是“把 *counter* 的数值读入 eax 寄存器”“将 eax 寄存器的数值变成原来的 2 倍”“把 eax 寄存器的数值写入 *counter*”这 3 个处理。

代码清单 10-14　将全局变量的值翻倍这一部分转换成汇编语言源代码的结果

```
mov eax,dword ptr[_counter]   ; 将 counter 的值读入 eax 寄存器
add eax,eax                   ; 将 eax 寄存器的值扩大至原来的 2 倍
mov dword ptr[_counter],eax   ; 将 eax 寄存器的数值存入 counter 中
```

在多线程处理中，用汇编语言记述的代码每运行 1 行，处理都有可能切换到其他线程（函数）中。因而，假设 MyFunc1 函数在读出 *counter* 的数值 100 后，还未来得及将它的 2 倍值 200 写入 *counter* 时，正巧 MyFunc2 函数读出了 *counter* 的数值 100，那么结果就会导致 *counter* 的数值变成了 200（图 10-8）。

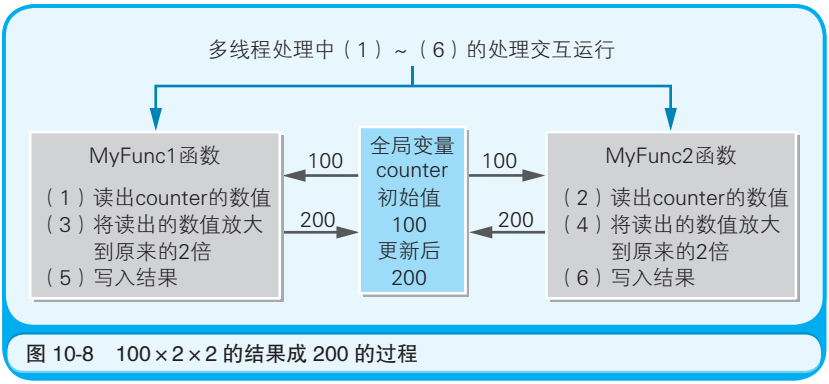

图 10-8　100×2×2 的结果成 200 的过程

为了避免该 bug，我们可以采用以函数或 C 语言源代码的行为单位来禁止线程切换的**锁定**方法。通过锁定，在特定范围内的处理完成之前，处理不会被切换到其他函数中。至于为什么要锁定 MyFunc1 函数和 MyFunc2 函数，大家如果不了解汇编语言源代码的话想必是不明白的吧。

现在基本上没有人用汇编语言来编写程序了。因为 C 语言等高级编程语言用 1 行就可以完成的处理，使用汇编语言的话有时就需要多行，效率很低。不过，汇编语言的经验还是很重要的。因为借助汇编语言，我们可以更好地了解计算机的机制。特别是对专业程序员来说，至少要有一次使用汇编语言的经验

下面让我们以开车为例进行说明。没有汇编语言经验的程序员，就相当于只知道汽车的驾驶方法而不了解汽车结构的驾驶员。对这样的驾驶员来说，如果汽车出现了故障或奇怪的现象，他们就无法自己找到原因。不了解汽车结构的话，开车的时候还可能会浪费油。这样的话，作为职业驾驶员是不合格的。与此相对，有汇编语言经验的程序员，也就相当于了解计算机和程序机制的驾驶员，他们不仅能自己解决问题，还能在驾驶过程中省油。

本章的内容确实有些绕，但是对了解计算机和程序的实际运行方式来说，体验汇编语言是最有效的。如果大家会使用 C 语言的话，希望大家对 C 语言的各种语法所对应的汇编语言都一一确认一下。最好能编写一些简短的程序来进行反复的测试。笔者自身也是通过进行这些尝试才使自己的编程技能有了大幅提高的。

下一章，我们将会对 I/O 端口的输入输出及中断处理等用程序来控制硬件的方法进行说明，同时也会介绍一个使用汇编语言的示例程序。

第 11 章 硬件控制方法

热身问答

阅读正文前，让我们先回答下面的问题来热热身吧。

1. 在汇编语言中，是用什么指令来同外围设备进行输入输出操作的?
2. I/O 是什么的缩写?
3. 用来识别外围设备的编号称为什么?
4. IRQ 是什么的缩写?
5. DMA 是什么的缩写?
6. 用来识别具有 DMA 功能的外围设备的编号称为什么?

怎么样？是不是发现有一些问题无法简单地解释清楚呢？下面是笔者的答案和解析，供大家参考。

答案

1. IN 指令和 OUT 指令
2. Input/Output
3. I/O 地址或 I/O 端口号
4. Interrupt Request
5. Direct Memory Access
6. DMA 通道

解析

1. 在 x86 系列 CPU 用的汇编语言中，通过 IN 指令来实现 I/O 输入，OUT 指令来实现 I/O 输出。
2. 用来实现计算机主机和外围设备输入输出交互的 IC 称为 I/O 控制器或简称为 I/O。
3. 所有连接到计算机的外围设备都会分配一个 I/O 地址编号。
4. IRQ 指的是用来执行硬件中断请求的编号。
5. DMA 指的是，不经过 CPU 中介处理，外围设备直接同计算机的主内存进行数据传输。
6. 像磁盘这样用来处理大量数据的外围设备都具有 DMA 功能。

本章重点

“计算机如果没有软件，就仅仅是个箱子”这个诙谐的描述大家都知道吧？也就是说，即使是计算机这种看起来很了不起的设备（硬件），离开了软件依然什么也做不了。虽然这句话极具讽刺意味，不过也正戳到了计算机的本质。因为软件的存在是硬件正常运行的必要条件。通过前面的章节我们已经知道，控制 CPU，只需把编译器或汇编器生成的本地代码加载到主内存并运行就可以了。那么，如何用程序来控制 CPU 和主内存以外的硬件呢？本章我们就会对这个问题进行解答。

11.1 应用和硬件无关?

在用 C 语言等高级编程语言开发的 Windows 应用中，大家很少能接触到直接控制硬件的指令。这是因为硬件的控制是由 Windows 全权负责的。

不过，Windows 提供了通过应用来间接控制硬件的方法。利用操作系统提供的**系统调用**功能就可以实现对硬件的控制。在 Windows 中，系统调用称为 API（图 11-1）。各 API 就是应用调用的函数。这些函数的实体被存储在 DLL 文件中。

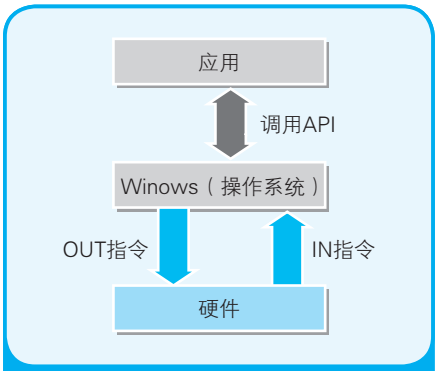

图 11-1 应用通过操作系统间接控制硬件

下面让我们来看一个利用系统调用来间接控制硬件的示例。例如，假设要在窗口中显示字符串，

就可以使用 Windows API 中的 TextOut 函数[①]。TextOut 的语法如代码清单 11-1 所示。在这段代码中，确实没有能让大家意识到硬件的参数。带有“设备描述表的句柄”这一注释的参数 hdc，是用来指定字符串及图形等绘制对象的识别值，表示的也不是直接硬件设备。

代码清单 11-1　TextOut 函数的语法（C 语言）

```
BOOL TextOut(
  HDC hdc,              // 设备描述表的句柄
  int nXStart,          // 显示字符串的 x 坐标
  int nYStart,          // 显示字符串的 y 坐标
  LPCTSTR lpString,     // 指向字符串的指针
  int cbString          // 字符串的文字数
);
```

那么，在处理 TextOut 函数的内容时，Windows 做了什么呢？从结果来看，Windows 直接控制了作为硬件的显示器。但 Windows 本身也是软件，由此可见，Windows 应该向 CPU 传递了某些指令，从而通过软件控制了硬件。

11.2　支撑硬件输入输出的 IN 指令和 OUT 指令

Window 控制硬件时借助的是输入输出指令。其中具有代表性的两个输入输出指令就是 IN 和 OUT。这些指令也是汇编语言的助记符。

IN 指令和 OUT 指令的语法如图 11-2 所示。这是 Pentium 等 x86 系列 CPU 用的 IN 指令和 OUT 指令的语法。**IN 指令**通过指定端口号的端口输入数据，并将其存储在 CPU 内部的寄存器中。**OUT 指令**则是把

① 在向窗口和打印机输出字符串时，可以使用 Windows 提供的 TextOut 函数作为 API。C 语言提供的 printf 函数，是用来在命令提示符中显示字符串的函数。使用 printf 函数，是无法向窗口和打印机输出字符串的。

CPU 寄存器中存储的数据，输出到指定端口号的端口。

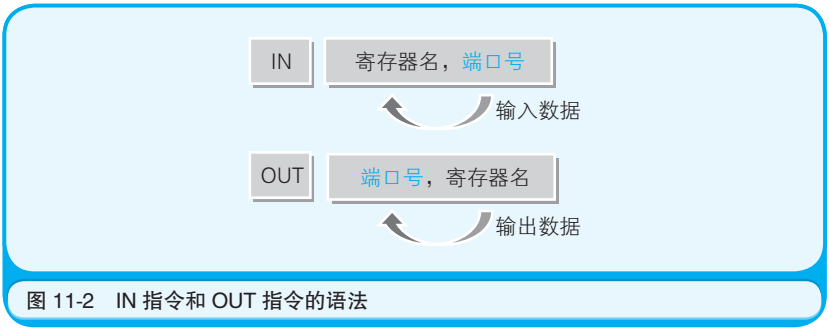

图 11-2 IN 指令和 OUT 指令的语法

下面让我们来看一下端口号和端口到底是什么。计算机主机中，附带了用来连接显示器及键盘等外围设备的连接器。而各连接器的内部，都连接有用来交换计算机主机同外围设备之间电流特性的 IC。这些 IC，统称为 **I/O 控制器**。由于电压不同，数字信号及模拟信号的电流特性也不同，计算机主机和外围设备是无法直接连接的。为了解决这个问题，I/O 控制器就很有必要了。

I/O 是 Input/Output 的缩写。显示器、键盘等外围设备都有各自专用的 I/O 控制器。I/O 控制器中有用于临时保存输入输出数据的内存。这个内存就是**端口**。端口（port）的字面意思是“港口”。由于端口就像是在计算机主机和外围设备之间进行货物（数据）装卸的港口，所以因此得名。I/O 控制器内部的内存，也称为寄存器。虽然都是寄存器，但它和 CPU 内部的寄存器在功能上是不同的。CPU 内部的寄存器是用来进行数据运算处理的，而 I/O 寄存器则主要是用来临时存储数据的。

在实现 I/O 控制器功能的 IC 中，会有多个端口。由于计算机中连接着很多外围设备，所以就会有多个 I/O 控制器，当然也会有多个端

口。一个 I/O 控制器既可以控制一个外围设备，也可以控制多个外围设备。各端口之间通过**端口号**进行区分。端口号也称为 **I/O 地址**。IN 指令和 OUT 指令在端口号指定的端口和 CPU 之间进行数据的输入输出。这和通过内存地址来进行主内存的读写是一样的道理（图 11-3）。

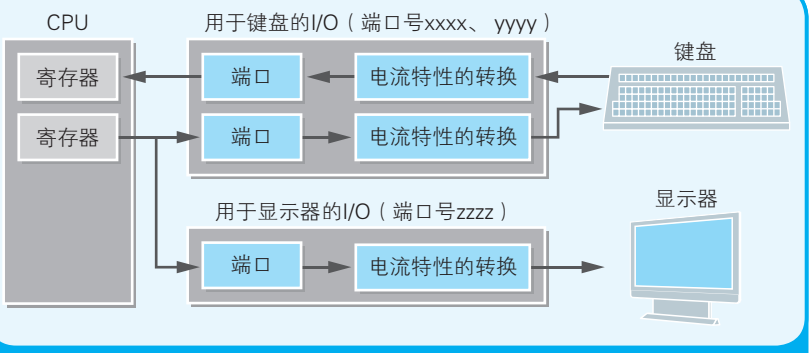

图 11-3　以端口为桥梁来实现 CPU 和外围设备之间的数据传递[①]

通过 Windows 的控制面板，我们可以查看外围设备所连接的 I/O 控制器的端口号。图 11-4 是通过 Windows 控制面板来查看软盘驱动控制器的属性时的情况[②]。“I/O 的范围”右侧的数值就是端口号。通过指定该端口号，并利用 IN/OUT 命令，就可以直接控制软驱这个硬件设备，实现输入输出处理了。

① I/O 装置，有的直接附带在计算机主机的主板（用来放置 CPU 的基板）上，有的则是各自独立的扩张板卡。键盘、鼠标、打印机等常用的 I/O，一般都在主板上，而显示高速图形的显示器及网卡等特殊的 I/O，通常是独立的扩张板卡。

② 近年来软驱已经不是标配了，Win7 后的版本中，可以通过控制面板→系统安全→系统→设备管理查看。——译者注

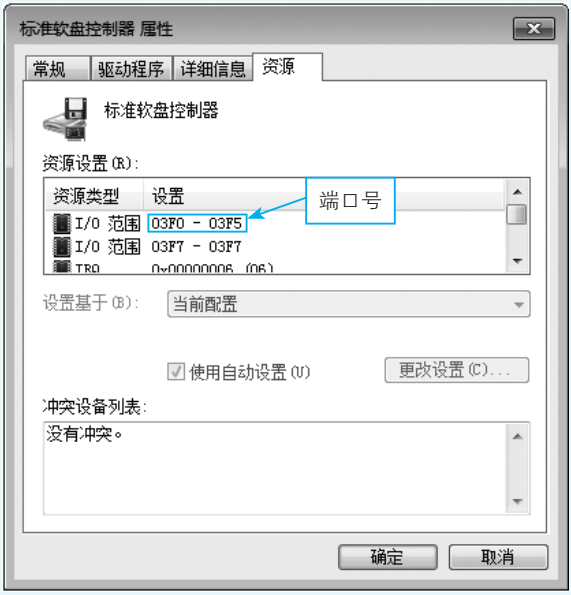

图 11-4 通过控制面板查看软盘驱动器的端口号

11.3 编写测试用的输入输出程序

首先让我们利用 IN 指令和 OUT 指令，来进行一个直接控制硬件的试验。假设该试验的目的是让计算机内配置的蜂鸣器（小喇叭）发音。虽然蜂鸣器内置在计算机内部，但其本身也是外围设备的一种。因为就算是把蜂鸣器取出，对计算机主机也不会有什么影响。

由于用汇编语言编写程序比较麻烦，因此这里我们采取在 C 语言源代码中插入助记符的方式来实现。在大部分 C 语言的处理（编译器的种类）中，只要使用 _asm{ 和 } 括起来，就可以在其中记述助记符。也就是说，这样就可以编写 C 语言和汇编语言混合的源代码。这里，