第 5 章　微型计算机输入输出接口

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，微型计算机和外部设备数据传输的方法，I/O 接口程序设计，PC 系列微型计算机常用外部设备的接口。

5.1　输入输出接口

5.1.1　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：数据信息从外部设备送往 CPU。

（2）输出设备：数据信息从 CPU 送往外部设备。

（3）复合输入输出设备：数据信息在 CPU 与外部设备之间双向传输。

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、显示器。

（2）数据存储设备：软盘、硬盘、光盘驱动器和 U 盘。

（3）媒体输入输出设备：扫描仪、打印机等。

（4）数据采集与设备控制：模拟量输入转换设备、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、图形信息。

（2）控制信号：CPU 以一组二进制向外部设备发出命令，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。

① 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。

② 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。

③ 有的设备有指示出错状态的信号，如打印机的纸尽（Paper Out）、故障（Fault）。

数据信号、控制信号、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。

5.1.2　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连接的界面。I/O 接口用来连接计算机系统

总线与外部设备，它具有如下功能。

(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部设备地址进行译码，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来自外部设备的数据或状态信息发往数据总线。

(3) 数据格式转换功能。当外部设备使用的数据格式与 CPU 数据格式不同时，就需要接口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、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检验。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、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。

5.1.3 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 和外部设备之间传输的数据、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 5-1 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 5-1　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。同时，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 采用的方式如下。

① 访问内存储器使用 20 根地址线 $A_0 \sim A_{19}$，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 16 根地址线 $A_0 \sim A_{15}$，同时使 $M/\overline{IO}=0$，I/O 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 中，使用 $A_0 \sim A_9$ 共 10 条地址线定义了 1024 个 I/O 端口(设 $A_{11} \sim A_{15}=0$)，地址范围为 0～3FFH。前 256 个端口地址供主板上接口芯片使用，后 768 个供扩展槽接口卡使用，部分主板用 I/O 端口分配情况列于表 5-2 中。

表 5-2　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	000～01FH	000～01FH
中断控制器 1	020～021H	020～021H
定时器	040～043H	040～05FH
并行接口芯片	060～063H	—
键盘控制器	—	060～06FH

续表

I/O 接口器件名称	PC/XT	PC/AT
RT/CMOS RAM	—	070～07FH
DMA 页面寄存器	080～083H	080～09FH
中断控制器 2	—	0A0～0BFH
NMI 屏蔽寄存器	0A0～0BFH	—
DMA 控制器 2	—	0C0～0DFH
协处理器	—	0F0～0FFH

5.1.4 输入输出指令

8086 CPU 采用内存与 I/O 端口独立编址方式，设置了一套独立的输入输出指令，用于 8 位/16 位端口的输入输出，如表 5-3 所示。

表 5-3 8086 输入输出指令

指　令	操　作	举　例	功　能
IN　ACC，PORT	AL/AX ←(PORT)	IN　AL，60H　;8 位输入指令 IN　AX，78H　;16 位输入指令	把指定端口中的数据读入 AL 或 AX 中
IN　ACC，DX	AL/AX←(DX)	MOV　DX，312H　;端口地址送入 DX IN　AX，DX　;16 位间接输入指令	
OUT　PORT，ACC	(PORT)←AL/AX	OUT　21H，AL　;8 位输出指令	把 AL 或 AX 中的数据向指定端口输出
OUT　DX，ACC	(DX)←AL/AX	MOV　DX，21H　;端口地址送入 DX OUT　DX，AL　;8 位间接输出指令	

输入输出必须通过累加器进行。8 位外部端口用 AL 进行输入输出，16 位外部端口用 AX 进行输入输出。

输入指令 IN 把外部设备接口输入端口(数据、状态)的信息读入累加器，8 位输入端口送 AL、16 位输入端口送 AX。输出指令 OUT 把累加器 AL/AX 的内容向 8 位/16 位输出端口(数据、命令)输出。

I/O 指令中，外部端口地址有两种寻址方式。端口地址为 0～255，可以用 8 位二进制数表示时，可以使用直接地址，端口地址以立即数的形式出现在指令中(高 8 位地址全为 0)。端口地址大于 255 时，必须把地址事先送入 DX 寄存器，通过该寄存器进行间接寻址。

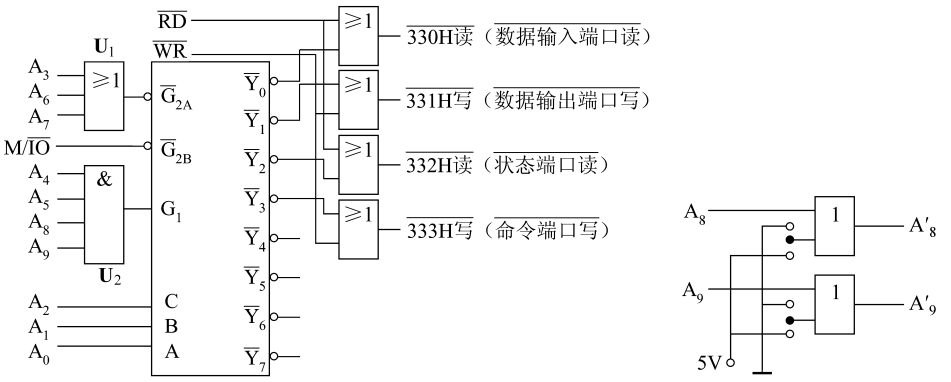
5.1.5 简单的 I/O 接口

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、命令端口的地址分别为 330H、331H、332H、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、命令端口。由此，在最小模式的系统中，可以设计如图 5-1 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 5-1　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 5-1(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、A'_9代替图 5-1(a)中的 A_8、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。

由于读操作和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、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 5-1(a)中右端的信号组合电路可做如图 5-2 的修改。

图中，Y_0、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$\overline{IORC}$，$\overline{IOWC}$代替上面的 M/$\overline{IO}$、$\overline{WR}$和$\overline{RD}$。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

在微型计算机系统数据总线上，连接着许多能够向 CPU 发送数据的设备，如内存储

器、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，要求所有的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 5-3 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。

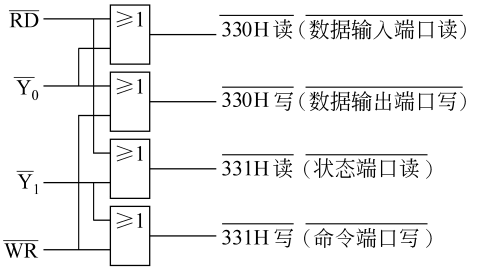


图 5-2　修改后的译码电路

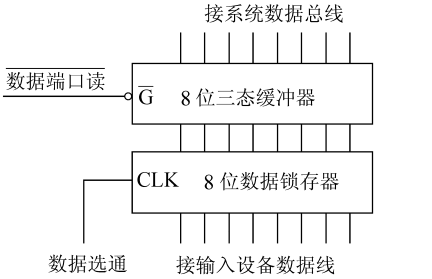


图 5-3　输入设备接口的数据锁存和缓冲电路

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 送往外部设备的数据或命令。一般应由接口进行锁存，以便使外部设备有充分的时间接收和处理。图 5-4 是一个 8 位输出锁存电路的例子。

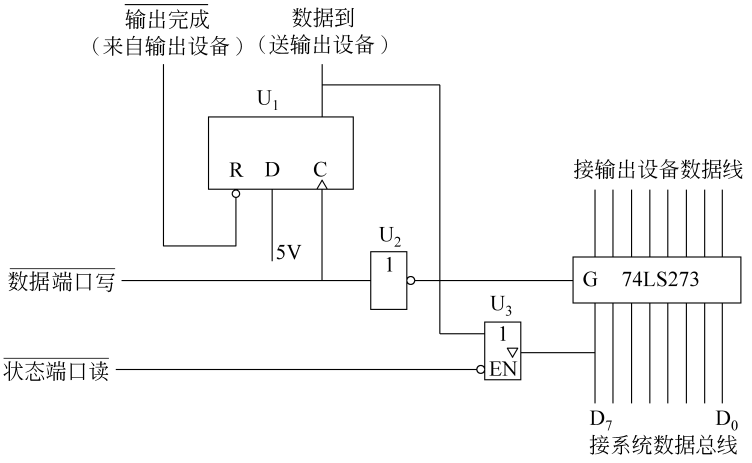


图 5-4　8 位输出锁存电路

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 $D_0 \sim D_7$ 输入 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“0”，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。如图 5-4 所示，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，如图 5-5 所示。它一方面与系统地址总线 A_0～A_{15}、数据总线 D_0～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。

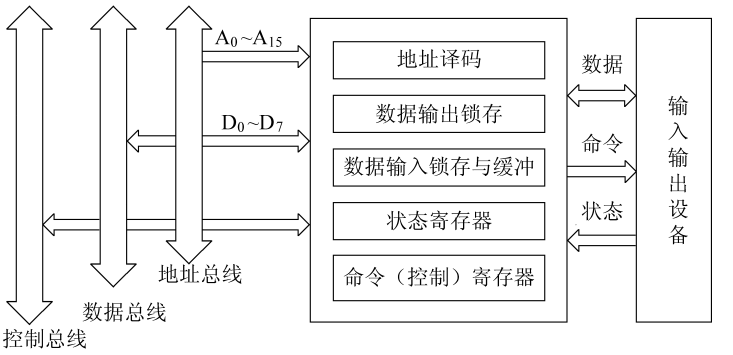


图 5-5　简单接口的组成

5.2　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：与内存储器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储器进行一次数据传输，而且这个过程可以连续进行。CPU 与外部设备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，两次击键的间隔时间往往是不确定的。因此，较之与内存储器的数据传输，CPU 与外部设备的数据传输有着不同的特点，因而也有着不同的处理方式。其传送方式概括起来有程序方式、中断方式、DMA 方式 3 种。

5.2.1　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，b)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，如图 5-6 所示。

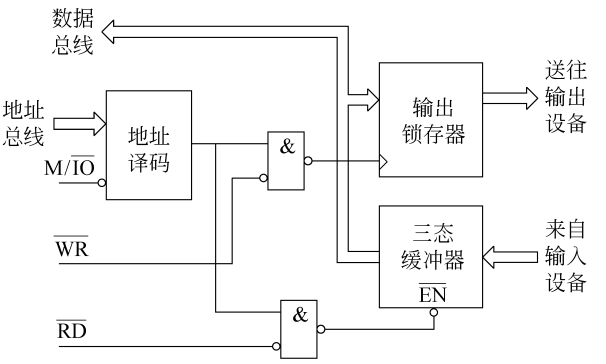


图 5-6　无条件输入输出接口

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。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＃电动机已通电运转，暗表示该电动机未通电等。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 5-6 给出了无条件传送时接口的组成方式，它是第 5.1 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部设备的状态。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，外部设备将数据准备好时，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，外部设备输出了一个数据后，接口便将状态端口的忙(BUSY)标志位清“0”。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) 如状态字表明外部设备已处于就绪状态，则传送数据。

图 5-7 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：一方面将外部设备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 从外部设备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部设备是否准备就绪，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“0”，设备又恢复到未就绪状态，本次数据传输到此结束。

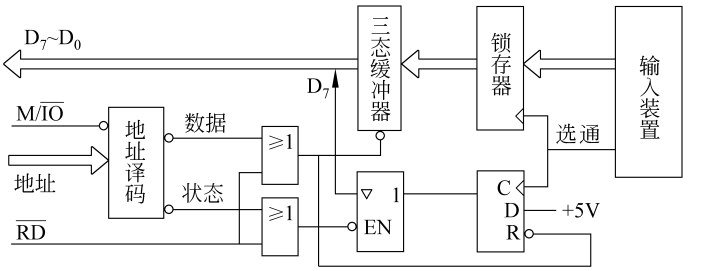


图 5-7　查询式输入接口电路

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 5-8 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 需要向一个外部设备输出数据时，先读取接口中的状态字，如果状态字表明外部设备空闲（不忙，BUSY＝0），说明可以向外部设备输出数据，此时 CPU 才执行数据输出指令，否则 CPU 必须等待。

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输入接口数据锁存器，同时将 D 触发器置“1”。D 触发器的输出信号一方面为外部设备提供一个联络信号 STB，告诉外部设备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部设备处于“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“0”，也把状态位清“0”，这样就可以开始下一个输出过程。

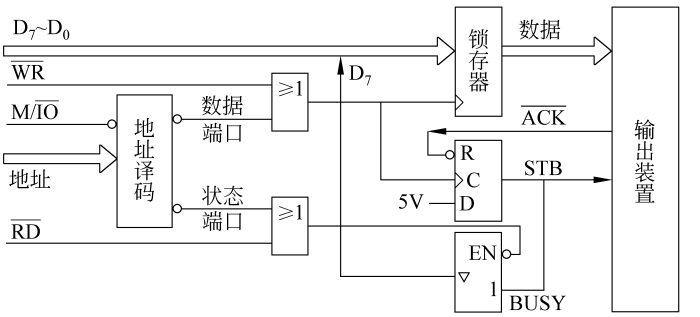


图 5-8　查询式输出接口电路

相应的汇编语言程序如下：

```
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 语言程序如下：

```
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 5-9 是查询式输入的程序流程图。

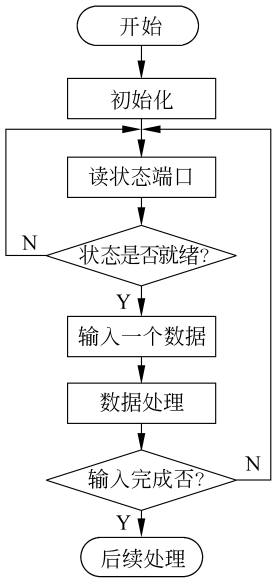


图 5-9　查询式输入流程

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。

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
```