# 第七章 输入/输出

7.1 外部设备

7.2 I/O模块

7.3 编程I/O

7.4 中断驱动I/O

7.5 直接内存访问（DMA）

7.6 直接Cache访问（Direct Cache Access）

7.7 I/0 通道和处理器

7.8 外部互联标准

7.9 IBM ZEnguleEC12 I/O结构

7.10 关键术语、复习题与问题

**学习目标**

通过学习本章节，你应当做到：

1. 解释I/O模块在计算机组成中的作用
2. 理解程序I/O和中断驱动I/O的区别，并讨论它们的相关优点
3. 概述直接内存访问（DMA）的操作过程
4. 概述直接高速存储器访问（direct cache access）过程
5. 解释I/O通道的功能和作用

除了处理器和一组内存模块外，计算机系统的第三个主要组成部分是一组I/O模块。每个模块与系统总线或中央交换机连接，并控制一个或多个外围设备。一个I/O模块并不是简单的用线连接到系统总线的一套机械设备。另外，I/O模块还包含在外围设备和总线直接执行通信功能的逻辑。

读者可能会疑惑为什么我们不把外设直接连接到系统总线上，其原因如下：

1. 有大量的外设需要多种操作方法。通过合并必要的逻辑到处理器上来控制一系列的设备是不切实际的。
2. 外设的数据传输速率常常比存储器和处理器要慢很多。因此，使用高速传输的系统总线来与外设直接通信是不切实际的。
3. 另一方面，有些外设的数据传输速率比存储器和处理器要快。同样的，如果不能处理得当，这种速率上的不匹配也会导致效率低下。
4. 外设常常会使用与它相连计算机不同的数据格式和字长。
5. 因此，I/O模块是必要的。此模块有两个主要的作用（如图Figure 7.1）：
   1. 通过系统总线或者中央交换机联接到处理器和内存上。
   2. 通过调整数据链路连接一个或多个外部设备。

我们先通过对外设进行一段简短的讨论来开启这一章的学习，然后将对I/O模块的结构和功能进行概述。之后我们将研究通过与处理器和内存协调合作，实现I/O功能的多种方法：内部I/O接口。之后，我们将考察直接内存访问（DMA）的一些细节以及直接高速缓存访问（direct cache access）的更前沿的创新点。最后，我们会研究建立在I/O模块和外部世界的外部I/O接口（external I/O interface）。

## 7.1 外部设备

I/O操作是通过各种各样的外部设备完成的，这些外部设备可在外部环境和计算机之间提供一种交换数据的方法。外部设备通过一个到I/O模块的链接连接到计算机（如图7.1）。该链接用于在I/O模块和外部设备之间交换控制、状态和数据信息。连接到I/O模块的外部设备通常被称为外围设备，或者简单地称为外设。

我们可以将外部设备大致分为三类：

* **人类可读的**：适合与计算机用户进行通信；
* **机器可读的**：适合与和设备沟通；
* **交流**：适合用于与远程设备通信。

人类可读设备的例子可以参考视频显示终端（VDT）和打印机。机器可读设备的例子可以参考磁盘和磁带系统，以及传感器和执行器，例如其在机器人应用中的使用。注意，在本章中我们将磁盘和磁带系统视为I/O设备，而在第六章中我们将它们视为内存设备。从功能的角度来看，这些设备是内存层次结构的一部分，在第6章中适当讨论了它们的使用。从结构的角度来看，这些设备由I/O模块控制，因此将在本章中进行讨论。

通信设备使计算机可以与远程设备交换数据，远程设备可以是人类可读设备，例如终端，机器可读设备，甚至可以是另一台计算机。

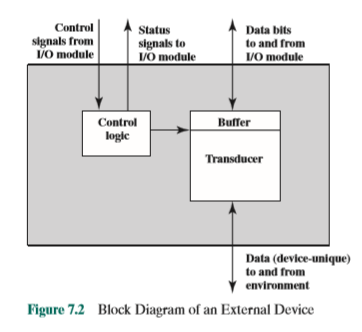
一般来说，外部设备的特性可表示为图7.2。I/O模块的接口采用控制、数据和状态信号的形式。控制信号确定设备将执行的功能，例如向I/O模块(INPUT或READ)发送数据、接受来自I/O模块(OUTPUT或WRITE)的数据、报告状态或执行特定于设备的一些控制功能(例如，定位磁盘头)。数据是一组要发送到I/O模块或从I/O模块接收的比特。状态信号指示设备的状态。例如READY/-NOT-READY，可以显示设备是否准备好进行数据传输。

与设备相关联的控制逻辑控制设备的操作来响应来自I/O模块的方向。该传感器在输出期间将数据从电能转换为其他形式的能量，在输入期间将数据从其他形式的能量转换为电能。通常，缓冲器与传感器相关联，以临时保持在I/O模块和外部环境之间传输的数据。8-16位的缓冲区大小对于串行设备来说是很常见的，而面向块存储的设备（如磁盘驱动器控制器）可能具有更大的缓冲区。

我们将在第7.7节中研究I/O模块与外部设备之间的接口。外部设备和环境之间的接口超出了本书的讨论范围，不过我们将在这里给出几个简单的例子。

**键盘/显示器**

最常见的计算机/用户交互方式是键盘/显示器交互。用户通过键盘输入信息，信息之后被传送到计算机中，并且还可以显示在显示器上。此外，显示器可显示由计算机提供的数据。



交换的基本单位是字符。每个字符都有一个对应的代码，通常长度为7或8位。最常用的文本编码是国际参考字母表（International Reference Alphabet，即IRA1）。此编码中的每个字符均由唯一的7位二进制代码表示；因此，一共可以表示128个不同的字符。字符可以分为两类：可打印字符和控制字符。可打印字符指的是字母、数字以及其他可以打印在纸上或显示在屏幕上的特殊字符。一些控制字符与控制字符的打印或显示有关；一个典型例子是回车字符。其他控制字符与通信过程有关。详情见附录H。

对于键盘输入，当用户按下键时，这产生一个电子信号，该信号由键盘中的传感器解释，并转换成相应的IRA码的位模式。然后，这个位模式被传输到计算机中的I/O模块。在计算机上，文本可以存储在相同的IRA代码中。在输出时，IRA代码字符通过I/O模块传输到外部设备。该装置处的传感器解释该代码，并将所需的电子信号发送到输出装置，以显示所指示的字符或执行所请求的控制功能。

**磁盘驱动器**

磁盘驱动器包含用于与I/O模块交换数据、控制和状态信号的电子设备以及用于控制磁盘读/写机制的电子设备。在固定磁头磁盘中，传感器能够在移动磁盘表面上的磁性模式和设备缓冲区中的位之间进行转换（如图7.2）。移动磁头磁盘还必须能够使磁盘臂在磁盘表面径向进出移动。

## 7.2 I/O模块

**模块功能**

I/O模块的主要功能或需求可以分为以下几类：

* 控制与计时
* 处理器通信
* 设备通信
* 数据缓冲
* 误差检测

在任何时间段内，处理器可以以不可预测的模式与一个或多个外部设备通信，这取决于程序对于I/O的需求。

注释：IRA定义在ITU-T提议T.50中，以前被称为国际字母表5（IA5）。美国国家版本的IRA被称为美国信息交换标准代码（即ASCII）。

内部资源，例如主存储器和系统总线，必须在各种活动之间共享，这其中包括数据I/O。因此，I/O功能包括**控制和计时**，以协调内部资源和外部设备之间的通信流。例如，控制从外部设备到处理器的数据传输可能涉及到以下的步骤：

1. 处理器询问I/O模块以确认其所连接设备的状态。

2. I/O模块返回设备状态。

3. 如果设备是可操作的并且已经准备好传输，则处理器通过命令向I/O模块请求数据传输。

4. I/O模块从外部设备获得数据单元(例如，8或16位)。

5. 数据从I/O模块传输到处理器。

如果系统采用总线，则处理器和I/O模块的每个交互都会涉及一个或多个总线仲裁。

前面的简化场景还说明了I/O模块必须与处理器和外部设备通信。**处理器通信**涉及以下内容：

* **命令译码**：I/O模块接受来自处理器的命令，通常作为信号在控制总线上发送。例如，磁盘驱动器的I/O模块可以接受以下命令：读取扇区（READ SECTOR）、写扇区（WRITE SECTOR）、寻找轨道号（SEEK）和扫描（SCAN）以及记录ID。
* **数据**：处理器和I/O模块之间通过数据总线进行数据交换。
* **状态报告**：由于外设速度太慢，了解I/O模块的状态非常重要。例如，如果要求I/O模块向处理器（读取）发送数据，那么模块有可能仍在处理之前的I/O命令而未能准备好处理当前命令。这种情况可以用状态信号来报告。常见状态信号有BUSY和READY。同时也会有报告各种错误情况的信号。
* **地址识别**：正如每个内存中每个字都有一个地址一样，每个I/O设备也是如此。因此，I/O模块必须为其控制的每个外围设备识别出一个唯一的地址。

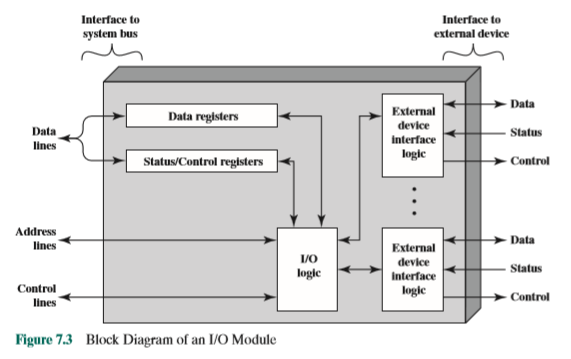
另一方面，I/O模块必须能够执行**设备通信**。这种通信包括命令、状态信息和数据（如图7.2）。

I/O模块的一个基本任务是**数据缓冲**。从图2.1来看，这种功能的需求是显而易见的。虽然进出主存储器或处理器的传输速率相当高，但对于许多外围设备来说，该速率要低几个数量级，并且覆盖范围很广。来自主存储器的数据急速突发地被发送到I/O模块，数据一开始被缓存在I/O模块中，然后以其数据速率发送到外围设备。相反方向上，数据被缓存起来，以免在慢速传输操作中占用内存。因此I/O模块必须能够同时以设备和存储器的速度进行操作。类似地，如果I/O设备以高于存储器访问的速率进行操作，那么I/O模块就需要执行必要的缓冲操作。

最后，I/O模块通常负责**错误检测**并随后向处理器报告错误。其中一类错误包括设备报告的机械和电气故障（例如，卡纸、磁盘磁道损坏）。另一类错误包括从设备传输到I/O模块时比特模式的意外更改。某些形式的错误检测代码通常用于检测传输错误。一个简单的例子是在数据的每个字符上使用奇偶校验位。例如，IRA字符代码占用字节的7位。第八位的值可由前面7位计算出，以保证该字节中的1的总数是偶数（偶奇偶）或奇数（奇偶校验）。当接收到1字节时，I/O模块检查奇偶校验以确定是否发生了错误。

**I/O模块结构**

I/O模块在复杂性和控制外部设备的数量上差异很大。这里我们只做一个大致的描述。（一个特定的设备，Intel 8255A，将在第7.4节中描述）图7.3是 一个I/O模块的一般框图。该模块通过一组信号线（例如，系统总线）连接到计算机的其余部分。传送到模块和从模块传送来的数据被缓冲在一个或多个数据寄存器中。同时还可能存在提供当前状态信息的一个或多个状态寄存器。状态寄存器还可以用作控制寄存器，用于接收来自处理器的详细控制信息。模块内的逻辑通过一组控制线与处理器交互。处理器使用控制线发出命令到I/O模块。



一些控制线可以由I/O模块使用(例如，用于仲裁和状态信号)。模块还必须能够识别和生成与其控制的设备相关联的地址。每个I/O模块具有唯一的地址，或者，如果它控制多个外部设备，则具有一组唯一的地址。最后，I/O模块包含特定于与它控制的每个设备的接口的逻辑。

I/O模块的功能可以允许处理器以一种简单的方式查看各种设备。I/O模块可能会给出外设的性能频谱。I/O模块可以隐藏外部设备的定时、格式和机电学的细节，以便处理器能够根据简单的读写命令以及可能的打开和关闭文件命令来工作。在其最简单的形式中，I/O模块仍然可以让它大部分的控制设备工作（例如，磁带倒带）对处理器可见。

承担大部分细节处理负荷的I/O模块，向处理器提供高级接口，其通常被称为I/O通道或**I/O处理器**。非常原始而且需要详细控制的I/O模块，通常被称为**I/O控制器**或**设备控制器**。I/O控制器在微型计算机上常见，而I/O通道则被使用在大型机上。

在下文中，在不存在混淆的情况下，我们将使用通用术语**I/O模块**，并在必要时使用更具体的术语。

## 7.3编程I/O

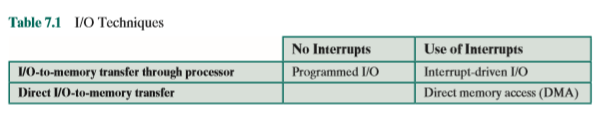
有三种技术可以用于I/O操作的技术。

通过编程I/O，数据可以在处理器和I/O模块之间交换。处理器执行可直接控制I/O操作的程序，包括感测设备状态、发送读或写命令以及传输数据。当处理器向I/O模块发出命令时，它必须等待I/O操作完成。如果处理器比I/O模块速度快，将会浪费处理器时间。

使用中断驱动的I/O，处理器发出I/O命令，继续执行其他指令，当I/O模块完成其工作时，处理器将被I/O模块中断。使用编程和中断I/O两种技术，处理器需要负责从主存储器中提取数据用于输出，并将数据存储在主存储器中用于输入。

另一种技术被称为直接存储器存取（DMA）。在这种模式下，I/O模块和主存储器直接交换数据，而不需要处理器的参与。

表7.1显示了这三种技术之间的关系。在本节中，我们将讨论编程I/O。中断I/O和DMA技术将对应在后续两节介绍。



**编程I/O概述**

当处理器正在执行程序并且遇到与I/O有关的指令时，它通过向适当的I/O模块发出命令来执行该指令。通过编程I/O，I/O模块将执行请求的操作，然后在I/O状态寄存器中置位（如图7.3）。I/O模块不会采取进一步操作来提醒处理器。特别注意，I/O模块不会中断中断处理器。因此，处理器必须定期检查I/O模块的状态，直到发现操作完成。

为了解释编程I/O技术，我们首先从处理器向I/O模块发出的I/O命令的角度来学习，然后从处理器执行I/O指令的角度来分析。

**I/O命令**

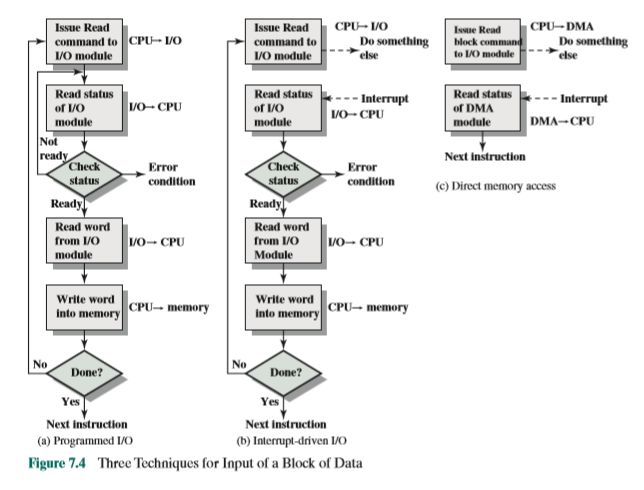
为了执行与I/O相关的指令，处理器发出地址，指定特定的I/O模块和外部设备，并发出I/O命令。当I/O模块被处理器寻址时，它可以接收四种类型的I/O命令：

* **控制命令**：用于激活外设并告诉它要做什么。例如：一个磁带单元可能被命令倒带或向前移动一条记录。这些命令是针对特定类型的外设设定的。
* **测试命令**：用于测试与I/O模块及其外设相关的各种状态条件。处理器希望知道相关联的外围设备是否开机并且可以使用。它还想知道最近一次I/O操作是否完成，以及是否发生错误。
* **读命令**：使I/O模块从外围设备获得数据项并将其放置在内部缓冲区中（如图7.3中的数据寄存器所示）。然后，处理器可以通过请求I/O模块将数据放置在数据总线上来获得数据项。
* **写命令**：使I/O模块从数据总线获取数据（字节或字）项，然后将该数据项发送到外围设备。

图7.4a给出了使用编程I/O将数据块从外围设备（例如，来自磁带的记录）读入内存的示例。数据一次读取一个字（例如，16位）。对于读入的每个字，处理器必须循环进行状态检查，直到它确定该字在I/O模块的数据寄存器中可用。该流程图突出表现出这种技术的主要缺点：这是一个耗时的过程，使处理器不必要地繁忙工作。

**I/O指令**

对于编程的I/O，处理器从存储器中获取的I/O相关的指令，与处理器向I/O模块发出用于执行指令的I/O命令之间存在密切的对应关系。也就是说，指令很容易被映射到I/O命令中，并且通常存在一种简单的一对一关系。指令的形式取决于外部设备的寻址方式。



通常，会有许多I/O设备通过I/O模块连接到系统中。每个设备都被赋予唯一的标识符或地址。当处理器发出I/O命令时，该命令包含所需设备的地址。因此，每个I/O模块必须能够解释地址线，来确认该命令是否是发给自己的。

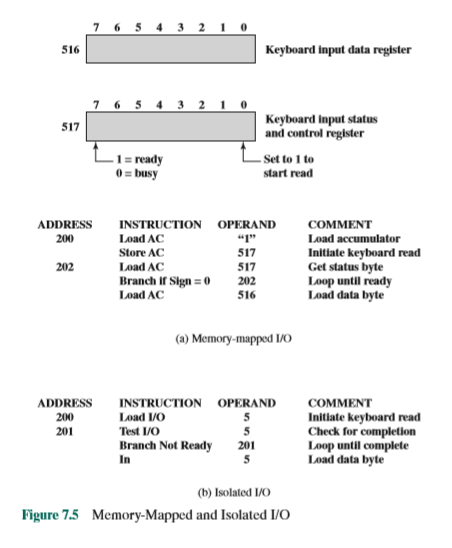
当处理器、主存储器和I/O共享公共总线时，有两种可能的寻址模式：内存映射和隔离。对于**内存映射I/O**，内存和I/O设备共享一个单独的地址空间。处理器将I/O模块的状态和数据寄存器视为内存位置，并使用相同的机器指令来同时访问内存和I/O设备。因此，例如，1对于0条地址线，对应的总计210 = 1024个任意的内存地址组合都可以作为I/O的地址。

有了内存映射的I/O，总线上需要一个读入线和一个写出线。或者，总线可以配置存储器读写以及输入和输出命令行。命令行指定地址引用的是内存位置还是I/O设备。所有的地址都可供双方同时使用。继续刚才的例子，对于10条地址线，系统现在可以同时支持1024个存储器地址和1024个I/O地址。因为用于I/O的地址空间与用于内存的地址空间是彼此隔离的，所以这称为**隔离I/O**。

图7.5对比了这两种编程I/O技术。图7.5a显示了使用内存映射技术，程序员可以看到的一个简单输入设备（例如终端键盘）的接口。假设一个10位的地址，以及一个512位的存储器（位置空间0-511）和最多512个I/O地址（位置空间512-1023）。两个地址专用于来自特定终端的键盘输入。地址516指向数据寄存器，地址517指向状态寄存器，状态寄存器还用作接收处理器命令的控制寄存器。图示的程序将把1字节的数据从键盘读入处理器中的累加寄存器。注意，处理器会循环读取直到数据字节可用。

对于隔离的I/O（图7.5b），I/O端口只能通过特殊的I/O命令访问，这些命令会激活总线上的I/O命令行。

对于大多数类型的处理器，有一组相对较大的用于引用内存的不同指令集。如果使用隔离I/O，则将会有很少的I/O指令。因此，内存映射I/O的一个优势在于，可以使用全部大量的指令，从而实现更高效的编程。缺点是有价值的内存地址空间用完了。内存映射和隔离I/O都是常用的。



## 7.4 中断驱动I/O

编程I/O的问题在于，处理器必须等待很长一段时间才能使所关心的I/O模块准备好接收或传输数据。处理器在等待时必须重复查询I/O模块的状态。结果导致整个系统的性能水平严重下降。

另一种方法是处理器向一个模块发出I/O命令，然后继续执行一些其他的有用的工作。然后，当I/O模块准备好与处理器交换数据时，它将中断处理器来请求服务。处理器于是像以前一样执行数据传输，结束后再恢复之前的工作处理。

让我们首先从I/O模块的角度考虑一下它是如何工作的。对于输入，I/O模块从处理器接收读取（READ）命令。然后，I/O模块继续从相关联的外设读取数据。一旦数据读取到模块的数据寄存器中，模块将通过控制线向处理器发出中断信号。然后，模块将等待，直到进程请求到其数据为止。当请求发出后，模块将其数据放置在数据总线上，然后准备好进行另一I/O操作。

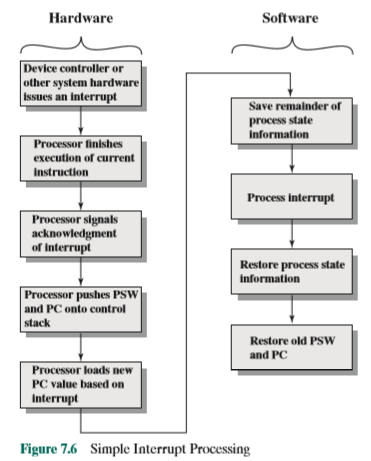
从处理器的角度来看，输入的操作如下。处理器发出一个读取（READ）命令。然后，它关闭并执行其他操作（例如，处理器可能同时处理多个不同的程序）。在每个指令周期结束时，处理器检查中断（如图3.9）。当来自I/O模块的中断发生时，处理器保存当前程序的上下文（例如，程序计数器和处理器寄存器）并处理中断。在这种情况下，处理器从I/O模块读取数据字并将其存储在存储器中。然后，它将恢复正在处理的程序（或其他程序）的上下文，并恢复执行。

图7.4b显示了使用中断I/O读取数据块的情况。与图7.4a相比，中断I/O比编程I/O更有效，因为它消除了不必要的等待。然而，中断I/O仍然会消耗大量的处理器时间，因为从存储器到I/O模块或从I/O模块到存储器的每个数据字都必须经过处理器。

**中断处理**

让我们更详细地考虑一下处理器在中断驱动的I/O中的作用。中断的发生会在处理器硬件和软件上都触发大量事件。图7.6显示了一个典型的事件序列。当I/O设备完成一次I/O操作，以下硬件事件会按顺序发生：

1. 该设备向处理器发出中断信号。
2. 处理器完成当前指令的执行后，回应中断信号，如图3.9所示。
3. 处理器测试中断，确认有一个中断后，并向发出中断的设备发送确认信号。确认信号允许设备移除其中断信号。



1. 处理器现在需要准备将控制转移到中断路由。首先，它需要在中断点保存恢复当前程序所需的信息。所需的最小信息是

(a)处理器的状态，它包含在称为**程序状态字(PSW)**的寄存器中；

(b)要执行的下一个指令的位置，它包含在程序计数器中。这些可以被推送到系统控制堆栈2上。

注释2：有关堆栈操作的讨论见附录I。

1. 处理器现在加载程序计数器，计数器的值为响应此中断的中断处理程序的入口地址。根据计算机体系结构和操作系统设计，可能有单个程序；或者每个中断类型有一个程序；又或者每个设备和每个中断类型有一个程序。如果有多个中断处理程序，处理器必须确定调用哪个。该信息可能已经包括在原始中断信号中，否则处理器可能必须向发出中断的设备发出请求，用以获得包含所需信息的响应。

一旦程序计数器被加载，处理器就进入下一个指令周期，该周期从取指令开始。因为取指令的结果是由程序计数器的内容决定的，所以控制被转移到中断处理程序处。执行这个程序会导致以下操作：

1. 此时，程序计数器和与中断程序相关的PSW已经保存在系统堆栈中。然而，还存在其他被认为是执行程序的“状态”的一部分的信息。特别地，需要保存处理器寄存器的内容，因为这些寄存器可能被中断处理程序使用。因此，需要保存所有这些值以及任何其他的状态信息。通常，中断处理程序将首先保存堆栈上所有寄存器的内容。图7.7a显示了一个简单的例子。在这种情况下，用户程序在位置N的指令之后被中断。所有寄存器的内容外加上下一个指令（N+1）的地址被压到堆栈上。堆栈指针更新为指向堆栈的新顶部，程序计数器更新为指向中断服务程序的起始位置。
2. 中断处理程序接着处理中断。这包括检查与I/O操作或导致中断的其他事件相关的状态信息。还可能涉及向I/O设备发送附加命令或确认。
3. 当中断处理完成时，保存的寄存器值从堆栈中检索并恢复到寄存器（例如，参见图7.7b）。
4. 最后一步是从堆栈中恢复PSW和程序计数器值。因此，下一个要执行的指令来自先前中断的程序。

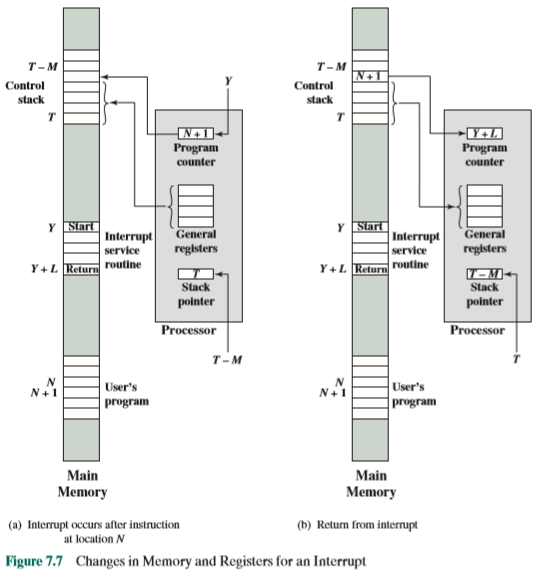
请注意，保存有关中断程序的所有状态信息以便以后恢复非常重要。这是因为中断不是从程序调用的例程。相反，中断可以在任何时间发生，因此可以在用户程序执行的任何点发生。它的发生是不可预知的。事实上，正如我们将在下一章中看到的，这两个程序可能没有任何共同之处，并且可能属于两个不同的用户。

**设计问题**

在实现中断I/O时出现两个设计问题。首先，因为几乎总是会有多个I/O模块，处理器如何确定哪个设备发出中断？其次，如果发生了多个中断，处理器如何决定处理哪个中断？

让我们先考虑设备识别。有四类常用的技术：

* + 多重中断线
  + 软件轮询
  + 菊花链（硬件轮询，矢量化）
  + 总线仲裁（矢量化）



解决这个问题的最直接的方法是在处理器和I/O模块之间提供**多中断线**。然而，要使用多于几个总线或处理器引脚来中断线路是不切实际的。因此，即使使用多条线，每条线也有可能附加有多个I/O模块。因此，每条中短线必须使用另外三种技术中的一种。

另一种选择是**软件轮询**。当处理器检测到中断时，它分支到中断服务例程，该例程通过轮询每个I/O模块以确定哪个模块发出了中断。轮询可以采用单独的命令行的形式（例如，TESTI/O）。在这种情况下，处理器发出TESTI/O命令并将特定I/O模块的地址放在地址线上。如果I/O模块设置了中断，则作出积极响应。或者，每个I/O模块可以包含一个可寻址状态寄存器。处理器可通过读取每个I/O模块的状态寄存器来识别中断模块。一旦识别出正确的模块，处理器就分支到对应该设备的设备服务例程。

软件轮询的缺点是耗时。一种更有效的技术是使用菊花链，它实际上是提供硬件轮询。图3.26所示的是一个配置菊花链的例子。对于中断，所有I/O模块共享一个公共中断请求线。中断确认线通过模块链接起来。当处理器检测到一个中断时，它发出一个中断确认。该信号通过一系列I/O模块传播，直到到达请求模块。请求模块通常通过在数据线上放置一个字进行响应。这个字被称为向量，要么是I/O模块的地址，要么是其他的一些唯一的标识符。无论哪种情况，处理器都使用向量作为指向适当的设备服务例程的指针。这避免了首先执行通用中断服务例程的必要。这种技术被称为矢量中断。

还有一种利用矢量中断的技术，那就是总线仲裁。通过总线仲裁，I/O模块必须首先获得总线的控制权，然后才能获得中断请求线。因此，一次只有一个模块可以获得总线请求线。当处理器检测到中断时，它在中断确认线上响应。然后，请求模块将其向量放置在数据线上。

上述技术用于识别请求I/O模块。当不止一个设备正在请求中断服务时，它们还提供了一种分配优先级的方法。对于多行请求的情况，处理器只选择具有最高优先级的中断行。使用软件轮询，模块被轮询的顺序决定它们的优先级。类似地，雏菊链上的模块的顺序也决定了它们的优先级。最后，总线仲裁可以采用优先级方案，如3.4节所述。

现在我们讨论两个中断结构的例子。

**英特尔82C59A中断控制器**

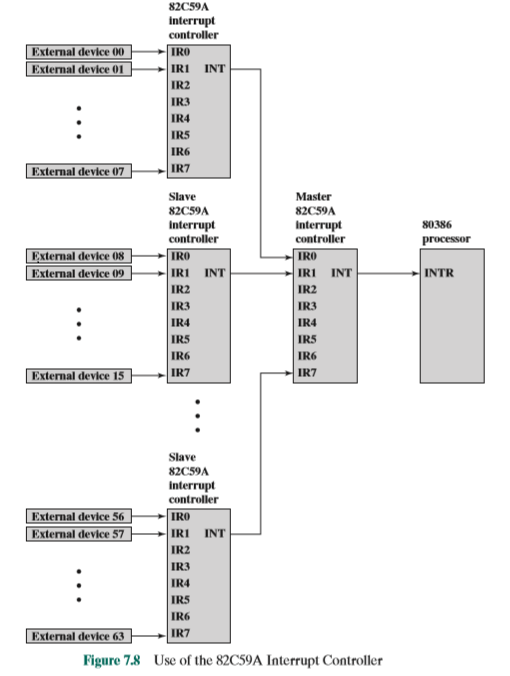
英特尔80386提供单个中断请求（**INTR**）线和单个中断确认（**INTA**）线。为了允许80386处理各种设备和优先级结构，它通常配置有外部中断仲裁器82C59A。外部设备连接到82C59A，82C59A反过来又连接到80386。

图7.8显示了82C59A连接多个I/O模块到80386的使用情况。单个82C59A可以处理多达八个模块。如果需要对八个以上的模块进行控制，则可以使用级联布置来处理多达64个模块。

82C59A的唯一职责就是管理中断。它接受来自附加模块的中断请求，确定哪个中断具有最高优先级，然后通过升起INTR线向处理器发信号。处理器通过INTA线进行确认。这将提示82C59A在数据总线上放置适当的向量信息。然后，处理器可以继续处理中断，并直接与I/O模块通信来读取或写入数据。

82C59A是可编程的。80386通过在82C59A中设置控制字来确定要使用的优先级方案。下面是几种可能的中断模式：

* **完全嵌套**：中断请求按优先级从0(IR0)到7(IR7)排序。



* **轮换的**：在一些应用中，许多中断设备具有相同的优先级。在这种模式下，设备在被服务之后，在组中会被降到到最低优先级。
* **特殊屏障**：允许处理器禁止来自某些设备的中断。

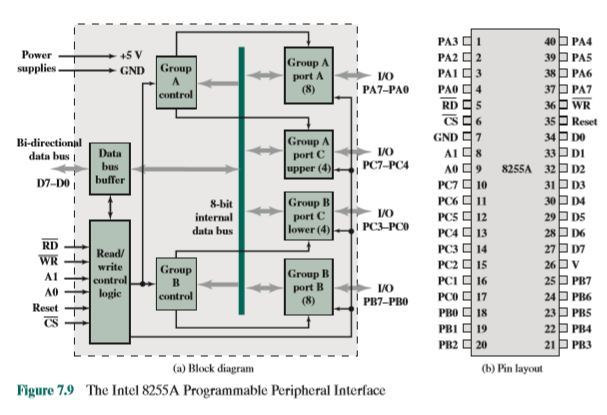
**英特尔8255A可编程外围接口**

现在我们来介绍Intel 8255A可编程外围接口，其可同时用于编程I/O和中断驱动I/O。8255A是一个最初设计用于Intel 80386处理器的单片、通用I/O模块。它如今已被其他制造商复制生产，是一种广泛使用的外围控制器芯片。它可以作为简单I/O设备的控制器用于微处理器，也可以用于包括微控制器系统的嵌入式系统。

**架构与操作**图7.9显示了40针封装的一般框图和引脚分配。如引脚布局所示，8255A包括以下线：

* + **D0- D7**:设备的数据I/O线。从8255A读取和写入的所有信息都通过这几条线传输。
  + **CS（芯片选择输入）**:如果这行是逻辑0，则微处理器可以读写到8255A。
  + **RD（读取输入）**：如果这行是逻辑0，同时CS输入是逻辑0，则8255A的数据可以被输出到系统数据总线上。
  + **WR（写入输入）**：如果该输入线是逻辑0，同时CS输入是逻辑0，则数据从系统数据总线写入8255A。

**复位**：如果该输入线是逻辑1，则8255A被置于其复位状态。所有外围端口都被设置为输入模式。

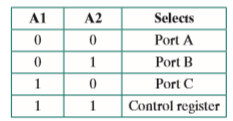
****

* **PA0- PA7，PB0- PB7，PC0-PC7**:这些信号线被用作8位I/O端口。它们可以连接到外围设备。
* **A0，A1**：这两条输入线的逻辑组合决定了8255A的数据被写入到哪个内部寄存器或从哪个内部寄存器读取。

图7.9a框图的右侧是8255A的外部接口，24条I/O线被分成三个8位组（A、B、C）。每个组可以充当8位I/O端口，从而可以为三个外围设备提供连接。此外，C组被细分为4位组（CA和CB），它们可以与A和B I/O端口一起使用。以这种方式配置，组C线携带控制和状态信号。

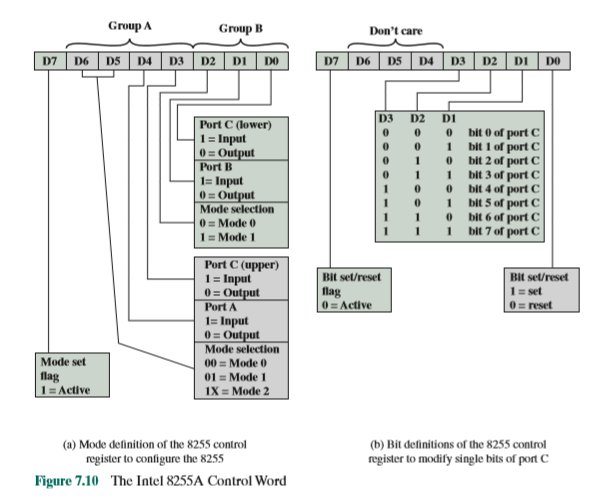
框图的左侧是连接到微处理器系统总线的内部接口。它包括8位双向数据总线(D0到D7)，用于在微处理器和I/O端口之间传输数据以及传输控制信息。

处理器通过处理器中的8位控制寄存器来控制8255A。处理器可以设置控制寄存器的值以指定各种操作模式和配置。从处理器的角度来看，存在一个控制端口，控制寄存器位在处理器中被设置后，通过线路D0-D7发送到控制端口。这两个地址线指定三个I/O端口或控制寄存器中的一个，如下所示：



因此，当处理器同时将A1和A2都设置为1时，8255A将数据总线上的8位值视为控制字。当处理器传输一个8位控制字，其D7行设置为1（图7.10 a），控制字用于配置24条I/O线的工作模式。这三种模式是：

* 模式0：这是基本的I/O模式。这三组8条外设线充当三个8位I/O端口。每个端口可以指定为输入或输出。如果端口被定义为输出，则数据只能被发送到端口，并且如果端口被设置为输入，则只能从端口读取数据。
* 模式1：在这种模式下，端口A和B可以被配置为输入或输出，连接端口C的线路被用作端口A和B的控制线。控制信号有两个主要作用：“握手”和中断请求。握手是一种简单的计时机制。发送器使用一条控制线作为**数据就绪（DATA READY）**线，以指示当前I/O数据线上有数据。接收机使用另一条线路作为**应答（ACKNOWLEDGE）**，用以指示数据已被读取，并且数据线可以被清除。另一行可被指定为**中断请求线(INTERRUPT REQUEST)**并接回系统总线。

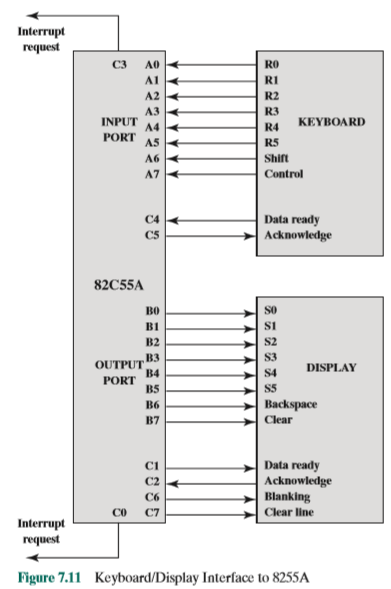


* 模式2：这是一个双向模式。在这种模式下，端口A可以被配置为端口B上的双向通信的输入或输出线，端口B线提供相反的方向。同样，端口C线用于控制信令。

当处理器将D7设置为逻辑0（图7.10 b）时，控制字被用来单独编程端口C的位值。这种特性很少使用。

**键盘/显示实例** 因为8255A是通过控制寄存器编程的，所以它可以用来控制各种简单的外围设备。图7.11显示出它可用于控制键盘/显示终端。键盘提供8位输入。其中的两位，即SHIFT和CONTROL，对于处理器中的键盘操控程序的执行有特殊的意义。然而，这种解释对于8255A是透明的，8255A简单地接受8位数据并将它们放在系统数据总线上。并将这两条握手控制线用于键盘。

显示器也使用8位数据端口连接。同样，有两位具有对8255A透明的特殊意义。除了两条握手线之外，还有两条提供额外控制功能的线。



## 7.5 直接内存访问（DMA）

**程序驱动和中断驱动I/O的缺点**

中断驱动的I/O虽然比简单的编程I/O更有效，但是仍然需要处理器的主动干预来在内存和I/O模块之间传输数据，并且任何数据传输都必须穿过一条通过处理器的路径。因此，这两种形式的I/O都具有两个固有的缺陷：

1. I/O传输速率受处理器测试和服务设备的速度限制。
2. 处理器被绑在管理I/O传输上；必须为每个I/O传输执行大量指令（例如，图7.5）。

在这两个缺点之间有几分权衡。考虑数据块的传输。使用简单的编程I/O，处理器被分配给I/O任务，并且可以以相当高的速率传输数据，而不做其他任何事情。中断I/O在某种程度上释放了处理器，但代价是损失I/O传输速率。然而，这两种方法对处理器活动和I/O传输速率都有不利影响。

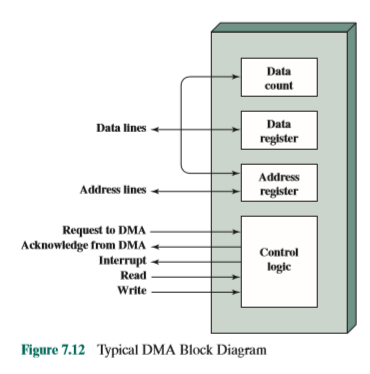
当需要移动大量数据时，需要一种更有效的技术：直接内存访问（DMA）。

**DMA功能**

DMA的实现涉及到系统总线上的附加模块。DMA模块（如图7.12所示）能够模拟处理器，并且确实能够从处理器接管对系统的控制。它需要这样做来通过系统总线向内存和从内存传输数据。为此，DMA模块仅能在处理器不需要总线时使用总线，或者必须强制处理器暂停操作。后一种技术更为常见，被称为周期窃取，因为DMA模块实际上是在窃取总线周期。

当处理器希望读写数据块时，它通过向DMA模块发送以下信息来给DMA模块发出命令：

* 无论请求读还是写，都使用处理器和DMA模块之间的读写控制线。
* 涉及到的在数据链路上通信的I/O的地址。



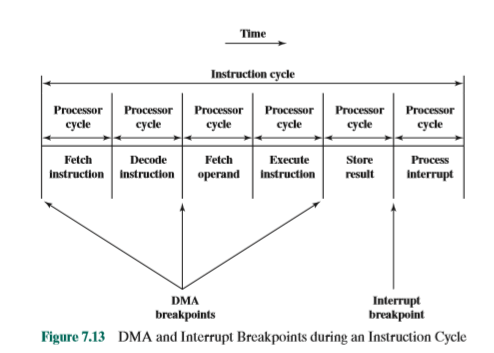
* 用于从数据线读取或写入、在数据线上通信并由DMA模块存储在其地址寄存器中内存的起始地址。
* 要读或写的字数，仍然通过数据线通信并存储在数据计数寄存器中。

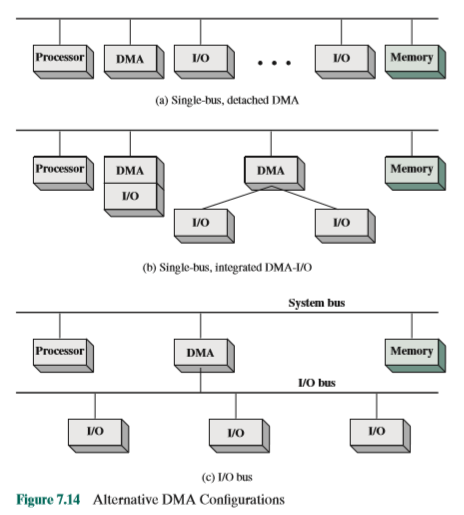
然后处理器继续进行其他工作。它已经将这个I/O操作委托给DMA模块。DMA模块将整个数据块，一次一个字地直接传送到存储器或从存储器传送过来，而不经过处理器。当传输完成时，DMA模块向处理器发送中断信号。因此，处理器只在传输开始和结束时工作（如图7.4 c）。

图7.13显示了在指令周期中处理器可能被挂起的位置。在每种情况下，处理器在正要使用总线之前被挂起。DMA模块随后会传送一个字并将控制权归还给处理器。注意，这不是中断；处理器不保存上下文或执行其他操作。事实上，处理器暂停了一个总线周期。总体上导致处理器执行变得更慢。然而，对于一个多字I/O传输来说，DMA比中断驱动I/O或编程I/O要快得多。

DMA机制可以通过多种方式配置。一些可能的配置如图7.14所示。在第一个例子中，所有模块共享同一个系统总线。DMA模块作为代理处理器，使用编程的I/O通过DMA模块在内存和I/O模块之间交换数据。这种配置虽然可能成本不高，但显然效率很低。与使用处理器控制的编程I/O一样，每传输一个字需要两个总线周期。

通过集成DMA和I/O功能，可以大大减少所需的总线周期数。如图7.14b所示，这意味着在DMA模块和一个或多个I/O模块之间存在一条路径，该路径不包括系统总线。DMA

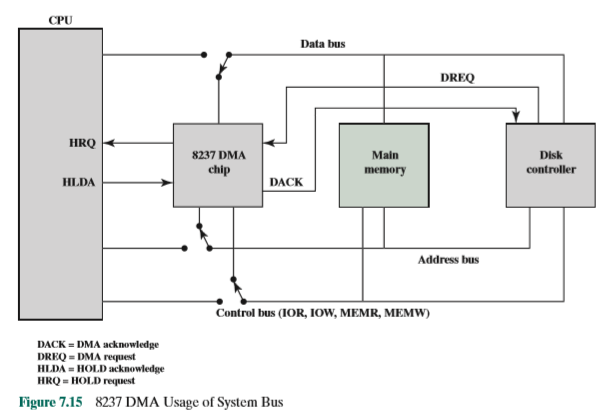




逻辑实际上可以是I/O模块的一部分，或者它可以是控制一个或多个I/O模块的独立模块。通过使用I/O总线将I/O模块连接到DMA模块（如图7.14c），可以进一步实现这个概念。这将DMA模块中的I/O接口数量减少到一个，并提供了易于扩展的配置。在这两种情况下（图7.14b和c），DMA模块与处理器和存储器共享的系统总线仅由DMA模块用于与存储器交换数据。DMA和I/O模块之间的数据交换在系统总线之外进行。

**英特尔8237A DMA控制器**

英特尔8237A DMA控制器接口连接到80\*86系列处理器和DRAM存储器，从而提供DMA能力。图7.15显示出DMA模块的位置。当DMA模块需要使用系统总线（数据、地址和控制）来传输数据时，它向处理器发送一个名为HOLD的信号。处理器使用HLDA(hold acknowledge)信号进行响应，表示DMA模块可以使用总线。例如，如果DMA模块要将一



块数据从存储器传输到磁盘，它将执行以下操作：

1. 外围设备（如磁盘控制器）通过置高DREQ（DMA请求）信号来请求DMA服务。
2. DMA将HRQ（保持请求）置高，通过它的HOLD引脚向CPU发出信号，表明它需要使用总线。
3. CPU将完成当前的总线周期（不一定就是当前的指令），并通过对其HDLA（保持确认）进行高置来响应DMA请求，从而通知8237DMA其可以继续使用总线来执行其任务。只要DMA正在执行其任务，HOLD信号就必须保持高电位有效。
4. DMA将激活DACK（DMA确认）信号，用以告诉外围设备它将开始传输数据。
5. DMA通过在地址总线上放置块的第一个字节的地址并激活MEMR信号，开始将数据从存储器传输到外围设备，从而将字节从存储器读入数据总线，然后激活IOW信号将数据写入外围设备。然后DMA减小计数器值，增大地址指针，并重复这个过程，直到计数达到零，任务完成。
6. 在DMA完成其工作后，它将释放HRQ，向CPU发出信号，表明它可以重新获得对其总线的控制。

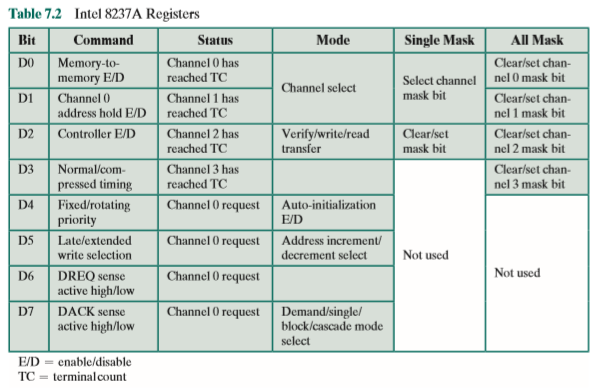
当DMA使用总线传输数据时，处理器是空闲的。通常，当处理器使用总线时，DMA是空闲的。8237 DMA被称为飞越DMA控制器。这意味着从一个位置移动到另一个位置的数据不经过DMA芯片，也不存储在DMA芯片中。因此，DMA只能在一个I/O端口和一个存储器地址之间传输数据，而不能在两个I/O端口或两个存储器位置之间传输数据。然而，随后我们也会介绍，DMA芯片可以通过寄存器执行存储器到存储器的传输。

8237包含四个可以独立编程的DMA信道，并且这些信道中的任何一个都可以在任何时刻处于活跃状态。这些信道被编号为0, 1, 2和3。

8237有成组的5个控制/命令寄存器，可用于编程和控制在其中一个信道上的DMA操作（表7.2）：

* **命令**：处理器加载这个寄存器来控制DMA的操作。D0可使能一个从存储器到存储器的传输，其中通道0用于将字节传输到8237暂存寄存器，通道1用于将字节从寄存器传输到存储器。当存储器到存储器传输方式已使能，D1可用于禁用通道0上的递增/递减，从而可以将固定值写入内存块。D2使能或禁用DMA。
* **状态**：处理器读取这个寄存器以确定DMA状态。位D0-D3用于指示信道0-3是否已达到它们的TC(终端计数)。位D4-D7由处理器用来确定是否有信道有DMA请求挂起。
* **模式**：处理器设置这个寄存器来确定DMA的操作模式。比特D0和D1用于选择信道。其他比特用于给选定的信道选择各种操作模式。位D2和D3确定传输是从I/O设备到存储器(写入)还是从存储器到I/O(读取)或者是验证操作。如果设置了D4，那么在DMA数据传输结束时，用它们的原始值重新加载内存地址寄存器和计数寄存器。位D6和D7决定使用8237的方式。在单模式下，传输一个字节的数据。块和需求模式用于块传输，需求模式允许传输提前结束。级联模式允许级联多个8237来将信道数量扩展到4个以上。
* **单掩模**：处理器可以设置这个寄存器。位D0和D1用来选择通道。位D2用来清除或设置该通道的掩模位。通过此寄存器，可以屏蔽（禁用）或解除屏蔽（开启）特定通道的DREQ输入。虽然命令寄存器可以用于禁用整个DMA芯片，但是单个掩模寄存器可以让程序员禁用或启用特定信道。
* **全掩模**：该寄存器类似于单个掩码寄存器，只是所有四个通道都可以通过一个写操作进行掩模或解掩模。

此外，8237A有八个数据寄存器：每个通道对应一个存储器地址寄存器和一个计数寄存器。处理器设置这些寄存器来显示受传输影响的主存大小的位置。



## 7.6 直接Cache访问

DMA是利用周边设备和网络I/O流量提高I/O性能的有效手段。然而，由于网络I/O的数据速率急剧增加，DMA无法扩展来满足增加的需求。这种需求主要来自于10Gbps 和100Gbps以太网交换机的广泛部署。这些交换机被用来处理大量往返于数据库服务器和其他高性能系统[STAL14a]的数据。次要但日益重要的流量则来源来自千兆位范围的Wi-Fi。处理3.2Gbps和6.76Gbps的网络Wi Fi设备正逐渐被广泛应用，产生了对企业系统的需求[STAL14b]。

在本节中，我们将展示如何使I/O函数能够直接访问cache，从而提高性能，这种技术称为**直接cache访问（DCA）**。在本节中，我们只关注最接近主内存的cache，称为**最后一级cache**。在一些系统中，会被称作L2高速缓存，在其他系统中，会被称作L3高速缓存。

首先，我们来介绍当代多核系统使用片上共享cache来提高DMA性能的方法。这种方法将使DMA功能能够直接访问上一级高速缓存。接下来，我们研究处理高速网络流量时出现的与高速缓存相关的性能问题。从这里，我们来看几个不同的DCA策略，它们均被设计用来提高网络协议的处理性能。最后，本节将介绍由Intel公司实现的DCA方法，其又被称为直接数据I/O。

**使用共享的最后一级缓存的DMA**

正如在第1章（参见图1.2）中所讨论的，现代多核系统包括专用于每个核的缓存和附加级别的共享高速缓存（L2或L3）。随着可用最后一级高速缓存大小的增加，系统设计人员增强了DMA功能，以便DMA控制器能够以类似于内核的方式访问共享高速缓存。为了阐明DMA和缓存之间的交互，首先描述一个特定的系统架构是很有用的。以下是Intel Xeon系统的概述。

**Xeon多核处理器**Intel Xeon是Intel的高端高性能处理器系列，用于服务器、高性能工作站和超级计算机。Xeon家族的许多成员都使用环形互连系统，如图7.16中的Xeon E5-2600/4600所示。

E5-2600/4600可以在一个芯片上配置最多8个核。每个内核都有专用的L1和L2高速缓存。有一个多达20 MB的共享L3高速缓存。L3缓存被划分为多个片，每个片与每个核心相关联，尽管每个核心可以寻址整个高速缓存。此外，每个片都有自己的缓存管道，因此请求可以与并行发送到片。

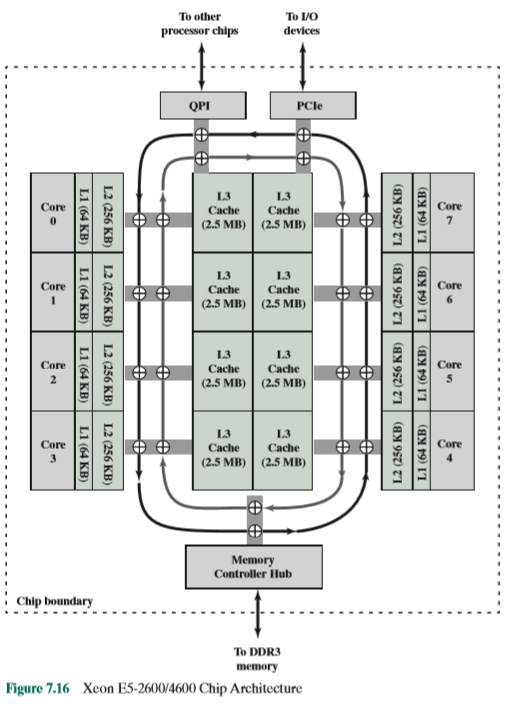
双向高速环形互连网络将内核、最后一级高速缓存、PCIe和集成存储器控制器（IMC）连接在一起。

本质上，环的操作如下：

1. 附加到双向环（QPI、PCIe、L3缓存、L2缓存）的每个组件都被视为环代理，并实现环代理逻辑。
2. 环代理通过分布式协议进行协作，以时隙的形式请求和分配对环的访问。
3. 当代理有数据要发送时，它会选择到目的地的路径最短的环方向，并在调度时隙可用时发送。

环结构在一定程度上为多核提供了良好的性能和规模。对于具有更多核的系统，使用多个环结构，每个环支持多核中的一部分。

**DMA在高速缓存中的应用** 在传统的DMA操作中，数据在主存储器和I/O设备之间通过系统互连结构进行交换，例如总线结构、环结构或QPI点对点矩阵结构。因此，例如说如果Xeon E5-2600/4600使用传统的DMA技术，输出过程将按照如下进行：在内核上运行的I/O驱动程序将向I/O控制器（在图7.16中标记为PCIe）发送I/O命令，其中包含要传输的数据的主存储器中缓冲区的位置和大小。I/O控制器发出读请求，该读请求被路由到存储器控制器集线器(MCH)，该MCH访问DDR3存储器上的数据并将其置于系统环上以传送到I/O控制器。L3高速缓存不参与该事务，并且需要一个或多个片外存储器读取。类似地，对于输入，数据从I/O控制器到达，通过系统环传送到MCH并被写入到主存储器。MCH还必须使与更新的存储器位置相对应的任何L3 缓存线无效。在这种情况下，需要一个或多个片外存储器写入。此外，如果应用程序希望访问新数据，则需要主存储器读取。



大量最后一级高速缓存的可用性，为一种更高效的技术提供了可能。Xeon E5-2600/4600就使用了这种技术。对于输出，当I/O控制器发出读请求时，MCH首先检查数据是否在L3高速缓存中。如果应用程序最近要将数据写入到要输出的内存块，在这种情况下，MCH将数据从L3高速缓存引导到I/O控制器；不需要主存储器访问。然而，它还导致数据从高速缓存中删除，即I/O设备的读取行为导致了数据被删除。因此，I/O操作之所以可以高效地进行，是因为它不需要主存储器访问。但是，如果应用程序将来确实需要该数据，则必须将其从主存储器读回L3高速缓存。Xeon E5-2600/4600上的输入操作如上段落图所述，不涉及L3高速缓存。因此，性能的改进仅涉及输出操作。

最后一点。虽然输出传输是直接从缓存到I/O控制器的，但是术语直接高速缓存访问并不用于该特性。相反，这个术语是为I/O协议应用程序保留的，如本节其余部分所述。

**高速缓存相关的性能问题**

网络流量以协议块序列的形式传输，称为分组或协议数据单元。最低层或链路层协议通常是以太网，因此每个到达和离开的数据块都由以太网数据包组成，数据包包含作为有效载荷的更高层协议分组的分组。高级协议通常是在以太网之上运行的Internet协议（IP）和在IP之上运行的传输控制协议（TCP）。因此，以太网有效负载由具有TCP报头和IP报头的数据块组成。对于传出数据，以太网分组在外围组件中形成，例如I/O控制器或网络接口控制器(NIC)。类似地，在传入通信时，I/O控制器剥离以太网信息，并将TCP/IP分组传送到主机CPU。

对于传出和传入流量，都涉及核心、主内存和高速缓存。在DMA方案中，当应用程序希望传输数据时，它将数据放置在主存储器中应用程序分配的缓冲区中。核心将此传输到主存储器中的系统缓冲区，并创建必要的TCP和IP报头，这些报头也缓存在系统存储器中。然后通过DMA拾取数据包，通过NIC进行传输。此活动不仅涉及主内存，还涉及高速缓存。对于传入通信，同样需要在系统和应用缓冲区之间进行类似的传输。

需要处理大量协议流量的情况下，有两个因素会降低性能。第一，核心在系统和应用程序缓冲区之间复制数据时会消耗宝贵的时钟周期。第二，因为内存速度跟不上CPU的速度，所以内核在等待内存读写时会浪费时间。在这种传统的处理协议流量的方式中，高速缓存无法起到作用，因为数据和协议头在不断变化，高速缓存必须不断更新。

为了使性能问题更清楚，同时阐述DCA作为改进性能的一种方法的优点，让我们更详细地研究下传入流量协议的处理过程。一般而言，处理步骤如下：

1. **数据包到达**：NIC接收到传入的以太网分组后，处理并剥离以太网控制信息。这其中包括进行错误检测的计算。然后，剩余的TCP/IP分组被传输到系统的DMA模块，DMA模块通常是NIC的一部分。NIC还创建带有数据包信息的数据包描述符，比如它在内存中的缓冲区位置。
2. **DMA**:DMA模块向主存储器传输数据，包括数据包描述符。它还必须使相应的高速缓存行无效（如果有的话）。
3. **NIC中断主机**：在多个数据包被传输之后，NIC向主机处理器发出一个中断。
4. **检索描述符和数据报头**：核心处理中断，调用中断处理程序，该程序读取所接收分组的描述符和报头。
5. **出现高速缓存缺失**：由于这是新的输入数据，所以响应包含新数据的系统缓冲区的缓存行是无效的。因此，核心必须停止并将数据从主存储器读入高速缓存，然后读入核心寄存器。
6. **报头被处理**：协议软件在核心上执行，分析TCP报头和IP报头的内容。这可能包括访问传输控制块（TCB），其中包含与TCP相关的上下文信息。TCB访问可能触发缓存丢失，也可能不触发缓存丢失，这需要主存储器访问。
7. **被转移的有效载荷**：数据包的数据部分从系统缓冲区被传输到适当的应用缓冲区。

传出数据通信过程和以上步骤类似，只是在高速缓存管理上有一些不同。传出数据通信过程如下步骤所示：

1. **请求分组传送**：当应用程序有一块数据要传输到远程系统时，它将数据放在应用程序缓冲区中，并用某种类型的系统调用向OS发出提醒。
2. **数据包创建**：操作系统调用TCP/IP进程来创建用于传输的TCP/IP分组。TCP/IP进程访问TCB（这可能涉及高速缓存未命中）并创建适当的报头。它还从应用程序缓冲区读取数据，然后将完成的数据包（头加上数据）放到系统缓冲区中。注意，写入系统缓存区的数据也存在于高速缓存中。TCP/IP进程还创建分组描述符，该分组描述符被放置在与DMA模块共享的存储器中。
3. **调用输出操作**：使用一个设备驱动程序来向DMA模块发出信号，通知其向NIC的输出已就绪。
4. **DMA传输**：DMA模块读取数据包描述符，然后执行从主存储器或最后一级缓存到NIC的DMA传输。注意，即使在（由DMA模块）读取的情况下，DMA传输也会使高速缓存中的对应的cache行无效。如果修改了该行，则会导致回写。内核不会执行使失效的操作。当DMA模块读取数据时，失效操作发生。
5. **NIC信号完成**：在传输完成之后，NIC向核心上的启动发送信号的驱动器发出通知信号。
6. **驱动器释放缓冲区**：一旦驱动程序收到完成通知，它就释放缓冲区空间供重用。内核还必须使包含缓冲区数据的高速缓存行无效。

可以看出，网络I/O涉及对缓存和主存储器的多次访问，以及数据在应用程序缓冲区和系统缓冲区之间的移动。由于核心和网络性能在内存访问时都超过了增益，因此主存储器的大量参与成为瓶颈。

**直接高速缓存访问策略**

已经提出了几种策略来更有效地利用网络I/O中的高速缓存，其中通用术语直接高速缓存访问应用于所有这些策略。

最简单的策略是在2006到2010年间，在多个英特尔Xeon处理器上作为原型实现的策略[KUMA07，INTE08]。这种形式的DCA只适用于传入的网络流量。一旦数据在系统存储器中可用，存储器控制器中的DCA函数就向核心发送预取提示。这使得核心能够从系统缓冲区预取数据分组，从而避免缓存丢失和内核周期的相关浪费。

虽然这种简单的DCA形式确实提供了一些改进，但是通过完全避免使用主存储器中的系统缓冲区，可以实现更多实质性的增益。对于协议处理的特定功能，请注意，核心只在系统缓冲区中访问一次数据包和分组描述符信息。对于传入的分组，核心从缓冲区读取数据，并将数据包有效负载传送到应用程序缓冲区中。它不需要再次访问系统缓冲区中的数据。类似地，对于传出数据包，一旦核心将数据放在系统缓冲区中，就不需要再次访问该数据。因此，假设I/O系统不仅具有直接访问主存储器的能力，而且具有访问高速缓存的能力，用于输入和输出操作。那么，可以使用最后一级缓存代替主存储器来缓存分组数据包以及传入和传出分组数据包的描述符。

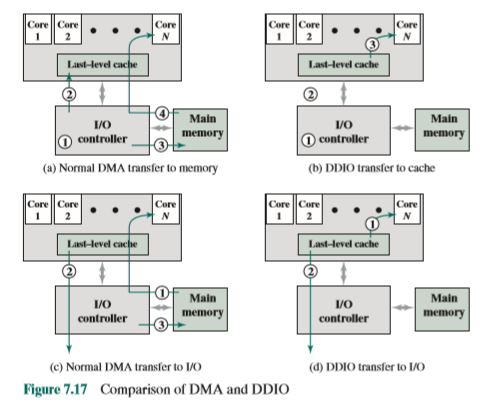
最后一种方法是真正的DCA，其在[HUGG05]中被提出。它也被称为**高速缓存注射（cache injection）**[LeON06]。这种更完整形式的DCA的版本在英特尔的Xeon处理器线路上实现，其被称为**直接数据I/O**[INTE12]。

**直接数据I/O**

Intel直接数据I/O（DDIO）是在所有Xeon E5系列处理器上实现的。通过并行比较使用和不使用DDIO的差别，可以最大程度地解释其操作过程。

**数据包输入**首先，我们来看一个数据包从网络到达NIC的情况。图7.17a显示了DMA操作所涉及的步骤。NIC启动存储器写入（1）。然后，NIC使与系统缓冲器(2)相对应的高速缓存行无效。接下来，执行DMA操作，将数据包直接存储到主存储器(3)。最后，在适当的核心接收到DMA中断信号之后，核心可以通过高速缓存(4)从存储器读取分组数据。

在讨论使用DDIO处理传入分组之前，我们需要总结一下第4章对高速缓存写策略的讨论，并介绍一种新的技术。对于下面的讨论，有一些在多处理器或多核环境中出现的高速缓存一致性的问题。这些会在第17章中讨论，我们不需要关注细节。回想一下，有两种用来处理高速缓存行更新的技术：



* **写通过**：所有写操作都对主存储器和高速缓存进行，确保主存储器总是有效的。任何其他核心高速缓存模块都可以监视到主存储器的流量，以保持其自身本地缓存的一致性。
* **写回**：仅在高速缓存中进行更新。当发生更新时，设置与该行关联的脏位。之后，当一个块被替换时，当且仅当脏位被设置时，它才能被写回主存储器。

DDIO在L3高速缓存中使用写回策略。

高速缓存执行写操作可能遇到高速缓存未命中的情况，可使用以下两种策略之一处理：

* **写入分配**：所需行从主存储器加载到高速缓存中。然后，通过写操作更新高速缓存中的行。该方案通常与写回方法一起使用。
* **非写入分配**：块直接在主存中被修改。没有对高速缓存进行更改。此方案通常与写通过方法一起使用。

基于以上考虑，我们可以描述下由NIC发起的入站传输的DDIO策略。

1. 如果有高速缓存命中，则更新高速缓存行，但不更新主内存；这只是高速缓存命中的回写策略。Intel的文献中称之为“**写更新**”。
2. 如果高速缓存未命中，则写入操作将发生在缓存中不会写入主存的行上。后续写入更新高速缓存行，同样没有引用主内存，也没有将来把此数据写入主存的操作。Intel文档[INTE12]将这称为“写分配”，遗憾的是，在一般高速缓存文献中，该术语并不具有相同的含义。

DDIO策略对于网络协议应用程序是有效的，因为输入数据不需要保留以供将来使用。协议应用程序会把数据写入应用程序缓冲区，没有必要将其临时存储在系统缓冲区中。

图7.17b显示了DDIO输入的操作。NIC启动存储器写入（1）。然后，NIC使与系统缓冲区相对应的高速缓存行无效，并将传入的数据存储在高速缓存（2）中。最后，在适当的内核接收到DCA中断信号之后，内核就可以从高速缓存（3）中读取分组数据。

**数据包输出**图7.17c显示了用于出站数据包传输的DMA操作的步骤。在内核上执行的TCP/IP协议处理程序从应用程序缓冲区读取数据并将其写入系统缓冲区。这些数据访问操作导致高速缓存未命中，使得数据需要从存储器读入L3高速缓存（1）。当NIC接收到开始传输操作的通知时，它从L3高速缓存中读取数据并发送它(2)。NIC的高速缓存访问使得数据从高速缓存中被删除并写回主存储器（3）。

图7.17d显示了数据包传输时DDIO操作所涉及的步骤。TCP/IP协议处理器创建要传输的数据包并将其存储在L3高速缓存（1）中分配的空间中，不存储在主存储器（2）中。NIC发起的读取操作由高速缓存中的数据满足，而不会导致主存储器的数据驱逐。

从以上的并行比较我们应该可以清晰地看出，无论是数据包的输入还是输出，DDIO都要比DMA要高效，因此相比较更能跟上数据包的高速传输率。

## 7.7 I/O通道和处理器

**I/O功能的演化**

随着计算机系统的发展，出现了复杂度不断增高精细度不断提升的单个设备。没有什么比I/O功能的变化更明显的了。我们已经看到了其演变的一部分。演变的过程步骤可以概述如下：

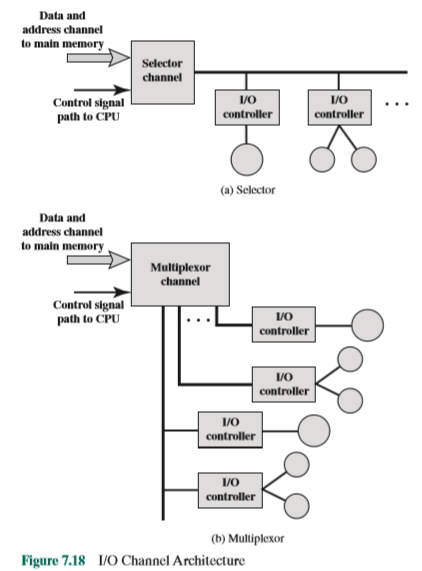
1. CPU直接控制外围设备。这已在简单的微处理器控制设备中得到了使用。
2. 增加了控制器或I/O模块。CPU使用编程I/O而不使用中断。通过这一步演变，CPU变得与外部设备接口的特定细节有些脱节。
3. 使用与步骤2相同的配置，但是现在使用中断。CPU不需要花费时间等待I/O操作被执行，从而提高了效率。
4. I/O模块通过DMA直接访问存储器。除了在传输开始和结束的时间，I/O模块可以在不涉及CPU的情况下将一个数据块移动到内存中或从内存中移动。
5. I/O模块被升级成具有专门针对I/O的指令集的处理器。CPU指示I/O处理器在内存中执行I/O程序。I/O处理器在没有CPU干预的情况下获取并执行这些指令。这使得CPU能指定一系列I/O活动，并且仅当执行完整个序列时才会被中断。
6. I/O模块具有它自己的本地存储器，并且实际上它本身就是一台计算机。采用这种体系结构，可以控制大量的I/O设备，并且只需要很少的CPU参与。这种体系结构的一个常见用途是控制与交互式终端的通信。I/O处理器负责控制终端所涉及到的大部分任务。

随着I/O功能发展的不断推进，越来越多的I/O功能在没有CPU参与的情况下执行。CPU不断从I/O相关任务中解脱出来，从而提高了性能。在最后两个步骤（5-6）中，随着能够执行程序的I/O模块的概念的引入，重大的变化发生了。对于步骤5，I/O模块通常被称为I/O通道。对于步骤6，术语I/O处理器经常被使用。然而，这两个术语有时也同时适用于这两种情况。在下文中，我们将使用术语I/O通道。

**I/O通道特性**

I/O信道是DMA概念的扩展。I/O通道具有执行I/O指令的能力，这使它能够完全控制I/O操作。在具有这种设备的计算机系统中，CPU不执行I/O指令。这些指令存储在主存储器中，以便被I/O通道本身专用的特殊用途处理器执行。因此，CPU通过指示I/O通道在内存中执行程序来启动I/O传输。程序将指定设备或多个设备、用于存储的内存区域或多个区域、优先级，以及针对某些错误情况采取的操作。I/O通道遵循这些指令并控制数据传输。

有两种常见类型的I/O通道，如图7.18所示。一个选择器通道控制多个高速设备，在任何时候，都尽力用这些设备之一传输数据。因此，I/O信道选择一个设备并影响着数据传输。每个设备或一小组设备由一个控制器或I/O模块处理，这与我们讨论的I/O模块非常相似。因此，在控制这些I/O控制器时，I/O通道代替CPU。多路复用器通道可以同时处理多个设备的I/O。对于低速设备，字节多路复用器尽可能快速地接受或传输字符到多个设备。例如，来自具有不同速率的三个单数据流A1A2A3A4 …，B1B2B3B4 …和C1C2C3C4 …的合成结果可能是A1B1C1A2C2 A3B2C3A4等等。对于高速设备，块多路复用器交织来自多个设备的数据块。



## 7.8 外部互联标准

在本节中，我们将简要概述最广泛使用的支持I/O的外部接口标准。标准中的两个，霹雳（Thunderbolt）和无限带宽（InfiniBand）将会在附录J中详细探究。

**通用串行总线（USB）**

USB被广泛用于外部连接。它是低速设备（如键盘和指针设备）的默认接口，但是也常用于高速I/O，包括打印机、磁盘驱动器和网络适配器。

USB已经经历了多代。第一个版本，USB 1.0，定义了1.5Mbps的低速数据速率和12Mbps的全速数据速率。USB 2.0提供480 Mbps的数据速率。USB 3.0则包含一种新型的有着更高速度，被称为“超高速（SuperSpeed）”的总线，与USB 2.0总线并行存在。“超高速”的信令速度是5Gbps，但是由于信令开销，可用的数据速率可以达到4Gbps。最近的规范是USB 3.1，它包括一个更快的传输模式，被叫作为“超高速+（SuperSpeed+）”。该传输模式实现了10Gbps的信令速率和9.7Gbps的理论可用数据速率。

USB系统由根主机控制器控制，根主机控制器连接到设备来创建出具有层级树拓扑的本地网络。

**火线（FireWire）串行总线**

FireWire是作为小型计算机系统接口（SCSI）的替代品开发的，用于小型系统，如个人计算机、工作站和服务器。目的是满足这些系统对高I/O速率的日益增长的需求，同时避免使用为大型机和超级计算机系统开发的笨重和昂贵的I/O通道技术。其结果是形成IEEE标准1394，用于高性能串行总线（High Performance Serial Bus，译者注），通常被称为FireWire。

FireWire使用菊花链配置，最多63个设备通过单个端口连接。此外，可以使用桥接器互连多达1022条FireWire总线，从而使系统能够根据需要支持多个外围设备。

FireWire提供了所谓的热插拔，这使得连接和断开外围设备，同时无需关闭计算机系统或重新配置系统成为可能。此外，FireWire还提供了自动配置；手动设置设备ID或关心设备的相关位置是没有必要的。对于FireWire，没有终止点，系统自动执行配置功能来分配地址。火线总线不必是严格的菊花链。事实上，树结构的配置也是可能的。

FireWire标准的一个重要特征是，它指定一组三层协议来标准化主机系统通过串行总线与外围设备交互的方法。物理层定义了在FireWire下允许的传输介质以及每个介质的电气和信令特性。定义了数据速率介于25 Mbps和3.2 Gbps之间。链路层描述了数据包中的数据传输。事务层定义了一个请求-响应协议，它向应用程序隐藏FireWire的底层细节。

**小型计算机系统接口（SCSI）**

SCSI曾经是连接外围设备（磁盘、调制解调器、打印机等）到中小型计算机的常用标准。虽然SCSI已经发展到更高的数据速率，但是在较小的系统中，它的受欢迎程度赶不上USB和FireWire。然而，高速版本的SCSI在企业系统的大容量存储支持上依然很受欢迎。例如，IBM zEnterprise EC12和其他IBM大型机仍为SCSI提供支持，另外大量Seagate的硬盘驱动器系统仍在使用SCSI。

SCSI的物理结构是共享总线，它可以支持多达16或32个设备，取决于标准的版本号。其总线可提供并行传输而不是串行传输，总线宽度在较早的版本上为16位，之后版本为32位。传输速度从最初的SCSI -1规格的5Mbps升级到SCSI -3 U3的160 Mbps。

**霹雳 （Thunderbolt）**

当前最新，也是最快的外围连接技术之一，同时可用于通用用途的接口标准被称为霹雳。它由英特尔公司与苹果公司合作开发。一根霹雳电缆可以完成以前需要多个电缆才能完成的工作。该技术将数据、视频、音频和电源合并为一个高速外围设备的连接，它可以连接例如硬盘驱动器、RAID（独立磁盘冗余阵列）阵列、视频捕获盒和网络接口等外设。其可在每个方向上提供了高达10 Gbps的吞吐量，以及多达10瓦的功率给连接的外设。

迅雷将在附录J中详细介绍。

**无限带宽（InfiniBand）**

InfiniBand是一个针对高端服务器市场的I/O规范。该规范的第一个版本发布于2001年初，同时吸引了众多供应商。例如，IBM的zEnterprise系列大型机重度依赖依赖InfiniBand多年。该标准描述了处理器和智能I/O设备之间的数据流的架构和规格。InfiniBand已经成为一种广泛流行于存储区域网络和其他大型存储配置的接口。本质上，InfiniBand使服务器、远程存储和其他网络设备能够连接到交换机和链接的中心结构中。基于交换机的体系结构可以连接多达64000台服务器、存储系统和网络设备。

InfiniBand将在附录J中详细介绍。

**PCI Express**

PCI Express是一个高速总线系统，用于连接各种类型和速度的外围设备。第3章详细讨论了PCI Express。

**SATA**

串行ATA（Serial Advanced Technology Attachment，即串行高级技术附件）是磁盘存储系统的接口。它提供高达6Gbps的数据速率，每个设备的最大数据速率为300Mbps。SATA广泛应用于台式电脑、工业和嵌入式应用中。

**以太网**

以太网是主要的有线网络技术，用于家庭、办公室、数据中心、企业和广域网。随着以太网的发展，其已经可以支持高达100Gbps的数据速率和从几米到几十公里的传输距离，以太网已经在支持个人计算机、工作站、服务器和大小组织中的海量数据存储中扮演着不可或缺的角色。

以太网最初是一个基于实验总线的3Mbps系统。利用总线系统，所有连接的设备，例如PC，都连接到公共同轴电缆，非常类似于住宅有线电视系统。第一个商业化的以太网，以及IEEE 802.3的第一个版本，是基于总线并运行在10Mbps的系统。随着技术的进步，以太网已经从基于总线转变为基于交换机，并且数据速率周期性地增加了一个数量级。在基于交换机的系统中，有一个中央交换机，所有的设备都直接连接到交换机。目前，以太网系统的可用速度高达100Gbps。以下是一个简短的发展年表。

* 1983年：10Mbps（兆位每秒，百万位每秒）
* 1995年：100 Mbps
* 1998年：1Gbps（千兆位每秒，十亿位每秒）
* 2003年：10 Gbps
* 2010年：40 Gbps和100 Gbps

**Wi-Fi（无线局域网）**

Wi-Fi是主要的无线互联网接入技术，用于家庭、办公室和公共场所。家庭中的Wi-Fi现在连接着电脑、平板电脑、智能手机以及许许多多电子设备，如摄像机、电视和恒温器。Wi-Fi已经成为企业用以提高员工生产力和网络沟通效率的重要手段。公共Wi-Fi热点已经急速扩大，大多数公共场所都可以免费提供互联网的接入。

随着天线、无线传输技术和无线协议设计的发展，IEEE 802.11委员会已经以前所未有的速度为Wi-Fi的新版本引入标准。一旦标准颁布，工业界就会迅速开发出产品。以下是一个从最初标准开始的简短的时间表，其中该标准被简单地称为IEEE 802.11，表中显示了每个版本的最大数据速率：

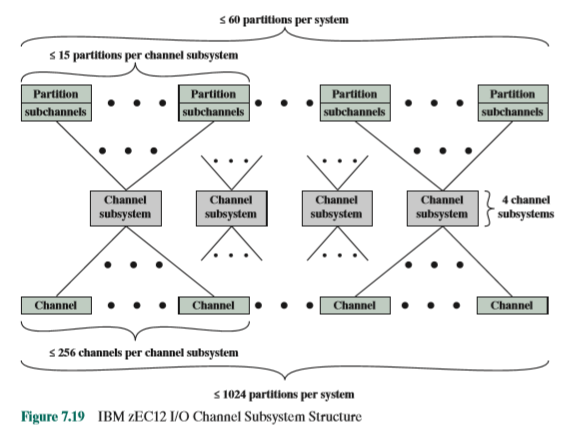
* 802.11（1997）：2Mbps（兆位每秒，百万位每秒）
* 802.11a（1999）：54 Mbps
* 802.11b（1999）：11 Mbps
* 802.11n（1999）：600 Mbps
* 802.11g（2003）：54 Mbps
* 802.11ad（2012）：6.76 Gbps（十亿比特/秒）
* 802.11ac（2014）：3.2 Gbps

## 7.9 IBM zENTERPRISE EC12 I/O结构

zEnterprise EC12是IBM最新的大型计算机产品（撰写本文时）。该系统基于zEC12处理器芯片的使用，它是一个5.5GHz的六个核心的多核芯片。zEC12架构最多可以有101个处理器芯片，总共有606个核心。在本节中，我们将介绍zEnterprise EC12的I/O结构。

**渠道结构**

zEnterprise EC12有一个专用的I/O子系统，用于管理所有I/O操作，完全解除了主服务器用于I/O处理和存储的负担。图7.21显示了I/O子系统的逻辑结构。在96个核心处理器中，



最多4个可以专用于I/O使用，可用以创建4个**通道子系统（CSS）**。每个CSS由以下元素构成：

* **系统辅助处理器（SAP）**：SAP是为I/O操作配置的核心处理器。它的作用是卸下I/O操作，管理通道和I/O操作队列。它减轻了其他处理器处理所有I/O任务的负担，使它们专注于应用程序逻辑。
* **硬件系统区域（HSA）**：HSA是包含I/O配置的系统存储器的保留部分。它被SAPS使用。固定数量的32GB被保留下来，它不是客户购买内存的一部分。通过消除有计划的或预先计划好的运行中断，可以带来更大的配置灵活性和更高的可用性。
* **逻辑分区**：逻辑分区是虚拟机的一种形式，实质上是在操作系统级别定义的逻辑处理器3。每个CSS最多可支持16个逻辑分区。

注释：3 虚拟机是操作系统的实例，同一个或多个运行应用程序一起在计算机内的隔离内存分区中运行。它使不同的操作系统能够在同一台计算机上同时运行，并且防止应用程序相互干扰。参见[ STAR12]对虚拟机的讨论。

* **子通道**：子通道在程序看来是逻辑设备，并且包含执行I/O操作所需的信息。对于CSS可寻址的每个I/O设备，都存在一个子通道。运行在分区上的通道子系统代码使用子通道向通道子系统传递I/O请求。子通道被分配给了定义在逻辑分区的每个设备。每个CSS支持高达196K的子通道。
* **通道路径**：通道路径是通道子部件和一个或多个控制单元之间经由通道通信的单个接口。命令和数据通过通道路径发送用以执行I/O请求。每个CSS可以有最多256个通道路径。
* **通道**：通道是与I/O控制单元(CU)通信的小型处理器。它们管理存储器和外部设备之间的数据传输。

这种精心设计的结构使大型机能够管理大量的I/O设备和通信链路。所有I/O处理都从应用程序和服务器处理器上移除，从而提高了性能。通道子系统处理器在配置上有些通用，这可以使它们能够管理各种I/O任务并跟上不断变化的需求。通道处理器专门为其接口的I/O控制单元编程。

**I/O系统组织**

为了解释I/O系统组织，我们需要首先简要解释zEnterprise EC12的物理布局。图7.20是水冷式机器的前视图（还有风冷式）。该系统具有以下特点：

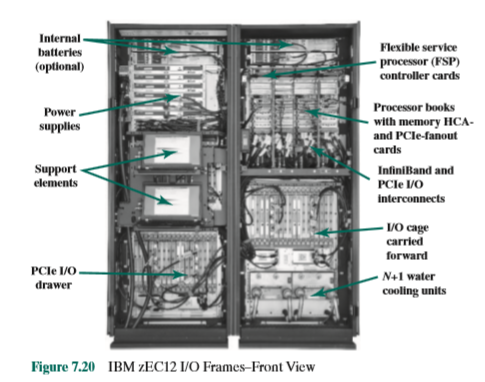
* 重量：2430公斤（5358磅）
* 宽度：1.568米（5.14英尺）
* 深度：1.69米（6.13英尺）
* 高度：2.015米（6.6英尺）

不是一台笔记本电脑。

该系统由两个称为框架的大空间组成，它们容纳了zEnterprise EC12的各种组件。右边的A框架包括两个大的盒子，外加用于布线和其他部件的空间。上边的盒子是一个处理器盒子，它有四个插槽可容纳多达四个完全互连的处理器库。每个库都包含一个多芯片模块（MCM）、存储卡和I/O盒子的连接。每个MCM都是一个板子，其中安装了六个多核芯片和两个存储控制芯片。

A框架中的下方的盒子是I/O盒子，它包含了I/O硬件，包括多路复用器和信道。I/O盒子是IBM根据工厂中客户的规格要求安装的固定单元。

左边的Z框架包含内部电池和电源，以及一个或多个支持元素的空间，系统管理员使用这些支持元素进行平台管理。Z框架还包括为两个或更多个I/O抽屉准备的插槽。



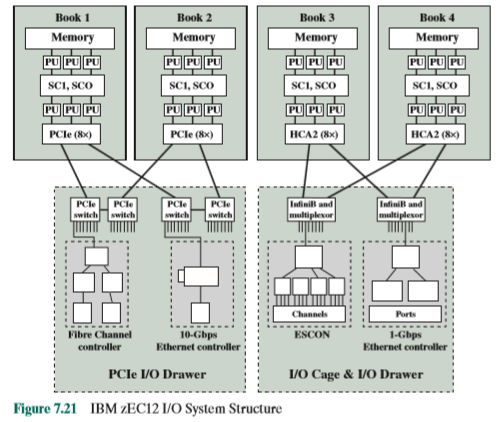
I/O抽屉包含了与I/O盒子类似的部件。不同之处在于，I/O抽屉比较小，并且易于在客户站点进行进出交换，以满足不断变化的需求。

在此背景下，我们现在介绍zEnterprise EC12 I/O系统结构的一种典型配置（图7.21）。每个zEC12处理器库支持两个内部（即A和Z帧的内部）I/O基础设施：用于I/O盒子和I/O抽屉的InfiniBand，以及用于I/O抽屉的PCI Express（PCIe）。这些信道控制器被称为**扇出**（fanouts）。

从处理器库到I/O盒子和I/O抽屉的InfiniBand连接是通过主机通道适配器（HCA）扇出的，它具有到I/O盒子或抽屉的InfiniBand多路复用器的InfiniBand链接。InfiniBand多路复用器被用于互连服务器、通信基础设施设备、存储和嵌入式系统。除了使用InfiniBand来互连所有使用InfiniBand的系统之外，InfiniBand多路复用器还支持其他I/O技术。ESCON（企业系统连接）支持使用专用的基于光纤的技术连接到磁盘、磁带和打印机设备。以太网连接可提供到支持这种流行的局域网技术的各种设备的1-Gbps和10-Gbps的连接。以太网的一个值得注意的用途是构建大型服务器群，特别是将刀片服务器彼此互连以及与其他大型机互连4。

注释：

4刀片服务器是一种服务器体系结构，它在单个机箱中容纳多个服务器模块（刀片）。它广泛用于数据中心，以节省空间和改善系统管理。无论是自立式还是架式安装，主板都提供电源，每个刀片都有自己的CPU、内存和硬盘。



从处理器库到I/O抽屉的PCIe连接是通过PCIe扇出到PCIe交换机的。PCIe开关可以连接到许多I/O设备控制器。zEnterprise EC12的典型示例是1-Gbps和10-Gbps以太网和光纤信道。

每个库包含最多8个InfiniBand HCA和PCIe扇区的组合。每个扇出支持多达32个连接，对于每个处理器库总共最多256个连接，每个连接都由通道处理器控制。

## 7.10关键术语、复习习题与难题

**关键术语**

|  |  |  |
| --- | --- | --- |
| 高速缓存注射 | I/O命令 | 外围设备 |
| 周期窃取 | I/O模块 | 程序I/O |
| 直接高速缓存访问（DCA） | I/O处理器 | 选择器通道 |
| 直接数据I/O | 隔离I/O | 串行I/O |
| 直接存储器存取（DMA） | 末级高速缓存 | 霹雳 |
| 无限带宽 | 内存映射I/O | 写分配 |
| 中断 | 多路复用通道 | 写回 |
| 中断驱动I/O | 非写入分配 | 写通过 |
| I/O通道 | 并行I/O | 写更新 |
|  |  |  |

**复习题**

7.1 列出外部或外围设备的三个主要分类。

7.2 为什么外围设备没有直接连接到系统总线？

7.3 I/O模块的主要功能是什么？

7.4 列出并简要讨论处理器通信过程中涉及的任务。

7.5 内存映射I/O和隔离I/O有什么区别？

7.6 当设备中断发生时，处理器如何确定哪个设备发出中断？

7.7 当DMA模块控制总线，并且保持总线控制时，处理器做什么？

**问题**

7.1 在典型的微处理器上，不同的I/O地址用于指代给定设备的I/O控制器中的I/O数据寄存器和用于控制和状态寄存器的不同的地址。这种寄存器称为**端口**。在英特尔8088中，使用两个I/O指令格式。其中一种格式，使用8位操作码指定I/O操作，后面跟着8位端口地址。其他I/O操作码表示端口地址在16位DX寄存器中。8088在每个I/O寻址模式下能寻址多少端口？

7.2 在Zilog Z8000微处理器系列中使用了类似的指令格式。在这种情况下，存在直接端口寻址能力，其中16位端口地址是指令的一部分。同时存在间接端口寻址能力，其中指令引用包含端口地址的16位通用寄存器中的一个。Z8000在每个I/O寻址模式下能寻址多少端口？

7.3 Z8000还包括块I/O传输能力，与DMA不同，它受处理器的直接控制。块传输指令指定端口地址寄存器（Rp）、计数寄存器（Rc）和目标寄存器（Rd）。Rd包含存储从输入端口读取的第一个字节的主存储器地址。Rc是16位通用寄存器中的任何一个。数据块的传输量有多大？

7.4 考虑一个微处理器，它具有块I/O传输指令，比如在Z8000上找到的指令。在第一次执行之后，这种指令需要两个时钟周期才能重新执行。然而，如果我们使用非阻塞I/O指令，则获取和执行总共需要18个时钟周期。当传输x字节的块时，计算使用块I/O指令带来的速度的增加。

7.5 有一个系统基于8位微处理器，其具有两个I/O设备。该系统的I/O控制器使用独立的控制和状态寄存器。这两种设备均按照一次处理一字节的规则来处理数据。第一个设备具有两条状态线和三条控制线。第二设备有三条状态线和四条控制线。

A. 对于每个设备的状态读取和控制，我们需要多少个8位I/O控制模块寄存器？

B. 如果第一个设备是只能输出的设备，那么需要的控制模块寄存器的总数量是多少？

C. 控制这两个设备需要多少不同的地址？

7.6 对于编程的I/O，图7.5表明处理器被卡在等待循环中，执行I/O设备的状态检查。为了提高效率，可以编写I/O软件，以便处理器定期检查设备的状态。如果设备尚未准备好，处理器可以跳转到其他任务。经过一段时间间隔后，处理器再次返回检查状态。

* 1. 考虑以上将数据一次输出一个字符到每秒10字符（cps）的打印机的方案。如果每200毫秒扫描一次状态会发生什么？
  2. 接下来考虑一个带有单个字符缓冲器的键盘。平均而言，字符以10 cps的速率输入。然而，两个连续按键之间的时间间隔可以短至60毫秒。I/O程序应该以什么频率扫描键盘？

7.7 微处理器每隔15ms扫描一个输出I/O设备的状态。这通过定时器每隔15ms向处理器发出警报来实现。该设备的接口包括两个端口：一个用于状态，一个用于数据输出。给定20MHz和32MHz的时钟速率，扫描和服务设备需要多长时间？为了简单起见，假设所有相关的指令周期都占用10个时钟周期。

7.8 图7.4描述了从外围设备读取数据块的三种技术。

比较图7.4(a)和7.4(b)，并讨论这两种技术的缺点。

7.9 特定的系统由操作员通过从键盘输入的命令来控制。在8小时间隔内输入的命令的平均数量是60。

a. 假设处理器每100ms扫描一次键盘，那么在8小时内要检查多少次？

b. 如果使用中断驱动的I/O，处理器访问键盘的数量会减少多少？

7.10 假设图7.9所示的8255A配置如下：端口A作为输入，端口B作为输出，端口C的所有位作为输出。给出控制寄存器的位来定义此配置。

7.11 考虑一个系统，它采用中断驱动的I/O，用于连续传输平均8KB/s数据的特定设备。

1. 假设中断处理花费大约100微秒（即，跳到中断服务例程（ISR）、执行它和返回主程序的时间）。如果该I/O设备为每个字节中断，则确定该I/O设备所消耗的处理器时间的比例。
2. 现在假设设备有两个16字节的缓冲区，并在其中一个缓冲区已满时中断进程。当然，中断处理需要更长的时间，因为ISR必须传输16个字节。在执行ISR时，处理器需要大约8微秒来传输每个字节。确定在这种情况下这个I/O设备消耗处理器时间的比例。
3. 现在假设处理器配备有块传输I/O指令，比如在Z8000上找到的指令。这使得关联的ISR将块中的每个字节转移时间减少到仅2μs。在这种情况下，确定使用这个I/O设备消耗的的处理器时间的比例。

7.12 在几乎所有包括DMA模块的系统中，DMA到主存储器的优先级总是高于CPU访问主存储器的优先级。为什么？

7.13 DMA模块通过周期性窃取从以9600bps的速度传输的设备向存储器传送字符。处理器以每秒100万指令（1 MIPS）的速率获取指令。由于DMA活动，处理器会减慢多少？

7.14 考虑一个系统，其中总线周期需要500纳秒。无论从处理器到I/O设备还是相反，总线控制的传输都需要250 ns。其中一个I/O设备具有50 kb/s的数据传输速率，并且采用DMA。数据一次传送1字节。

1. 假设我们在突发模式中使用DMA。也就是说，DMA接口在块传输开始之前获得总线主控权，并保持总线的控制，直到整个块被转移。当传输一个128字节的块时，设备会占用总线多长时间？
2. 如果是周期窃取模式，计算结果如何？

7.15 对8237A的时序图的研究表明一旦块传输开始，每个DMA周期需要三个总线时钟周期。在DMA循环期间，8237A在内存和I/O设备之间传输一个字节的信息：

1. 假设我们以8兆赫的速率来计时8237A。传输一个字节需要多长时间？
2. 最大可达到的数据传输速率是多少？
3. 假设内存不够快，我们必须在每个DMA周期插入四个等待状态。实际的数据传输速率是多少？

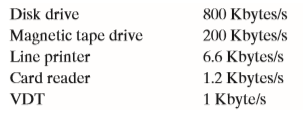
7.16 假设在前面问题的系统中，一个内存周期需要450ns。在不影响可达到的数据传输速率的情况下，我们能够将总线的时钟速率降低到什么值呢？

7.17 DMA控制器服务于四个仅接收电信链路（每个DMA信道一个），每个通信链路具有64Kbps的速度。

a. 你会采用突发模式还是周期窃取模式来操作控制器？

b. 对于DMA信道的服务，您会采用什么优先级方案？

7.18 32位计算机有两个选择器通道和一个多路复用器通道。每个选择器通道支持两个磁盘和两个磁带单元。多路复用器通道有两个线路打印机、两个读卡器和10个VDT端子连接到它。假设以下传输速率：



估算系统中最大总I/O传输速率。

7.19 计算机由处理器和I/O设备D组成，I/O设备D通过共享总线连接到主存储器M，数据总线宽度为一个字。处理器每秒最多可执行106条指令。平均每个指令执行需要五个机器周期，其中三个使用内存总线。存储器读写操作使用一个机器周期。假设处理器正在持续执行“后台”程序，这些程序需要95%的指令执行率，但不需要任何I/O指令。假设一个处理器周期等于一个总线周期。现在假设I/O设备用于在M和D之间传输非常大的数据块。

1. 如果使用编程的I/O，并且每个单字I/O传输要求处理器执行两个指令，那么通过D，估计可能的最大I/O数据传输速率，即每秒的字数。
2. 如果使用DMA，再次估算此速率。

7.20 数据源产生7位IRA字符，每个IRA字符附加一个奇偶校验位。导出在R-bps线上的最大有效数据速率(IRA数据比特率)的表达式，用于以下情况：

1. 异步传输，带上1.5位单位的停止位；
2. 位同步传输，帧由48个控制位和128个信息位组成；
3. 与（b）相同，带一个1024位的信息字段；
4. 字符同步，每帧9个控制字符，16个信息字符；
5. 与（d）相同，带一个128个的信息字符。

7.21 两个女人在一个高栅栏的两边。其中一个女人叫苹果服务员，在她的篱笆旁边有一棵漂亮的苹果树，树上长满了美味的苹果，她很乐意随时给另一个女人提供苹果。另一个女人叫苹果食客的人，她喜欢吃苹果，但是没有苹果。事实上，她必须以固定的速度吃苹果（每天一苹果，医生远离我）。如果她吃得比固定的速度快，她会生病的。如果她吃得慢一点，她就会营养不良。两个女人都不能说话，所以问题的关键是以正确的速度把苹果从苹果服务员那里送给苹果食客。

1. 假设在篱笆顶上有一个闹钟，并且这个闹钟可以有多个闹钟设置。时钟如何被用来解决这个问题？绘制一个时序图来说明解决方案。
2. 现在假设没有闹钟。取而代之的是，苹果食客只要需要苹果，就可以挥动旗子。提出一个新的解决方案。苹果服务员也有一个标志，会对你有帮助吗？如果是的话，把它加入到解决方案中。讨论这种方法的缺点。
3. 现在去掉这个旗子，假设有一条长长的绳子。提出一种优于（b）使用绳子的解决方案。

7.22 假设一个16位微处理器和两个8位微处理器要连接到系统总线。给出以下细节：

1. 所有微处理器都具有传输任何类型的数据所必需的硬件特性：编程I/O、中断驱动I/O和DMA。

2. 所有微处理器都有一个16位地址总线。

3. 两个存储板，每个都有64KB的容量，与总线连接。设计者希望使用尽可能大的共享内存。

4. 系统总线最多支持四个中断线和一个DMA线。作出必要的其他假设，并解决：

1. 根据线路的数量和类型给出系统总线规范。
2. 描述用于在总线上通信的可能协议（即，读-写、中断和DMA序列）。
3. 解释上述设备如何与系统总线连接。

終わり！！