第 3 章 微型计算机输入输出接口

CHAPTER

外部设备是构成微型计算机系统的重要组成部分。程序、数据和各种外部信息要通过外部设备输入到计算机内，计算机内的各种信息和处理的结果要通过外部设备进行输出。微型计算机和外部设备的数据传输，在硬件线路与软件实现上都有其特定的要求和方法。本章重点讨论连接微机系统总线和外部设备的硬件电路——输入输出接口（Input/Output Interface，I/O 接口）的结构和组成方法，微型计算机和外设数据传输的方法，I/O 接口程序设计，以及 PC 系列微机常用外设的接口。

3.1 输入输出接口

3.1.1 外部设备及其信号

按照外部设备与 CPU 之间数据信息传输的方向，外部设备可以划分为以下 3 类。

（1）输入设备：数据信息从外设送往 CPU。

（2）输出设备：数据信息从 CPU 送往外设。

（3）复合输入输出设备：数据信息在 CPU 与外设之间双向传输。

按照设备的功效，外部设备又可以划分为以下 4 类。

（1）人机交互设备：在操作员与微机之间交换信息，例如键盘、鼠标和显示器。

（2）数据存储设备：例如软盘、硬盘、光盘驱动器和 U 盘。

（3）媒体输入输出设备：例如扫描仪和打印机等。

（4）数据采集与设备控制：模拟量输入、转换设备和过程控制设备等。

外部设备的信号因设备而异，但是，它们与主机之间交换的信号还是可以归类为以下 3 种。

（1）数据信号：以二进制形式表述的数值、文字、声音和图形信息。

（2）控制信号：CPU 以一组二进制向外设发出命令，控制设备的工作。

(3) 状态信号：一组二进制表示的外部设备当前工作状态，从外部设备送往CPU。

- 输入设备在完成一次输入操作(例如用户在键盘上按下一个按键)之后，发出就绪信号(READY)，等待CPU进行数据传输。
- 输出设备在接收了来自CPU的数据信息，实施输出的过程中，发出忙信号(BUSY)，表明目前不能接收新的数据信息。
- 有的设备有指示出错状态的信号，如打印机的纸尽(Paper Out)和故障(Fault)。

数据信号、控制信号和状态信号都是以数据的形式通过数据总线与CPU进行传输的。

3.1.2 I/O接口的功能

接口是计算机一个部件与另一个部件之间的连接界面。I/O接口则用来连接计算机系统总线与外部设备，它具有如下的功能。

1. 设备选择功能

CPU通过地址代码来标识和选择不同的外部设备。接口对系统总线上传输的外设地址进行译码，在检测到本设备地址代码时，产生相应的选中信号，并按CPU的要求进行信号传输。

2. 信息传输与联络功能

在设备被选中时，接口从CPU接收数据或控制信息，或者将数据或状态信息发往数据总线。

3. 数据格式转换功能

外设使用的数据格式与CPU的数据格式不同时，接口要进行两种数据格式之间的相互转换。例如，把来自键盘的串行格式信息转换为并行信息。

4. 中断管理功能

中断管理功能主要包括向CPU申请中断、向CPU发中断类型号以及中断优先权的管理等。在以80x86为CPU的系统中，这些功能大部分由专门的中断控制器实现。

5. 复位功能

接口在接收系统的复位信号后，将接口电路及其所连接的外部设备置成初始状态。

6. 可编程功能

有些接口具有可编程特性，可以用指令来设定接口的工作方式、工作参数和信号的极性。可编程功能扩大了接口的适用范围。

7. 错误检测功能

许多数据传输量大、传输速率高的接口具有信号传输错误的检测功能。常见的信号传输错误有以下两种：

1) 物理信道上的传输错误

信号在线路上传输时，如果遇到干扰信号，可能发生传输错误。检测传输错误的常见方法是奇偶校验。以偶校验为例，发送方在发送正常位数据信息的同时，增加一位校验位，通过对校验位设置为0或1，使信息位连同校验位中1的个数为偶数；接收方核对接收到的信息位和校验位中1的个数，若1的个数是奇数，则可以断定产生了传输错误。需

要说明的是，奇偶校验是一种比较简单的检验方法，它能够确定某次数据传输是错误的，但却不能确定某次数据传输一定是正确的。

2）数据传输中的覆盖错误

输入设备完成一次输入操作后，把所获得的数据暂存在接口内。如果在该设备完成下一次输入操作之前，CPU 没有从接口取走数据，那么，在新的数据送入接口后，上一次的数据被覆盖，导致数据的丢失。输出操作中也可能产生类似的错误。

3.1.3　I/O 端口的编址方法

1. 端口

接口内通常设置有若干个寄存器，用来暂存 CPU 和外设之间传输的数据、状态和命令。这些寄存器被称为端口（port）。根据寄存器内暂存信息的种类和传输方向，可以有**数据输入端口**、**数据输出端口**、**命令端口**（也称**控制端口**）和**状态端口**。每一个端口有一个独立的地址。CPU 用地址来区别各个不同的端口，对它们进行读、写操作，如表 3-1 所示。CPU 对状态端口进行一次读操作，可以得到该端口暂存的状态代码，从而获得与这个接口相连接的外部设备的状态信息。CPU 对数据端口进行一次读或写操作，也就是与该外部设备进行一次数据传输。而 CPU 把若干位代码写入命令端口，则意味着对该外部设备发出一个控制命令，要求该设备按代码的要求进行工作。由此可见，CPU 与外部设备的输入输出操作，都是通过对相应端口的读写操作来完成的。所谓外部设备的地址，实际上是该设备接口内各端口的地址，一台外部设备可以拥有几个通常是相邻的端口地址。

表 3-1　CPU 对 I/O 端口的读写操作

端口种类	读　端　口	写　端　口
数据输出端口	非法操作*	把输出的数据写入端口，继而送往输出设备
数据输入端口	把从输入设备输入的数据读入 CPU	非法操作
命令（控制）端口	非法操作*	向端口写入对外部设备的控制命令
状态端口	从端口读入外部设备的当前状态	非法操作

* 某些接口的输出类端口允许把当前已经输出的内容读入 CPU，称为回读。该操作需要接口内相关电路的支持。

2. I/O 端口的编址

I/O 端口地址的编排有两种不同的方法。

1）I/O 端口与内存统一编址

这种编址方式也称为存储器映射编址方式，它从内存的地址空间里划出一部分，分配给 I/O 端口，一个 8 位端口占用一个内存字节单元地址。已经用于 I/O 端口的地址，存储器不能再使用。

I/O 端口与内存统一编址后，访问内存储器单元和 I/O 端口使用相同的指令，这有助于降低 CPU 的复杂性，给使用者提供方便。但是，I/O 端口占用内存地址，相对减少了内存的可用范围。而且，由于难以区分访问内存和 I/O 的指令，降低了程序的可读性和可维护性。

2）I/O端口与内存独立编址

这种编址方法中，内存储器和I/O端口各自有自己独立的地址空间。访问I/O端口需要专门的I/O指令。

8086/8088 CPU采用这种编址方式。

（1）访问内存储器使用20根地址线A_0～A_{19}，同时使$M/\overline{IO}=1$，内存地址范围为00000～0FFFFFH共1MB。

（2）访问I/O端口使用低16根地址线A_0～A_{15}，同时使$M/\overline{IO}=0$，IO端口地址范围为0000～0FFFFH。

采用这种方式后，两个地址空间相互独立，互不影响。

3. IBM PC微型计算机I/O端口地址分配

在早期PC系列微机中，使用A_0～A_9共10条地址线定义了1024个I/O端口（设A_{11}～$A_{15}=0$），地址范围为0～3FFH。前256个端口地址供主板上的接口芯片使用，后768个端口供扩展槽接口卡使用，部分主板上的I/O端口分配情况列于表3-2中。

表3-2 系统板（主板）上I/O部分接口器件的端口地址

I/O接口器件名称	PC/XT	PC/AT
DMA控制器1	000H～01FH	000H～01FH
中断控制器1	020H～021H	020H～021H
定时器	040H～043H	040H～05FH
并行接口芯片	060H～063H	—
键盘控制器	—	060H～06FH
RT/CMOS RAM	—	070H～07FH
DMA页面寄存器	080H～083H	080H～09FH
中断控制器2	—	0A0H～0BFH
NMI屏蔽寄存器	0A0H～0BFH	—
DMA控制器2	—	0C0H～0DFH
协处理器	—	0F0H～0FFH

3.1.4 简单I/O接口的组成

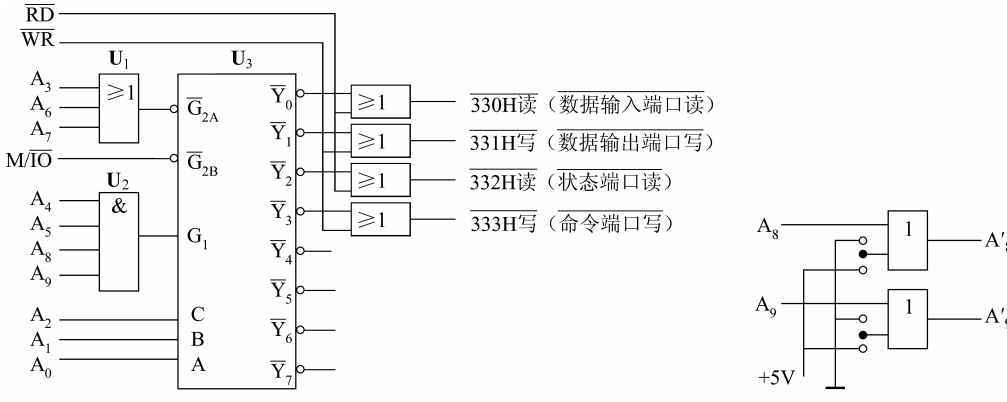
1. 地址译码电路

地址译码是接口的基本功能之一。CPU在执行输入输出指令时，向地址总线发送外部设备的端口地址。在接收到与本接口相关的地址后，译码电路应能产生相应的选通信号，使相关端口寄存器进行数据、命令或状态的传输，完成一次I/O操作。

由于一个接口上的几个端口地址通常是连续排列的，可以把地址代码分解为两个部分：高位地址用作对接口的选择，低位地址用来选择接口内不同的端口。

例如，某接口数据输入端口、数据输出端口、状态端口和命令端口的地址分别为330H、331H、332H和333H。假设该系统使用10位端口地址。那么，当8位高位地址为11001100时，表明本接口被选中，2位低位地址的4种组合00、01、10、11分别表示选中了本接口的数据输入端口、数据输出端口、状态端口和命令端口。由此，在最小模式的系统

中，可以设计如图 3-1 所示的译码电路。选中本接口时，地址码 A_7、A_6、A_3 均为 0，或门 U_1 输出 0。同时，选中本接口时，地址码 A_9、A_8、A_5、A_4 全为 1，于是与门 U_2 输出 1，它们连同 M/$\overline{IO}$的“0”使 3-8 译码器 U_3 工作。根据 A_2 A_1 A_0 的 8 种组合，可以得到 8 个地址选择信号（本接口只使用其中的 4 个），与$\overline{RD}$、$\overline{WR}$的进一步组合，就得到了本例中 4 个端口的读/写选通信号。



(a) 地址译码电路　　(b) 跳线器

图 3-1　端口的地址译码电路

设定端口地址时，注意不能和已有设备的端口地址重复。为了避免重复的发生，许多接口电路允许用跳线器（jumper）改变端口地址。图 3-1（b）给出了一个跳线器的例子。异或门（半加器）的一个输入引脚来自跳线器的中间引脚，它可以由使用者选择连接 1（+5V）或 0（接地），另一个引脚分别连接 A_8 和 A_9。将异或门的输出 A_8'和 A_9'代替图 3-1（a）中的 A_8 和 A_9 引脚连接到 U_2。两个跳线引脚均接地时，上面译码电路仍然产生 330H～333H 的端口译码信号，而当两个跳线引脚均接 1 时，则上面译码电路会产生 030H～033H 的端口译码信号。同理还可以产生 130H～133H、230H～233H 的译码信号。

由于读、写操作不会同时进行，一个输入端口和另一个输出端口可以使用同一个地址代码。例如，可安排数据输入端口和数据输出端口使用同一个地址 330H，命令端口和状态端口共同使用地址 331H，则图 3-1（a）中右端的信号组合电路可作如图 3-2 所示的修改。

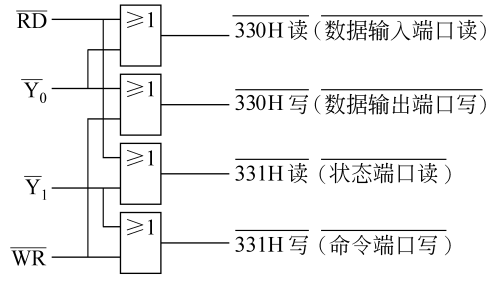


图 3-2　修改后的译码电路

图中，Y_0 和 Y_1 是译码器输出的两个地址选择信号。需要注意的是，数据输入端口和数据输出端口虽然使用相同的地址，但却是两个各自独立的不同的端口。

8086 工作于最大模式时，由 8288 总线控制器发出$\overline{\text{IORC}}$和$\overline{\text{IOWC}}$代替上面的 M/$\overline{\text{IO}}$、$\overline{\text{WR}}$和$\overline{\text{RD}}$。读者不妨自己设计一个相应的地址译码电路。

2. 数据锁存器与缓冲器

在微机的系统数据总线上，连接着许多能够向 CPU 发送数据的设备，如内存、外部设备的数据输入端口等。为了使系统数据总线能够正常地进行数据传送，要求所有这些连接到系统数据总线的设备具有三态输出的功能。也就是说，在 CPU 选中该设备时，它能向系统数据总线发送数据信号。在其他时间，它的输出端必须呈高阻状态。为此，所有的输入端口必须通过三态缓冲器与系统总线相连。

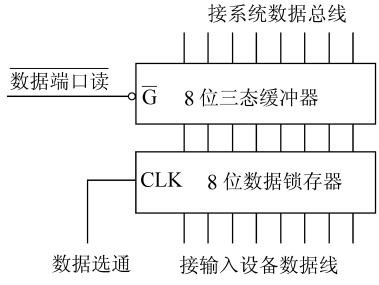


图 3-3 输入设备接口的数据锁存和缓冲电路

图 3-3 中，输入设备在完成一次输入操作后，在送出数据的同时，产生数据选通信号，把数据打入 8 位锁存器 74LS273。锁存器的输出信号通过 8 位三态缓冲器 74LS244 连接到系统数据总线。数据端口读信号由地址译码电路产生。该信号为高电平（无效）时，三态缓冲器输出端呈高阻态。一旦该端口被选中，数据端口读变为低电平（有效），已锁存的数据就可以通过 74LS244 送往系统数据总线继而被 CPU 所接收。

如果输入设备自身具有数据的锁存功能。输入接口内可以不使用锁存器。输入设备的数据线可通过三态缓冲器与系统数据总线相连接。但是，由于系统总线的工作特点，输入接口中的三态缓冲器是绝对不能省略的。

CPU 送往外设的数据或命令一般应由接口进行锁存，以便使外设有充分的时间接收和处理。图 3-4 是一个 8 位输出锁存电路的例子。

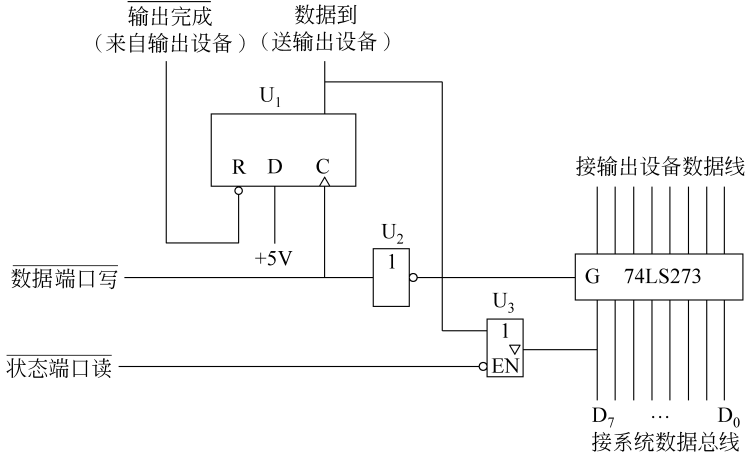


图 3-4 输出锁存电路

由地址译码电路产生的数据端口写选通信号是一个负脉冲，经 U_2 反相后把来自系统数据总线的 8 位数据 $D_0 \sim D_7$ 打入 8 位寄存器 74LS273，经输出端 $Q_0 \sim Q_7$ 送往外部设备。数据端口写信号同时把 D 触发器(U_1)置 1，通过 Q 端发出“数据到”信号，通知外部设备及时接收已输出的一字节数据。外部设备在输出完成之后，向接口回送一个输出完成负脉冲，将 D 触发器(U_1)清零，准备接收下一个数据。

外部设备在接收和输出数据期间，D 触发器 Q 端保持为 1。所以，它同时也成为该设备的状态信号 BUSY。图 3-4 中该信号通过 1 位三态缓冲器(U_3)连接到双向数据总线 D_7，三态缓冲器由地址译码获得的状态端口读信号控制。CPU 通过对状态端口的读指令，在 D_7 上可以获知它的状态。该位为 1 时，CPU 不能向数据输出端口发送新的数据，否则将发生“覆盖错误”。

综上所述，把地址译码、数据锁存与缓冲、状态寄存器、命令寄存器各个电路组合起来，就构成一个简单的输入输出接口(见图 3-5)。它一方面与系统地址总线 $A_0 \sim A_{15}$、数据总线 $D_0 \sim D_7$，控制总线 M/$\overline{IO}$、$\overline{RD}$、$\overline{WR}$(最小模式时)或$\overline{IOWC}$、$\overline{IORC}$(最大模式时)相连接，另一方面又与外部设备相连。由于常用的字符输入输出设备均使用 8 位数据，上述例子中均使用 8 位的数据输入输出端口。对于 16 位的 I/O 设备，其接口的基本原理是相同的。

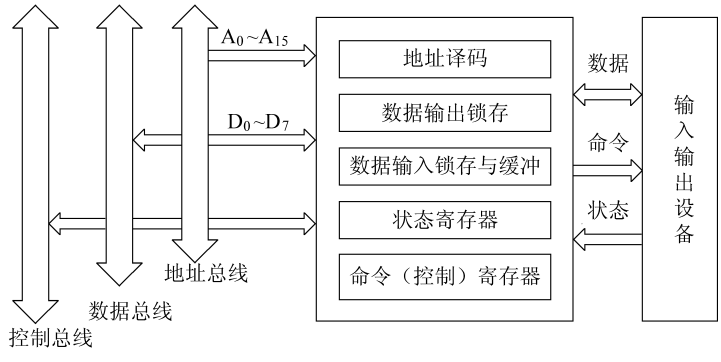


图 3-5　简单接口的组成

3.2　输入输出数据传输的控制方式

CPU 与外部主要进行两种类型的数据传输：与内存的数据传输和与外部设备的数据传输。CPU 使用一个总线周期就可以与内存进行一次数据传输，而且这个过程可以连续进行。CPU 与外设的数据传输则要复杂得多。CPU 从输入设备读入一个数据之后，要等到该设备完成了第二次数据输入之后，才能读入第二个数据。等待的时间不但与该设备的工作速度有关，有时也带有许多随机的成分。例如，用户在键盘输入过程中，两次按键的间隔时间往往是不确定的。因此，较之与内存的数据传输，CPU 与外设的数据传输有着不同的特点，因而也有着不同的处理方式。概括起来有以下 3 种传送方式：程序方式、中断方式和 DMA 方式。

3.2.1 程序方式

程序方式传送是指在程序控制下进行信息传送，具体实现又可分为无条件传送和条件传送两种方式。

1. 无条件传送方式

一些简单的I/O设备，对它们的I/O操作可以随时进行。例如，一些设备常用一组开关指示设备的配置情况和操作人员设定的工作方式：每个开关只有两种不同状态（ON/OFF，对应于1/0），它与某个输入端口中的一位（bit）相对应。程序员可以随时用输入指令读取该端口内每个开关的状态，而无须考虑它的状态。这一类简单设备的输入信号一般不需要锁存，可以通过三态缓冲器与系统数据总线直接相连（见图3-6）。

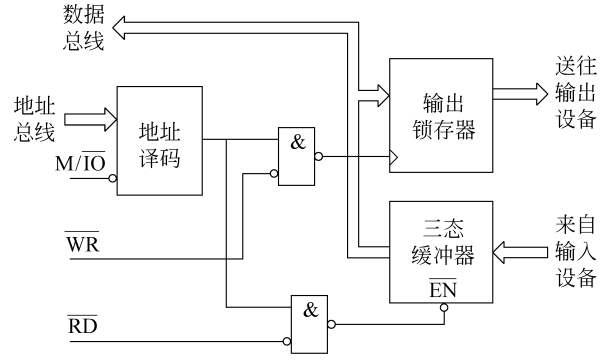


图3-6 无条件输入输出接口

一些简单的输出设备也有类似的情况。例如，常常用一组发光二极管（LED）来指示设备当前的工作状态。每一个LED对应于输出端口的一位（bit）。它的亮/暗代表某个设备两个特定的状态。例如，某LED亮表示2#电机已通电运转，暗表示该电机未通电等。这样的输出信号通常要在接口内进行锁存，以便在新的输出到来之前保持现在的输出状态。

图3-6给出了无条件传送时接口的组成方式，它是3.1节中介绍的几个部分的简单组合。一个8位的无条件输入接口只使用一个端口（数据输入端口），占用一个端口地址，16位无条件输入接口也只有一个数据输入端口，占用两个端口地址。无条件输出接口的情况类似。

2. 条件传送方式

条件传送也称为查询式传送。使用条件传送方式时，CPU通过程序不断读取并测试外设的状态。如果输入设备处于准备好状态，CPU执行输入指令从该设备输入。如果输出设备处于空闲状态，则CPU执行输出指令向该设备输出。为此，接口电路除了有传送数据的端口以外，还应有传送状态的端口。对于输入过程来说，外设将数据准备好时，将接口内状态端口的“准备好”标志位（READY）置1。对于输出过程来说，外设输出了一个数据后，接口便将状态端口的“忙”（BUSY）标志位清零。表示当前输出寄存器已经处于

“空”状态，可以接收下一个数据。

可见，对于条件传送来说，一个数据的传送过程由 3 个环节组成：

(1) CPU 从接口中读取状态字。

(2) CPU 检测状态字的对应位是否满足“就绪”条件，如果不满足，则回到(1)重新读取状态字。

(3) 如状态字表明外设已处于“就绪”状态，则传送数据。

图 3-7 展示了查询方式的输入接口电路。该接口内有两个端口：用于输入数据的“数据输入端口”由数据锁存器和一个 8 位三态缓冲器组成；用于存储设备状态的状态端口由一个 D 触发器和一个 1 位三态门组成，三态门的输出连接到数据总线的任选一根(本例为 D_7)。输入设备在数据准备好以后向接口发一个选通信号。这个选通信号有两个作用：一方面将外设的数据送到接口的锁存器中；另一方面使接口中的 D 触发器置 1。按照数据传送过程的 3 个步骤，CPU 从外设输入数据时先读取状态字(本例中状态字仅一位)，通过检查状态字确定外设是否“准备就绪”，即数据是否已进入接口的锁存器中。如准备就绪，则执行输入指令读取数据。读取数据的同时把 D 触发器清零，设备又恢复到“未就绪”状态，本次数据传输到此结束。

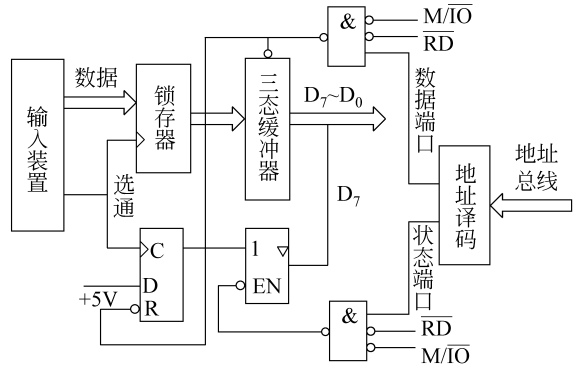


图 3-7　查询式输入接口电路

相应的汇编语言指令如下：

```
AGAIN:  IN      AL, STATUS_PORT        ;读状态端口,如果 D7=1 表示"数据就绪"
        TEST    AL, 80H                ;测试"数据就绪"位
        JZ      AGAIN                  ;未就绪,继续读状态端口
        IN      AL, DATA_PORT          ;已就绪,从数据端口读取数据,同时清除状态位
        …
```

用 C 语言编写如上过程，则更为简便：

```
do {stat=inportb(status_port);}
    while (stat & 0x80==0);            /* 数据未准备好则反复读状态 */
data=inportb(data_port);               /* 数据已准备好则读取数据 */
```

图 3-8 展示了查询方式的输出接口电路。同样地，该接口也有两个端口：数据输出端口由 8 位锁存器构成，状态（输入）端口由一个 D 触发器和一个 1 位三态门组成。CPU 需要向一个外设输出数据时，先读取接口中的状态字，如果状态字表明外设空闲（“不忙”，BUSY＝0），说明可以向外设输出数据，此时 CPU 才执行数据输出指令，否则 CPU 必须等待。

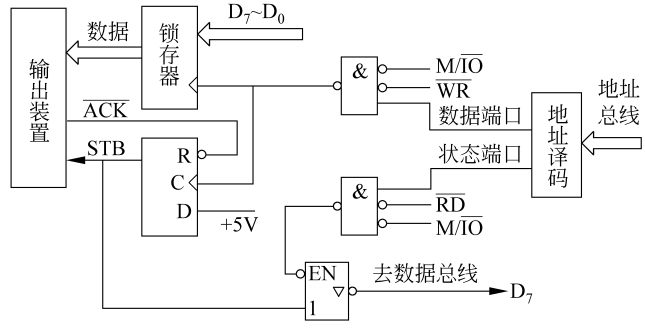


图 3-8　查询式输出接口电路

CPU 执行数据输出指令时，地址译码电路产生的数据锁存信号将数据总线上的数据打入接口数据锁存器，同时将 D 触发器置 1。D 触发器的输出信号一方面为外设提供一个联络信号 STB，告诉外设现在接口中已有数据可供提取；另一方面也用作该设备的状态标志“忙”（BUSY）。CPU 读取状态端口后可以得知该外设处于“忙”状态，从而阻止输出新的数据。输出设备从接口取走数据，或者完成了本次输出后，通常会回送一个应答信号$\overline{\text{ACK}}$。该信号使接口内的 D 触发器清零，也是把状态位清零，这样就可以开始下一个输出过程。

相应的汇编语言程序为

```
ONE:  IN      AL, STATUS_PORT        ;读状态端口
      TEST    AL, 80H                ;测试"忙"位
      JNZ     ONE                    ;忙,再读状态端口
      MOV     AL, DATA               ;不忙,取来数据
      OUT     DATA_PORT, AL          ;数据送入数据端口,同时置 BUSY=1
      …
```

相应的 C 语言程序为

```
do {stat=inportb(status_port);}
    while (stat & 0x80==0x80);        /*设备"忙"则反复读状态*/
outportb(data_port, data);            /*设备空闲则输出数据*/
```

可以发现，它和用于输入的程序段有不少相似之处。

进行多个数据的输入输出时，每进行一次输入或者输出都要首先查询它的状态字，只有当设备就绪时才可以进行数据的传输。图 3-9 是查询式输入的程序流程图。

下面通过一个例子介绍查询式输入输出的程序设计方法。

某字符输入设备以查询方式工作，数据输入端口地址为0054H，状态端口地址为0056H。如果状态寄存器中 D_0 位为1，表示输入缓冲器中已经有一个字节准备好，可以进行输入；D_1 位为1表示输入设备发生故障。要求从该设备上输入80个字符，然后配上水平和垂直校验码（本例中采用偶校验），向串行口输出。如果设备出错，则显示错误信息后停止。

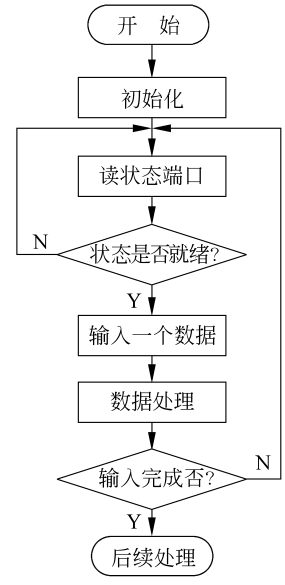


图 3-9　查询式输入流程

汇编语言程序如下：

```
.MODEL    SMALL
.DATA
Buffer    DB       81 dup(?)        ;多留1字节用于存放
                                    ;垂直校验码
Message   DB       'Device  Fault!',0DH,0AH,'$'
.CODE
Start:    MOV      AX, @DATA        ;对DS初始化
          MOV      DS, AX
          LEA      SI, Buffer       ;置SI为缓冲区指针
          MOV      CX, 80           ;设置CX为计数器
          MOV      DL, 0            ;设置DL为垂直校验码初值
Next:     IN       AL, 56H          ;读入状态
          TEST     AL, 02H          ;测状态寄存器 D1
          JNZ      ERROR            ;设备故障,转ERROR
          TEST     AL, 01H          ;测状态寄存器 D0
          JZ       Next             ;未准备好,则等待,再测
          IN       AL, 54H          ;已准备好, 输入字符
          AND      AL, 7FH          ;清最高位,同时进行校验
          JPE      Store            ;已经是偶数个1,则转Store
          OR       AL, 80H          ;奇数个1,将最高位置为1
Store:    XOR      DL, AL           ;产生垂直校验码
          MOV      [SI], AL         ;将字符送缓冲区
          INC      SI               ;修改地址指针
          LOOP     Next             ;80个字符未输入完成,继续
          MOV      [SI], DL         ;80个字符输入完成,保存垂直校验码
Transfer: LEA      SI, Buffer       ;准备发送,SI中置字符串首址
          MOV      CX, 81           ;发送字符数,连同垂直校验码为81个
One:      MOV      AH, 04H          ;设置串口输出功能号
          MOV      DL, [SI]         ;取出一个字符
          INT      21H              ;从串口输出
          INC      SI               ;修改指针
          LOOP     One              ;输出下一个字符
          JMP      Done
Error:    MOV      AH, 09H          ;设备故障,输出出错信息
          LEA      DX, Message
```

```
          INT     21H
Done:     MOV     AX, 4C00H
          INT     21H             ;返回操作系统
          END     Start
```

对以上程序作几点说明：

程序由两段并列的循环程序组成：第一段程序从设备输入80个字符，同时产生它的水平/垂直校验码存入缓冲区；第二段程序将缓冲区内容通过串口输出。

测试状态位要注意先后次序：由于设备故障可能导致该设备不能正常输入，使完成标志(D_0)恒为零。所以，在设备发生故障时先测试完成标志可能导致程序死循环。

一般来说，计算机内的字符用7位ASCII代码表示，需要进行奇偶校验时通常用一个字节的最高位作为校验位，低7位存放字符的ASCII代码。程序中产生水平校验码的方法是：从设备读入数据后，清除最高位，然后根据剩余7位的奇偶特性决定最高位置1或不变(保持为0)。垂直校验码则由80个字节半加(异或)得到。

第一段程序用查询的方法进行字符输入，第二段程序用系统功能调用的方法输出。可见，对外部设备输入输出的方法不是唯一的，要根据具体情况决定采用哪一种方法。

上述程序完成了一个设备的输入输出处理，如果系统有多个设备需要使用查询方式进行输入输出，则可采用循环查询的方法。下例中假定有3个设备，它们的状态端口地址分别为STAT1、STAT2和STAT3，并假定3个状态端口均使用05位作为准备好标志。

```
TREE:     MOV     FLAG,07H        ;设置3个设备的初始状态，每个设备占用一位
INPUT:    IN      AL,STAT1
          TEST    AL,20H
          JZ      DEV2
          CALL    PROC1           ;子程序PROC1完成设备1的数据输入输出
DEV2:     IN      AL,STAT2
          TEST    AL,20H
          JZ      DEV3
          CALL    PROC2           ;子程序PROC2完成设备2的数据输入输出
DEV3:     IN      AL,STAT3
          TEST    AL,20H
          JZ      NOINPUT
          CALL    PROC3           ;子程序PROC3完成设备3的数据输入输出
NOINPUT:  CMP     FLAG,0          ;测试3个设备是否均已完成数据输入输出
          JNE     INPUT
          …
```

上述程序中，PROC1、PROC2和PROC3分别是3个设备输入输出的子程序。为了避免3个设备输入输出完成后程序陷入死循环，上例中设置了一个内存单元FLAG作为3个设备是否输入完成的标志，它的初值为7(0000 0111B)，每一位二进制代表一个设备的完成状态。每当一个设备输入完成，就在各自的输入输出处理子程序PROC1、PROC2和PROC3中将FLAG单元中代表本设备的那一位二进制清零。在标号NOINPUT处

判断 FLAG 是否为零：若 FLAG 值为零，说明 3 个设备均已输入完成，程序执行其他后续任务，否则转 INPUT 处继续 3 个设备的输入过程。

上例仅适用于 3 个设备工作速度都比较慢的情况。如果其中一个设备工作速度很快，而其他设备的输入输出处理程序运行时间又较长，则可能发生覆盖错误。在这种情况下，应优先执行工作速度较快的外设的 I/O 过程，然后再执行其他设备的 I/O 过程。

3.2.2　中断方式

程序查询方式的数据传送解决了 CPU 与外设工作速度的协调问题，但是却大大降低了 CPU 的使用效率。在程序查询方式的数据传送中，CPU 需要不断地查询外设接口中的状态标志，这样就会占用 CPU 大量的工作时间。在与一些中、慢速的外设交换信息时，真正用于传送数据的时间是极少的。为了避免发生覆盖错误而导致丢失数据，CPU 又不便同时从事其他工作，只能把大部分时间花费在查询状态上，工作效率十分低下。

在程序查询方式中，CPU 处于主动地位，外设处于消极等待查询的被动地位。在一个实际控制系统中，外设常常可能有数十个甚至上百个，由于它们的工作速度不相同，要求 CPU 服务的时间带有随机性，有些要求是很急迫的。查询方式的数据传送很难使系统中每一个外设都能工作在最佳状态。

为了使 CPU 能有效地管理多个外设，提高 CPU 的工作效率，可以赋予系统中的外设某种主动申请、配合 CPU 工作的“权利”。例如，某外设已把数据准备好（例如，模-数转换接口已经把一个模拟量转换成数字量，等待 CPU 来取数据），它可以主动向 CPU 发出一个请求信号。CPU 在接收到这个请求信号之后，暂停当前的工作，转而进行该设备的数据传送操作。数据传送结束之后，CPU 再继续刚才的工作。赋予外设这样一种“主动权”之后，CPU 可以不必反复查询该设备的状态，而是正常地处理系统任务，CPU 与外设处于某种“并行工作”的状态，这就是中断方式的数据传送。

利用中断方式进行数据传送，可以大大提高 CPU 的效率。例如，某外设在 1s 内传送 100B，若用程序查询的方式传送，则 CPU 为传送 100B 所花费的时间等于外设传送 100B 所用的时间，也是用了 1 秒的时间。如果用中断控制方式传送，CPU 为执行一个字节的传送需要进入一次中断服务程序。若 CPU 执行一次中断服务程序需要 100μs，则传递 100B CPU 所使用的时间为 100μs×100＝10ms，只占 1s 时间的 1%，其余 99%的时间 CPU 可用于执行主程序，CPU 的工作效率得到显著提高。

中断方式的数据传送仍在程序的控制下执行，所以也称为程序中断方式，适应于中、慢速的外部设备数据传送。

3.2.3　直接存储器存取（DMA）方式

计算机系统中某些外部设备工作速度很高。例如，现代硬磁盘驱动器工作时，数据传输的间隔时间小于 1μs。对于这样高速的外部设备，程序中断方式和程序查询方式的数据传输速度会跟不上外部设备的工作速度。

使用程序中断方式时，每传送一次数据，CPU 必须执行一次中断服务程序。由于外部中断是随机产生的，执行中断服务程序时必须将若干寄存器的内容压入堆栈，在返回时

再把它们弹出堆栈。在中断服务程序中还要判别设备的工作状态(例如区分是读磁盘还是写磁盘),寻找缓冲区的地址等。虽然直接用于数据传输的只有两三条指令,但进入一次中断服务程序,CPU却要执行几十条甚至上百条指令。有时就会发生这样的情况：一个数据尚未从接口的数据寄存器内取走,新的数据又送入了该寄存器,从而发生了丢失数据的覆盖错误。

程序查询方式的响应速度比中断方式要快一些,但完成一次数据传输也需要执行七八条以上的指令。CPU的工作速度不高时仍有可能跟不上外设数据传输的需要。

所谓直接存储器存取(Direct Memory Access,DMA)是指将外设的数据不经过CPU直接送入内存,或者,从内存储器不经过CPU直接送往外部设备。一次DMA传送只需要执行一个DMA周期(大体相当于一个总线读/写周期),因而能够满足高速外设数据传输的需要。

使用DMA方式传输时,需要一个专门的器件来协调外设接口和内存的数据传输,这个专门的器件称为DMA控制器,简称DMAC。

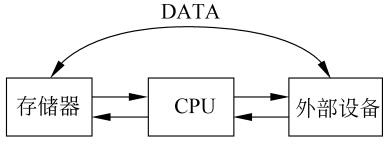


图3-10 DMA数据传输示意图

图3-10是DMA传输示意图。

3.3 开关量输入输出接口

开关量的输入输出一般不受状态的制约,所以都采用无条件的输入输出。本节通过开关量接口的分析及其程序设计,使读者对简单接口建立一个初步的、具体的概念。

3.3.1 开关量输入接口

1. 基本的开关量输入接口

常见的输入开关有3种形态：单刀单掷开关、单刀双掷开关和按钮。它们的基本连接及其接口见图3-11。

开关量通过三态缓冲器与系统数据总线相连接。常用的三态缓冲器有74LS244(输入输出同相)和74LS240(输入输出反相)。执行一条输入指令可以同时读入8位或16位这样的开关量。

2. 矩阵式开关量输入接口

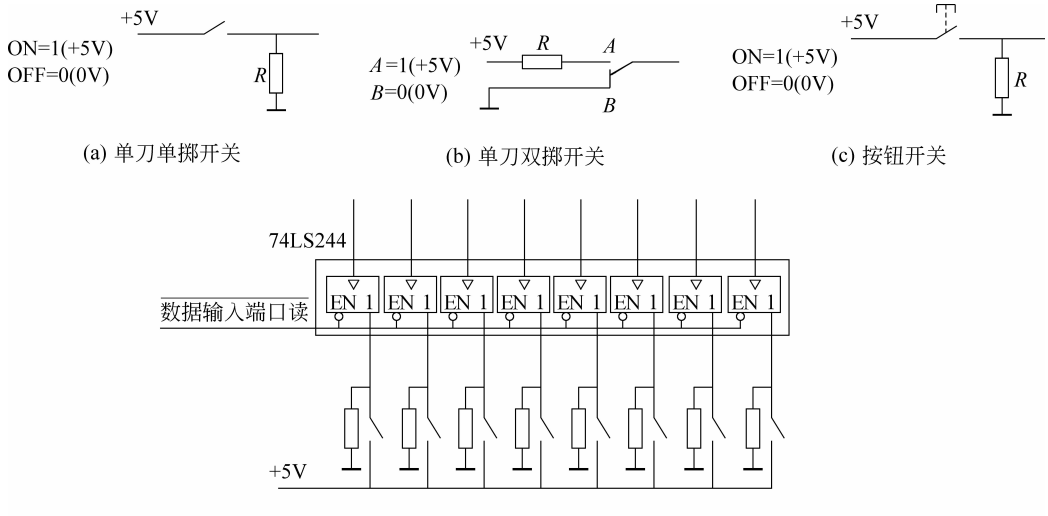
上述接口适用于少量开关量输入的场合。开关数量多时,常常把它们排列成矩阵以简化电路,数字设备上常用的键盘就是一个典型的例子。

图3-12中,一个8位数据输出端口的输出端连接了8根行线(Row,R_0～R_7),另一个数据输入端口由8根列线(Column,C_0～C_7)输入。在每根行线和列线的交叉点上连接了一个按钮,共64个。该电路有如下特点：

(1) 没有键按下时,输入端口输入为全1。

(2) 输出端口输出全1时,不论有无键按下,输入端口输入仍然为全1。

(3) 某一行线输出0时,如果该行上有一个键按下,则输入端口输入代码为7个1、1个0,0的位置与被按下键的位置相对应。



(a) 单刀单掷开关　(b) 单刀双掷开关　(c) 按钮开关

(d) 开关量输入接口

图 3-11　基本开关量输入接口

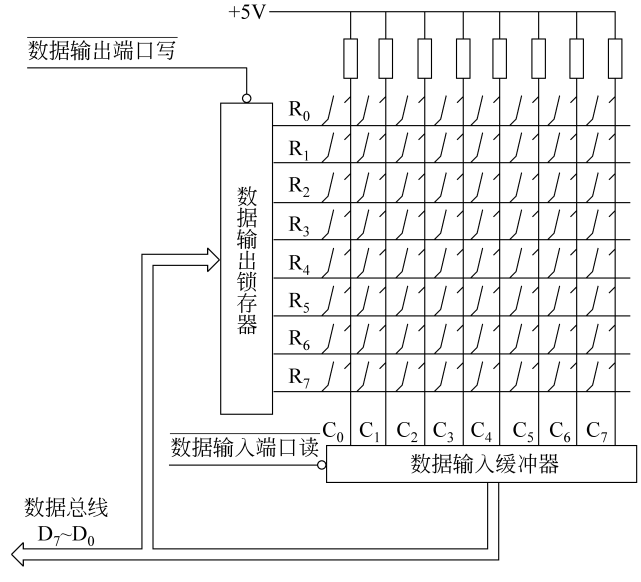


图 3-12　键盘接口

根据以上规则，可以通过程序对 8 根行线逐行扫描，识别按键的所在行、列，从而获得该键的代码。

一个键刚按下时，会产生抖动。这时读它的代码，容易产生误判。因此，在发现有键按下后要适当延时。

一个键的编码可以用它的二字节行列码表示。例如，键(R_3，C_2)的二字节行列码为 F7FBH，F7H＝11110111 表示按键在 R_3 行上，FBH＝11111011 表示按键在 C_2 列上。用行列码查表，可以得到这个键的代码(ASCII 代码，或者用户自己的编码)。一个键的编码

也可以用它的一字节扫描码表示。上例中按键的一字节扫描码为32H。高4位0011和低4位0010分别给出了键所在的行、列编号。

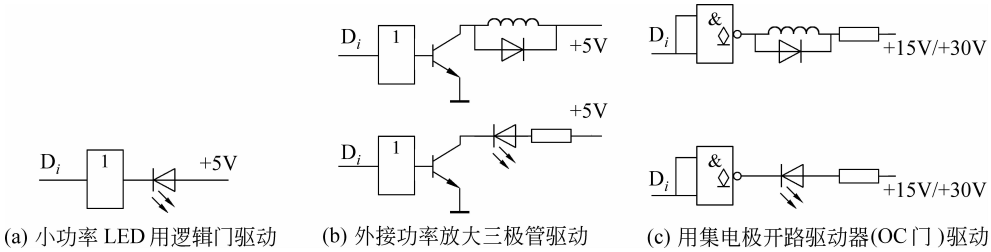
下面的程序对键盘进行扫描：没有键被按下，返回－1，否则，返回按键的二字节行列码（高8位为行码，低8位为列码）。程序中，符号常量RPORT和CPORT已定义为行、列端口的地址。

```
unsigned int kbinput()
{   unsigned int row, column, code;
    outportb(RPORT, 0);                    /* 各行输出全0,测试有无键按下 */
    if(inportb(CPORT)& 0xff==0xff) return(0xffff);  /* 没有键按下,返回全1 */
    delay(20);                             /* 有键按下,延时20ms,消除抖动 */
    if(inportb(CPORT) & 0xff==0xff) return(0xffff);
                                           /* 延时后再次测试,确认有键被按下 */
    row=0x7f;
    column=0xff;                          /* 置行码初值:从最高位对应行开始逐行扫描 */
    while(row!=0xff && column==0xff)      /* 循环:8行未测试完,并且未找到按键所在行 */
    {   outportb(RPORT, row);              /* 输出行码,测试行输出0,其余行输出1 */
        column=inportb(CPORT);             /* 读入列码 */
        row=row >>1;
        row=row+0x80;                      /* 形成下一个行码 */
    }
    if(column==0xff) return(0xffff);      /* 未找到按键所在行,返回全1 */
    code=row * 0x100+column;               /* 由行码、列码组合得到行列码 */
    outportb(RPORT, 0);                    /* 各行输出全零 */
    do { column=inportb(CPORT);}
        while(column & 0xff !=0xff);       /* 等待按键被释放 */
    return(code);
}
```

3.3.2 开关量输出接口

1. 基本的开关量输出接口

常见的开关量输出有两种：LED发光二极管（用作状态指示灯）和执行元件驱动线圈。它们的连接见图3-13。



(a) 小功率LED用逻辑门驱动　(b) 外接功率放大三极管驱动　(c) 用集电极开路驱动器(OC门)驱动

图3-13　基本的开关量输出电路

小功率 LED 指示灯用于指示室内仪表状态，它们可以由逻辑电路直接驱动。一般的 TTL 逻辑电路输出高电平时输出电流较小（μA 级），输出低电平时吸收电流稍大（mA 级，见表 3-3），所以逻辑电路输出端均与 LED 的阴极连接。这样，输出 0 时，LED 发光；输出 1 时，LED 熄灭。

表 3-3　部分逻辑电路输出端电流

器 件 型 号	高电平输出电流（I_{OH}）	低电平吸收电流（I_{OL}）
74LS00，04，10，20，30（逻辑门）	400μA	8mA
74LS01，03，05，12，22（OC 门）	100μA	8mA
7407（OC 驱动器）	250μA	40mA（V_{oh}＝30V）
74LS244（总线驱动器）	15mA	24mA
74LS273（D 触发器）	400μA	8mA
74LS373（三态输出锁存器）	2.6mA	24mA

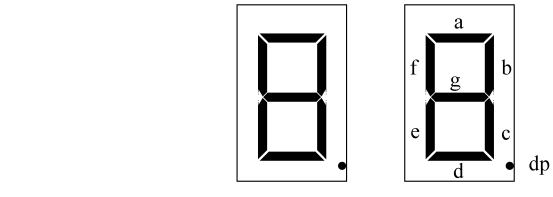
大功率 LED 驱动或执行元件驱动线圈的驱动有两种可选的方法：

（1）逻辑电路输出，外接功率放大三极管驱动。

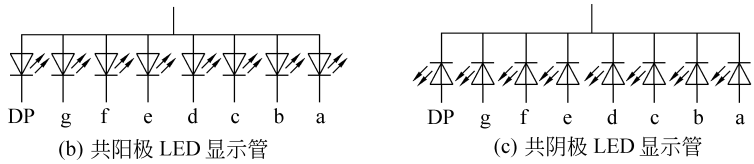
（2）采用集电极开路驱动器（OC 驱动器，如表 3-3 中的 7407），输出端通过上拉电阻接高压。输出 0 时，输出端内部三极管导通，线圈/LED 失电，执行元件不工作，LED 熄灭；输出 1 时，集电极开路，线圈/LED 通电，执行元件工作，LED 发光。

2. LED 七段数码显示管接口

LED 七段数码显示管是常用的数码显示设备。它的内部由 7 个（实际上有 8 个）发光二极管按一定顺序排列而成（见图 3-14）。把各二极管的阳极连接在一起，就构成共阳极 LED 显示管。同理，也有共阴极 LED 显示管。



(a) LED 显示管外形与二极管编码



(b) 共阳极 LED 显示管　　(c) 共阴极 LED 显示管

图 3-14　LED 数码显示管

七段显示数码管是电流驱动型器件，为了获得足够的亮度，需要为它提供较大的电流。可以采用图 3-13 中（b），（c）的方案进行驱动。

为了获得所需要的字形，通常把数字 0～9 的字形代码（称为七段码或段码）存入内存的一张表格，称为七段码表。通过查这张七段码表，可以得到一个数字的字形代码，向七

段显示管输出这个字形代码就实现了对该数字的显示。

有多位数字需要同时显示时，最简单的办法是为每一位数字设立一个独立的输出端口。但是，当数字位数较多时，使用的元器件较多。由于LED显示具有余辉效应，可以通过共用端口的方法来简化电路。以共阳极数码管为例，用一个公用端口储存七段码，同时连接到各个数码管的阴极。用另一个端口来控制各个数码管的阳极，每一位输出线通过驱动电路连接到一个数码管的阳极，从而控制这个数码管亮或暗，这个端口储存的代码称为位码。多位数码循环显示过程如下：

(1) 设置位码，熄灭所有数码管。

(2) 将一个数码管的字形代码(段码)送入段码端口。

(3) 设置位码，点亮一个数码管。

(4) 准备下一个数字的段码和位码，适当延时。

重复以上过程，多位不同的数字就同时显示在不同的数码管上。送段码之前熄灭所有数码管可以消除段码和位码不同步产生的闪烁。

以图3-15为例，为了在8个LED数码管上同时显示不同的数字，可以编写以下的8086汇编语言源程序(本例中，假设需要输出的数字分别是1、2、3、4、5、6、7、8。段码和位码的端口地址分别是segport和bitport)：

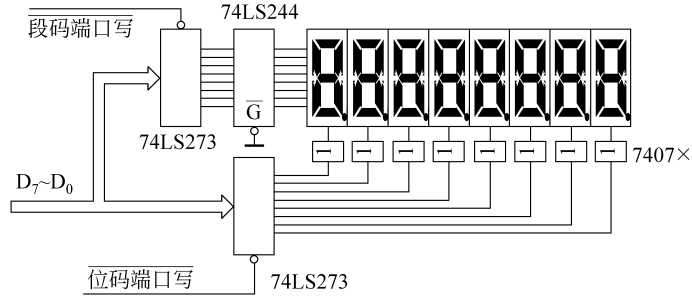


图3-15 多位LED数码显示接口

```
.model    small
.data
segtab    db 40h, 4fh, 24h, 30h, 19h
          db 12h, 02h, 78h, 00h, 10h
buffer    db 1, 2, 3, 4, 5, 6, 7, 8
bitcode   db  ?
.stack    100h
.code
leddisp   proc    far
          push    ds                    ;保护各寄存器内容
          push    ax
          push    bx
          push    cx
          push    si
```

```
            mov     ax, @data           ;装载 ds
            mov     ds, ax
            lea     bx, segtab          ;bx 置为七段码表首址
            mov     bitcode, 80h        ;置位码初始值为 80H(从左边 LED 开始显示)
            mov     si, 0               ;si 用作输出缓冲区指针,初值 0
            mov     cx, 8               ;cx 用作循环计数器,初值 8
one:        mov     al, 0
            out     bitport, al         ;送位码 0,熄灭各 LED
            mov     al, buffer[si]      ;从输出缓冲区取出一个待输出数字
            xlat                        ;转换成七段码
            out     segport, al         ;向段码端口输出
            mov     al, bitcode
            out     bitport, al         ;输出位码,点亮一个 LED
            ror     bitcode, 1          ;修改位码,得到下一个位码
            inc     si                  ;修改输出缓冲区指针
            call    delay               ;延时
            loop    one                 ;循环,点亮下一个 LED
            pop     si
            pop     cx                  ;恢复各寄存器
            pop     bx
            pop     ax
            pop     ds
            ret                         ;返回主程序
            leddisp  endp
            end
```

3.4　PC 系列微型计算机外部设备接口

本章 3.1～3.3 节介绍了计算机接口的基本原理,本节具体介绍 PC 微型计算机内部使用的各种外部设备接口。

3.4.1　传统低速外部设备接口

PC 系列微机配备使用的低速外部设备接口有串行通信接口、键盘接口、鼠标接口、打印机接口和软盘驱动器接口。在新一代 PC 微型计算机内,上述部分接口由于改用 USB 总线而不再出现。

1. 串行通信接口(COM1 和 COM2)

一个数据的各位在多根信号线上同时传送,称为并行传输。如果将一个数据的各位在 1 根/1 对信号线上按时间先后依次传送,称为串行传输。

早期 PC 提供两个低速的串行通信接口,命名为 COM1 和 COM2,可以用来连接鼠标、用于网络通信的调制解调器、串行接口的打印机等设备。早期串口使用 DB25 接插件与外部连接,后期改为 DB9 接插件。现在大多数微型计算机已不再配置这种低速的串行

接口。

第5章将进一步介绍串行通信的原理与实现。

2. 键盘接口

键盘由单片微处理器构成的控制器和16行×8列的按键阵列组成。早期的PC使用83键的键盘，现在以104键的键盘为主。键盘接口电路集成在主板的芯片组中，通过一个5芯电缆与键盘相连接。电缆内各信号如表3-4所示。

表3-4 键盘电缆信号

信号名称	传输方向	作用
时钟信号	接口↔键盘	双向时钟信号
数据信号	键盘↔接口	串行发送扫描码
空/复位	接口→键盘	接口对键盘进行复位操作
GND		电源地线
V_{CC}/+5V	接口→键盘	电源线

键盘控制器在监测到某个键按下后，根据按键的行、列位置，生成这个键的扫描码，以串行方式传送到键盘接口电路。接口内部的移位寄存器逐位接收，将它们组合成为并行代码，最终送入可编程并行接口芯片8255A的A端口，并向主机发出键盘中断。

键盘中断服务程序主要功能如下：

(1) 从键盘数据输入端口(8255A的A端口，地址60H)读取键盘扫描码。

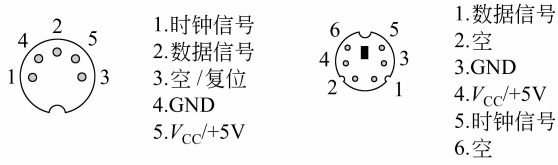
(2) 将扫描码转换成ASCII码或扩展码，存入键盘缓冲区。

(3) 如果是换档键(如Caps Lock、Ins等)，将状态存入BIOS数据区的键盘标志单元。

(4) 如果是组合键(如Ctrl+Alt+Del)则直接执行，完成其对应的功能。

(5) 对于中止组合键(如Ctrl+C或Ctrl+Break)，强行中止程序的执行，返回系统。

PC上使用的传统键盘插口有两种：直径13mm的5芯PC键盘插口和直径8mm的6芯PS/2键盘插口(见图3-16)。新型键盘使用USB接口和无线接口。



(a) 5芯接口　　(b) PS/2接口

图3-16 键盘接插件

3. 鼠标接口

鼠标是一种相对定位设备，功能与键盘的光标移动键相似。通过移动鼠标可以快速定位屏幕上的对象，是计算机图形界面人机交互的必备外部设备。

鼠标按其结构可分为为光电机械式、光电式、轨迹球和新型的无线鼠标等。

光电机械式鼠标内置了3个滚轴：X方向滚轴和Y方向滚轴，另1个是空轴。这3个滚轴都与一个可以滚动的橡胶球接触，并随着橡胶球的滚动一起转动。X、Y滚轴上装