硬件部分很重要，但是硬件只起一半作用。计算机的魔力在于软件。软件把计算机从固定用途的设备变成了高度灵活的设备，它能轻松地实现新的功能！本章将介绍底层软件——机器码和汇编语言。我发现交互方法是理解这些主题的最好方法，所以本章的大部分内容都体现在设计中。

8.1 软件术语

要讨论软件，需要首先介绍一些术语。告诉计算机做什么的指令被称为“软件”，这与硬件形成了对比，硬件是计算机的物理元素。完成任务的一组有序的软件指令被称为“程序”，编程就是编写这种程序的行为。

术语“应用程序”有时被用作程序的同义词，尽管应用程序往往指的是直接与人进行交互的程序，而不是与软件或硬件交互的程序。应用程序也可能由多个程序构成。app这个词大约在2008年开始流行，其中含有的其他意思将在第13章中介绍。

另一个指代一组软件指令的名称是“计算机代码”或“代码”。CPU执行“机器码”，而软件开发人员一般用高级编程语言编写源代码。术语“源代码”是指由开发人员最初编写的程序文本。这种代码一般不会用CPU能直接理解的形式来编写，因此在其能被计算机运行之前，还需要进行其他处理。我将在第9章详细介绍源代码和高级编程语言，但是现在，我们先来看看软件的基础：机器码。

机器码是二进制机器语言指令形式的软件。如第7章所述，CPU的架构决定了CPU能理解的指令。就像人类语言是用词汇表构成的一样，机器语言是用CPU系列已知的指令列表构成的。词汇表中的词语排列成句子来表达含义，CPU指令排列成程序后也具有相同功能。