我们使用微软的 Visual C++[①] 来作成应用。

在 AT 兼容机中，蜂鸣器的默认端口号是 61H（末尾的 H，表示的是十六进制数（Hexadecimal）的意思）。用 IN 指令通过该端口号输入数据，并将数据的低 2 位设定为 ON，然后再通过该端口号用 OUT 指令输出数据，这时蜂鸣器就会响起来。采用同样的操作方法，将数据的低 2 位设定为 OFF 并输出后，蜂鸣器就停止了。

位设定为 ON 指的是将该位设定为 1，位设定为 OFF 指的是将该位设定为 0。把位设定为 ON，只需把想要设定为 ON 的位设定为 1，其他位设定为 0 后进行 OR 运算即可。由于这里需要把低 2 位置为 1，因此就是和 03H 进行 OR 运算。03H 用 8 位二进制数来表示的话是 00000011。由于即便高 6 位存在着具体意义，和 0 进行 OR 运算后也不会发生变化，因而就和 03H 进行 OR 运算。把位设定为 OFF，只需把想要置 OFF 的位设定为 0，其他位设定为 1 后进行 AND 运算即可。由于这里需要把低 2 位设定为 0，因此就要和 FCH 进行 AND 运算。在源代码中，FCH 是用 0FCH 来记述的。在前面加 0 是汇编语言的规定，表示的是以 A～F 这些字符开头的十六进制数是数值的意思。0FCH 用 8 位二进制数来表示的话是 11111100。由于即便高 6 位存在着具体意义，和 1 进行 AND 运算后也不会产生变化，因而就是同 0FCH 进行 OR 运算（代码清单 11-2）。

代码清单 11-2　利用 IN/OUT 指令来控制蜂鸣器的程序示例

```
void main ( ) {
    // 计数器
    int i;

    // 蜂鸣器发声
```

① 可以免费下载的 Borland C++ 5.5，不支持加入了汇编语言的源代码的编译。使用该版本时需要购买特定的汇编器。因此这里我们用的是 Visual C++ 6.0。

```
    _asm {
        IN  EAX, 61H
        OR  EAX, 03H                                    (1)
        OUT 61H, EAX
    }

    // 等待一段时间
    for (i = 0; i < 1000000 ; i++) ;                    (2)

    // 蜂鸣器停止发声
    _asm {
        IN  EAX, 61H
        AND EAX, 0FCH                                   (3)
        OUT 61H, EAX
    }
}
```

接下来就让我们对代码清单 11-2 进行详细说明。main 是位于 C 语言程序运行起始位置的函数。在该函数中，有两个用 _asm{ 和 } 围起来的部分，它们中间有一个使用 for 语法的空循环（不做任何处理的循环）。

（1）部分是控制蜂鸣器发音的部分。首先，通过 IN EAX,61H（助记符不区分大小写）指令，把端口 61H 的数据存储到 CPU 的 EAX 寄存器中。接下来，通过 OR EAX,03H 指令，把 EAX 寄存器的低 2 位设定成 ON。最后，通过 OUT 61H,EAX 指令，把 EAX 寄存器的内容输出到 61 号端口，使蜂鸣器开始发音。虽然 EAX 寄存器的长度是 32 位，不过由于蜂鸣器端口是 8 位，所以只需对下 8 位进行 OR 运算和 AND 运算就可以正常运行了。

（2）部分是一个重复 100 万次的空循环，主要是为了在蜂鸣器开始发音和停止发音之间稍微加上一些时间间隔。因为现在计算机的 CPU 运行速度非常快，哪怕是 100 万次的循环，也几乎是瞬间完成的。

（3）部分是用来控制蜂鸣器发音停止的部分。首先，通过 IN EAX,61H 指令，把端口 61H 的数据存储到 CPU 的 EAX 寄存器中。接

下来，通过 AND EAX,0FCH 指令，把 EAX 寄存器的低 2 位设定成 OFF。最后，通过 OUT 61H,EAX 指令，把寄存器 EAX 的内容输出到 61 号端口，使蜂鸣器停止发音。大家可以把 61H 端口的低 2 位认为是蜂鸣器的开关。

最后，让我们对代码清单 11-2 进行编译，并尝试运行一下。这时，蜂鸣器应该会发出“嘀!”的短促声音。此外，有一点需要注意的是，该程序虽然在旧版本 Windows（95、98）中可以正常运行，但在这以后的 Windows（XP、Vista 等）版本中是无法正常运行的。这是因为，为了保护系统安全，现在的 Windows 禁止了应用直接控制硬件的方式。如果将该程序在最近的 Windows 版本上运行的话，就会出现如图 11-5 所示的错误信息，而且蜂鸣器也不会发出声音。

图 11-5　由于现在的 Windows 禁止应用直接控制硬件，因而出现了错误信息

11.4　外围设备的中断请求

让我们再来看一下图 11-4。在“I/O 范围”下面有一个“IRQ”项目，对应的值是 0x00000006（06）。IRQ（Interrupt Request）是**中断请求**的意思。那么，IRQ 主要是用来做什么的呢?

IRQ 是用来暂停当前正在运行的程序，并跳转到其他程序运行的

必要机制。该机制称为**中断处理**。中断处理在硬件控制中担当着重要角色。因为如果没有中断处理，就有可能出现处理无法顺畅进行的情况。

从中断处理开始到请求中断的程序（中断处理程序）运行结束之前，被中断的程序（主程序）的处理是停止的。这种情况就类似于在处理文档的过程中有电话打进来，电话就相当于中断处理。假如没有中断功能的话，就必须等到文档处理完毕才可以接听电话。这样就太不方便了。由此可见，中断处理有着很大的价值。就像接听完电话后返回到原来的文档作业一样，中断处理程序运行结束后，处理也会返回到主程序中继续（图 11-6）。

图 11-6　中断处理就类似于在处理文档时接电话

实施中断请求的是连接外围设备的 I/O 控制器，负责实施中断处理程序的是 CPU。为了进行区分，外围设备的中断请求会使用不同于 I/O 端口的其他编号，该编号称为**中断编号**。在控制面板中查看软盘驱动器的属性时，IRQ 处显示的数值 06，表示的就是用 06 号来识别软盘驱

动器发出的中断请求。另一方面，操作系统及 BIOS[①] 则会提供响应中断编号的中断处理程序。

假如同时有多个外围设备进行中断请求的话，CPU 也会为难。为此，我们可以在 I/O 控制器和 CPU 中间加入名为**中断控制器**的 IC 来进行缓冲。中断控制器会把从多个外围设备发出的中断请求有序地传递给 CPU。大家对中断控制器的认识可能比较薄弱，不过只需了解该设备的存在和角色就可以了（图 11-7）。

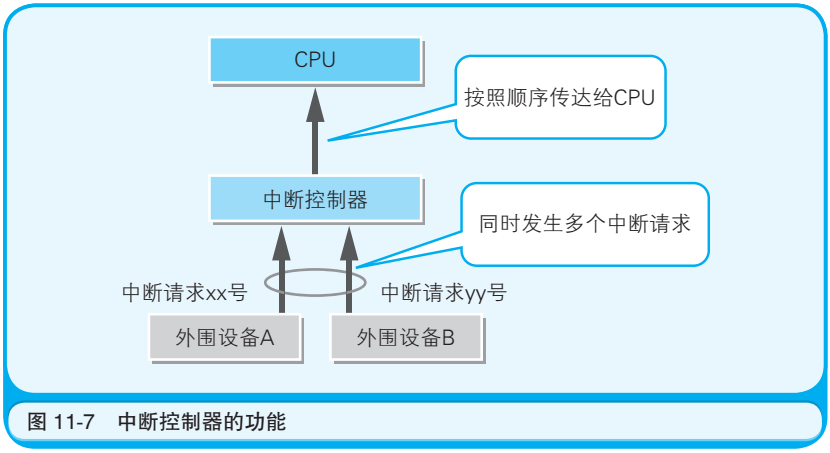

图 11-7　中断控制器的功能

CPU 接收到来自中断控制器的中断请求后，会把当前正在运行的主程序中断，并切换到中断处理程序。中断处理程序的第一步处理，就是把 CPU 所有寄存器的数值保存到内存的栈中。在中断处理程序中完成外围设备的输入输出后，把栈中保存的数值还原到 CPU 寄存器中，然后再继续进行对主程序的处理。假如 CPU 寄存器的数值没有还

① BIOS（Basic Input Outpu System）位于计算机主板或扩张板卡上内置的 ROM 中，里面记录了用来控制外围设备的程序和数据。这一点在第 7 章中进行过说明。

原的话，就会影响到主程序的运行，甚至还有可能会使程序意外停止或者发生运行异常。这是因为主程序在运行过程中，出于某些原因用到 CPU 寄存器。而这时如果突然插入别的程序，主程序必然会受到影响。因此，在中断请求完毕后，各寄存器的数值必须要还原到中断前的状态。只要寄存器的值保持不变，主程序就可以像没有发生任何事情一样继续处理（图 11-8）。

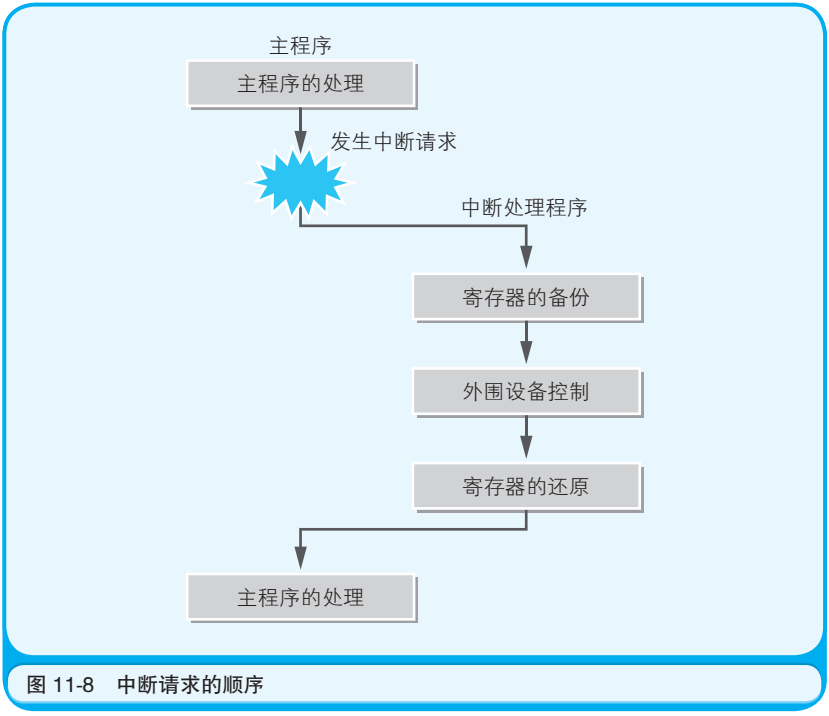

图 11-8　中断请求的顺序

11.5　用中断来实现实时处理

在主程序运行的过程中，中断发生的频率有多大呢？实际上，大部分的外围设备，都会频繁地发出中断请求。其原因就是为了实时处

理从外围设备输入的数据。虽然不利用中断也可以从外围设备输入数据。但那种情况下，主程序就必须要持续不断地检测外围设备是否有数据输入。

由于外围设备有很多个，因此就有必要按照顺序来调查。按照顺序调查多个外围设备的状态称为**轮询**。对几乎不产生中断的系统来说，轮询是比较合适的处理。不过，对计算机来说就不适合了。举例来说，假如主程序正在调查是否有鼠标输入，这时如果发生了键盘输入的话，该如何处理呢？结果势必会导致键盘输入的文字无法实时地显示在显示器上。而通过使用中断，就可以实现实时显示了。

打印机等输出用的外围设备中，外围设备接收数据的状态，有时是需要用中断来通知的。由于外围设备的处理速度比计算机主机的处理速度要慢很多，因此，这种情况下就不需要对打印机的状态进行多次调查，只需在中断请求发生时输出数据即可，这样一来，其他时间 CPU 就可以集中处理别的程序了。中断处理是不是很方便呢。

11.6　DMA 可以实现短时间内传送大量数据

在了解 I/O 输入输出及中断处理的同时，还希望大家记住另外一个机制，这就是 DMA（Direct Memory Access）。DMA 是指在不通过 CPU 的情况下，外围设备直接和主内存进行数据传送。磁盘等都用到了这个 DMA 机制。通过利用 DMA，大量数据就可以在短时间内转送到主内存。之所以这么快速，是因为 CPU 作为中介的时间被节省了（图 11-9）。

图 11-10 和在前面看到的软盘控制器的属性画面是相同的。在资源[①]

① 资源是计算机具备的有限资源的统称。端口号、IRQ、DMA 等可以指定的数值范围都是有限的，因此它们也是资源的一种。

标签中有 DMA 设定，可以看出此处该设定为 02。02 这个编号称为 **DMA 通道**。CPU 借助 DMA 通道，来识别是哪一个外围设备使用了 DMA。

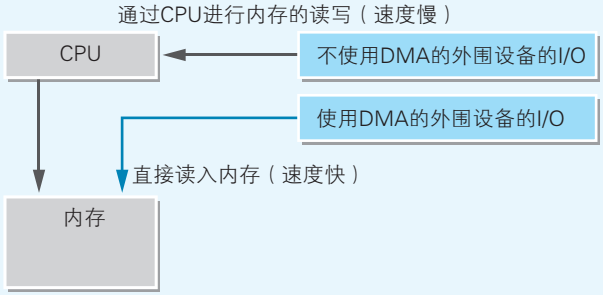

图 11-9　使用 DMA 的外围设备和不使用 DMA 的外围设备的不同

图 11-10　软盘控制器的 DMA 通道

I/O 端口号、IRQ、DMA 通道可以说是识别外围设备的 3 点组合。不过，IRQ 和 DMA 通道并不是所有的外围设备都必须具备的。计算机主机通过软件控制硬件时所需要的信息的最低限，是外围设备的 I/O 端口号。IRQ 只对需要中断处理的外围设备来说是必需的，DMA 通道则只对需要 DMA 机制的外围设备来说是必需的。假如多个外围设备都设定成同样的端口号、IRQ 及 DMA 通道的话，计算机就无法正常工作了。这种情况下，就会出现“设备冲突”的提示。

11.7　文字及图片的显示机制

在本章的最后，让我们一起来看一下显示器显示文字及图形的机制。如果用一句话来简单地概括该机制，那就是显示器中显示的信息一直存储在某内存中。该内存称为 VRAM（Video RAM）。在程序中，只要往 VRAM 中写入数据，该数据就会在显示器中显示出来。实现该功能的程序，是由操作系统或 BIOS 提供，并借助中断来进行处理的。

在 MS-DOS 时代，对大部分计算机来说，VRAM 都是主内存的一部分。例如 PC-9801 这种机型的计算机，主内存地址 A0000 地址以后是 VRAM 区域。如果用程序往 VRAM 内存地址中写入数据，文字及图形就可以显示出来。不过，文字和图形的颜色最多只能有 16 种。这是因为 VRAM 的内存空间太小了（图 11-11(a)）。

在现在的计算机中，**显卡**等专用硬件中一般都配置有与主内存相独立的 VRAM 和 GPU（Graphics Processing Unit，图形处理器，也称为图形芯片）。这是因为，对经常需要描绘图形的 Windows 来说，数百兆的 VRAM 是必需的。而为了提升图形的描绘速度，有时还需要专用的图形处理器（图 11-11(b)）。但不管怎样，内存 VRAM 中存储的数据就是显示器上显示的信息，这一机制是不变的。

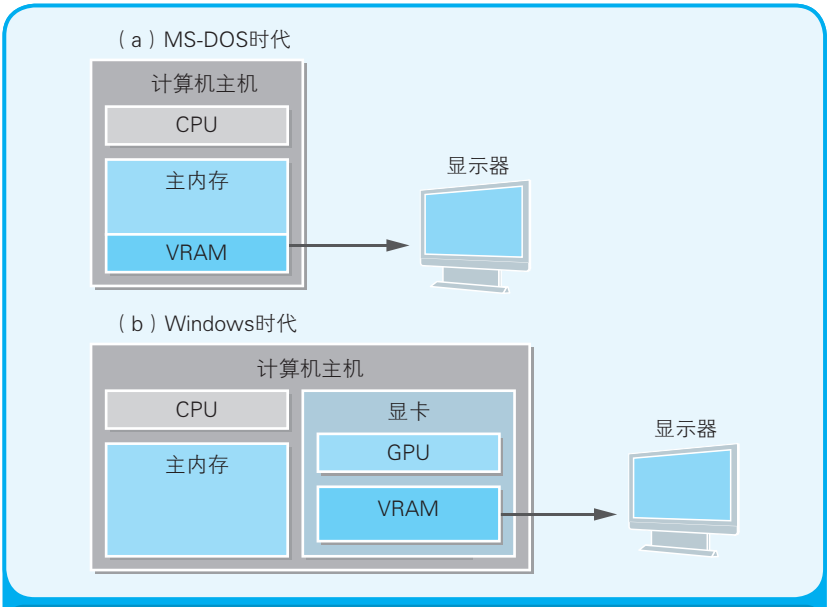

图 11-11 VRAM 中写入的数据被显示在显示器上

用软件来控制硬件听起来好像很难，但实际上只是利用输入输出指令同外围设备进行输入输出的处理而已。中断处理是根据需要来使用的选项功能，DMA 则直接交给对应的外围设备即可。由此可见，对程序员来说，其实并不困难。

虽然计算机领域的新技术在不断涌现，但计算机能处理的事情，始终只是对输入的数据进行运算，并把结果输出，这一点是不会发生任何变化的。不管程序内容是什么，最终都是数据的输入输出和运算。本章介绍的开启和停止蜂鸣器的程序，就是一个很好的例子。 而无论是计算机还是程序，其实都很简单。

下一章，我们会通过开发一个简单的游戏程序，来对计算机的“思考”机制进行说明。